Toward Accurate, Efficient, and Robust Hybridized Discontinuous Galerkin Methods

by

Johann Paul Schwind Dahm

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Aerospace Engineering and Scientific Computing)
in The University of Michigan
2017

Doctoral Committee:

Associate Professor Krzysztof Fidkowski, Chair
Professor Robert Krasny
Professor Georg May, RWTH Aachen
Professor Kenneth Powell

Johann Dahm

jdahm@umich.edu

ORCID iD: 0000-0001-9657-3564

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my advisor, Professor Krzysztof Fidkowski, for his guidance and insights into my research throughout my graduate work. I am lucky to have such an understanding and patient advisor and mentor. His direction adapted to what I needed: from high-level guidance to detailed attention to the project when I needed it. I admire that he not only has a deep commitment to research but also to creating outstanding courses at both the graduate and undergraduate levels, and in both cases clearly explains even the most complicated concepts. The example he set drove me to accomplish this work.

I would also like to thank my committee members: Professors Robert Krasny and Ken Powell at the University of Michigan, and Professor Georg May with whom I had the good fortune of working in Aachen. Thanks especially to Robert Krasny, for stepping in and serving as a cognate, and for providing detailed feedback on my entire dissertation; to Ken Powell for providing me guidance since the beginning of my graduate work; and to Professor Georg May for his help with and detailed knowledge of hybridized discontinuous Galerkin methods.

My fellow research group members, both past - Marco Ceze, Isaac Asher, and Steve Kast - and present - Kyle Ding, Yukiko Shimizu, Devina Sanjaya, Guodong Chen, and Kevin Doetsch - deserve much thanks. To Marco Ceze for his tireless efforts developing the XFlow code into what it was when I started so I could focus my contributions to the code where it was most needed; to my former roommate Isaac Asher for teaching me cooking skills; and to Steve Kast for his close friendship over the years. To all the present group members, your friendships kept me sane throughout the years. Thanks also to Peter Klein, with whom I started the hybridized discontinuous Galerkin project, for often working late into the night together on campus.

There are a number colleagues and academics outside my research group who have helped me along this journey that deserve a great deal of thanks. To Sara Spangelo, my office-mate and exercise partner, for helping me toward the path I am on now, and Derek Dalle for setting a great example with his coding practices and expert knowledge of even the most esoteric MATLAB. Christian Heinrich, for being such an

inspirational GSI and office mate over the years, and Daniel Zaide for providing a positive distraction from work. Ian Tobasco, for always offering to discuss research questions and helping with proofs. Professor Smadar Karni, for her inspirational hyberbolic equations reading group and offering encouragement throughout my graduate work. Michael Woopen, for our work together on hybridized discontinuous Galerkin methods, letting me live with him while in Aachen, and helping me navigate the city.

Finally I would like to thank my family. To my parents for their constant support over the years and my brother for setting a good example of work ethic as he finished his graduate work. Most of all I would like to thank my finacée Mara Kutter, for standing by me and tolerating my stress over the last four years, and my dog Phoebe for making sure I regularly enjoyed the outdoors.

Funding for this work was provided by the Predictive Science Academic Alliance Program (PSAAP), the National Science Foundation Graduate Research Fellowship Program (GRFP), and the Beyster Computational Innovation Fellowship administered by the University of Michigan College of Engineering.

TABLE OF CONTENTS

LIST OF FIGURES

Figure

LIST OF TABLES

Table

ABSTRACT

Toward Accurate, Efficient, and Robust Hybridized Discontinuous Galerkin Methods

by

Johann Paul Schwind Dahm

Chair: Krzysztof Fidkowski

Computational science, including computational fluid dynamics (CFD), has become an indispensible tool for scientific discovery and engineering design, yet a key remaining challenge is to simultaneously ensure accuracy, efficiency, and robustness of the calculations. This research focuses on advancing a class of high-order finite element methods and develops a set of algorithms to increase the accuracy, efficiency, and robustness of calculations involving convection and diffusion, with application to the inviscid Euler and viscous Navier-Stokes equations. In particular, it addresses high-order discontinuous Galerkin (DG) methods, especially hybridized (HDG) methods, and develops adjoint-based methods for simultaneous mesh and order adaptation to reduce the error in a scalar functional of the approximate solution to the discretized equations. Contributions are made in key aspects of these methods applied to general systems of equations, addressing the scalability and memory requirements, accuracy of HDG methods, and efficiency and robustness with new adaptation methods.

First, this work generalizes existing HDG methods to systems of equations, and in so doing creates a new primal formulation by applying DG stabilization methods as the viscous stabilization for HDG. The primal formulation is shown to be even more computationally efficient than the existing methods. Second, by instead keeping existing viscous stabilization methods and developing a new convection stabilization, this work shows that additional accuracy can be obtained, even in the case of purely convective systems. Both HDG methods are compared to DG in the same computational framework and are shown to be more efficient.

Finally, the set of adaptation frameworks is developed for combined mesh and order refinement suitable for both DG and HDG discretizations. The first of these

frameworks uses hanging-node-based mesh adaptation and develops a novel local approach for evaluating the refinement options. The second framework intended for simplex meshes extends the mesh optimization via error sampling and synthesis (MOESS) method to incorporate order adaptation.

Collectively, the results from this research address a number of key issues that currently are at the forefront of high-order CFD methods, and particularly to output-based hp-adaptation for DG and HDG methods.

CHAPTER 1

Introduction

1.1 Motivation

Today, advances in computing power and the availability of sophisticated numerical methods with supporting algorithms have made computational science, including computational fluid dynamics (CFD), an indispensable tool for scientific discovery and engineering design. An enormous range of scientific and engineering applications now routinely make use of – and even rely on – large-scale multi-physics computational simulations for analysis and design of complex problems. Examples range from fluid dynamics and aeronautics to combustion and industrial processes, from material science and structural mechanics to heat transfer and thermomechanical systems, and countless other fields that span in their objectives from the pursuit of scientific discovery to engineering optimization and design of practical devices.

Each of these application areas involves governing equations that describe their underlying physics, and thus each requires numerical methods that are suited to solving the particular equations at hand. Increases in computing power and the development of computing architectures have been responsible for much of the expanded use and increased capabilities of simulation in these fields. Equally important, however, has been the development of numerical methods in applied mathematics and their implementation in sophisticated algorithms that allow these computing resources to be accurately, efficiently, and robustly applied to solving some of the most complex and relevant problems in these fields.

Although some of these application areas extend beyond computational fluid dynamics, the focus of this dissertation is on methods that are especially suited to numerically solving the partial differential equations (PDEs) that govern fluid dynamics, particularly the inviscid Euler equations and the viscous Navier-Stokes equations. Specifically, this work focuses on developing a particular class of numerical

algorithms that is potentially suited for simulating convection and diffusion processes with substantially greater accuracy and computational efficiency than is possible with currently available methods.

A key challenge facing all such algorithms is the need to simultaneously achieve improved computational accuracy, efficiency, and robustness over as wide a range of simulation parameter values as possible. These may include complexities in the geometry of the solution domain itself, as well as values of physical parameters such as the viscosity that can affect the accuracy and stability of the computational solution. In practice, achieving a useful balance of accuracy, efficiency, and robustness over more than a narrow range of parameter values can be difficult. As a consequence, many algorithms that achieve improved accuracy and efficiency are insufficiently robust for widespread use – they often require delicate management of interrelated inputs by expert practitioners to obtain a stable simulation, thus limiting their overall impact.

The central question is therefore whether it is possible, keeping in mind the broad range of applications, to conceive and implement specific numerical algorithms that demonstrate substantially improved accuracy and efficiency, which at the same time are sufficiently robust in the hands of any user, regardless of the application area. The remaining sections of this chapter outline the particular class of solution approaches being considered here, and the particular aspects of the solution accuracy and efficiency that the numerical methods and corresponding algorithms developed herein have sought to advance.

1.2 Background

This work focuses on advancing adaptive high-order finite element methods for computational simulations involving convection and diffusion. Governing equations involving convection and diffusion are central to CFD, though the ideas presented here are applicable to many other types of governing equations as well. In particular, this work addresses the broad class of solution approaches referred to as high-order adaptive discontinuous Galerkin (DG) methods. More specifically, there are two primary areas of emphasis in this work. The first is to develop advances in hybridized discontinous Galerkin (HDG) methods for CFD applications, and the second is to create adaptation algorithms for simultaneously adapting the mesh and solution polynomial representation to achieve improved accuracy for a given computational cost.

Section 1.2.1 thus first provides background information on high-order DG methods in general, which is essential for understanding the contributions of this work to

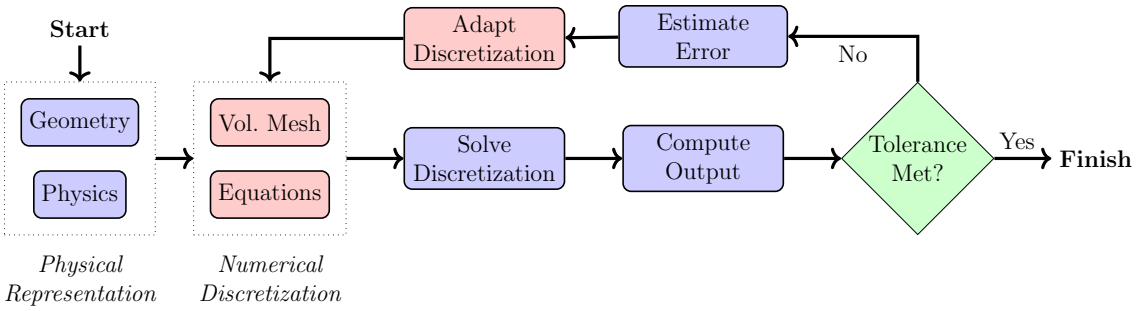


Figure 1.1: General solution procedure when using an error estimation and mesh adaptation process for a finite element computation. The main contributions from this work are in the areas shown in red.

HDG methods. Section 1.2.2 then gives additional background on the error estimation and adaptation methods used in this work, specifically the adjoint-based algorithms for determining how both the error estimate and the simulation should be adapted in response to this estimate. It also provides further background on mesh adaptation, in which the mesh size h is locally reduced for a given order p to obtain greater simulation accuracy. Section 1.2.3 then provides background on the hybridization approach that leads to HDG methods, which are more computationally efficient than existing DG methods for high orders.

Before presenting this background information, it is useful to first consider the broad overview illustrated in Figure 1.1 of the steps that are involved when applying these types of computational methods. For a given system of governing equations corresponding to the underlying physics, the first step is to describe the problem geometry and the equation set to be solved. Depending on the particular problem, the geometry may be quite simple and thus can be generated manually, but often it is sufficiently intricate to require separate computational generation of the geometry, as for example when simulating the flow over an entire aircraft. Then, since finite element discretizations use meshes to describe approximate solutions within the computational domain, a volumetric mesh of the domain must be created. Since the underlying physics will be represented on the mesh, to better represent the solution for a given approximation order the mesh size distribution should ideally be smaller in those regions of the domain that have a greater effect on the output of interest from the simulation. Conversely, the mesh size h can be coarser in other regions to which the output of interest is less sensitive. For instance, if the output of interest is the lift or drag on a wing, then the streamlines of the flow leading up to and including the boundary layer will be critical to obtaining these quantities accurately, but the

domain far from the wing may have little effect.

For complex geometries or problem descriptions, *ab initio* generation of such an efficient mesh suited to the governing equations, the geometry, and the key output of interest can be time-consuming at best, even when guided by expert practitioners. More important, it is exceedingly difficult even for experts to know *a priori* what the relative importance is of different regions in the domain for predicting the key output of interest. For these reasons it is desirable to provide a solution methodology that can, from an initial approximation of such an efficient mesh, adapt the mesh in such a way as to refine or reduce the mesh size h where doing so provides the greatest benefit in terms of accuracy in the output of interest, and coarsen or increase h in other regions of the domain. A combined hp adaptation approach goes even further by reducing the mesh h or increasing the order p in those regions that have the greatest benefit on the output of interest, and in a way that best captures the output of interest, while increasing h and reducing p in other regions.

Such adaptation in turn requires a method for error estimation to determine the extent to which each region in the computational domain affects the output of interest. These methods estimate the error in the output of interest from the simulation, and provide the adaptation algorithm a field indicating the relative importance of each region for approximating the output. This procedure relies on an error indicator that is relatively inexpensive to obtain. The adaptation algorithm can either refine the representation of the solution in those regions, or it can even try to optimize the representation altogether. By repeatedly solving and adapting, the error in the output of interest can be minimized to a prescribed tolerance.

1.2.1 High-Order Adaptive Discontinuous Galerkin Methods

In general, finite element methods provide access to high-order accuracy by increasing the polynomial order of the underlying numerical representations. The present work falls in this class of methods, and in particular it focuses on discontinuous Galerkin methods, namely finite element methods that use piecewise high-order polynomials for the approximate solution, or state, on mesh elements in the solution domain.

The general interpolation error of the solution of a mesh-based numerical method with an average element length scale h can be written as $\mathcal{O}(h^r)$, where r characterizes the order of accuracy in the global error with decreasing mesh size h. Any consistent numerical method will converge to the exact solution in certain norms as $h \to 0$, but different methods do so at differing rates r. It is generally agreed that a high-order

method is one with $r > 2$. Thus to achieve a given numerical accuracy, a low-order numerical method – namely one with comparatively small r – must use a highly refined mesh h to achieve a desired solution accuracy, while a high-order numerical method, with a larger r value, can achieve the same error with a coarser mesh.

However, there exists a trade-off between mesh resolution and order of accuracy for any method, as low-order methods will be comparatively less computationally expensive for a given mesh. Thus high-order methods are not automatically more efficient, though they are generally found to be more efficient for high-fidelity CFD calculations [1]. More concretely, Fidkowski [2] finds that assuming a solution error norm converging at rate r, so that the error $E \sim \mathcal{O}(h^{p+1})$, the time to solution T in terms of the physical dimension d, the computational speed F, and the algorithm complexity a is

$$\log T = d\left(-\frac{1}{p+1}\log E + a\log\left(p+1\right)\right) - \log F + \text{constant}.$$

For high-fidelity calculations $E \ll 1$, so the computational time T depends exponentially on the ratio $d/(p+1)$, clearly favoring high-order methods.

DG methods use piecewise polynomials to describe the field inside each element, so these methods extend easily to high order while maintaining a compact stencil. It is the compact stencil that is central to the effectiveness of DG methods, since the state within any element depends only on those of nearest neighboring elements. While high-order methods such as DG are known to provide high convergence orders for sufficiently smooth or regular problems, the convergence order can be limited by the presence of singularities. However by focusing mesh refinement around singularities, adaptive DG methods are able to effectively recover the high-order convergence inherent in DG methods.

Yet, whether high-order methods are more computationally efficient than low-order methods has been a topic of intense study, especially in CFD, where high-order methods have thus far not seen widespread use in practical codes aimed at non-expert users. Several high-order CFD workshops have sought answer this question, by comparing state-of-the-art implementations of adaptive high-order methods against industry-optimized low-order methods that typically emphasize robustness over accuracy and efficiency. From these it is generally agreed that for smooth flow fields, where high-order accuracy is possible, high-order methods such as DG outperform low-order methods. However, for complex geometries involving non-smooth fields, these benefits are not always easily obtained. The current consensus can be generally

summarized as follows [3]:

- For problems with smooth solutions, high-order methods outperform highly-optimized second-order finite volume codes in terms of error for a given cost measured in a non-dimensionalized compute time.
- Even for non-smooth problems, where high-order methods fail to optimally converge, high-order methods can outperform low-order methods when coupled with effective adaptation algorithms. However more work is needed to make adaptation robust, and insufficient evidence exists to state with certainty that high-order methods will perform better.
- High-order methods often require too much memory for routine use.
- Coupling mesh (h) and order (p) adaptation in a hybrid scheme can be a highly effective technique for reducing error, especially in difficult non-smooth problems.
- High-order methods are insufficiently robust for Reynolds-averaged Navier-Stokes (RANS) simulations.

Thus, while work to date has shown that high-order methods such as DG, and especially hybridized methods such as HDG, can potentially outperform conventional CFD approaches for certain types of problems, DG and HDG methods to date remain insufficiently mature to serve as a basis for practical CFD codes aimed at non-expert users.

1.2.2 Error Estimation and Adaptation

Error estimation is central to the development of computational algorithms. Even for algorithms that do not use adaptation, an estimate of the error is needed to quantify the extent to which the simulation result can be considered trustworthy. For simple test cases, best-practice guidelines can be used to indicate how solution features may affect the outputs. However designing such guidelines can be time-consuming and requires expert knowledge, and even then their applicability to more complex problems may be questionable. For algorithms that use adaptation, error estimation by the simulation itself is essential to determine the degree of adaptation needed and where adaptation can be most productively applied.

Since there are multiple sources of error in a simulation, it is important to first understand what question the simulation is being asked to answer, and how various

sources of error will affect the corresponding solution. Discretization error arises from differences between the exact solution to the governing equations and the discrete solution of the discretized forms of these equations. Since the exact solution is generally unknown, discretization error must typically be estimated in some way. Approaches for doing so include both *a priori* estimates, which appeal to interpolation arguments, and *a posteriori* estimates using the difference between the approximate solution and a finer representation of the solution, usually by means of a reconstruction process. Another source of error, termed residual error, results from measuring how well an approximate solution satisfies the exact equations. This itself can be approximated by evaluating the residual error in a finer space discretization. The locations of such non-zero residuals indicate, in a sense, where the approximate solution is not representing the exact solution well. Both residual and discretization errors can be used to form a global error estimate, or to form a local indicator that identifies regions in which adaptation can be most productively implemented to better approximate the overall solution.

This work, however, concerns itself with reducing the error in a scalar output functional of the approximate solution. Although using indicators of the type noted above to refine the approximation will eventually lead to a more accurate output value, doing so is unlikely to correctly prioritize the regions where refinement can be most productively applied, and may altogether miss some regions where adaptation may be most beneficial. To make matters worse, such indicators only identify where the errors *occur*, but they do not identify where the refinement should be targeted to reduce the *sources* of these errors that affect the output functional. Fortunately, there are approaches for estimating the error in such output functionals that correctly identify the regions of the domain that affect the *source* of the errors in the outputs.

This work focuses on one such approach, using the solution of the *adjoint* problem for the given output functional – where the adjoint field represents the sensitivities of the output functional to the residuals – to identify regions in which adaptation can be most productively applied. By using the adjoint field based on the specified output functional, the error estimation can provide a local adaptation indicator that is focused on reducing the error in the output of interest. Such an indicator aligns the adaptation with the question that the simulation is being asked to answer. The indicator is then passed to the adaptation procedure to change the mesh or the discretization to reduce the error in the output. Chapter 3 introduces the adjoint and describes its connection to error estimation, then describes how it is used to create both a global error estimate and localized indicator.

With regard to the adaptation itself, in effect the goal is to equidistribute the error. The adaptation can either take an initial mesh and discretization order and successively refine regions until a prescribed tolerance is met, or it can use an optimization approach in which regions are refined or coarsened so that the output error is minimized for a given number of degrees of freedom (DOF) in the solution approximation. High-order finite element methods, such as DG, can refine the solution approximation by changing the local size of the mesh elements (h-adaptation), the local polynomial order of the solution (p-adaptation), or by a combination of these (hp-adaptation). DG methods make p-adaptation especially convenient, since the polynomial representations are discontinuous across mesh elements, allowing purely local changes to be readily made.

Mesh adaptation generally resizes the elements to focus resolution where it is needed. The mesh may be edited by a number of operations, including locally inserting new elements or by globally re-meshing. In regions with strong spatial inhomogeneities, such as in boundary layers or other anisotropic features, the mesh adaptation algorithm should refine elements so their length scale is reduced along high-gradient directions and increased along largely homogeneous directions, to minimize the overall number of elements needed to represent the solution.

The error for a given mesh size h converges as h^r and thus is polynomial in the number of degrees of freedom, scaling as asymptotically as DOF^{-r}. In regions with strong spatial gradients, where the underlying solution has low regularity, the discrete solution can be better represented by adding elements locally to reduce h than by increasing the local polynomial order p within mesh elements. For instance near a discontinuity, increasing DOF by increasing the order p will have little effect on the accuracy. On the other hand, when raising the order p in regions of high regularity, the power r increases, so an extremely fast exponential rate of convergence $\mathcal{O}(\mathrm{DOF}^{-r})$ is possible. This is not always the case since the constant in this relationship can be quite large, in which case it could be preferable to refine the mesh, even in smooth regions. Thus a given accuracy may be most efficiently achieved by a combined hp-adaptation method that automatically selects the most productive type of refinement for each individual location throughout the mesh.

1.2.3 Hybridization

Part of this work deals with HDG methods, so this section provides background on hybridized finite element methods in general as a basis for the technical description in Chapter 4. Hybridization reduces the effective dimensionality of a method and

thereby greatly reduces the number of DOF that must be solved for, albeit at the cost of an additional solve that must be performed. However this additional solve is purely *local*, and thus can be done entirely in parallel to maintain the increased computational efficiency that hybridization offers.

Finite element methods in general seek a solution on elements that comprise the computational domain, resulting in a linear system, for example when using an implicit time-stepping method. The traditional approach to solving the system using parallelization is to partition the domain into non-overlapping subdomains and assign one to each processor. Such a domain-decomposition method then solves for the approximate solution by communicating with neighboring processes. Removing parallelization for the moment, this procedure can in certain cases be considered as solving for a set of coupling conditions between the processors so that the discretization is globally satisfied. This idea is used in the creation of various preconditioners, for instance the family of finite element tearing and interconnect (FETI) and balancing domain decomposition (BDDC) methods. The coupling conditions take the form of Lagrange multipliers for the system, such that each subdomain only couples to the multipliers rather than to the neighboring sub-domain.

A hybridized finite element method occurs when the number of sub-domains equals the number of finite elements. In this case the multipliers can be thought of as an additional approximate solution variable on the dual mesh – linking faces across elements – associated with the original computational mesh. For the class of hybridized finite element methods – specifically for HDG methods considered in this work – the set of multipliers, called traces, can be considered a sort of boundary condition for a local problem on each of the elements. Solving for the traces on the faces instead of for the approximate solution itself is an attractive property since a discretization in d dimensions using order p polynomials requires only DOF $\sim \mathcal{O}(p^{d-1})$ to represent the traces, instead of DOF $\sim \mathcal{O}(p^d)$ to represent the elemental approximate solutions. Note however that this is only the scaling; for low orders it is often the case that the number of DOF are similar or higher for HDG discretization. Since the scaling on the size of the resulting global linear system is thus reduced by one order, for high orders p the resulting system is substantially smaller than its non-hybridized counterpart.

There is a price to pay, however, since now additional local solvers are required to solve for the elemental approximate solution in terms of the multipliers. In effect, the hybridized method trades a larger linear system that requires more memory for a smaller system that requires an additional local solve. However parallellization allows these additional local solves to be done entirely in parallel, providing a huge benefit

from hybridization in terms of the overall computational efficiency.

The HDG method in Chapter 4 belongs to the class of methods that takes advantage of massively parallel simulation architectures to gain this inherent efficiency benefit from hybridization. When coupled with the additional accuracy and efficiency advantages of the *hp* adaptation in Chapter 5, the resulting high-order HDG approach offers substantially increased computational accuracy and efficiency.

1.3 Dissertation Overview

This dissertation presents results from research into development of numerical methods for increasing the accuracy, efficiency, and robustness of high-order hybridized discontinous Galerkin methods through combined *hp* adaptation. The methods developed herein are primarily directed at simulations involving convection and diffusion, particularly simulations of the inviscid Euler equations and the viscous Navier-Stokes equations, though they can be applied to other types of equations as well. The main objective is to provide substantial improvements in the accuracy and computational efficiency of such simulations, while providing sufficient robustness to allow their implementation in codes intended for non-expert users.

1.3.1 Organization

The remainder of this dissertation is organized as follows:

- Chapter 2 introduces the governing equations and notation used to describe the methods, and discusses a class of discontinuous Galerkin (DG) methods that can be used to discretize these equations.
- Chapter 3 presents the output-based error estimation that will be used in a later chapter as part of an *hp* adaptation algorithm for this class of DG discretization methods.
- Chapter 4 introduces a hybridized discontinous Galerkin (HDG) discretization for a general convection-diffusion transport system, and presents results of a thorough comparison to DG methods for both scalar advection-diffusion and the Navier-Stokes system.
- Chapter 5 presents a new approach for combined *hp* adaptation for both DG and HDG methods, and provides numerical results from test cases that demonstrate

the benefits in terms of improved computational accuracy and efficiency from this new approach.

- Chapter 6 summarizes the principal conclusions drawn from the results presented in the preceding chapters.

1.3.2 Major Contributions

This dissertation addresses a number of key issues that were identified in the *1st International Workshop on High-Order CFD Methods* [3], including the scalability of methods and memory requirements, robustness of the resulting simulations, and the practical value of output-based *hp*-adaptation in high-order methods. The specific contributions made by this dissertation are as follows:

- Generalization of the existing HDG methods to systems of equations. This involves application of existing DG stabilization methods to serve as the viscous stabilization for HDG.

- Development of a new form of the HDG method to systems of convection-diffusion equations suitable for highly-convective systems resulting from high-Reynolds number conditions with Navier-Stokes equations.

- A direct comparison of DG to both forms of HDG methods in the same computational framework for both the scalar convection-diffusion equation and the Navier-Stokes system.

- Development of a novel hanging-node-based *hp*-adaptation framework for both DG and HDG discretizations.

- Extension of the mesh optimization via error sampling and synthesis (MOESS) method to *hp*-optimization for both DG and HDG discretizations.

CHAPTER 2

Governing Equations and Discretization

While the methods investigated here are developed with the intention of applying them to particular sets of equations, specifically the compressible Navier-Stokes equations and their Reynolds-averaged form, they are formulated for a general set of convection-diffusion equations arising from a system of conservation laws. This chapter thus first introduces the general convection-diffusion system in the notation that will be used throughout, and then identifies the specific equations that will be solved with the methods developed herein. It then describes the existing DG discretization of these equations.

2.1 Equations and Notation

The general system of convection-diffusion equations for a state vector $\mathbf{u} \in \mathbb{R}^s$ of s solution components defined on a d-dimensional domain, $\Omega \subset \mathbb{R}^d$, will be written here as

$$\partial_t \mathbf{u} + \partial_i \mathbf{H}_i \left(\mathbf{u}, \nabla \mathbf{u} \right) + \mathbf{s}(\mathbf{u}, \nabla \mathbf{u}) = \mathbf{0}, \tag{2.1}$$

where $\mathbf{H}_i \in \mathbb{R}^s$ is the i^{th} spatial component of the total flux, $1 \leq i \leq d$ indexes the spatial dimension d with summation implied on repeated indices, and $\mathbf{s}$ is a source term. The total flux is decomposed into convective and diffusive terms

$$\mathbf{H}_i \left(\mathbf{u}, \nabla \mathbf{u} \right) = \mathbf{F}_i \left(\mathbf{u} \right) + \mathbf{G}_i \left(\mathbf{u}, \nabla \mathbf{u} \right), \tag{2.2}$$

where $\mathbf{F}_i \in \mathbb{R}^s$ is the i^{th} component of the convective or inviscid flux, and $\mathbf{G}_i \in \mathbb{R}^s$ is the i^{th} component of the diffusive or viscous flux for all equations. The diffusive flux is always linear in the gradient, involving $\mathbf{K}_{ij} \in \mathbb{R}^{s \times s}$, namely the (i, j) component of

the viscous diffusivity tensor, in the form

$$\mathbf{G}_i\left(\mathbf{u}, \nabla\mathbf{u}\right) = -\mathbf{K}_{ij}\left(\mathbf{u}\right)\partial_j\mathbf{u}. \tag{2.3}$$

2.1.1 Compressible Navier-Stokes Equations

The Navier-Stokes equations describe momentum conservation in the motion of a viscous fluid via Newton's second law of motion, including convection and diffusion effects. Together with the conservation of mass and energy these represent a system of $d+2$ equations in the convection-diffusion form. Specifically, the equations are

$$\begin{aligned} &\partial_t\rho && +\partial_j\left(\rho v_j\right) && = 0 \\ &\partial_t\left(\rho v_i\right) && +\partial_j\left(\rho v_i v_j + p\delta_{ij}\right) && = \partial_j\tau_{ij}, \quad i = 1\ldots d \\ &\partial_t\left(\rho E\right) && +\partial_j\left(\rho v_j H\right) && = \partial_j\left(v_j\tau_{ij} + q_i\right), \end{aligned} \tag{2.4}$$

where the variables are defined as:

Symbol	Definition
ρ	Fluid density
v_i	Velocity component (i^{th} dir.)
E	Specific total energy
H	Specific total enthalpy
$\vec{q}$	Heat flux vector
τ_{ij}	Viscous stress tensor

Using the state vector of solution components $\mathbf{u} = (\rho, \rho v_1, \ldots, \rho v_d, \rho E)^t$, where t denotes transpose, these can be rewritten in the general transport system form (2.1). The resulting convective and diffusive flux components in the i^{th} direction defined in (2.2) are

$$\mathbf{F}_i = \begin{pmatrix} \rho v_i \\ \rho v_d v_i + p\delta_{id} \\ \rho v_i H \end{pmatrix}, \quad \mathbf{G}_i = -\begin{pmatrix} 0 \\ \tau_{id} \\ v_j\tau_{ij} - q_i \end{pmatrix}, \tag{2.5}$$

where the second row containing the momentum conservation for ρv_d represents d collapsed equations.

Fourier's law dictates that the heat flux is directly proportional to the gradient of the temperature T via the thermal conductivity k_T as

$$q_i = -k_T\partial_i T, \tag{2.6}$$

where T is the temperature, a thermodynamic quantity related to the average translational kinetic energy of molecules in the fluid. The pressure p is a property that measures the average perpendicular force per unit area from molecular collisions against a surface. Here we assume that the fluid has negligible intermolecular forces and is always in thermodynamic equilibrium, and thus the ideal gas law connects the temperature, pressure, and density via the gas constant $R \equiv \bar{R}/\mathrm{MW}$, where $\bar{R}$ is the universal gas constant and MW is the average molecular weight of the gas, as

$$p = \rho R T. \tag{2.7}$$

The (specific) total enthalpy H is simply a definition of the energy that additionally accounts for the amount of work done to make room for a unit mass of fluid, namely

$$H = E + \frac{p}{\rho}. \tag{2.8}$$

Furthermore, if the constant-volume and constant-pressure specific heats of the gas can both be taken as constants, the gas is calorically perfect and can be closed by the equation of state

$$p = (\gamma - 1)\left(\rho E - \frac{\rho |\mathbf{v}|^2}{2}\right) \tag{2.9}$$

where the ratio of the specific heat γ is then also a constant property of the fluid.

The stress tensor τ_{ij} was derived by Stokes in 1845 according various observations to take the form $\tau_{ij} = 2\mu\epsilon_{ij} + \lambda\, \partial_l v_l\, \delta_{ij}$, where

$$\epsilon_{ij} = \frac{1}{2}(\partial_i v_j + \partial_j v_i) \tag{2.10}$$

is the strain rate tensor and μ is the dynamic viscosity. Adopting Stokes' hypothesis that equates the average normal stress in a deforming fluid to the thermodynamic definition of pressure is equivalent to assuming $\lambda = -2/3\mu$, which then simplifies the stress tensor as

$$\tau_{ij} = 2\mu\left(\epsilon_{ij} - \frac{1}{3}\,\partial_l v_l\, \delta_{ij}\right). \tag{2.11}$$

The fluid properties $\mu, k_T, \gamma,$ and R need to be specified to complete the model. Results in this work use the properties listed in the table below unless otherwise stated.

Dynamic viscosity	$\mu = \mu_{\text{ref}} \left(\frac{T}{T_{\text{ref}}}\right)^{\frac{3}{2}} \left(\frac{T_{\text{ref}} + T_s}{T + T_s}\right)$ $T_{\text{ref}} = 288.15\,\text{K},\ T_s = 110\,\text{K}$
Thermal conductivity	$\kappa_T = \frac{\gamma \mu R}{(\gamma - 1)\,\text{Pr}}, \quad \text{Pr} = 0.71$
Gas constant	$R = 8.314\ \text{J}/(\text{mol} \cdot \text{K})$
Specific-heat ratio	$\gamma = \frac{7}{5}$

2.1.2 Reynolds-Averaged (RANS) Equations

The Spalart-Allmaras (SA) model closes the ensemble (Reynolds) averaged Navier-Stokes equations via a further transport equation for a turbulent viscosity ν. Turbulence is inherently an unsteady phenomenon, and initial condition perturbations in time lead to chaotic results, so the system is usually analyzed by decomposing the equations into mean velocity $\overline{\vec{v}}$ and perturbations about the mean $\vec{v}\,'$. When solving for the mean velocities, or in the case of compressible flow the momenta $\rho\vec{v}$, a problem exists in that the stress tensor contains terms of the form $\rho\overline{v_i' v_j'}$, called the Reynolds' stresses, which have no known universal form. SA is one of many approaches to modeling this quantity by augmenting the fluid viscosity via an added eddy viscosity. This eddy viscosity is modeled in terms of a kinematic turbulent viscosity $\tilde{\nu}$ that satisfies a transport equation with an empirically defined source term. Many of the model relations employ the non-dimensionalized kinematic viscosity $\chi = \tilde{\nu}/\nu$.

In the SA model the eddy viscosity μ_t is represented as

$$\mu_t = \begin{cases} \rho\tilde{\nu} f_{v1}(\chi) & \tilde{\nu} \geq 0 \\ 0 & \tilde{\nu} < 0 \end{cases} \qquad f_{v1}(\chi) = \frac{\chi^3}{\chi^3 + c_{v1}^3}. \tag{2.12}$$

With this, the heat flux and stress tensor are augmented in the original compressible Navier-Stokes system to

$$q_j = \frac{\gamma R}{\gamma - 1} \left(\frac{\mu}{\text{Pr}} + \frac{\mu_t}{\text{Pr}_t}\right) \partial_i T, \quad \tau_{ij} = 2\,(\mu + \mu_t) \left(\epsilon_{ij} - \frac{1}{3}\,\partial_l v_l\,\delta_{ij}\right). \tag{2.13}$$

An additional equation is added to the system to solve for $\rho\tilde{\nu}$ in (2.12) via

$$\partial_t(\rho\tilde{\nu}) + \partial_j(\rho u_j\tilde{\nu}) = \frac{1}{\sigma}\rho\partial_j\left((\nu+\nu')\,\partial_j\tilde{\nu}\right) + \frac{c_{b2}\rho}{\sigma}\partial_j\tilde{\nu}\partial_j\tilde{\nu} + P - D \tag{2.14}$$

where P and D are production and destruction terms, respectively. The additional viscosity ν' in the equation is given by the SA-Neg model variant as

$$\nu' = \begin{cases} \nu\chi, & \chi \geq 0 \\ \nu f_n(\chi) & \chi < 0, \end{cases} \qquad f_n(\chi) = \frac{c_{n1}+\chi^3}{c_{n1}-\chi^3}. \tag{2.15}$$

Work on turbulence modeling related to high-order discretizations such as DG has been focused on behavior when the kinematic turbulent viscosity becomes negative, as some authors have observed that in this regime traditional SA model variants become unstable. Oliver and Allmaras [4] proposed modifications to the model to account for negative $\tilde{\nu}$. Since then, other models have extended this, including recently the SA-Neg model by Allmaras *et al.* [5] used throughout this work.

The empirical production and destruction source terms dictate, respectively, the increase and decrease of turbulent kinetic viscosity. The production, P, is modeled as

$$P = \begin{cases} c_{b1}\tilde{S}\rho\tilde{\nu} & \chi \geq 0 \\ c_{b1}S\rho\tilde{\nu} & \chi < 0 \end{cases} \qquad \tilde{S} = \begin{cases} S + \bar{S} & \bar{S} \geq -c_{v2}S \\ S + \frac{S\left(c_{v2}^2 S + c_{v3}\bar{S}\right)}{S(c_{v3}-2c_{v2})-\bar{S}} & \bar{S} < -c_{v2}S \end{cases} \tag{2.16}$$

where the vorticity magnitude $S = \sqrt{2\Omega_{ij}\Omega_{ij}}$ from the rotation rate tensor $\Omega_{ij} = \left(\partial_i v_j - \partial_j v_i\right)/2$ is modeled as

$$\bar{S} = \frac{\tilde{\nu} f_{v2}(\chi)}{\kappa^2 d^2}, \qquad f_{v2}(\chi) = 1 - \frac{\chi}{1+\chi f_{v1}(\chi)},$$

where d is the distance to the nearest wall. The destruction term, D, is modeled as

$$D = \begin{cases} c_{w1} f_w(\tilde{\nu})\frac{\rho\tilde{\nu}^2}{d^2} & \chi \geq 0 \\ -c_{w1}\frac{\rho\tilde{\nu}^2}{d^2} & \chi < 0 \end{cases} \qquad f_w(\tilde{\nu}) = g(\tilde{\nu})\left(\frac{1+c_{w3}^6}{(g(\tilde{\nu}))^6 + c_{w3}^6}\right)^{1/6} \tag{2.17}$$

where

$$g(\tilde{\nu}) = r(\tilde{\nu}) + c_{w2}\left((r(\tilde{\nu}))^6 - r(\tilde{\nu})\right), \quad r(\tilde{\nu}) = \frac{\tilde{\nu}}{\tilde{S}\kappa^2 d^2}.$$

Table 2.1 lists the remaining parameters necessary to close the SA turbulence model.

When discretizing the transport equation for $\rho\tilde{\nu}$ (2.14), an additional term appears

Pr_t	$= 0.9$	c_{w1}	$= \frac{c_{b1}}{\kappa^2} + \frac{1+c_{b2}}{\sigma}$
c_{b1}	$= 0.1355$	c_{w2}	$= 0.3$
c_{b2}	$= 0.622$	c_{w3}	$= 2$
c_{v1}	$= 7.1$	c_{n1}	$= 16$
c_{v2}	$= 0.7$	κ	$= 0.41$
c_{v3}	$= 0.9$	σ	$= 2/3$

Table 2.1: Closure parameters for the SA-Neg2 Spalart-Allmaras eddy viscosity turbulence model model.

when rewriting the diffusion term in conservative form, namely

$$\frac{\rho}{\sigma}\partial_j\left(\left(\nu+\nu'\right)\partial_j\tilde{\nu}\right) = \partial_j\left(\frac{\rho}{\sigma}\left(\nu+\nu'\right)\partial_j\tilde{\nu}\right) \underline{-\frac{1}{\sigma}\left(\nu+\nu'\right)\partial_j\rho\partial_j\tilde{\nu}}. \tag{2.18}$$

The underlined term above is added as a source on the right side.

Ceze [6] notes that the additional equation in the current form is difficult to implement in a nonlinear solver because the turbulent kinematic viscosity will typically be many orders of magnitude smaller than the other state components. The equations are more easily solved for a non-dimensionalized $\tilde{\nu}$ of the form

$$\tilde{\nu}_D = \frac{\tilde{\nu}}{L} \tag{2.19}$$

where L is a problem-specific scaling parameter, usually on the order of the laminar viscosity ν. The equation for $\tilde{\nu}_D$ is also divided by L so that the residuals associated with this equation scale similarly to those of the other equations. This rescaling results in an easier system for the nonlinear and linear solvers.

2.2 Finite Element Discretization

The domain Ω is tessellated with a mesh T_h of N_e non-overlapping elements Ω_e with the index $e = 1 \ldots N_e$. The mesh skeleton consists of the set of both interior and exterior (boundary) faces $E_h = E_h^I \cup E_h^B$. The set of faces $\partial T_h = \{F : F \in \partial\Omega_e\}_{\Omega_e \in T_h}$ references E_h^B faces once, and faces in E_h^I once from each side, denoted individually by $(\cdot)^+$ and $(\cdot)^-$, twice in total. On boundary faces, E_h^B, the element side is always denoted by $(\cdot)^+$. *Face* will be used throughout to refer to edges in two dimensions and faces in three dimensions, since the discretizations can be written almost entirely independent of dimension. Normal vectors on the mesh faces will always refer to the outward-pointing direction, so for simplicity $\vec{n}^- = 0$ on boundary faces E_h^B. For a

scalar function in the domain, for example $w \in L^2(\Omega)$, the average and normal jump quantities are defined as

$$\{w\} = \frac{1}{2}(w^+ + w^-), \quad [\![w\vec{n}]\!] = w^+\vec{n}^+ + w^-\vec{n}^-, \quad [\![w]\!]^+ = [\![w\vec{n}]\!]\cdot\vec{n}^+$$

and for a vector-valued function $\vec{w} \in [L^2(\Omega)]^d$ are defined as

$$\{\vec{w}\} = \frac{1}{2}(\vec{w}^+ + \vec{w}^-), \quad [\![\vec{w}\cdot\vec{n}]\!] = \vec{w}^+\cdot\vec{n}^+ + \vec{w}^-\cdot\vec{n}^-.$$

On boundaries, the jump and average operators are redefined to ignore the contributions from across the face, with $\vec{n}^- = 0$ and $\{w\} = w^+$, $\{\vec{w}\} = \vec{w}^+$. Note that while $[\![w]\!]$ is not direction agnostic, it will only be used later where changing the sign of the difference does not affect the result.

The DG methods considered here approximate each component of the solution on each element by a polynomial of finite degree p_e, denoted by the space $\mathcal{P}^{p_e}(\Omega_e)$, the order of which is allowed to vary between elements. The approximate solution, or state, is denoted by $\mathbf{u}_h$ or $\mathbf{u}_H$, where h denotes a finer mesh refinement level, with h finer than H. When the order and mesh resolution vary independently in a way that matters for the description, the state will be instead written as $\mathbf{u}_{h,p}$. A component of the state vector, u_h, is said to formally exist in the space $\mathcal{V}_h$ where

$$\mathcal{V}_h := \left\{ v \in L^2(\Omega) \;\middle|\; v|_{\Omega_e} \in \mathcal{P}^{p_e}(\Omega_e) \;\; \forall\, \Omega_e \in T_h \right\}, \tag{2.20}$$

therefore the state vector for the system of equations is $\mathbf{u}_h \in \boldsymbol{\mathcal{V}}_h := [\mathcal{V}_h]^s$. This space is depicted for one solution component in Figure 2.1. In order to define the hybridized methods an additional space of functions on the faces, or skeleton, of the mesh

$$\mathcal{M}_h := \left\{ m \in L^2(E_h) \;\middle|\; m|_F \in \mathcal{P}^{p_f}(F) \;\; \forall\, F \in E_h \right\} \tag{2.21}$$

is needed. If $\lambda_h \in \mathcal{M}_h$ then $\boldsymbol{\lambda}_h \in \boldsymbol{\mathcal{M}}_h := [\mathcal{M}_h]^s$. The spaces $\mathcal{M}_h^I$ and $\boldsymbol{\mathcal{M}}_h^I$ are defined in a similar fashion for polynomials only on interior faces.

The finite element spaces are implemented numerically using a linear combination of polynomial basis functions on each element and face. Since no continuity is required between elements, these sums are independent of the neighboring elements, so the

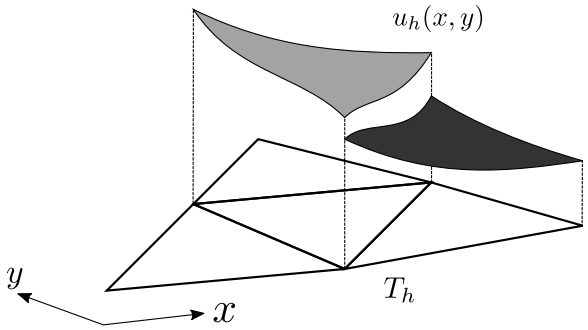


Figure 2.1: Depiction of a possible approximate solution in the piecewise polynomial approximation space used by DG state components on two elements.

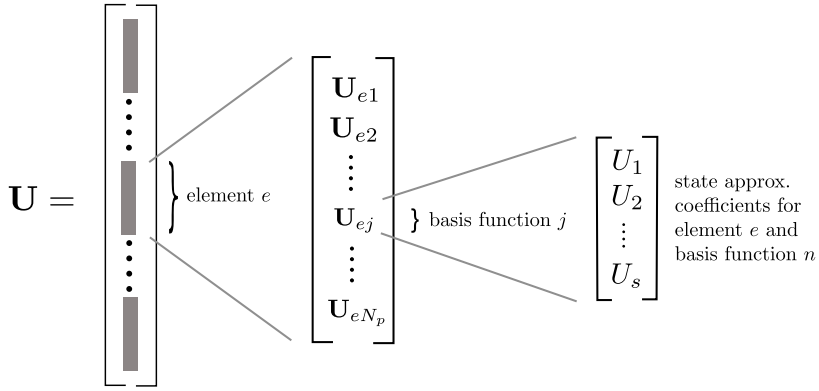


Figure 2.2: Unrolled storage used when storing the DG state vector $\mathbf{u}_h$ denoted by $\mathbf{U} \in \mathbb{R}^N$.

expansion on a single element Ω_e takes the form

$$\mathbf{u}_h(\vec{x})\big|_{\Omega_e} = \sum_{j=1}^{N_p} \mathbf{U}_{ej}\phi_j(\vec{x}), \tag{2.22}$$

where N_p is the dimension of the polynomial basis and the state vector $\mathbf{U}_{ej}$ is the vector of s unknowns, so the same basis is used for all equations. The global state vector is unrolled for all elements as shown in Figure 2.2. If the order is the same on all elements, the total system size, or number of DOF is $N = N_e \cdot N_p \cdot s$.

The finite element spaces additionally have associated inner products which induce a natural norm. An inner product of functions w and v in $L^2(\mathcal{V})$ will be denoted by $(w,v) = \int_{\mathcal{V}} w\,v\,d\Omega$ if $\dim(\mathcal{V}) = d$ and $\langle w,v\rangle = \int_{\mathcal{V}} w\,v\,ds$ if $\dim(\mathcal{V}) = d-1$. Similarly, an inner product of functions $\mathbf{w}$ and $\mathbf{v}$ in $[L^2(\mathcal{V})]^d$ will be denoted by $(\mathbf{w},\mathbf{v}) = \int_{\mathcal{V}} \mathbf{w}\cdot\mathbf{v}\,d\Omega$ if $\dim(\mathcal{V}) = d$ and $\langle \mathbf{w},\mathbf{v}\rangle = \int_{\mathcal{V}} \mathbf{w}\cdot\mathbf{v}\,ds$ if $\dim(\mathcal{V}) = d-1$. In order to concisely present the methods, it is also useful to employ the volume-based

inner products

$$(w,v)_{T_h} = \sum_{\Omega_e \in T_h} (w,v)_{\Omega_e}, \quad (\mathbf{w},\mathbf{v})_{T_h} = \sum_{\Omega_e \in T_h} (\mathbf{w},\mathbf{v})_{\Omega_e}, \quad (\vec{w},\vec{v})_{T_h} = \sum_{\Omega_e \in T_h} (\vec{w},\vec{v})_{\Omega_e}$$

and boundary-based inner products

$$\langle w,v\rangle_{E_h} = \sum_{F \in E_h} \langle w,v\rangle_F, \quad \langle w,v\rangle_{\partial T_h} = \sum_{\Omega_e \in T_h} \langle w,v\rangle_{\partial\Omega_e},$$

with extension to systems of equations involving $\mathbf{w}$ and $\mathbf{v}$.

2.3 Discontinuous Galerkin (DG) Discretization

Traditional DG methods are the footing on which the present work is set. This section reviews the development of these methods, then formulates the methods used for the convection-diffusion system, focusing on the aspects they have in common with the HDG discretization.

2.3.1 Overview of DG Methods

The history of DG applied to convection-diffusion conservation laws is particularly interesting because it includes individual development contributions for both model elliptic and hyperbolic PDEs. When applied to elliptic problems, DG requires a stabilization technique to weakly impose continuity. Alternatively, the development of DG for nonlinear hyperbolic equations benefited greatly from the development of finite volume methods, including Riemann solvers and limiters.

DG is certainly not the first choice for elliptic second-order equations. The presence of a second-derivative term suggests the use of an approximation that is everywhere differentiable. Methods for approximating with these functions, called continuous Galerkin (CG) methods, were indeed how original finite element methods for these equations were formulated. DG development began in the late 1960s when Lions and then Nitsche developed penalty methods. These methods enforce boundary Dirichlet data g on the solution u through a penalty term, integrated against test functions v, of the form $\int_{\partial\Omega} v\,\alpha\,(u-g)\,ds$ where the penalty parameter α is taken to infinity under refinement [7]. These were developed because it is difficult to construct functions that go to zero on an arbitrary boundary in order to properly enforce boundary conditions in the prior methods. Nitsche went on the prove that if the penalty parameter were taken to be $\mathcal{O}(h)$, then the state converges to the exact solution with optimal order

in $H^1(\Omega)$ and $L^2(\Omega)$. This was a vast improvement upon the existing methods, for which it was often difficult to construct functions with continuous derivatives that vanish on $\partial\Omega$. Development began in earnest when it was realized that inter-element continuity could be imposed in similar fashion to the methods developed for weakly enforcing Dirichlet boundary conditions. However, despite this work, DG methods for elliptic equations without adaptivity remain less efficient than classical conforming finite element methods, so it was only relatively recently, in the 1990s, that these methods have again received attention.

On the other hand, DG for hyperbolic systems has received considerable attention and development. This is largely due to the fact that at least in a single space dimension (1D), the element interfaces locally may be considered as Riemann problems for a short time, and in multiple dimensions a Riemann solver may be used to pointwise couple element approximations. All that is then needed is to ensure stability and high-order accuracy around discontinuities. Much of this work can be borrowed from the finite volume literature. Unlike the methods for elliptic equations, for hyperbolic systems DG has been shown to be superior to many classical finite element methods.

2.3.2 Formulation for Diffusion Terms

To illustrate application of the discontinuous Galerkin method, first consider its discretization of the model Poisson equation with homogeneous Dirichlet boundary conditions written as a first-order system

$$\vec{\sigma} = \nabla u, \quad -\nabla \cdot \vec{\sigma} = f \quad \text{in } \Omega \quad u = 0 \ \text{ on } \partial\Omega. \tag{2.23}$$

Given the discontinuous nature of the approximation, DG requires the discretization of the equation in the form in (2.23) to obtain a non-singular weak solution of the equation. The resulting system will solve for a u_h satisfying the set of equations

$$a_h(u_h, v) = (f, v)_{T_h} \quad \forall v \in \mathcal{V}_h, \tag{2.24}$$

where $a(\cdot, \cdot)$ is the bilinear form, and v are the test functions. This already identifies a key weakness of these methods, namely that they result in a system of equations that involve all the DOF for the state. Hybridization methods reduce this system size, but still use a discontinuous approximation space for the state, so successful discretization strategies for this setup will carry over.

After integrating (2.23) by parts against test functions, the flux formulation of

the problem is to find $u_h \in \mathcal{V}_h$ and $\vec{\sigma} \in [\mathcal{V}_h]^d$ such that,

$$\begin{aligned} (\vec{\sigma}, \vec{\tau})_{T_h} &= \langle \hat{u}, \vec{\tau} \cdot \vec{n} \rangle_{\partial T_h} - (u_h, \nabla \cdot \vec{\tau})_{T_h} \qquad \forall \vec{\tau} \in [\mathcal{V}_h]^d \\ (\vec{\sigma}, \nabla v)_{T_h} - \langle \hat{\vec{\sigma}} \cdot \vec{n}, v \rangle_{\partial T_h} &= (f, v)_{T_h} \qquad \forall v \in \mathcal{V}_h. \end{aligned} \tag{2.25}$$

This cannot yet be written in the form of (2.24) because the trace $\hat{u}$ and flux $\hat{\vec{\sigma}} \cdot \vec{n}$ coupling solutions on neighboring elements together are still undefined, and more importantly because $\vec{\sigma}$ should not remain in the system. Choosing $\vec{\tau} = \nabla v$ in the first of (2.25), integrating the last term by parts, and substituting this into the second set, the equations reduce to the more useful form

$$\Big(\nabla u_h, \nabla v\Big)_{T_h} - \Big\langle \hat{\vec{\sigma}} \cdot \vec{n}, v \Big\rangle_{\partial T_h} - \Big\langle (u_h - \hat{u}), \nabla v \cdot \vec{n} \Big\rangle_{\partial T_h} = \Big(f, v\Big)_{T_h} \quad \forall v \in \mathcal{V}_h. \tag{2.26}$$

Now after selecting choices for the trace and flux in terms of u_h and ∇u_h this can be written as in (2.24). The trace and flux should be conservative in that they are single-valued on the interior skeleton, and consistent in that if the exact solution is substituted into the equations, they match this value at every point.

This work will employ methods that set $\hat{u} = \{u_h\}$ on interior faces, and by the boundary condition $\hat{u} = 0$ on $\partial\Omega$. With this definition for the trace, if additionally $\hat{\vec{\sigma}} = \{\nabla u_h\}$, the bilinear form is non-coercive and the resulting system can become singular. The methods recover coercivity by defining the interior and boundary fluxes on a face F as

$$\hat{\vec{\sigma}}|_F = \{\nabla u_h\} - \eta\{\vec{\delta}_F([\![u_h \vec{n}]\!])\}, \quad \hat{\vec{\sigma}}^B|_F = \nabla u_h^+ - \eta\{\vec{\delta}_F(u_h^+ \vec{n}^+ - 0)\} \tag{2.27}$$

where η is a constant independent of the mesh size and $\vec{\delta}(\vec{\phi}) \in [\mathcal{V}_h^F]^d$ needs to be defined in

$$\mathcal{V}_h^F := \big\{ v \in L^2(\Omega_F) : v|_{\Omega_e} \in \mathcal{P}^p(\Omega_e)\ \forall \Omega_e \in \Omega_F \big\}$$

with Ω_F the set of elements Ω_e neighboring face F over which $\hat{\vec{\sigma}}$ has support. There are many methods for defining the $\vec{\delta}(\vec{\phi})$ operators, but relatively few are compact, in that only nearest-neighbor blocks of the matrix are non-zero. One of these, the interior penalty (IP) method, similar to the method Nitsche used for enforcing boundary conditions, uses a piecewise-constant $\vec{\delta}_F(\vec{\phi}) = h^{-1}\vec{\phi}$ with h some measure of the size of the face F [8]. The second form of Bassi and Rebay (BR2) instead defines the operators $\vec{\delta}_F$ on neighboring elements through an external lifting for every face F as

[9]

$$\sum_{\Omega_e \in \Omega_F} \left(\vec{\delta}_F(\vec{\phi}), \vec{\tau} \right)_{\Omega_e} = \left\langle \vec{\phi}, \{\vec{\tau}\} \right\rangle_F \quad \forall \vec{\tau} \in [\mathcal{V}_h^F]^d. \tag{2.28}$$

In this way the operator spreads a vector quantity $\vec{\phi}$ defined on the face F onto Ω_F, over the neighboring elements. On boundary faces, the set Ω_F contains only the element neighboring the boundary. For the IP method, the values of η that yield a coercive bilinear form vary for different polynomial orders, and can only be proven for linear elements [10]. On the other hand, the BR2 method is robustly stable when η is at least the maximum number of faces of neighboring elements, so this work sets

$$\eta^I = \kappa \max\left(N_f^{K^+}, N_f^{K^-}\right) \quad \eta^B = \kappa N_f^{K^+}, \tag{2.29}$$

where $\kappa \geq 1$ is an $\mathcal{O}(1)$ stabilization parameter. This work uses $\kappa = 2$ unless otherwise specified, for added stability.

Although the form (2.26) is used when implementing the method, it is difficult to use this to verify that the resulting bilinear form is symmetric. To define such a bilinear form, sets of integrals over ∂T_h should be moved to single integrals over E_h. The key in this step is to use the identity [7]

$$\langle w, \vec{\tau} \cdot \vec{n} \rangle_{\partial T_h} = \langle [\![w\vec{n}]\!], \{\vec{\tau}\} \rangle_{E_h} + \langle \{w\}, [\![\vec{\tau} \cdot \vec{n}]\!] \rangle_{E_h \setminus \partial\Omega}.$$

Assuming $\hat{u} = \{u_h\}$, the bilinear form for the diffusion equation (2.23) is

$$\begin{aligned} a_h(u_h, v) = (\nabla u_h, \nabla v)_{T_h} &- \langle [\![u_h\vec{n}]\!], \{\nabla v\} \rangle_{E_h} \\ &- \langle \{\nabla u_h\}, [\![v\vec{n}]\!] \rangle_{E_h} + \langle \vec{\delta}([\![u_h\vec{n}]\!]), [\![v\vec{n}]\!] \rangle_{E_h}. \end{aligned} \tag{2.30}$$

This is a simplified form due to the homogeneous Dirichlet boundary conditions; in general there are additional boundary terms. From (2.30) it is clear that as long as the last functional involving $\vec{\delta}(\cdot)$ is symmetric, the resulting bilinear form is symmetric.

2.3.3 Formulation for Transport System

The remainder of this work involving DG methods will refer to the application of these methods to convection-diffusion systems. These will be analyzed mostly in the steady-state form as a nonlinear system, so it is convenient to work with the following semidiscrete weak form of the equations

$$(\partial_t \mathbf{u}_h, \mathbf{v})_{T_h} + \mathcal{R}_h(\mathbf{u}_h, \mathbf{v}) = 0 \quad \forall \mathbf{v} \in \boldsymbol{\mathcal{V}}_h \tag{2.31}$$

where $\mathcal{R}_h(\mathbf{u}_h, \mathbf{v})$ is the semilinear form associated with the (possibly nonlinear) steady-state equations, and the time discretization is left undetermined. The equations are obtained in a similar fashion to that of the elliptic equation, except that there are now additional source and convection terms. The semilinear weak form can be written as contributions from each of the elements,

$$\mathcal{R}_h(\mathbf{u}_h, \mathbf{v}) = \sum_{e=1}^{N_e} \mathcal{R}_h(\mathbf{u}_h, \mathbf{v}|_{\Omega_e}) = 0, \quad \forall \mathbf{v} \in \boldsymbol{\mathcal{V}}_h. \tag{2.32}$$

After a single integration by parts and a second for the diffusion terms the semilinear form contribution for element Ω_e is

$$\begin{aligned}
\mathcal{R}(\mathbf{u}_h, \mathbf{v}|_{\Omega_e}) = &- \int_{\Omega_e} \partial_i \mathbf{v}^T \mathbf{H}_i \, d\Omega + \int_{\Omega_e} \mathbf{v}^T \, \mathbf{s} \, d\Omega \\
&+ \int_{\partial\Omega_e \setminus \partial\Omega} \mathbf{v}^T \left(\widehat{\mathbf{F}} + \widehat{\mathbf{G}} \right) ds + \int_{\partial\Omega_e \cap \partial\Omega} \mathbf{v}^T \left(\widehat{\mathbf{F}}^B + \widehat{\mathbf{G}}^B \right) ds \\
&- \int_{\partial\Omega_e \setminus \partial\Omega} \partial_i \mathbf{v}^T \, \mathbf{K}_{ij}(\mathbf{u}_h)(\mathbf{u}_h - \hat{\mathbf{u}}) n_j \, ds \\
&- \int_{\partial\Omega_e \cap \partial\Omega} \partial_i \mathbf{v}^T \, \mathbf{K}_{ij}(\mathbf{u}_h^B)(\mathbf{u}_h - \hat{\mathbf{u}}) n_j \, ds
\end{aligned} \tag{2.33}$$

where the test functions $\mathbf{v}$ and states $\mathbf{u}_h$ are those which are compact on Ω_e. The last two terms involving the viscous diffusivity tensor symmetrize the semilinear form for adjoint consistency. As for a single equation, the interface state is eliminated by setting $\hat{\mathbf{u}} = \{\mathbf{u}_h\}$ on interior faces and $\hat{\mathbf{u}} = \mathbf{u}^{\mathrm{B}}(\mathbf{u}_h^+)$ on boundaries.

Convection terms are handled by using a numerical flux function $\hat{\mathbf{F}}(\mathbf{u}_h^{\pm}, \vec{n})$ to define a single-valued normal flux on interior faces. The numerical flux resolves the double-valued convective flux $\vec{\mathbf{F}}(\mathbf{u}_h)$ on faces by upwinding based on the local direction of propagation. This is constructed in general by using the solution of an approximation to the Riemann problem pointwise on the faces. This work uses the Roe approximate solver [11] with an entropy fix, which takes the form

$$\hat{\mathbf{F}}(\mathbf{u}_h^{\pm}, \vec{n}) = \frac{1}{2} \left(\mathbf{F}_i(\mathbf{u}_h^+) + \mathbf{F}_i(\mathbf{u}_h^-) \right) n_i - \frac{1}{2} \mathbf{L} \, |\mathbf{\Lambda}| \, \mathbf{R} \left(\mathbf{u}_h^+ - \mathbf{u}_h^- \right), \tag{2.34}$$

where $\mathbf{L}$, $\mathbf{R}$, and $\mathbf{\Lambda}$ are the matrices, respectively, of left and right eigenvectors, and eigenvalues of the flux Jacobian $\mathbf{A}_i(\tilde{\mathbf{u}}) \, n_i$ based on the Roe-averaged state $\tilde{\mathbf{u}}(\mathbf{u}_h^{\pm})$ [12]. On boundary faces a numerical flux function is not required, and in fact adversely affects error convergence. Instead of using a Riemann solver on boundaries, this work

sets

$$\hat{\mathbf{F}}^B(\mathbf{u}_h^B, \vec{n}) = \mathbf{F}_i(\mathbf{u}_h^B) n_i.$$

The boundary state $\mathbf{u}_h^B(\mathbf{u}_h^+, \mathrm{BC})$ is determined by the state neighboring the boundary and the boundary condition on the face.

The diffusion terms are treated similarly to that for a scalar equation. A numerical diffusive flux is necessary on interfaces analogous to the definition (2.27) dotted with the normal vector for a single equation. On interior faces f,

$$\hat{\mathbf{G}}(\mathbf{u}_h^{\pm}, \nabla \mathbf{u}_h^{\pm}, \vec{n}) = \frac{1}{2}\big(\mathbf{G}_i(\mathbf{u}_h^+, \nabla \mathbf{u}_h^+) + \mathbf{G}_i(\mathbf{u}_h^-, \nabla \mathbf{u}_h^-)\big) n_i + \frac{\eta}{2}\left(\boldsymbol{\delta}_{f,i}^+ + \boldsymbol{\delta}_{f,i}^-\right) n_i, \quad (2.35)$$

with η the stabilization chosen for BR2 according to (2.29). There is a certain level of ambiguity in how the BR2 stabilization is defined for systems, specifically regarding where the diffusivity tensor is added. One option is to apply the operator to the state multiplied by the diffusivity tensor from (2.3), $\vec{\boldsymbol{\delta}}_f(\llbracket \mathbf{K}_{ij}\mathbf{u}_h \rrbracket)$.

$$\int_{\Omega_e^+} \boldsymbol{\tau}_i^{+T} \boldsymbol{\delta}_{f,i}^+ \, d\Omega + \int_{\Omega_e^-} \boldsymbol{\tau}_i^{-T} \boldsymbol{\delta}_{f,i}^- \, d\Omega = \frac{1}{2}\int_f \left(\boldsymbol{\tau}_i^{+T} + \boldsymbol{\tau}_i^{-T}\right)\left(\mathbf{K}_{ij}(\mathbf{u}_h^+)\mathbf{u}_h^+ - \mathbf{K}_{ij}(\mathbf{u}_h^-)\mathbf{u}_h^-\right) n_j \, ds, \quad \forall \boldsymbol{\tau} \in [\boldsymbol{\mathcal{V}}_h^f]^d. \quad (2.36)$$

Equation (2.36) is adjoint consistent, as is the original formulation [9], which instead splits $\vec{\boldsymbol{\delta}}_f \in [\boldsymbol{\mathcal{V}}_h^f]^d$ into $\vec{\boldsymbol{\delta}}_f^{\pm}$ on either side of the face, solving

$$\int_{\Omega_e^+} \boldsymbol{\tau}_i^{+T} \boldsymbol{\delta}_{f,i}^+ \, d\Omega + \int_{\Omega_e^-} \boldsymbol{\tau}_i^{-T} \boldsymbol{\delta}_{f,i}^- \, d\Omega = \frac{1}{2}\int_f \left(\boldsymbol{\tau}_i^{+T}\mathbf{K}_{ij}(\mathbf{u}_h^+) + \boldsymbol{\tau}_i^{-T}\mathbf{K}_{ij}(\mathbf{u}_h^+)\right)\left(\mathbf{u}_h^+ - \mathbf{u}_h^-\right) n_j \, ds, \quad \forall \boldsymbol{\tau} \in [\boldsymbol{\mathcal{V}}_h^f]^d. \quad (2.37)$$

This is the formulation used throughout this work. On boundary faces the single $\vec{\boldsymbol{\delta}}_f \in [\boldsymbol{\mathcal{V}}_h^f]^d$ solves

$$\int_{\Omega_e^+} \boldsymbol{\tau}_i^T \boldsymbol{\delta}_{f,i} \, d\Omega = \int_F \boldsymbol{\tau}_i^T \mathbf{K}_{ij}(\mathbf{u}_h^B)\left(\mathbf{u}_h^+ - \mathbf{u}_h^B\right) \, ds, \quad \forall \boldsymbol{\tau} \in [\boldsymbol{\mathcal{V}}_h^f]^d, \quad (2.38)$$

and only has support over the element neighboring the boundary. Note that the diffusivity matrix is calculated using the boundary state, even when the test function comes from the neighboring element. The numerical diffusive flux on these faces is

$$\hat{\mathbf{G}}^B = \Pi_G^B\left[\mathbf{G}_i(\mathbf{u}_h^B, \nabla \mathbf{u}_h^+) n_i + \eta \boldsymbol{\delta}_{f,i} n_i\right], \quad (2.39)$$

where the projection Π_G^B in (2.39) imposes the Neumann-type boundary conditions on the gradient term affecting the viscous flux; *e.g.* , when imposing the heat flux as a boundary condition.

CHAPTER 3

Output-Based Error Estimation and Adaptation

Adjoint-based techniques in finite element methods for error estimation and adaptation have long been a topic of interest, in part because the concept of an adjoint is used for superconvergence proofs in these methods. Adjoint-based error estimation techniques are not limited to finite element methods, but the error estimates have an adjoint error correction term that is zero for many of these methods, which makes adjoint-based error estimation ideally suited for these methods [13]. Their use for both error estimation and adaptation in CFD is also well-studied [13, 14, 15, 16, 17, 18, 19]. Section 3.1 introduces the discretization of the adjoint for a general discretization and discusses how it is usually implemented. Section 3.2 then describes how the adjoint is used for error estimation in the output functional.

3.1 Error Estimates and Relation to Adjoints

As was alluded to in Section 1.2.2, accurate calculation of output functionals requires resolution at locations that cannot be properly identified by error indicators based solely on local information. Such locations can however be identified by adjoint-based methods. The key idea is that the adjoint solution relates the residual error in the discretization to the error in the output functional, and since it results from the dual discretization of the problem, it is able to do so in a way that captures the essential propagation effects.

3.1.1 Duality and Linear Theory

The ideas presented here have been extensively discussed in previous works, but the development of the theory and applications of concepts such as adjoint consistency will be used later in the analysis.

Consider a linear differential equation of the form,

$$Lu = f \tag{3.1}$$

with homogeneous boundary conditions on a domain $\Omega \subset \mathbb{R}^d$. Here L is a linear differential operator, for instance the advection operator $\vec{a} \cdot \nabla$ or the Laplacian ∇^2, $f \in L^2(\Omega)$ is a given source term, and u is the solution. To illustrate the point, consider the simple output $J(u) = (g, u)_\Omega$, where the notation $(\cdot, \cdot)_\Omega$ denotes the integral inner product over the domain Ω. There exists another differential equation,

$$L^* \psi = g \tag{3.2}$$

called the adjoint form, or alternatively the *dual* form, with the output $J(\psi) = (\psi, f)_\Omega$ resulting in the same value. It should not be surprising that there is a different equation that yields the same output since there is a different source term, but it is surprising that this source term is precisely the weight in the output of the original equation. The outputs are equivalent as long as the differential operator L^* is the *adjoint operator* of the original differential operator L, defined by

$$(\psi, f)_\Omega = (\psi, Lu)_\Omega = (L^* \psi, u)_\Omega = (g, u)_\Omega. \tag{3.3}$$

The solution of the adjoint form of the equation is called the *adjoint*. The operators satisfy the *adjoint identity*, which states

$$(v, Lu)_\Omega = (L^* v, u)_\Omega \qquad \forall v, u \in \mathcal{V}$$

where $\mathcal{V}$ is the suitable function space for the inner product, for instance $L^2(\Omega)$. The process here reveals, in a sense, how to derive the adjoint equation for a given linear operator L. For instance, ignoring boundary conditions for the moment, if $L = \vec{a} \cdot \nabla$ then an integration by parts introduces a negative sign, so $L^* = -\vec{a} \cdot \nabla$. On the other hand, if the operator was the Laplacian, $L = \Delta$, then integration by parts twice to move the operator from the trial to the test functions cancels the negative sign, so $L^* = L$.

The adjoint form is also useful in deriving the continuous form of the adjoint error estimate. This is not the form used most of the time, but it will reveal the rate of convergence expected from output with Galerkin finite element methods. Assume that the original differential equation is solved with a variational formulation resulting from a Galerkin finite element method in the form (2.24) for an approximate solution

$u_h \in \mathcal{V}_h$. Then, the error in the scalar output J is

$$\begin{aligned}\delta J := J(u) - J(u_h) =& (u - u_h, g)_\Omega \\ =& (u - u_h, L^* \psi)_\Omega \\ =& (Lu - Lu_h, \psi)_\Omega \\ =& -(Lu_h - f, \psi)_\Omega = -(r(u_h), \psi)_\Omega\end{aligned}$$

where $r(u_h)$ is the non-zero residual obtained by evaluating the linear PDE with the approximate solution u_h. So far this statement is not of much use, since it involves the exact adjoint ψ which in general is unknown. However, for a Galerkin finite element method the error estimate above is equivalent to

$$\delta J = -(r(u_h), \psi - \psi_h)_\Omega, \tag{3.4}$$

where $\psi_h \in \mathcal{V}_h$ is the approximate solution of the adjoint from the same discretization of the adjoint equation. These are equivalent because the Galerkin method solves, before integration by parts,

$$(r(u_h), v)_\Omega = 0 \quad \forall v \in \mathcal{V}_h,$$

and one such $v = \psi_h$ since these are in the same space. While the error estimate in the form (3.4) is quite remarkable, it should be somewhat expected, since the adjoint links the residual error to the error in the output, so this simply integrates that error over the domain. The error estimate (3.4) gives a simple bound on the output error as,

$$|\delta J| \leq \|r(u_h)\|_1 \|\psi - \psi_h\|_1, \tag{3.5}$$

where $\|a\|_1 = \int_\Omega |a| \, d\Omega$. If the adjoint ψ_h comes from an order p space and is solved with the same DG method, then $\|\psi - \psi_h\|_1$ typically converges like $\mathcal{O}(h^{p+1})$, so at rate $p+1$. The residual, assuming L contains order m derivatives, converges at rate $p+1-m$, since derivatives of the order p quantities have to be taken. Putting these together it is expected that the output converges at $\mathcal{O}(h^{2p+2-m})$, which is precisely the rate typically observed for DG methods. Although the bound above breaks when the adjoint ψ contains a singularity, affecting the norm $\|\psi - \psi_h\|_1$, the convergence rate above is often unaffected. This shows a key benefit of using Galerkin methods, namely that outputs superconverge in this way. If the method were not able to subtract off the approximate adjoint ψ_h, then the output would only converge at h^p, unless the

correction term itself were added to the error estimate.

Note that the HDG method introduced in Chapter 4 provides a discretization of the gradient, and assuming that the gradients converge optimally, convergence is often observed to be one order higher.

3.1.2 Discrete Analysis

The approach taken in a typical implementation is usually based on a discrete adjoint calculated from the original *primal* discretization itself, without the need to additionally discretize the *adjoint* equations as well. The starting point for deriving this system is the set of nonlinear equations derived from converting the time-independent semilinear form in (2.32) to a set of residual equations in discrete form, namely

$$\mathbf{R}_H(\mathbf{U}_H) = \mathbf{0}. \tag{3.6}$$

Here $\mathbf{U}_H$ is the vector of unknowns for an approximate solution $\mathbf{u}_H \in \boldsymbol{\mathcal{V}}_H$ similar to Figure 2.2 for a mesh T_H and order p. After computing the output $J_H(\mathbf{U}_H)$ it is useful to determine how much error is in this output, and where the sources of the error occur.

Computing the error estimate

$$\delta J = J(\mathbf{U}) - J_H(\mathbf{U}_H)$$

would be ideal, but the exact solution vector $\mathbf{U}$ is in general unknown, and even if this could be computed, it does not help in targeting the sources contributing to the error. Since the exact solution is in general unattainable, the error estimate must be computed between the current approximate solution and the solution from a finer space $\mathbf{U}_h$, namely

$$\delta J_h := J_h(\mathbf{U}_h) - J_H(\mathbf{U}_H) \approx \delta J. \tag{3.7}$$

This implies that while δJ_h is an error estimate, it is by no means a strict bound on the error. The finer space $\boldsymbol{\mathcal{V}}_h$ is usually defined as the space obtained by uniformly refining the mesh T_H or increasing the order p. Note that often $J_H(\cdot) = J(\cdot)$, except when order-dependent parameters are present, such as geometry definitions or BR2 stabilization η values. Usually, however, this approximation is a fair assumption. While the approximate solution on the finer space could be directly computed to calculate the global error estimate, this would require the solution of an additional even larger nonlinear system and thus would be immensely impractical. Instead, a

similar adjoint technique will turn the need for an extra nonlinear system into an additional linear solve for the fine-space adjoint variable $\mathbf{\Psi}_h$.

It is possible to compute the fine-space adjoint $\mathbf{\Psi}_h$ approximately without having to solve for the fine-space primal state $\mathbf{U}_h$. First, realize that perturbations of the state $\delta\mathbf{U}_h$ induce corresponding perturbations of the residual $\delta\mathbf{R}_h$ by linearizing the original discretization about the primal state $\mathbf{U}_H$ represented on the fine-space via injection as $\mathbf{U}_h^H$, as

$$\left.\frac{\partial \mathbf{R}_h}{\partial \mathbf{U}_h}\right|_{\mathbf{U}_h^H} \delta\mathbf{U}_h + \delta\mathbf{R}_h = 0. \tag{3.8}$$

Further using the perturbations, define the discrete adjoint $\mathbf{\Psi}_h$ as the sensitivity of J_h to an infinitesimal residual perturbation added to the nonlinear system, namely

$$\delta J_h = J_h(\mathbf{U}_h) - J_h(\mathbf{U}_h + \delta\mathbf{U}_h) \approx -\mathbf{\Psi}_h^T \delta\mathbf{R}_h. \tag{3.9}$$

Assuming the output J_h is differentiable, the left and right side expressions above can be related by inserting (3.8) as

$$\left.\frac{\partial J_h}{\partial \mathbf{U}_h}\right|_{\mathbf{U}_h^H} \delta\mathbf{U}_h = \mathbf{\Psi}_h^T \delta\mathbf{R}_h = \mathbf{\Psi}_h^T \left.\frac{\partial \mathbf{R}_h}{\partial \mathbf{U}_h}\right|_{\mathbf{U}_h^H} \delta\mathbf{U}_h. \tag{3.10}$$

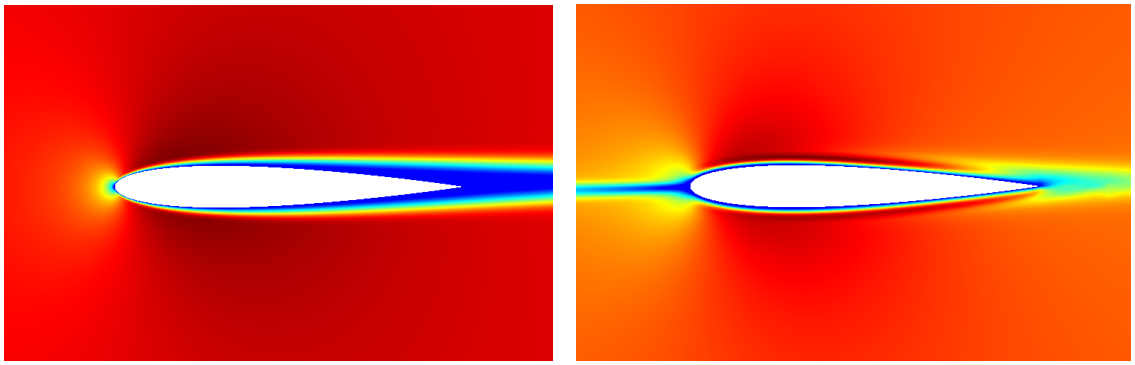
This must hold for all permissible perturbations, so these can be dropped, and after rearranging the *discrete adjoint system* for the fine-space adjoint is given by

$$\left.\frac{\partial \mathbf{R}_h}{\partial \mathbf{U}_h}\right|_{\mathbf{U}_h^H}^T \mathbf{\Psi}_h = \left.\frac{\partial J_h}{\partial \mathbf{U}_h}\right|_{\mathbf{U}_h^H}^T. \tag{3.11}$$

Often even this system does not have to be solved exactly to get a meaningful representation of the adjoint necessary to estimate the error. Other works [1] have investigated the use of Jacobian-free methods for inexpensively computing an approximation to this adjoint. For the results presented in later chapters, this adjoint system is computed with an iterative matrix solver close to machine accuracy.

For certain applications, even the computation of the Jacobian on the fine space is prohibitively expensive. In these situations the adjoint on the current space $\mathbf{\Psi}_H$ can be computed by solving (3.11) entirely on the space $\boldsymbol{\mathcal{V}}_H$, then reconstructed to act as a surrogate for the fine-space adjoint. With the HDG discretization, there is an additional option available, discussed at the end of Chapter 4.

As an example of the adjoint principle, consider the two-dimensional Navier-Stokes simulation of a NACA 0012 airfoil shown in Figure 3.1 at 1° angle of attack, Re =



(a) Primal state x-momentum (b) Adjoint state x-momentum

Figure 3.1: Contours of x-momentum for the solution and the drag adjoint for a NACA0012 airfoil at $\text{Re} = 5 \times 10^3$. Drag is sensitive to perturbations in the dark red and blue in the adjoint.

5×10^3, and $M_\infty = 0.5$. The contour plot on the left shows the low-velocity (blue) wake created behind the airfoil in the x-momentum component of the primal state $\mathbf{U}_H$. If the output of interest is the drag on the airfoil, setting this as J_H, then in practice the corresponding *adjoint* state $\mathbf{\Psi}_H$ is solved for using (3.11) linearizing about the state $\mathbf{U}_H$ instead of the injected state. Computing the adjoint in the current space in this way does not require the construction of a fine-space Jacobian matrix, and therefore the additional cost incurred by solving for the adjoint state is only the solution of the linear system. The x-momentum component of the adjoint state, shown on the right of Figure 3.1, reveals the characteristic "reverse-wake"; perturbations in the x-momentum conservation equation upstream of the airfoil affect its drag, while perturbations behind it have relatively little effect on its drag.

It may not be immediately obvious, but the dual form (3.2) from linear theory bears a remarkable resemblance to (3.11), as it should if the discrete adjoint is an approximation of the continuous equation. If the output is a linear functional then it can be written as $J_h(\mathbf{U}_h) = \mathbf{G}^T\mathbf{U}_h$, and if the Jacobian matrix is denoted by $\mathbf{A}$, (3.11) reduces to

$$\mathbf{A}^T\mathbf{\Psi} = \mathbf{G}.$$

Since the adjoint of a matrix is the conjugate transpose this seems to be a discretization of the continuous dual problem.

In order to compute the error estimate δJ_h, first write the output $J_h(\mathbf{U}_h)$ in terms

of an expansion about the injected state $\mathbf{U}_h^H$ as

$$J_h(\mathbf{U}_h) = J_h(\mathbf{U}_h^H) + \left.\frac{\partial J_h}{\partial \mathbf{U}_h}\right|_{\mathbf{U}_h^H} \delta\mathbf{U} + \mathcal{O}(\delta\mathbf{U}^2) \tag{3.12}$$

The state difference $\delta\mathbf{U} = \mathbf{U}_h - \mathbf{U}_h^H$ needs to be solved for to compute the error estimate and can be obtained by setting the linearized fine-space residuals, $\mathbf{R}_h(\mathbf{U}_h)$, to zero

$$\mathbf{R}_h(\mathbf{U}_h) = \mathbf{R}_h(\mathbf{U}_h^H) + \left.\frac{\partial \mathbf{R}_h}{\partial \mathbf{U}_h}\right|_{\mathbf{U}_h^H} \delta\mathbf{U} \approx \mathbf{0} \quad \rightarrow \quad \delta\mathbf{U} \approx -\left(\left.\frac{\partial \mathbf{R}_h}{\partial \mathbf{U}_h}\right|_{\mathbf{U}_h^H}\right)^{-1} \mathbf{R}_h(\mathbf{U}_H^h).$$

Note that in general the fine-space residual $\mathbf{R}_h(\mathbf{U}_h^H)$ will not be zero, and the locations of the nonzeros indicate where the current solution does not well represent a finer solution. Inserting the above result into (3.12) yields

$$J_h(\mathbf{U}_h) = J_h(\mathbf{U}_h^H) - \left.\frac{\partial J_h}{\partial \mathbf{U}_h}\right|_{\mathbf{U}_h^H} \left(\left.\frac{\partial \mathbf{R}_h}{\partial \mathbf{U}_h}\right|_{\mathbf{U}_h^H}\right)^{-1} \mathbf{R}_h(\mathbf{U}_H^h) + \mathcal{O}(\delta\mathbf{U}^2).$$

The quantity multiplying the residuals with the injected state $\mathbf{U}_h^H$ is simply the fine-space adjoint $\mathbf{\Psi}_h$ from (3.11). As long as the geometry and other parameters do not change, then $J_h(\mathbf{U}_h^H) = J_H(\mathbf{U}_H)$ and the error estimate is based on the *adjoint-weighted residual*

$$\delta J_h = -\mathbf{\Psi}_h^T \mathbf{R}_h\left(\mathbf{U}_h^H\right) + \mathcal{O}\left(\delta\mathbf{U}^2\right). \tag{3.13}$$

Since the adjoint was obtained using DG, which uses a Galerkin variation form, the injected adjoint can be subtracted without affecting the error estimate. The final form is therefore

$$\delta J_h \approx -\left(\mathbf{\Psi}_h - \mathbf{\Psi}_h^H\right)^T \mathbf{R}_h\left(\mathbf{U}_h^H\right). \tag{3.14}$$

Just as with the continuous linear theory, the error estimate involves the product of a difference in the adjoints and the fine-space residual evaluated with the injected state. This form of the error estimate is second-order accurate in the state difference $\delta\mathbf{U}$, owing to the linearizations performed above. There are a few other forms of the adjoint-based error estimate that can be formed, for instance a third-order accurate version by Becker and Rannacher [14].

3.1.3 Adjoint Consistency and Accuracy

Section 3.1.2 showed that the discrete adjoint system bears resemblance to the continuous adjoint PDE. In fact, it has been shown theoretically and numerically [20, 21] that such *adjoint consistency* is necessary for optimal convergence of the output functionals. This is an interesting finding, since it implies that in order to construct a method from which outputs optimally converge, it is also necessary to ensure consistency of a different equation.

The initial step in determining adjoint consistency for a discretization and output is to derive both the dual form of the system of PDE similar to (3.2) for a single linear equation, and the adjoint system implied by the discretization. Both these derive from the principle that the corresponding adjoint equation for a general semilinear form $\mathcal{R}(\mathbf{u}, \mathbf{v})$ with an output functional $\mathcal{J}(\mathbf{u})$ is [20]

$$\mathcal{R}'[\mathbf{u}](\mathbf{v}, \boldsymbol{\psi}) = \mathcal{J}'[\mathbf{u}](\mathbf{v}) \qquad \forall \mathbf{v},$$

where the prime symbol denotes the Frechet derivative about the argument in brackets. In particular, the semilinear form becomes linearized about $\mathbf{u}$ and thus is itself a bilinear form. Therefore this is simply a linear functional $\mathcal{L}^*_{\mathbf{u},\mathbf{v}}(\boldsymbol{\psi}) = \mathcal{R}'[\mathbf{u}](\mathbf{v}, \boldsymbol{\psi})$ analogous to the adjoint operator L^* in (3.2). Applying this concept, the adjoint equations implied by the DG discretization (2.32) seek $\boldsymbol{\psi}_h \in \boldsymbol{\mathcal{V}}_h$ such that

$$\mathcal{R}'[\mathbf{u}_h](\mathbf{v}, \boldsymbol{\psi}_h) = \mathcal{J}'[\mathbf{u}_h](\mathbf{v}) \quad \forall \mathbf{v} \in \boldsymbol{\mathcal{V}}_h. \tag{3.15}$$

These equations bear resemblance to the discrete adjoint system (3.11). Similarly, in order to derive the dual form of a general nonlinear PDE, first create a semilinear form by integrating by parts against the adjoint variables, yielding $\mathbf{R}(\mathbf{u}, \boldsymbol{\psi})$ with the differential operator acting on the adjoint variables. Then the dual form of the system satisfies

$$\mathcal{R}'[\mathbf{u}](\delta\mathbf{u}, \boldsymbol{\psi}) = \mathcal{J}'[\mathbf{u}](\delta\mathbf{u}), \tag{3.16}$$

for all $\delta\mathbf{u}$ such that $\mathbf{u} + \delta\mathbf{u}$ satisfy the boundary conditions. Since this must hold for all suitable variations, these can then be "removed", yielding the system and boundary conditions. The original DG discretization is *adjoint consistent* if the exact adjoint $\boldsymbol{\psi}$ from (3.16) satisfies (3.15). Even if this does not hold true, a method can be *asymptotically adjoint consistent* if this holds in the limit $h \to 0$, can recover the correct convergence rates asymptotically.

Adjoint inconsistency in DG methods usually derives from the treatment of bound-

ary conditions [20] or standard treatments for source terms that contain state gradients [21]. In either case, when this occurs the convergence rate of the output functional will decrease and if the boundary treatment is to blame, even the primal approximate state $\mathbf{u}_h$ near these boundaries will become oscillatory. Source terms that contain state gradients, such as those for the RANS turbulence models, require special treatment in DG to ensure adjoint consistency. HDG, on the other hand, automatically provides adjoint consistency for these source terms due to the mixed formulation for the state gradients.

3.2 Error Localization and Adaptation

The error estimate in (3.14) is written as a global inner product of the discrete vectors, so it implicitly includes a summation over the elements

$$\delta J_h = -\sum_{\Omega_e \in T_h} (\mathbf{\Psi}_{h,e} - \mathbf{\Psi}^H_{h,e})^T \mathbf{R}_{h,e}(\mathbf{U}^H_h), \tag{3.17}$$

where $(\cdot)_{,e}$ denotes the restriction of the vector to the element Ω_e on the fine-space. A local error indicator can be defined as the absolute value of the contribution from each element as

$$\epsilon_e = (\mathbf{\Psi}_{h,e} - \mathbf{\Psi}^H_{h,e})^T \mathbf{R}_{h,e}(\mathbf{U}^H_h). \tag{3.18}$$

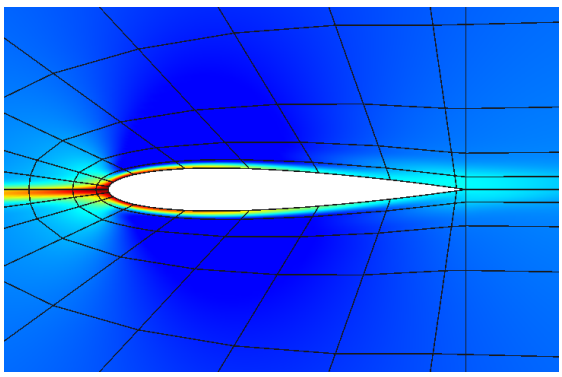
If the fine space is order enriched, then the meshes coincide so elements with large absolute value $|\epsilon|$ should be chosen for refinement. If instead the fine space is obtained with h-refinement, then ϵ_e should be summed for each of the fine-space elements corresponding to the elements of the current mesh to obtain the error indicator. This process is shown in Figure 3.2 for the drag output on a NACA 0012 airfoil, the same case presented in Section 3.1.2. The solution was initially obtained at $p = 2$ on the relatively coarse mesh shown, and the fine-space adjoint $\mathbf{\Psi}_h$ and residual are obtained at $p = 3$.

To summarize, the following procedure is used when applying this form of error estimation:

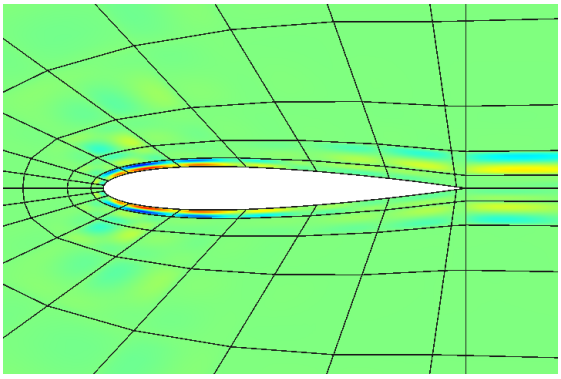
1. Solve the discrete equations $\mathbf{R}_H(\mathbf{U}_H)$ for $\mathbf{U}_H$ in the *coarse* space $\boldsymbol{\mathcal{V}}_H$, and compute the output $J(\mathbf{U}_H)$

2. Represent the coarse solution on a uniformly refined fine space $\boldsymbol{\mathcal{V}}_h$ as $\mathbf{U}^H_h$

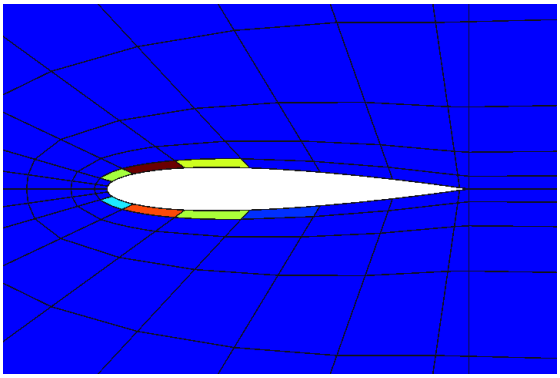
3. Compute the fine-space residual with this injected state $\mathbf{R}_h(\mathbf{U}^H_h)$



(a) Fine-space drag adjoint Ψ_h density comp.



(b) Residual $\mathbf{R}_h(\mathbf{U}_h^H)$ density comp.



(c) Localized error indicator ϵ

Figure 3.2: Components used to compute the error estimate and localize on each element, shown for a NACA 0012 airfoil at $\mathrm{Re} = 5 \times 10^3$.

4. Solve for the fine-space adjoint $\mathbf{\Psi}_h$ either directly with the linear system (3.11) or by first solving for the coarse adjoint $\mathbf{\Psi}_H$ and then reconstructing

5. Evaluate the approximate error estimate δJ_h via (3.14) and use this to correct the output

6. Localize the error estimate to form $|\epsilon_e|$ for each element

7. Select a certain percentage of the elements with the highest error to adapt

The indicator only identifies the elements in which the errors originate, since ϵ is itself a scalar value. In particular, there is no indication of directionality for constructing anisotropic meshes, and additionally no indication of whether refining the mesh or increasing the order on the element will be more effective. Techniques for achieving these goals will be discussed in Chapter 5.

CHAPTER 4

Hybridized Discontinous Galerkin (HDG) Methods

Hybridized discontinous Galerkin methods indirectly represent the solution by an auxiliary state on the skeleton of the mesh, from which the local state on each element can be solved. By first solving for this auxiliary state the size of the global system is dramatically reduced.

The primary objective of this chapter is to introduce two variants of HDG, both of which improve on the accuracy and computational efficiency of the existing methods. This chapter begins by giving a brief history and overview of the existing HDG methods for linear convection-diffusion systems. It then describes the extension of the existing HDG methods to general systems of convection-diffusion equations and discusses implementation aspects, before finally introducing the two new variants. The first of these removes the extra gradient variable, thereby obtaining a more computationally efficient method. The second method retains the gradients, and introduces a new flux form for coupling elements, thereby allowing for optimal gradient convergence and even faster convergence for the element-by-element post-processed solution.

4.1 Overview of Existing Methods

Finite element methods, such as DG described in Section 2.3, result in a linear system to solve for the approximate solution. This happens, for example, when solving with a Newton solver for the solution of the nonlinear system. The number of DOF, or the size of the system, resulting from these equations grows rapidly with the dimension d as $\mathcal{O}(p^d)$ for increasing polynomial order p. HDG methods reduce this growth rate by instead solving a linear system that only couples a set of trace unknowns on the

Triangles:

method	$p=1$	$p=2$	$p=3$	$p=4$
DG	6	12	20	30
CG	1	4	9	16
HDG	6	9	12	15
EDG	1	4	7	10

Quadrilaterals:

method	$p=1$	$p=2$	$p=3$	$p=4$
DG	4	9	16	25
CG	1	4	9	16
HDG	4	6	8	10
EDG	1	3	5	7

Tetrahedra:

method	$p=1$	$p=2$	$p=3$	$p=4$
DG	24	60	120	210
CG	1	8.2	27.4	64.6
HDG	36	72	120	180
EDG	1	8.2	27.4	58.6

Hexahedra:

method	$p=1$	$p=2$	$p=3$	$p=4$
DG	8	27	64	125
CG	1	8	27	64
HDG	12	27	48	75
EDG	1	7	19	37

Table 4.1: Number of DOF (per equation of a system) that are required per mesh vertex to represent the HDG trace variable.

interior faces of the mesh. The approximate solution on each element is then recovered from the traces on the interior faces, and the boundary conditions on the exterior faces. Since the faces are a lower-dimensional object than the elements themselves, the trace requires only $\mathcal{O}(p^{(d-1)})$ DOF. The traces for HDG are independent of the overall mesh topology, so no continuity is required, as shown in Figure 4.1. Although the growth rate of the DOF is lowered, with relatively few exceptions there are more faces than elements in a mesh, so the HDG system can in fact be larger than the corresponding DG system at the same order. Table 4.1 shows the number of globally-coupled DOF required for different methods for a sufficiently large and regular mesh. While embedded discontinuous Galerkin (EDG) ultimately has a smaller system size, HDG increasingly reduces the system size for higher order p over DG. Embedded discontinuous Galerkin (EDG) is obtained by enforcing global continuity of the traces so they are single-valued at mesh vertices. However, it does so at the cost of optimal flux convergence, a property that affects the accuracy of the overall solution.

Hybridization was originally developed for the solution of mixed finite element methods. Mixed methods define the dual of the system - usually the flux or gradient - as an unknown variable, thus resulting in an even larger system than primal methods such as DG. These methods are popular because the gradient or flux is approximated without the loss of accuracy incurred from differentiating the approximate solution itself. While these methods may be feasible to solve in one spatial dimension, the system becomes very large in multiple dimensions and high orders. In 1965, De Veubeque recognized in the context of linear elasticity that by using a trace the resulting system size could be dramatically reduced through static condensation [22].

Cockburn, Guzmaǹ and Wang [23] determine a link between the hybridized versions of two popular mixed methods using the Raviart-Thomas (RT) and Brezzi-Douglas-Marini (BDM) spaces, respectively, and derive conditions under which the methods yield identical results. Later it was recognized that this procedure unifies a number of seemingly different hybridizable finite element methods simply by switching various parameters [24].

Although the term HDG refers to a family of methods, it is most often used to refer to the hybridized local discontinuous Galerkin (LDG-H) method using the space $\mathcal{V}_h$ from (2.20) for the approximate solution and $\vec{\mathcal{V}}_h$ for its flux. This method has become popular for its straightforward L^2 finite element spaces, and in certain situations optimal flux convergence and enhanced accuracy through a suitable projection. This projection can be implemented as an element-by-element post-processing step yielding a solution that converges at order $p+2$ in L^2 [23], where other DG methods usually converge at $p+1$. These methods have been studied and characterized extensively for the solution of purely second-order equations and steady-state convection-diffusion equations in [25, 26, 27].

4.1.1 Formulation for Linear Convection-Diffusion

Much of the HDG development has been focused on its application to scalar equations, such as the steady-state convection-diffusion equation

$$\vec{q} = \nabla u, \quad \nabla \cdot (\vec{a}u - b\,\vec{q}) + f = 0, \quad u = g \ \text{ on } \partial\Omega \tag{4.1}$$

where $\vec{a}$ is the velocity, b is the relative diffusivity, and f is a passive source term. In order to form an HDG discretization, first consider the flux formulation of the equations similar to (2.25) for the Poisson equation. For the convection-diffusion equation above, on an element Ω_e this takes the form

$$(\vec{q}_h, \vec{v})_{\Omega_e} + (u_h, \nabla \cdot \vec{v})_{\Omega_e} - \langle \hat{u}, \vec{v} \cdot \vec{n} \rangle_{\partial\Omega_e} = 0 \qquad \forall \vec{v} \in [\mathcal{P}^p(\Omega_e)]^d \tag{4.2}$$

$$-\big(\vec{a}u_h - b\vec{q}_h, \nabla w\big)_{\Omega_e} + \big\langle \widehat{\vec{a}u_h} - \widehat{b\vec{q}_h}, w \big\rangle_{\partial\Omega_e} = \big(f, w\big)_{\Omega_e} \qquad \forall w \in \mathcal{P}^p(\Omega_e). \tag{4.3}$$

HDG methods define the trace $\hat{u}$ as an unknown to be solved for $\lambda_h \in M_h^I$ on interior faces and let the boundary conditions define the state on the boundary faces. On these faces, the upwind state, $g_h = \mathbb{P}g$ a projected $L^2(\partial\Omega)$ function when $\vec{a} \cdot \vec{n}^+ > 0$, or u_h^+ is inserted. Note that this definition of the trace differs from that used in most other past HDG formulations. This is immediately a difference with DG, which

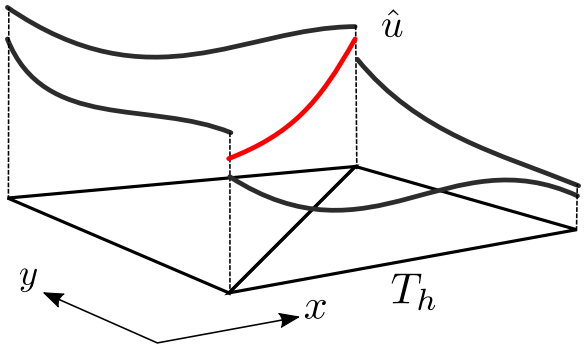


Figure 4.1: HDG trace around two elements. The central trace, shown in red is coupled to all other traces on neighboring elements.

eliminates $\hat{u}$ as an unknown by specifying its form.

For a finite element method to be hybridizable, the trace $\hat{u}$ must be single-valued, except at mesh vertices, though fortunately all adjoint consistent methods are of this form [24]. Additionally, instead of eliminating the gradients as DG would, mixed HDG methods keep $\vec{q}_h$ and the set of equations (4.2). Assuming the trace is known around an element, all that is required for the local solver defined by (4.2) and (4.3) is to define on the element faces the convective and diffusive fluxes, $\widehat{\vec{a}u_h}$ and $\widehat{b\vec{q}_h}$, respectively. These fluxes are the scalar HDG analogs of $\hat{\mathbf{F}}$ and $\hat{\mathbf{G}}$ in (2.34) and (2.35). The local solver is an injective mapping from the space of traces to the intra-element solution. In order to hybridize the method, the local solver cannot depend directly on the neighboring element's approximate solution, so the total flux is

$$\widehat{\vec{a}u_h} - \widehat{b\vec{q}_h} = \vec{a}\hat{u}\cdot\vec{n} - b\,\vec{q}_h\cdot\vec{n} + \vec{\delta}(u_h,\hat{u})\cdot\vec{n} \tag{4.4}$$

where the stabilization $\vec{\delta} = \vec{\delta}^F + \vec{\delta}^G$ is the flux stabilization split into convective and diffusive parts. Traditionally the stabilization takes the form

$$\vec{\delta}^F = \tau^F(u_h - \hat{u})\vec{n}, \quad \vec{\delta}^G = \tau^G(u_h - \hat{u})\vec{n}. \tag{4.5}$$

Whereas in DG the total flux $\hat{\mathbf{H}}$ was required to be single-valued, here the flux is in general double-valued. Note, however, that the total flux is still continuous in a weak sense.

A proper convective flux for the convection-dominated regime is one based on the upwind state, so that overall the flux is $\vec{a}u^+\cdot\vec{n}$ when $\vec{a}\cdot\vec{n}^+ > 0$ and $\vec{a}u^-\cdot\vec{n}$ otherwise. The convective stabilization for each side of the face, τ^F, should be chosen to satisfy

this property, so the overall convective flux

$$\widehat{\vec{a}u_h} = \vec{a}\hat{u}\cdot\vec{n} + \tau^F\left(u_h - \hat{u}\right) \tag{4.6}$$

is upwinded. Since the flux (4.6) is defined in terms of the trace, the stabilization should be chosen so that the trace is the upwind state. To do this, the method should pick $\tau^F > 0$ when $\vec{a}\cdot\vec{n} > 0$; a suitable choice is

$$\tau^F = \begin{cases} \frac{1}{2}\left(|\vec{a}\cdot\vec{n}| + \vec{a}\cdot\vec{n}\right) & \text{interior faces} \\ 0 & \text{boundary faces.} \end{cases} \tag{4.7}$$

The implicitly defined boundary state $\hat{u}(u_h^+)$ is already properly upwinded, so $\tau^F = 0$ on boundaries; in fact this is required for adjoint consistency. The resulting convective flux (4.6) is always upwinded,

$$\widehat{\vec{a}u_h} = \begin{cases} \vec{a}u_h^+\cdot\vec{n}, & \vec{a}\cdot\vec{n} > 0 \\ \vec{a}\hat{u}\cdot\vec{n} & \text{otherwise.} \end{cases} \tag{4.8}$$

This is the convective stabilization from the HDG II scheme for scalar linear systems proposed in Nguyen *et. al.* [27].

Fewer choices for the diffusion stabilization are provided. In [23] the authors prove that the gradients $\vec{q}_h$ converge optimally in L^2 for a purely diffusive problem ($\tau^F \equiv 0$) provided $\tau^G = \mathcal{O}(1)$ under mesh refinement. From a numerical analysis standpoint this is intuitive: if the stabilization is not $\mathcal{O}(1)$ with refinement, then either the flux (gradient) or u_h is being penalized too heavily. The authors go on to analyze the convective-dominated regime of steady-state convection-diffusion in [26] and propose using either $\tau^G = \eta$ and $\tau^G = \eta h^{-1}$, where $\eta = b/\ell_{\text{visc}}$ and ℓ_{visc} is a viscous length-scale for dimensional correctness. This length-scale is usually chosen based on the size of the domain or the length over which the viscosity is acting, for example the size of an airfoil in two dimensions. Other authors [27] also use this choice of viscous stabilization.

To couple the local solvers together and create a hybrid bilinear form for the trace variable, conservativity of the fluxes must also be enforced. Since the trace unknowns exist in the space M_h^I, to accomplish the conservation in a well-posed manner, the hybridized methods globally enforce

$$\left\langle \left[\!\left[\widehat{\vec{a}u_h} - b\widehat{\vec{q}_h}\right]\!\right], \mu \right\rangle_{E_h^I} = 0 \quad \forall \mu \in M_h^I, \tag{4.9}$$

where the fluxes are determined by the local solvers. Equations (4.9), and in fact the entire definition of the global system, form an implicit system for u_h and $\vec{q}_h$ on elements, since the boundary conditions are accounted for in the definition of the fluxes and elsewhere in the local solvers.

It is also valid to formulate HDG methods that define an unknown trace on boundaries, but doing so shows no advantages. When formulating in this way, the convective stabilization τ_c should be chosen to penalize the jump between the state and trace on these faces, and the conservativity condition (4.9) must be extended to all of $\mathcal{M}_h$, including the boundaries. The boundary traces add to the size of the global system and evidence shows no increase in accuracy when doing so.

For the linear steady-state convection-diffusion equation (4.1) with τ as above, the conservativity condition (4.9) equates the total flux pointwise. By equating the fluxes and solving for the trace, this can be written explicitly as

$$\lambda_h = \begin{cases} (\vec{a}\cdot\vec{n}^+ + 2\tau_d)^{-1}\left(\vec{a}\cdot\vec{n}^+ u^+ - b[\![\vec{q}\cdot\vec{n}]\!] + 2\tau_d\{u\}\right) & \text{for } \vec{a}\cdot\vec{n}^+ > 0 \\ (\vec{a}\cdot\vec{n}^- + 2\tau_d)^{-1}\left(\vec{a}\cdot\vec{n}^- u^- - b[\![\vec{q}\cdot\vec{n}]\!] + 2\tau_d\{u\}\right) & \text{otherwise.} \end{cases} \tag{4.10}$$

This verifies a well-known property that the HDG trace depends on the gradients and thus is not in the class of LDG methods for any finite value of τ^F. Although (4.10) in general has no elegant form, it does have the limits

$$\lambda = \begin{cases} \{u\} - \frac{\ell_{\text{visc}}}{2}[\![\vec{q}\cdot\vec{n}]\!] & |\vec{a}\cdot\vec{n}| \to 0 \\ u^{\text{up}} & b \to 0, \end{cases} \tag{4.11}$$

where u^{up} is the upwind state, u_h^+ when $\vec{a}\cdot\vec{n}^+ > 0$ and u_h^- otherwise. Thus in the convective limit the traces recover the upwind state, and in the diffusive limit HDG, specifically LDG-H, is different from any of the DG methods surveyed in [7].

The corresponding weak formulation of HDG involves only the trace as is written as

$$a(\lambda_h, \mu) = b(\mu) \quad \forall \mu \in \mathcal{M}_h^I, \tag{4.12}$$

and after converting to a linear system is

$$\mathbf{K}\boldsymbol{\Lambda} = \mathbf{F}. \tag{4.13}$$

In this system $\boldsymbol{\Lambda}$ is the vector of trace unknowns for all interior faces and the matrix $\mathbf{K}$ is referred to as the reduced Jacobian matrix. After the global linear solve, the

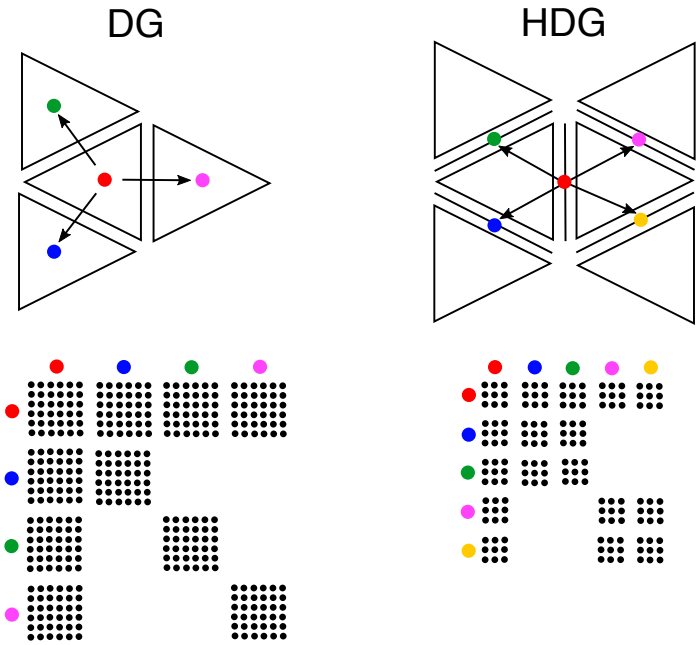


Figure 4.2: Jacobian elements (colored dots) over a triangular mesh with arrows showing coupling from the center element in red for both DG and HDG, and the resulting Jacobian matrix with $p = 2$.

local solvers provide $u_h|_{\Omega_e}$ and $\vec{q}_h|_{\Omega_e}$ for each element given $\hat{u}$ on the element faces.

The sparsity of the reduced Jacobian $\mathbf{K}$ matrix can be deduced by realizing that since the traces link together the local solvers, the local solvers depend on the surrounding traces. Thus, the set of rows in the matrix corresponding to the DOF for the lighter trace in Figure 4.1 have non-zero entries on the diagonal block, as well as columns corresponding to DOF of the darker traces. In contrast, the EDG method would have additional coupling to the traces sharing the mesh vertices.

4.1.2 Numerical Example: Interior Error Norm

Before continuing it is instructive to compare the L^2 errors of the above form of HDG for scalar convection-diffusion to those of DG. To do so, we consider the linear convection-diffusion equation on $\Omega = [0, 1]^2$ and add a source term s to render the exact solution $u = \sin(\pi x)\sin(\pi y)$. The projection $\mathbb{P}$ used to specify boundary conditions can slightly affect the convergence, so specifying homogeneous boundary conditions eliminates this possible source of differences. In these results DG uses an upwind convective flux and BR2 diffusion treatment with $\kappa = 2$ in (2.29), and HDG uses the upwind flux stabilization in (4.7) and constant diffusive flux stabilization with $\tau = b$.

Tables 4.2 and 4.3 compare the L^2 errors and rates of convergence for both numerical methods. The norms presented in the tables are defined as

$$\begin{aligned}
\|u_h - u\|_{L^2(\Omega)} &= \sqrt{\sum_{\Omega_e \in \Omega_h} \int_{\Omega_e} (u_h - u)^2 \, d\Omega} \\
\|\vec{q}_h - \nabla u\|_{L^2(\Omega)} &= \sqrt{\sum_{\Omega_e \in \Omega_h} \int_{\Omega_e} \sum_{i=1}^{d} \left(\vec{q}_{h,i} - \partial_i u\right)^2 \, d\Omega} \\
\|\hat{u} - u\|_{L^2(\partial\Omega_h)} &= \sqrt{\sum_{\Omega_e \in \Omega_h} \operatorname{diam}(\Omega_e) \int_{\partial\Omega_e} \left(\hat{u} - u\right)^2 \, ds}
\end{aligned} \tag{4.14}$$

where $\operatorname{diam}(\Omega_e)$ is a measure of the diameter of the element. For DG, the gradient is instead taken by differentiating the u_h basis functions so $\vec{q}_h = \nabla u_h$. For the purely diffusive results in Table 4.2, it is interesting to note (1) that HDG is consistent for $p = 0$, while DG fails to converge with mesh refinement, (2) that the rate of convergence for the gradients is one order higher using HDG than DG, and (3) that the rate of convergence of the trace is $p+2$ for both methods. As previously noted, if the stabilization had been $\tau = bh^{-1}$, the gradients would have converged at the same rate as the primal. For the convection-dominated case, since the diffusive stabilization is still $\mathcal{O}(1)$ the HDG rate of convergence is again higher, and in fact even if $b = 0$ the gradient optimally converges using the convective flux stabilization (4.7). This occurs because the gradient $\vec{q}_h$ is determined using the upwinded trace states in the integration by parts; for other choices of stabilization that do not purely upwind the trace state, this superconvergence does not occur.

4.1.3 Numerical Example: Boundary Output Error Norm

The primary goal of this work, targeting CFD methods, is to better estimate output functionals of the state as described in Chapter 3. The outputs of engineering interest often take the form of boundary integrals such as the lift or drag on a wing. Accordingly, this example tests the accuracy of DG and HDG on the linear convection diffusion equation (4.1), again on $\Omega = [0, 1]^2$, but with a boundary output. Instead of using homogeneous Dirichlet boundary conditions and adding a source term to specify the exact solution inside the domain as in Section 4.1.2, the source term is omitted and the following boundary function is set

$$u^B(x, y) = \exp\left(0.5 \sin(-4x + 6y) - 0.8 \cos(3x - 8y)\right) \quad \text{on } \partial\Omega. \tag{4.15}$$

p	N_e	$\|u_h - u\|_{L^2(\Omega)}$		$\|\nabla u_h - \nabla u\|_{L^2(\Omega)}$		$\|\hat{u} - u\|_{L^2(\partial T_h)}$	
		error	rate	error	rate	error	rate
	64	2.563e-1		0.222e1		6.533e-2	
0	256	2.516e-1	0.03	0.222e1	0.00	3.163e-2	1.05
	1024	2.504e-1	0.01	0.222e1	0.00	1.567e-2	1.01
	64	7.536e-3		2.516e-1		7.272e-4	
1	256	1.896e-3	1.99	1.259e-1	1.00	9.010e-5	3.01
	1024	4.749e-4	2.00	6.295e-2	1.00	1.123e-5	3.00
	64	1.987e-4		1.287e-2		3.851e-5	
2	256	2.481e-5	3.00	3.219e-3	2.00	2.444e-6	3.98
	1024	3.100e-6	3.00	8.048e-4	2.00	1.539e-7	3.99
	64	5.538e-6		4.237e-4		9.814e-7	
3	256	3.482e-7	3.99	5.296e-5	3.00	3.080e-8	4.99
	1024	2.180e-8	4.00	6.621e-6	3.00	9.635e-10	5.00
	64	9.316e-8		1.062e-5		1.780e-8	
4	256	2.907e-9	5.00	6.640e-7	4.00	2.815e-10	5.98
	1024	9.082e-11	5.00	4.151e-8	4.00	4.426e-12	5.99

(a) DG

p	N_e	$\|u_h - u\|_{L^2(\Omega)}$		$\|\vec{q}_h - \nabla u\|_{L^2(\Omega)}$		$\|\hat{u} - u\|_{L^2(\partial T_h)}$	
		error	rate	error	rate	error	rate
	64	1.174e-1		4.446e-1		2.078e-2	
0	256	5.987e-2	0.97	2.244e-1	0.99	5.237e-3	1.99
	1024	3.020e-2	0.99	1.126e-1	0.99	1.314e-3	1.99
	64	8.320e-3		3.084e-2		1.469e-3	
1	256	2.184e-3	1.93	8.007e-3	1.95	1.930e-4	2.93
	1024	5.598e-4	1.96	2.040e-3	1.97	2.474e-5	2.96
	64	2.672e-4		9.946e-4		4.723e-5	
2	256	3.449e-5	2.95	1.271e-4	2.97	3.048e-6	3.95
	1024	4.379e-6	2.98	1.607e-5	2.98	1.935e-7	3.98
	64	6.512e-6		2.423e-5		1.151e-6	
3	256	4.165e-7	3.97	1.539e-6	3.98	3.682e-8	4.97
	1024	2.633e-8	3.98	9.694e-8	3.99	1.164e-9	4.98
	64	1.273e-7		4.738e-7		2.251e-8	
4	256	4.052e-9	4.97	1.499e-8	4.98	3.581e-10	5.97
	1024	1.277e-10	4.99	4.714e-10	4.99	5.645e-12	5.99

(b) HDG

Table 4.2: Error convergence on uniformly refined quadrilateral meshes for the manufactured solution with $\vec{a} = (0, 0)$, $b = 1$.

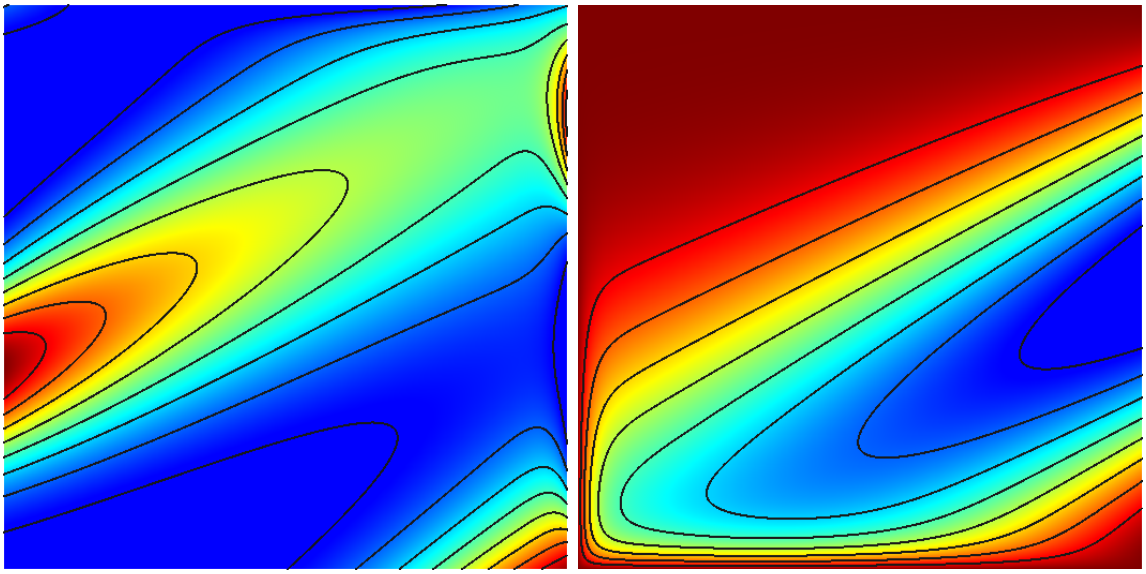
p	N_e	$\|u_h - u\|_{L^2(\Omega)}$		$\|\nabla u_h - \nabla u\|_{L^2(\Omega)}$		$\|\hat{u} - u\|_{L^2(\partial T_h)}$	
		error	rate	error	rate	error	rate
	64	2.193e-1		0.222e1		5.448e-2	
0	256	1.450e-1	0.60	0.222e1	0.00	1.844e-2	1.56
	1024	1.085e-1	0.42	0.222e1	-0.00	6.866e-3	1.43
	64	4.620e-3		2.565e-1		1.639e-3	
1	256	1.081e-3	2.10	1.267e-1	1.02	2.028e-4	3.02
	1024	2.634e-4	2.04	6.308e-2	1.01	2.522e-5	3.01
	64	3.046e-4		1.439e-2		8.112e-5	
2	256	3.084e-5	3.30	3.358e-3	2.10	3.645e-6	4.48
	1024	3.350e-6	3.20	8.142e-4	2.04	1.779e-7	4.36
	64	5.211e-6		4.688e-4		1.095e-6	
3	256	3.406e-7	3.94	5.523e-5	3.09	3.135e-8	5.13
	1024	2.167e-8	3.97	6.706e-6	3.04	9.627e-10	5.03
	64	9.883e-8		1.126e-5		2.055e-8	
4	256	2.906e-9	5.09	6.759e-7	4.06	2.829e-10	6.18
	1024	9.075e-11	5.00	4.170e-8	4.02	4.416e-12	6.00

(a) DG

p	N_e	$\|u_h - u\|_{L^2(\Omega)}$		$\|\vec{q}_h - \nabla u\|_{L^2(\Omega)}$		$\|\hat{u} - u\|_{L^2(\partial T_h)}$	
		error	rate	error	rate	error	rate
	64	3.670e-1		9.558e-1		3.408e-2	
0	256	2.246e-1	0.71	7.913e-1	0.27	1.130e-2	1.59
	1024	1.124e-1	1.00	5.714e-1	0.47	3.301e-3	1.78
	64	1.801e-2		6.555e-2		1.378e-3	
1	256	1.951e-3	3.21	9.052e-3	2.86	1.557e-4	3.15
	1024	4.227e-4	2.21	2.048e-3	2.14	1.862e-5	3.06
	64	1.944e-4		9.379e-4		3.411e-5	
2	256	2.471e-5	2.98	1.188e-4	2.98	2.185e-6	3.96
	1024	3.126e-6	2.98	1.554e-5	2.93	1.382e-7	3.98
	64	5.660e-6		2.778e-5		9.746e-7	
3	256	3.303e-7	4.10	1.592e-6	4.13	2.917e-8	5.06
	1024	1.998e-8	4.05	9.847e-8	4.02	8.828e-10	5.05
	64	8.995e-8		4.290e-7		1.590e-8	
4	256	2.896e-9	4.96	1.427e-8	4.91	2.559e-10	5.96
	1024	9.178e-11	4.98	4.672e-10	4.93	4.056e-12	5.98

(b) HDG

Table 4.3: Error convergence on a uniformly refined quadrilateral meshes for the manufactured solution with $\vec{a} = (\cos(\frac{\pi}{6}), \sin(\frac{\pi}{6}))$, $b = 10^{-2}$.



(a) Primal solution (b) Adjoint solution

Figure 4.3: Solution with $a = (\cos(\pi/6), \sin(\pi/6))$ and $b = 1/50$ and adjoint for the output defined by (4.16).

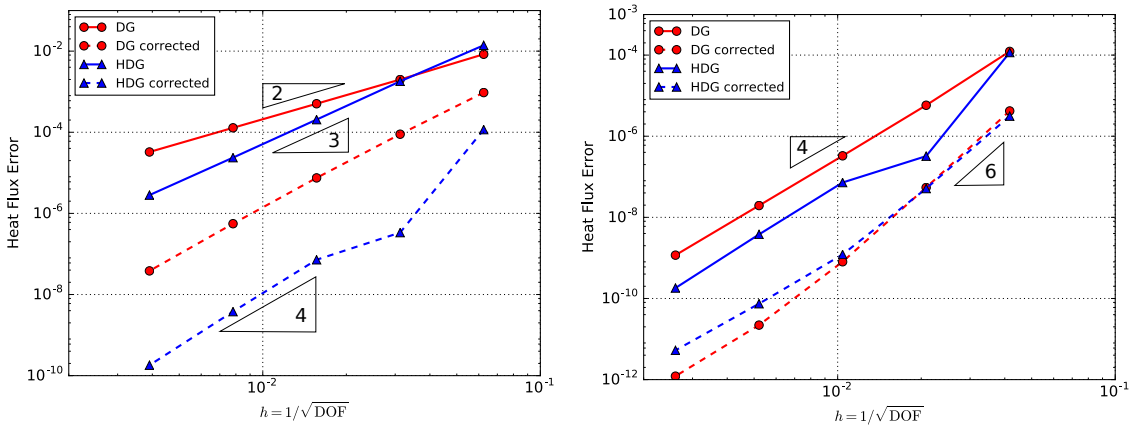
The output of interest will be the functional integrated over the right side of the domain

$$J(u) = \int_0^1 w(y) \frac{\partial u}{\partial x}(1, y)\, dy \quad w(y) = \frac{1}{2}\left(1 - \cos(2\pi y)\right). \tag{4.16}$$

The solution to this equation with $a = (\cos(\pi/6), \sin(\pi/6))$ and $b = 1/50$, and the adjoint for the output, are shown in Figure 4.3.

When the output is on a boundary, only part of the domain affects the results, so refinement should target these regions to effectively capture the output. In this example, however, since the output is affected by the gradient on most of the boundary, and the equations advect the solution primarily from left to right, most of the domain will affect the output and uniform refinement is a relatively effective approach to capturing the output.

Figure 4.4 plots the error in the weighted "heat flux" boundary output $|J(u) - J(u_h)|$ for different mesh refinement strategies for both DG and HDG. The results are obtained by refining each element of a uniform 64-element quadrilateral mesh four times. After each solve, the error estimate is computed according to (3.17) following the solution of the fine-space adjoint. The fine-space employed here and throughout uses order enrichment, so the adjoint used in the error estimation is solved at order $p + 1$. The discrete adjoint procedure for HDG is discussed in Section 4.2.6, but essentially computes the adjoint-weighted residual for all equations.



(a) $p = 1$ results

(b) $p = 2$ results

Figure 4.4: Comparison of the output (4.16) (solid) and corrected output (dashed) error calculated from the DG and HDG discretizations of the linear convection diffusion equation.

While the solution itself exhibits boundary layers and singularities, since the output contains a weight function that approaches zero at the endpoints, the adjoint field is smooth and the optimal rates of convergence are expected. The DG uncorrected results converge at approximately $2p$ as expected, and HDG $p = 1$ results converge at $2p+1$ thanks to the optimal convergence of $\vec{q}_h$ at the boundary. The output corrected by the error estimate for $p = 1$ converges at one order higher, so at third order for DG and fourth order for HDG. The $p = 2$ outputs for both methods seem to converge at exactly fourth order, and while the corrected output initially converges faster at sixth order, it finally slows to fourth order on the finer meshes.

4.2 Extension to Systems

The first contribution of this work extends the HDG method from linear scalar convection-diffusion theory to enable its application to systems of equations. Furthermore, the goal of this section is to outline an HDG discretization of general convection-diffusion systems that is computationally efficient and sufficiently robust to apply to general compressible Navier-Stokes computations.

4.2.1 Formulation for Transport System

The HDG discretization for a system discretizes $\mathbf{u}_h$, $\vec{\mathbf{q}}_h$, and $\boldsymbol{\lambda}_h$ according to the polynomial spaces specified in Table 4.4. The flux formulation or local solver for (2.1) is obtained for every element Ω_e by choosing approximation spaces locally for $\mathbf{u}_h$ and

Unknown	Test function	Local approx. space	Support
$\mathbf{u}_h$	$\mathbf{w}$	$\boldsymbol{\mathcal{P}}^{p_e}(\Omega_e) \equiv [\mathcal{P}^{p_e}(\Omega_e)]^s$	$\forall \Omega_e \in T_h$
$\vec{\mathbf{q}}_h$	$\vec{\mathbf{v}}$	$\vec{\boldsymbol{\mathcal{P}}}^{p_e}(\Omega_e) \equiv [\boldsymbol{\mathcal{P}}^{p_e}(\Omega_e)]^d$	$\forall \Omega_e \in T_h$
$\boldsymbol{\lambda}_h$	$\boldsymbol{\mu}$	$\boldsymbol{\mathcal{P}}^{p_f}(F) \equiv [\mathcal{P}^{p_f}(F)]^s$	$\forall F \in E_h^I$

Table 4.4: Spaces for unknowns in an HDG discretization.

$\vec{\mathbf{q}}_h$ and integrating by parts

$$\begin{aligned}
&\int_{\Omega_e} \mathbf{v}_i^T \mathbf{q}_{h,i}\, d\Omega + \int_{\Omega_e} \partial_i \mathbf{v}_i^T\, \mathbf{u}\, d\Omega - \int_{\partial\Omega_e} \mathbf{v}_i^T n_i \hat{\mathbf{u}}\, ds = 0, && \forall \vec{\mathbf{v}} \in \vec{\boldsymbol{\mathcal{P}}}^{p_e}(\Omega_e) \\
&\int_{\Omega_e} \mathbf{w}^T \partial_t \mathbf{u}_h\, d\Omega - \int_{\Omega_e} \partial_i \mathbf{w}^T \mathbf{H}_i\, d\Omega + \int_{\partial\Omega_e \setminus \partial\Omega} \mathbf{w}^T \left(\hat{\mathbf{F}} + \hat{\mathbf{G}}\right) ds \\
&\quad + \int_{\partial\Omega_e \cap \partial\Omega} \mathbf{w}^T \left(\hat{\mathbf{F}}^B + \hat{\mathbf{G}}^B\right) ds + \int_{\Omega_e} \mathbf{w}^T \mathbf{s}\, d\Omega = 0, && \forall \mathbf{w} \in \boldsymbol{\mathcal{P}}^{p_e}(\Omega_e).
\end{aligned} \tag{4.17}$$

Given a definition of the trace $\hat{\mathbf{u}}$ and total flux on an interface $\hat{\mathbf{H}}$, this defines the local solver for the state and gradient on Ω_e. The trace for the HDG discretization of a system, defined as

$$\hat{\mathbf{u}} = \begin{cases} \boldsymbol{\lambda}_h & \text{interior faces} \\ \mathbf{u}^B(\mathbf{u}_h^+, \mathrm{BC}) & \text{boundary faces,} \end{cases} \tag{4.18}$$

is left as an unknown on interior faces and on boundaries is again defined implicitly using the neighboring element state and boundary condition, so no unknown DOF are required. The general form for the total flux dotted with an outward-pointing normal is

$$\hat{\mathbf{H}}(\mathbf{u}_h, \vec{\mathbf{q}}_h, \hat{\mathbf{u}}, \vec{n}) = \hat{\mathbf{F}}(\mathbf{u}_h, \hat{\mathbf{u}}) + \hat{\mathbf{G}}(\mathbf{u}_h, \mathbf{q}_h, \hat{\mathbf{u}}) = \mathbf{H}_i(\hat{\mathbf{u}}, \vec{\mathbf{q}}_h) n_i + \boldsymbol{\delta}_i(\mathbf{u}_h, \hat{\mathbf{u}}, \vec{n}) n_i \tag{4.19}$$

where $\mathbf{u}_h$ and $\vec{\mathbf{q}}_h$ are taken from inside the element. Again, $\vec{\boldsymbol{\delta}} = \vec{\boldsymbol{\delta}}^F + \vec{\boldsymbol{\delta}}^G$ where

$$\vec{\boldsymbol{\delta}}^F = \boldsymbol{\tau}^F (\mathbf{u}_h - \hat{\mathbf{u}})\, \vec{n}, \quad \vec{\boldsymbol{\delta}}^G = \boldsymbol{\tau}^G (\mathbf{u}_h - \hat{\mathbf{u}})\, \vec{n}. \tag{4.20}$$

For nonlinear systems the stabilization matrix $\boldsymbol{\tau} \in \mathbb{R}^{s \times s}$ is not only critical for accuracy but also affects the stability of the resulting system.

For the convective stabilization, motivated by the analogy with a Lax-Friedrichs flux, a basic choice is $\boldsymbol{\tau}^F = c_{\max}\mathbf{I}$ involving the maximum wavespeed $c_{\max}$. Assuming no diffusive flux, $\lambda_h = \{\mathbf{u}_h\}$ with this stabilization, thus the flux is appropriately

upwinded but the resulting trace is not. A more sophisticated stabilization borrows from the Roe approximate Riemann solver, setting instead [28]

$$\boldsymbol{\tau}^F = \mathbf{R}|\boldsymbol{\Lambda}|\mathbf{L}, \tag{4.21}$$

where $\mathbf{L}$, $\mathbf{R}$, and $\boldsymbol{\Lambda}$ are the matrices of left and right eigenvectors, and eigenvalues of the flux Jacobian $\partial \mathbf{F}_i(\boldsymbol{\lambda}_h)/\partial \boldsymbol{\lambda}_h n_i$, respectively. With this choice the resulting flux is

$$\hat{\mathbf{F}}(\mathbf{u}_h^+, \hat{\mathbf{u}}) = \mathbf{F}_i(\hat{\mathbf{u}})n_i + \mathbf{R}|\boldsymbol{\Lambda}|\mathbf{L}\left(\mathbf{u}_h^+ - \hat{\mathbf{u}}\right). \tag{4.22}$$

The trace is again the average of the states with this formulation, and thus is incorrectly upwinded. Properly upwinding the trace is critical for the stability of HDG for convection-dominated problems and when post-processing. Section 4.4.2 contains additional information regarding methods to properly upwind the trace.

The form of convective flux depends on the boundary condition. In general a Riemann solver is necessary to correctly resolve the incoming and outgoing characteristics if the desired state vector $\mathbf{u}$ is specified; on these boundaries $\hat{\mathbf{F}}^B = \hat{\mathbf{F}}_j(\mathbf{u}_h^+, \mathbf{u})$. This formulation is consistent with the treatment of the Dirichlet condition for the scalar formulation, in which the flux is based on the upwind state at the boundaries. For other boundary conditions, the convective flux based on the implicitly-defined boundary state $\hat{\mathbf{u}}$ is used, without the stabilization, so $\vec{\boldsymbol{\delta}}^F = \vec{\mathbf{0}}$ on those faces. The reason for not including stabilization on these faces is that the convective flux is already used and no unknown trace state exists, so the additional upwinding is unnecessary and would in fact be adjoint inconsistent.

Diffusive stabilization, on the other hand, is added on all faces. On boundary faces, the resulting diffusive flux is projected as

$$\hat{\mathbf{G}}^B = \Pi_G^B \left[\mathbf{G}_i(\mathbf{u}_h^B, \mathbf{q}_h^+)n_i + \boldsymbol{\delta}_i^G n_i\right], \tag{4.23}$$

identically to the treatment for DG in (2.39). As for a single equation, a constant viscous stabilization, for instance $\boldsymbol{\tau}^G = \eta\mathbf{I}$, is necessary for optimal convergence of the state gradients $\vec{\mathbf{q}}_h$. Evidence has shown, however, that in general this choice, depending on the value of η, can result in stiff nonlinear systems. Some past works, *e.g.* [28], have ignored $\vec{\boldsymbol{\delta}}^G$ altogether, and relied on the convective stabilization providing enough coupling to $\hat{\mathbf{u}}$. While this is acceptable for relatively simple cases, for more

difficult cases this again increases stiffness. Another approach [29] sets

$$\boldsymbol{\tau}^G(\hat{\mathbf{u}}) = \frac{n_i \mathbf{K}_{ij}(\hat{\mathbf{u}}) n_j}{\ell_{\text{visc}}}. \tag{4.24}$$

While using a simple constant $\eta\mathbf{I}$ is also consistent with the PDE, it assumes there is diffusion in all of the equations. Using (4.24) has the benefit of adding stabilization only to the equations that require it, and adds it proportionally to the amount of viscosity in the equation. Again the ambiguously-defined viscous length-scale ℓ_{visc} must still be chosen.

The local solvers (4.17) for each element Ω_e are coupled together by the conservativity condition

$$\int_F \boldsymbol{\mu}^T \left[\!\!\left[\hat{\mathbf{F}} + \hat{\mathbf{G}} \right]\!\!\right] ds = 0 \qquad \forall \boldsymbol{\mu} \in \boldsymbol{\mathcal{M}}_h^I, \tag{4.25}$$

where $\boldsymbol{\mathcal{M}}_h^I$ is the product of the local approximation spaces in Table 4.4 for all interior faces F. Together these define a linear system to be solved for the traces $\boldsymbol{\lambda}_h$ from which $\mathbf{u}_h$ and $\vec{\mathbf{q}}_h$ are determined by the local solvers.

4.2.2 Implementation

The state vectors $\mathbf{q}_h$ for each dimension d, as well as $\mathbf{u}_h$, and $\boldsymbol{\lambda}_h$, are each stored as an expansion in N_p basis functions as

$$\begin{aligned}
\vec{\mathbf{q}}_h(\vec{x})\big|_{\Omega_e} &= \sum_{i=1}^{d} \sum_{n=1}^{N_p} \mathbf{Q}_{ein} \phi_n(\vec{x}) \hat{x}_i \\
\mathbf{u}_h(\vec{x})\big|_{\Omega_e} &= \sum_{n=1}^{N_p} \mathbf{U}_{en} \phi_n(\vec{x}) \\
\boldsymbol{\lambda}_h(\vec{x})\big|_{f} &= \sum_{n=1}^{N_p} \boldsymbol{\Lambda}_{fn} \mu_n(\vec{s}),
\end{aligned} \tag{4.26}$$

where $\vec{x} \in \mathbb{R}^d$ denotes a coordinate inside element Ω_e, $\vec{s} \in \mathbb{R}^{d-1}$ denotes the coordinate on interior face f, $\hat{x}_i$ is the x_i-coordinate unit normal, and n indexes the local approximation space basis functions ϕ and μ for the element and face, respectively. Most implementations use basis functions of the same order for $\vec{\mathbf{q}}_h$ and $\mathbf{u}_h$, since evidence shows no extra accuracy from using a higher order for either variable. With a different form of the HDG stabilization, Qui *et al.* [30] have shown a benefit to using one order higher in the elemental state.

Using the discrete approximations for the unknowns in (4.26), the local solvers

coupled with the conservativity condition can be written as a system of equations driving the following discrete residual vectors to zero,

$$\begin{aligned}
\mathbf{R}^{\mathbf{Q}}_{ein} &= \int_{\Omega_e} \phi_n \mathbf{q}_{h,i}\, d\Omega + \int_{\Omega_e} \partial_i \phi_n\, \mathbf{u}_h\, d\Omega - \int_{\partial\Omega_e} \phi_n n_i \hat{\mathbf{u}}\, ds \\
\mathbf{R}^{\mathbf{U}}_{en} &= \int_{\Omega_e} \phi_n \partial_t \mathbf{u}_h\, d\Omega - \int_{\Omega_e} \partial_i \phi_n \mathbf{H}_i(\mathbf{u}_h, \vec{\mathbf{q}}_h)\, d\Omega + \int_{\partial\Omega_e} \phi_n \hat{\mathbf{H}}(\mathbf{u}_h, \vec{\mathbf{q}}_h, \hat{\mathbf{u}}, \vec{n})\, ds + \int_{\Omega_e} \phi_n \mathbf{s}\, d\Omega \\
\mathbf{R}^{\mathbf{\Lambda}}_{fn} &= \int_f \mu_n \left(\hat{\mathbf{H}}(\mathbf{u}_h^+, \vec{\mathbf{q}}_h^+, \boldsymbol{\lambda}_h, \vec{n}^+) + \hat{\mathbf{H}}(\mathbf{u}_h^-, \vec{\mathbf{q}}_h^-, \boldsymbol{\lambda}_h, \vec{n}^-) \right) ds
\end{aligned} \tag{4.27}$$

for all elements Ω_e and interior faces f. Note that the boundary terms that were explicitly written in (4.17) are absorbed into the $\hat{\mathbf{H}}$ integral for simplicity. The most important observation about these equations is that the $\mathbf{R}^{Q}_{ein}$ and $\mathbf{R}^{U}_{en}$ residuals incorporate only the states on the element Ω_e themselves and the traces $\hat{\mathbf{u}}$ on $\partial\Omega_e$.

4.2.3 Static Condensation

If the associated fluxes are linear, or if these equations are solved with a series of Newton iterations, the equations (4.27) result in the following linear system structure

$$\begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix} \begin{bmatrix} \mathbf{QU} \\ \mathbf{\Lambda} \end{bmatrix} = \begin{bmatrix} \mathbf{P} \\ \mathbf{E} \end{bmatrix}. \tag{4.28}$$

The first row of blocks, $\mathbf{A}$ and $\mathbf{B}$, include both the set of local solver equations so it convenient to define $\mathbf{QU} = (\mathbf{Q}, \mathbf{U})^T$, along with $\mathbf{R}^{\mathbf{QU}} = (\mathbf{R}^{\mathbf{Q}}, \mathbf{R}^{\mathbf{U}})^T$. The definitions of the individual blocks are

$$\mathbf{A} = \frac{\partial \mathbf{R}^{\mathbf{QU}}}{\partial \mathbf{QU}}, \quad \mathbf{B} = \frac{\partial \mathbf{R}^{\mathbf{QU}}}{\partial \mathbf{\Lambda}}, \quad \mathbf{C} = \frac{\partial \mathbf{R}^{\Lambda}}{\partial \mathbf{QU}}, \quad \mathbf{D} = \frac{\partial \mathbf{R}^{\Lambda}}{\partial \mathbf{\Lambda}} \tag{4.29}$$

Although the system could be solved in this form, the system size is unnecessarily large. Benefiting from the earlier observation that the local solver depends only on element-local values at the surrounding traces $\boldsymbol{\lambda}_h$, $\mathbf{QU}$ can be efficiently removed through static condensation yielding the reduced system

$$\mathbf{K\Lambda} = \mathbf{F} \qquad \begin{aligned} \mathbf{K} &= \mathbf{D} - \mathbf{C}\mathbf{A}^{-1}\mathbf{B} \\ \mathbf{F} &= \mathbf{E} - \mathbf{C}\mathbf{A}^{-1}\mathbf{R}. \end{aligned} \tag{4.30}$$

This is possible to build efficiently because $\mathbf{A}$ is block-diagonal element-wise. After this is solved, $\mathbf{QU}$ can be obtained by the local solver alone

$$\mathbf{QU} = \mathbf{A}^{-1}\left(\mathbf{R} - \mathbf{B}\boldsymbol{\Lambda}\right). \tag{4.31}$$

The reduced Jacobian matrix couples traces together that are indirectly linked by the local solvers, and so contains the "transmission" of residual across the element shared by the faces. Although the reduced system is almost always smaller than before static condensation, there is still a price to be paid. Static condensation of the Jacobian can be performed efficiently using a single loop over the elements. On each element, first all the integrals and their linearizations appearing in in (4.27) are calculated. Then all pairs of faces, call these f and g, are looped over to add the effect of face f on g through element Ω_e

$$\mathbf{K}_{fg} \overset{+}{=} \underbrace{\frac{\partial \mathbf{R}_f^\Lambda}{\partial \boldsymbol{\Lambda}_g}\delta_{fg}}_{\mathbf{D}_{fg}} - \underbrace{\begin{bmatrix}\frac{\partial \mathbf{R}_f^\Lambda}{\partial \mathbf{Q}_e} & \frac{\partial \mathbf{R}_f^\Lambda}{\partial \mathbf{U}_e}\end{bmatrix}}_{\mathbf{C}_{fe}} \underbrace{\begin{bmatrix}\frac{\partial \mathbf{R}_e^Q}{\partial \mathbf{Q}_e} & \frac{\partial \mathbf{R}_e^Q}{\partial \mathbf{U}_e} \\ \frac{\partial \mathbf{R}_e^U}{\partial \mathbf{Q}_e} & \frac{\partial \mathbf{R}_e^U}{\partial \mathbf{U}_e}\end{bmatrix}^{-1}}_{\mathbf{A}_e} \underbrace{\begin{bmatrix}\frac{\partial \mathbf{R}_e^Q}{\partial \boldsymbol{\Lambda}_g} \\ \frac{\partial \mathbf{R}_e^U}{\partial \boldsymbol{\Lambda}_g}\end{bmatrix}}_{\mathbf{B}_{eg}}. \tag{4.32}$$

The inversion of the local solver block $\mathbf{A}$ for high polynomial orders is not inexpensive, so this must be handled carefully in the implementation. Fortunately again $\frac{\partial \mathbf{R}_e^Q}{\partial \mathbf{Q}_e}$ admits a block-diagonal structure, one block for each dimension, each the size of $\frac{\partial \mathbf{R}_e^U}{\partial \mathbf{U}_e}$. Fast block-inversion formulas can therefore be applied for $\mathbf{A}_e^{-1}$.

Each of these matrices takes the form

$$\mathbf{A}_e = \begin{bmatrix}\frac{\partial \mathbf{R}_{e1}^Q}{\partial \mathbf{Q}_{e1}} & & & \frac{\partial \mathbf{R}_{e1}^Q}{\partial \mathbf{U}_e} \\ & \frac{\partial \mathbf{R}_{e2}^Q}{\partial \mathbf{Q}_{e2}} & & \frac{\partial \mathbf{R}_{e2}^Q}{\partial \mathbf{U}_e} \\ & & \ddots & \vdots \\ \frac{\partial \mathbf{R}_e^U}{\partial \mathbf{Q}_{e1}} & \frac{\partial \mathbf{R}_e^U}{\partial \mathbf{Q}_{e2}} & \cdots & \frac{\partial \mathbf{R}_e^U}{\partial \mathbf{U}_e}\end{bmatrix} = \begin{bmatrix}\ddots & & & \vdots \\ & \mathbf{A}_{QQ} & & \mathbf{A}_{Q_iU} \\ & & \ddots & \vdots \\ \cdots & \mathbf{A}_{UQ_j} & \cdots & \mathbf{A}_{UU}\end{bmatrix}. \tag{4.33}$$

The $\mathbf{A}_{QQ}$ blocks along the diagonal are identical as long as the same polynomial order is used for each vector component, so these are each denoted by $\mathbf{A}_{QQ}$. In order to efficiently invert this type of matrix a block-inversion formula can be applied. This is a well-known technique, but lesser referenced is the algorithm for efficiently implementing the block inversion. $\mathbf{A}_e^{-1}$ must be applied twice, so the first step is to pre-compute blocks necessary for the inversion, storing them in place. This process is shown in Algorithm 1, which applies in all cases, even when $\vec{\mathbf{q}}_h$ do not exist by

setting $d = 0$. The pre-processing function overwrites the blocks to

$$\mathbf{A}_e = \begin{bmatrix} \mathbf{A}_{QQ} & \mathbf{A}_{QQ}^{-1}\mathbf{A}_{Q_iU} \\ \mathbf{K}^{-1}\mathbf{A}_{UQ_j} & \text{PLU}(\mathbf{K}) \end{bmatrix},$$

where $PLU(\mathbf{K})$ denotes the in-place LU-factorization storage with permutation vector P stored separately. After pre-processing, the matrix can be applied relatively inexpensively to another block $\mathbf{A}_e^{-1}(\mathbf{B}_Q, \mathbf{B}_U)^T$ according to Algorithm 2, overwriting the $\mathbf{B}$ block. The application of the block-inversion matrices can altogether alternatively be thought of as a second level of static condensation using (4.30) then (4.31).

The inverse must be applied twice, once to each $\mathbf{B}_{ef}$ for each face of the element, and once to the vector $\mathbf{R}_e$ to build the linear system. After the linear solve, the inverse must again be applied to solve for $\mathbf{P}_e$.

Algorithm 1 Pre-process block $\mathbf{A}_e$ matrix for fast application of $\mathbf{A}_e^{-1}$ to the statically condensed matrix and residual.

function PREPROCESSA($\mathbf{A}$)
  for $i = 1 \ldots d$ **do**
    $\mathbf{A}_{Q_iU} \leftarrow \mathbf{A}_{QQ}^{-1}\mathbf{A}_{Q_iU}$
  end for
  for $i = 1 \ldots d$ **do**
    $\mathbf{A}_{UU} \leftarrow \mathbf{A}_{UU} - \mathbf{A}_{UQ_i}\mathbf{A}_{Q_iU}$
  end for
  PLU factor $\mathbf{A}_{UU}$ in place
  for $i = 1 \ldots d$ **do**
    $\mathbf{A}_{UQ_i} \leftarrow \mathbf{A}_{UU}^{-1}\mathbf{A}_{UQ_i}$
  end for
end function

4.2.4 Nonlinear and Linear Solver

A Newton solver with pseudo-time continuation (PTC) is employed to solve the resulting possibly nonlinear system of equations [6], resulting in a set of linear systems to solve. Pseudo-time continuation adds the effect of a backward-Euler time discretization to each Newton iteration on the linearization, but not on the residuals themselves. This has the effect of reducing the stiffness of the linear systems, since there exists less of a difference between the current approximate solution and the solution of the Newton system for a finite time-step. For DG, this could be implemented by only giving the nonlinear solver knowledge of the residuals and mass matrix, since the pseudo-time term is added to the entire Jacobian matrix. On the other hand,

Algorithm 2 Left-multiply $\mathbf{B}_{ef}$ by pre-processed $\mathbf{A}_e^{-1}$ matrix.

function ApplyAInverse($\mathbf{A}$)
  for $i = 1 \dots d$ **do**
    $\mathbf{B}_{Q_i} \leftarrow \mathbf{A}_{QQ}\mathbf{B}_{Q_i}$
  end for
  $\mathbf{X} \leftarrow -\mathbf{A}_{UU}^{-1}\mathbf{B}_U$ ▷ Use PLU-factored $\mathbf{A}_{UU}$
  for $i = 1 \dots d$ **do**
    $\mathbf{X} \leftarrow \mathbf{X} + \mathbf{A}_{UQ_i}\mathbf{B}_{Q_i}$
  end for
  for $i = 1 \dots d$ **do**
    $\mathbf{B}_{Q_i} \leftarrow \mathbf{B}_{Q_i} + \mathbf{A}_{Q_iU}\mathbf{X}$
  end for
  $\mathbf{B}_U = -\mathbf{X}$
end function

the term is only added to the local residuals associated with the $\mathbf{w}$ test functions in HDG, so the local solver needs to add the effect of the pseudo-timestep Δt before static condensation via

$$\frac{\partial \mathbf{R}_{ei}^U}{\partial \mathbf{U}_{en}} \stackrel{+}{=} \frac{1}{\Delta t} \int_{\Omega_e} \phi_i \phi_n d\Omega$$

The time-continuation strategies to take $\Delta t \to \infty$ for DG also yield successful convergence for HDG; this work employs the Exponential progression with under-relaxation (EXPur) method. A line search is used for RANS cases, which largely follows the definition in [6] Algorithm 3.4, with the exception that the HDG residuals are not statically condensed for each step.

The resulting linear systems from the Newton iterations are solved using a preconditioned Generalized minimal residual method (GMRES) method. This has been shown to be a very efficient iterative method for the solution of these systems for DG [31]. Several preconditioners are employed: block incomplete LU factorization without fill (ILU0) with MDF ordering [31], and block line-Jacobi smoothing [2]. The results in this work apply the preconditioner on the right side of the system matrix.

Performance of preconditioners applied to HDG systems can be expected to degrade compared to DG systems due to the increased linkage in the Jacobian matrix. Jacobian blocks from DG represent element contributions, while those from HDG link interior faces, so preconditioners that used direct analogies to the mesh need to be generalized. While DG systems have the number of off-diagonal blocks equal to the number of mesh faces per element n_f, HDG systems have $2(n_f - 1)$ off-diagonal blocks. Thus for multiple spatial dimensions HDG will always have more, albeit usually smaller, off-diagonal blocks.

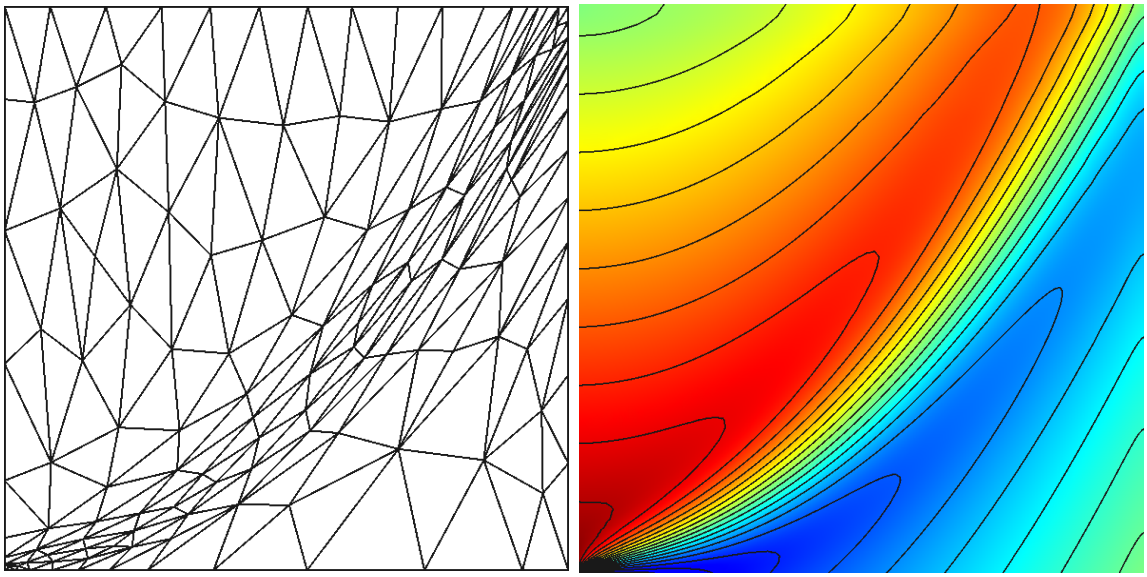
Since Line-Jacobi smoothing inverts the matrix along elemental lines of greatest coupling in DG, it does so along lines of greatest interior face coupling in HDG, overall inverting less of the overall Jacobian. The low-storage LU factorization method [31] assumes that the situation where Jacobian element k is a neighbor of j and i when j and i are also neighbors does not occur. With this assumption the entire LU factorization can be stored in-place using the existing allocated Jacobian matrix. In DG this situation is highly anomalous, but unfortunately, in HDG this always occurs due to the higher level of connectivity. This can be observed even with the simple case shown in Figure 4.1; the two darker traces on either element are mutual neighbors, and also neighbors of the shared lighter trace.

Figure 4.6 compares the number of GMRES iterations necessary to solve the systems resulting from DG and HDG discretizations of the scalar convection-diffusion case similar to that described in [31] Section 6.2. The results here differ slightly by using a different mesh, shown in Figure 4.5a, and setting a zero-gradient condition on the top and right, shown in Figure 4.5b, instead of a free boundary condition. The convective and diffusive limits are on the left and right side of both Figures 4.6a and 4.6b, respectively. The first two entries in the legend, DG-BR2, DG with BR2, and HDG-M-C, HDG with the constant-scaling viscous stabilization (4.24), correspond to the methods discussed in Sections 4.1.1 and 2.3.3. The performance degradation when using BR2-type stabilization is expected *e.g.* from Theorem 3.6 in [32]. In fact the performance degrades for both preconditioners except when ILU0 is used with constant viscous stabilization. Although this result is for a specific linear case, experience suggests this finding generalizes to systems.

4.2.5 Parallelization

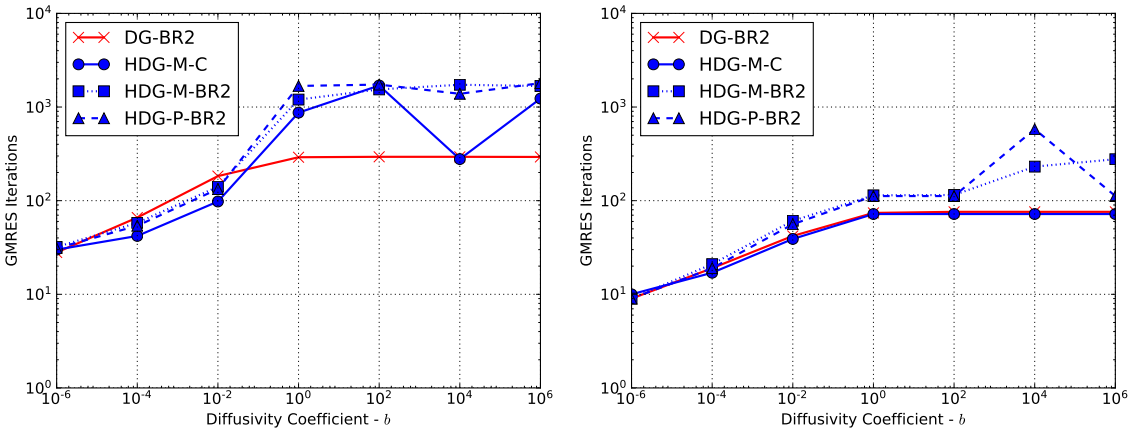
One of the most interesting aspects of HDG is the indirect coupling of the states on elements, and this certainly affects how the algorithm should be designed for parallelism compared to DG. In general two levels of parallelism should be considered: splitting the large problem onto multiple processes that pass information back and forth, and taking advantage of the multithreading capabilities of CPUs and coprocessors. This work focuses on distributed memory parallelism to split the large global problem and enable scaling to massive computations, leaving threading for future investigation.

In order to split the problem effectively, it should be split into sets of connected elements and faces, as shown in Figure 4.7 to minimize the number of faces straddling interfaces, thereby minimizing the communication necessary between processes. HDG



(a) Mesh for computations (b) Solution for $b = 10^{-2}$

Figure 4.5: Mesh and example solution with $b = 10^{-2}$ used for the comparison in Fig. 4.6.



(a) Line-Jacobi (b) ILU0

Figure 4.6: Number of GMRES iterations necessary to solve the linear system to machine tolerance using ILU0 and Line-Jacobi preconditioners with different discretizations at $p = 4$.

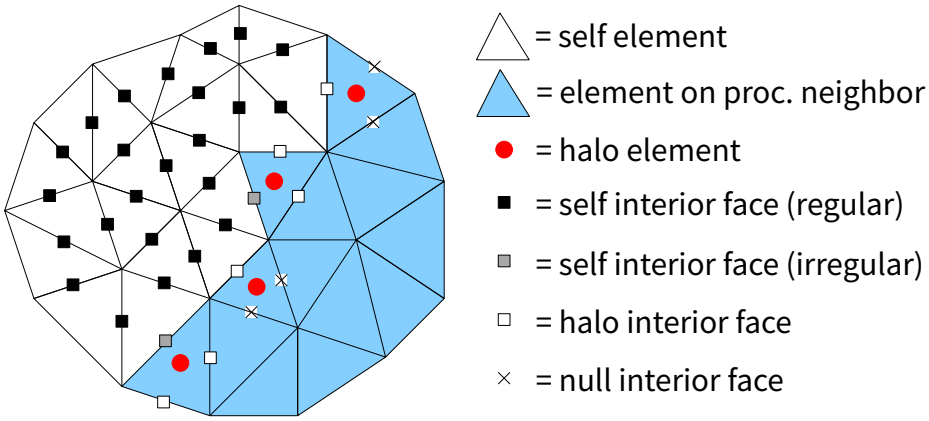


Figure 4.7: Notation used in parallelizing an unstructured HDG solver.

shares this approach, but has the added complication of using both local solvers evaluated element-wise and also an interior face-based linear system. Thus both elements and faces need to be properly parallelized so no interior face or element is computed by updating more than one process.

Firstly elements are assigned to processes using a weighted partition balancing algorithm via the METIS library [33]. Using this method, the elements with the strongest coupling remain on the same process so the preconditioner remains effective while the load is balanced. Clearly for DG the element (graph node) weights should be chosen based on the polynomial order of the state approximation, and the inter-element (graph edge) weights based on the level of coupling between elements. For nonlinear problems this can be difficult to identify as the state changes between nonlinear solver iterations affect the coupling, so the partitions could be rebalanced. The DG edge weights are the average of the norms of the linearization, so for the edge connecting elements i and j

$$w_{ij}^{\mathrm{DG}} = \left\| \frac{\partial \mathbf{R}_i}{\partial \mathbf{u}_j} \right\|_2 + \left\| \frac{\partial \mathbf{R}_j}{\partial \mathbf{u}_i} \right\|_2 . \tag{4.34}$$

In HDG, faces are coupled together through the local solvers on elements, so for the edge connecting elements i and j

$$w_{ij}^{\mathrm{HDG}} = \left\| \frac{\partial \mathbf{R}_i^U}{\partial \boldsymbol{\lambda}_h} \right\|_2 + \left\| \frac{\partial \mathbf{R}_j^U}{\partial \boldsymbol{\lambda}_h} \right\|_2 . \tag{4.35}$$

After the partitions are defined, HDG still needs to determine ownership of the faces straddling processes. This is done by looping over the faces and assigning them to the process with the fewest total interior faces. These become irregular interior

faces for the assigned process, shown in gray boxes for the left process in Figure 4.7, and they depend on state and element information as well as surrounding faces owned by the neighboring process. As non-blocking communication is usually possible, it is best to begin computing the residuals and linearizations of certain faces while communicating, then computing those residuals that depend on the neighbor. In HDG, after the linear solve for the traces, the updated values have to be transferred to neighboring processes for the local solver to update the elemental state.

This parallelization scheme was adopted to be most compatible with DG. An alternative method could, however, parallelize based on the dual mesh, balancing the faces and including element data as necessary. This approach would be more natural for the face-based HDG method, but would require an extra loop to assign elements to processors for DG.

4.2.6 Discrete Adjoint and Error Estimation

The discrete form of the equations resulting from HDG, shown in (4.27), is

$$\begin{bmatrix} \mathbf{R}^Q(\mathbf{Q},\mathbf{U},\mathbf{\Lambda}) \\ \mathbf{R}^U(\mathbf{Q},\mathbf{U},\mathbf{\Lambda}) \\ \mathbf{R}^\Lambda(\mathbf{Q},\mathbf{U},\mathbf{\Lambda}) \end{bmatrix} = \mathbf{R}_H(\mathbf{S}_H) = \mathbf{0}, \tag{4.36}$$

where $\mathbf{S}_H$ denotes the concatenated array of unknowns. The subscript H, while not present in the original definition of the discrete system or the state vector storage definition (4.26), is used here to denote the relative refinement of the approximation space as in Chapter 3. The remainder of the discrete adjoint theory is then applicable to this discrete system, treating the residuals as a single system. The transposed discrete adjoint system, as implemented, is

$$\left(\frac{\partial \mathbf{R}_H}{\partial \mathbf{S}_H}\right)^T \mathbf{\Psi}_H + \left(\frac{\partial J}{\partial \mathbf{S}_H}\right)^T = 0. \tag{4.37}$$

Note the sign change compared to the treatment presented in Section 3.1.2.

As expected, this shows that discrete adjoint variables exist for each set of residuals, corresponding to different test functions. Shütz and May [34] showed for more traditional variant of HDG that these adjoint variables correspond to the adjoint, its gradient, and its trace on the skeleton of the mesh. Appendix A shows that for the present formulation, with $\vec{\mathbf{q}}$ approximating $\nabla \mathbf{u}$, that the adjoint components approximate the adjoint, the adjoint diffusive flux, and the trace of the adjoint on the

skeleton of the mesh.

To efficiently compute the adjoint system, static condensation must be applied to the system resulting from (4.37) above:

$$\begin{bmatrix} \mathbf{A}^T & \mathbf{C}^T \\ \mathbf{B}^T & \mathbf{D}^T \end{bmatrix} \begin{bmatrix} \boldsymbol{\Psi}^{\mathbf{Q}\mathbf{U}} \\ \boldsymbol{\Psi}^{\Lambda} \end{bmatrix} = \begin{bmatrix} \frac{\partial J}{\partial \mathbf{Q}\mathbf{U}}^T \\ \frac{\partial J}{\partial \boldsymbol{\Lambda}}^T \end{bmatrix}. \tag{4.38}$$

After statically condensing the element-interior DOF the reduced system is

$$\underbrace{\left(\mathbf{D}^T - \mathbf{B}^T\mathbf{A}^{-T}\mathbf{C}^T\right)}_{\mathbf{K}^T} \boldsymbol{\Psi}^{\Lambda} = \left(\frac{\partial J}{\partial \boldsymbol{\Lambda}} - \mathbf{B}^T\mathbf{A}^{-T}\frac{\partial J}{\partial \mathbf{Q}\mathbf{U}}\right).$$

Therefore, the matrix is simply the transpose of the original reduced Jacobian system. The operation that creates the right side of this system, however, is different than that from the original system, which is shown in (4.30).

The transposed version of the static condensation is, in fact, more computationally efficient, since after applying Algorithm 2 in creating the reduced Jacobian matrix, the quantity $\mathbf{A}^{-1}\mathbf{B}$ is already stored, and $(\mathbf{A}^{-1}\mathbf{B})^T = \mathbf{B}^T\mathbf{A}^{-T}$ as needed. Only a single matrix-vector multiply is required, whereas for the original static condensation of the right side the operations are

1. $\mathbf{R} \leftarrow \mathbf{A}^{-1}\mathbf{R}$ (ApplyAInverse)
2. $\mathbf{F} \leftarrow \mathbf{E} + \mathbf{C}\mathbf{R}$ (Matrix Multiplication).

Therefore, the $2 + 3d$ matrix multiplications (step 1 above) necessary to create the right side of the original system are not present in the static condensation for the adjoint problem.

Creating the fine-space adjoint and the error estimate with HDG follows the procedure in Section 3.1.2 for a general system. The localization procedure, however, is less straightforward for HDG than for DG, in which the adjoint-weighted residuals were defined element-wise. The adjoint-weighted residual in HDG, on the other hand, contains contributions from each of the equations (4.27),

$$\epsilon^Q = \left(\boldsymbol{\Psi}_h^Q\right)^T \mathbf{R}_h^Q(\mathbf{S}_h^H), \quad \epsilon^U = \left(\boldsymbol{\Psi}_h^U\right)^T \mathbf{R}_h^U(\mathbf{S}_h^H), \quad \epsilon^\Lambda = \left(\boldsymbol{\Psi}_h^\Lambda\right)^T \mathbf{R}_h^\Lambda(\mathbf{S}_h^H).$$

While the global error estimate should sum all contributions, the ϵ^Λ, for instance, cannot be directly attributed to an element. It has been observed that the contributions from ϵ^Q and ϵ^Λ are usually many orders of magnitude smaller, but in certain

cases can become comparable to ϵ^U.

In order for HDG to retain the best possible approximation, the order, p, of the trace states $\boldsymbol{\lambda}_h$, and the dual variable $\vec{\mathbf{q}}_h$ must be kept at the same order as the approximate solution $\mathbf{u}_h$ itself. For this reason, half of the localized error estimates on faces $|\epsilon^\Lambda|$ are added to each of the neighboring element indicators $|\epsilon^Q| + |\epsilon^U|$, so

$$\epsilon_e = |\epsilon_e^Q| + |\epsilon_e^U| + \frac{1}{2}\sum_{f\in\partial\Omega_e} |\epsilon_f^\Lambda|. \tag{4.39}$$

4.3 Diffusion Formulation

Central to HDG methods is the choice of the stabilization parameter, or tensor for a system. This is especially critical because in the existing framework stabilization is the single parameter responsible for coupling the local solvers through the conservativity condition. Interestingly, precursors to modern LDG-H methods, the hybridized RT and BDM methods, did not require diffusive stabilization at all [24]. By contrast, diffusion systems resulting from LDG-H are singular without diffusive stabilization, and how the stabilization is chosen has consequences for the accuracy and system conditioning.

First consider some background about the methods themselves using the Laplace equation with Dirichlet boundary conditions. Cockburn *et al.* [35] prove an error estimate using a projected solution $(\vec{\Pi}_h\vec{q},\, \Pi_h u)$ in $[\mathcal{P}^p(\Omega_e)]^d \times \mathcal{P}^p(\Omega_e)$. The projected solution $(\vec{\Pi}_h\vec{q},\, \Pi_h u)$ is the element of $[\mathcal{P}^p(\Omega_e)]^d \times \mathcal{P}^p(\Omega_e)$ satisfying,

$$\begin{aligned}
\left(\vec{\Pi}_h\vec{q} - \vec{q}, \vec{v}\right)_{\Omega_e} &= 0 \quad \forall \vec{v} \in \vec{\mathcal{P}}^{p-1}(\Omega_e) \\
\left(\Pi_h u - u, w\right)_{\Omega_e} &= 0 \quad \forall w \in \mathcal{P}^{p-1}(\Omega_e) \\
\left\langle \vec{\Pi}_h\vec{q}\cdot\vec{n} + \tau\Pi_h u - u, \mu\right\rangle_F &= 0 \quad \forall \mu \in \mathcal{P}^p(F)
\end{aligned} \tag{4.40}$$

for all faces F of the mesh element Ω_e assuming $p > 0$. The approximation properties are given by the following theorem proved in the paper.

Theorem 4.1. *Suppose $p > 1$, $\tau|_{\partial\Omega_e}$ is non-negative and $\tau_{\Omega_e}^{max} := \max \tau|_{\partial\Omega_e} > 0$, then the projection $(\vec{\Pi}_h\vec{q},\, \Pi_h u)$ is uniquely solvable. Furthermore, there is a constant C, independent of Ω_e and τ such that for any s_q, $s_u \in (\frac{1}{2}, p+1]$*

$$\|\vec{\Pi}_h\vec{q} - \vec{q}\|_{\Omega_e} \le C h_{\Omega_e}^{s_q} |\vec{q}|_{H^{s_q}(\Omega_e)} + C h_{\Omega_e}^{s_u} \tau_{\Omega_e}^{*} |u|_{H^{s_u}(\Omega_e)}$$

$$\|\Pi_h u - u\|_{\Omega_e} \le C h_{\Omega_e}^{s_u} |u|_{H^{s_u}(\Omega_e)} + C \frac{h_{\Omega_e}^{s_q}}{\tau_{\Omega_e}^{max}} |\nabla\cdot\vec{q}|_{H^{s_q}(\Omega_e)}$$

where $\|\cdot\|_D$ *denotes the* $L^2(D)$*-norm for any domain* D*. Here* $\tau^*_{\Omega_e} := \max \tau|_{\partial\Omega_e \setminus F^*}$*, where* F^* *is a face of* Ω_e *at which* $\tau|_{\partial\Omega_e}$ *is maximum.*

Using the above estimates, Cockburn [32] proves the following estimates,

$$\begin{aligned}
&\|\vec{q}_h - \vec{q}\|_{T_h} \leq \|\vec{\Pi}_h\vec{q} - \vec{q}\|_{T_h} + \|\vec{\Pi}_h\vec{q} - \vec{q}_h\|_{T_h} \leq 2\|\vec{\Pi}_h\vec{q} - \vec{q}\|_{T_h} \\
&\|u_h - u\|_{T_h} \leq \|\Pi_h u - u\|_{T_h} + \|\Pi_h u - u_h\|_{T_h} \leq C\|\Pi_h u - u\|_{T_h} + b_\tau C\|\vec{\Pi}_h\vec{q} - \vec{q}\|_{T_h}
\end{aligned} \tag{4.41}$$

where

$$\begin{aligned}
h &= \max\{h_{\Omega_e} : \text{for all } \Omega_e \in T_h\} \\
b_\tau &= \max\{1 + h_{\Omega_e}\tau^*_{\Omega_e} + h_{\Omega_e}/\tau^{\max}_{\Omega_e} : \text{for all } \Omega_e \in T_h\}.
\end{aligned}$$

Although this result looks rather complicated, both $\vec{q}_h$ and u_h converge optimally in $L^2(T_h)$ as long as $\tau^{\max}_{\Omega_e}$ is either constant or $\mathcal{O}(h^{-1})$ and $\tau^*_{\Omega_e}$ is constant, $\mathcal{O}(h)$, or 0, for sufficiently smooth $\vec{q}$ and u. When this happens and both variables converge optimally, post-processing will produce a solution that converges even faster. Setting $\tau^{\max}_{\Omega_e} = \mathcal{O}(h^{-1})$ unfortunately ceases to be a viable option when considering the condition number estimate also proved in [32],

$$\kappa \leq Ch^{-2} \cdot \max\{1 + (\tau^*_{\Omega_e} h_{\Omega_e})^2 : \text{for all } \Omega_e \in T_h\}, \tag{4.42}$$

for a C independent of h. This implies that the condition number κ has a higher constant when super-penalized. The logical approach to obtaining optimal convergence of both variables is therefore to use constant stabilization.

Applying the above formulation to systems such as Navier-Stokes can lead to difficulties. In general the equations making up the system have differing levels of effective diffusion, and some equations, for instance mass conservation, have no diffusion at all. It is therefore fitting to set the viscous stabilization to a constant for each face proportional to the effect of viscosity on that equation, as in (4.24).

There exists a class of problems for which even the above stabilization method does not seem to routinely yield convergence. For these situations, it has been observed that using stabilization methods similar to those used for DG give more robust convergence [36].

4.3.1 One-Sided DG Stabilization

Similar viscous stabilizations to those introduced in Section 2.3.2 for DG can, with only slight modifications, be defined and used for HDG with the one-sided fluxes. The reason for using these is that constant-scaling stabilization methods lead to difficulty converging in certain cases. It is unfortunately very difficult to analyze exactly when this difficulty occurs.

These methods have now been employed by a number of works [37, 36], which have found that these improve solvability of the resulting HDG discretization. One distinct difficulty with applying these methods, however, is that they scale like $\mathcal{O}(h^{-1})$, thus super-penalizing by the definition in Section 4.3. Optimal convergence of the fluxes is thus not possible, and furthermore, the condition number of the resulting matrices increases.

Instead of defining the jump between the left and right states, the jump for the HDG diffusion stabilization is between the state and the trace, $(\mathbf{u}_h^+ - \hat{\mathbf{u}})$. The interior-penalty method can be used with

$$\vec{\boldsymbol{\delta}}^G = \eta \frac{n_i \mathbf{K}_{ij} n_j}{h^+} (\mathbf{u}_h^+ - \hat{\mathbf{u}}) \vec{n}, \tag{4.43}$$

and the BR2 lifting is redefined as $\vec{\boldsymbol{\delta}}^G = \eta \vec{\boldsymbol{\delta}}_f$, where $\boldsymbol{\delta}_f$ solves the system on one side of the face,

$$\int_{\Omega_e^+} \boldsymbol{\tau}_i \boldsymbol{\delta}_{f,i} \, d\Omega = \int_F \boldsymbol{\tau}_i \mathbf{K}_{ij}(\hat{\mathbf{u}})(\mathbf{u}_h^+ - \hat{\mathbf{u}}) n_j \, ds, \quad \forall \vec{\boldsymbol{\tau}} \in \vec{\boldsymbol{\mathcal{P}}}^{p_e}(\Omega_e). \tag{4.44}$$

Empirical evidence suggests that when constant stabilization leads to difficulty converging, it occurs on coarse meshes, or on meshes with highly anisotropic elements. In these cases a practitioner needs to do one of the following:

1. Make the viscous length-scale ℓ_{visc} very small in order to increase the total amount of stabilization; this can work, but often requires multiple attempts with different ℓ_{visc};

2. Rescale the resulting one-sided BR2 to

$$\vec{\boldsymbol{\delta}}^G = \eta \frac{|\Omega_e|}{\min_{F \in \partial\Omega_e} |F|} \boldsymbol{\delta}_f :$$

this can be more successful than (1) above, but still requires tuning;

3. Super-penalize the flux using the $\vec{\boldsymbol{\delta}}^G$ from BR2 as-is; this nearly always works, at the expense of optimal gradient convergence and increased stiffness;

4. Re-mesh to better resolve features; this is the best decision, if possible.

The issue of convergence with diffusive flux stabilization is related to the convective stabilization. Using the changes to the convection formulation discussed below in Section 4.4.2 greatly improves the performance of the constant stabilization methods, allowing solutions to be obtained for many cases that were otherwise unreachable.

4.3.2 Primal Formulation

Although one of the most attractive properties of the HDG method is the mixed formulation in the local solver while retaining a small overall global system, the addition of the gradient variables adds computational cost to the local solves. If accuracy of the gradients is unnecessary or unattainable, or if computational cost is extremely important, it is possible to remove these in the local solver, thereby obtaining a hybridized version of the DG method introduced in this work. These are called *primal* methods, by contrast to dual methods like LDG-H, since they remove the dual variable $\vec{\mathbf{q}}_h$. Although a class of primal hybridized methods was introduced in [24] (IP-H methods) for a model elliptic equation, they have not been widely studied, especially for systems of equations.

These methods use the DG local solver, shown in (2.33), with properly defined HDG one-sided interface fluxes $\hat{\mathbf{F}}$ and $\hat{\mathbf{G}}$ and leave the traces unknown in interior faces $\boldsymbol{\lambda}_h$. The resulting system of discrete residuals is

$$\begin{aligned}\mathbf{R}_{en}^{U} = &\int_{\Omega_e} \phi_n \partial_t \mathbf{u}_h \, d\Omega - \int_{\Omega_e} \partial_i \phi_n \mathbf{H}_i \left(\mathbf{u}_h, \vec{\mathbf{q}}_h\right) \, d\Omega + \int_{\partial\Omega_e} \phi_n \hat{\mathbf{H}}(\mathbf{u}_h, \vec{\mathbf{q}}_h, \hat{\mathbf{u}}, \vec{n}) \, ds \\ &- \int_{\partial\Omega_e} \partial_i \phi_n \mathbf{K}_{ij} \left(\hat{\mathbf{u}}\right) \left(\mathbf{u}_h - \hat{\mathbf{u}}\right) n_j \, ds + \int_{\Omega_e} \phi_n \mathbf{s} \, d\Omega \qquad (4.45) \\ \mathbf{R}_{fn}^{\Lambda} = &\int_f \mu_f \left(\hat{\mathbf{H}} \left(\mathbf{u}_h^+, \vec{\mathbf{q}}_h^+, \boldsymbol{\lambda}_h, \vec{n}^+\right) + \hat{\mathbf{H}} \left(\mathbf{u}_h^-, \vec{\mathbf{q}}_h^-, \boldsymbol{\lambda}_h, \vec{n}^-\right) \right) \, ds\end{aligned}$$

Therefore, the primal HDG equations cannot simply be obtained by hybridizing the flux formulation local solver (4.17) after removing the $\vec{\mathbf{v}}$ equations and setting $\vec{\mathbf{q}}_h = \nabla \mathbf{u}_h$. Just as in the scalar Poisson example (2.33), where eliminating the set of equations defining the $\vec{v}$ by substituting $\vec{v} = \nabla w$ yielded the additional adjoint consistency term, this term naturally arises by eliminating the equation. It is well known that this term is required for a symmetric, coercive bilinear form for DG, but

this is less clear for the hybridized bilinear form of HDG methods.

It can, however, be shown more easily that the additional term is required for adjoint consistency. Consider the primal HDG discretization of the Poisson equation $-\Delta u = f$ for some $f \in L^2(\Omega)$, in $\Omega \subset \mathbb{R}^d$, and $u = g$ on $\partial\Omega$ on a mesh T_h, as

$$\begin{aligned}\mathcal{R}_h(u,\lambda;w,\mu) = (\nabla u, \nabla w)_{T_h} - (f,w)_{T_h} \\ - \langle \nabla u \cdot \vec{n} + \tau(u-\lambda), w-\mu\rangle_{\partial T_h \setminus \partial\Omega} - \langle \nabla u \cdot \vec{n} + \tau(u-g), w\rangle_{\partial\Omega} \\ - \langle u-\lambda, \nabla w \cdot \vec{n}\rangle_{\partial T_h \setminus \partial\Omega} - \langle u-g, \nabla w \cdot \vec{n}\rangle_{\partial\Omega} = 0.\end{aligned} \tag{4.46}$$

Note that this includes the adjoint consistency term. The output functional $J(u, \nabla u)$ defined as

$$J(u,\nabla u) = \int_\Omega j_\Omega u d\Omega + \int_{\partial\Omega} j_\Gamma \nabla u \cdot \vec{n}\, ds$$

is compatible and the primal HDG discretization is adjoint consistent.

The output functional is compatible with the Poisson equation, since it integrates the flux along a Dirichlet boundary (*c.f.* [38] section 4). Since the primal and the adjoint satisfy the weak form $u, \psi \in H_0^1(\Omega)$, it must be shown that these satisfy the discrete adjoint equation

$$\mathcal{R}_h'[u](\delta u, 0; \psi, \psi|_{E_h^I}) + \mathcal{R}_h'[\lambda](0, \delta\lambda; \psi, \psi|_{E_h^I}) = J'[u](\delta u)$$

for all suitable variations: $\delta u \in H_0^1(\Omega)$ and variations on interior faces of T_h, $\delta\lambda \in H^1(E_h^I)$ such that $\delta\lambda = 0$ on $\partial\Omega$. This expression uses the observation that the adjoint associated with the μ test functions is simply the exact adjoint evaluated on the interior faces, as shown by Schütz and May [34].

The discrete adjoint equation terms corresponding to $\delta\lambda$ are

$$\langle \delta\lambda, [\![\nabla\psi]\!]\rangle_{E_h^I} - \left\langle \tau^+\delta\lambda, \psi^+ - \psi|_{E_h^I}\right\rangle_{E_h^I} - \left\langle \tau^-\delta\lambda, \psi^- - \psi|_{E_h^I}\right\rangle_{E_h^I} = 0,$$

and the terms involving δu are

$$(\nabla\delta u, \nabla\psi)_{T_h} - \langle \delta u, \nabla\psi\cdot\vec{n}\rangle_{\partial T_h} - \langle \nabla\delta u\cdot\vec{n}, \psi\rangle_{\partial\Omega} - (\delta u, j_\Omega)_\Omega - \langle \nabla\delta u\cdot\vec{n}, j_\Gamma\rangle_{\partial\Omega} = 0. \tag{4.47}$$

The exact adjoint satisfies the equation $-\Delta\psi = j_\Omega$ in Ω and $\psi = j_\Gamma$ on $\partial\Omega$ [38], thus for a function $z \in H_0^1(\Omega)$ also solves

$$-(\nabla z, \nabla\psi)_{T_h} + \langle z, \nabla\psi\cdot\vec{n}\rangle_{\partial T_h} = (z, j_\Omega)_\Omega.$$

Subtracting this from (4.47) yields

$$\langle \nabla \delta u \cdot \vec{n}, \psi - j_\Gamma \rangle_{\partial\Omega} = 0$$

which enforces the correct adjoint boundary condition.

Eliminating the dual variables and adding the additional term to the local solver overall increases the computational efficiency of the method. To what extent this affects the discretization requires deeper study, and even then it is difficult to understand exactly how different the two methods will be, as implementation aspects such as compute caches and memory layout will affect the performance. Mixed HDG has to locally invert a large system compared to primal HDG, three times the size for two-dimensional discretizations, but the primal version requires an additional term be computed and linearized. While the precise effect of computing the additional term is nearly impossible to assess analytically, it is possible to measure precisely how many fewer matrix-matrix multiplications are involved by considering Algorithms 1 and 2, which describe efficient processes for inverting and multiplying by the local solver. Table 4.5 summarizes the additional floating point operations necessary with the mixed form of HDG. Indeed, even without going through the exercise to tabulate

Step	**# Multiplications**	**# Operations**
Pre-processing	$3d$	$3dN_p^2(2N_p - 1)$
$\mathbf{A}_e^{-1}\mathbf{R}_e$	$3d$	$3dN_p(2N_p - 1)$
$\mathbf{A}_e^{-1}\mathbf{B}_e$	$3dn_f$	$3dn_f N_p^2(2N_p - 1)$
Total	$3d(2 + n_f)$	$3dN_p(2N_p - 1)(N_p + n_f N_p + 1)$

Table 4.5: Additional operations required when assembling the linear system for the HDG mixed form relative to the primal form, assuming a d-dimensional element with n_f faces and N_p basis functions.

these, it is clear that this takes $\mathcal{O}(N_p^3)$ operations, since this is the number for a single matrix multiplication. Using standard Gaussian quadrature rules, integrating over a d-dimensional object with $2p + 1$ order requires $\mathcal{O}(p^d)$ points and operations. The polynomial spaces that are used to approximate the u_h and components $\mathbf{q}_{h,i}$ require $\mathcal{O}(p^d)$ basis functions, so the additional matrix multiplication takes $\mathcal{O}(p^{3d})$ operations. By contrast, the additional face-based adjoint consistency term in primal HDG requires $\mathcal{O}(p^{d-1})$ operations, so for high orders primal HDG is guaranteed to exceed the mixed formulation in operation counts.

The above comparison ignores multiplicative constants and does not account for implementation details. To make this more concrete, consider actual run times for

	Pec = 10^{-2}					Pec = 10^{2}				
	1	2	3	4	5	1	2	3	4	5
pre-solve	0.94	0.69	0.63	0.54	0.53	1.00	0.81	0.64	0.52	0.50
solve	1.43	1.15	1.18	1.13	1.03	1.22	1.13	1.28	1.09	1.27
post-solve	0.87	0.85	0.72	0.66	0.57	0.92	1.00	0.83	0.69	0.66

(a) $t_{\text{primal,BR2}}/t_{\text{mixed,const.}}$

	Pec = 10^{-2}					Pec = 10^{2}				
	1	2	3	4	5	1	2	3	4	5
pre-solve	0.82	0.73	0.64	0.49	0.53	0.80	0.77	0.60	0.53	0.50
solve	0.92	1.03	0.92	0.94	1.08	0.71	0.97	0.99	0.89	1.12
post-solve	0.89	0.84	0.81	0.65	0.62	0.83	0.90	0.79	0.66	0.67

(b) $t_{\text{primal,BR2}}/t_{\text{mixed,BR2}}$

Table 4.6: Ratio of run times for a two-dimensional scalar linear convection-diffusion equation with orders $p = 1 - 5$. Entries less than one indicate faster run time for primal HDG with BR2.

the problem shown in Figure 4.5 with different orders. Table 4.6 compares the ratio of the actual run-times of mixed HDG with constant and BR2 viscous stabilization to primal HDG for both low and high Peclet numbers, controlled by adjusting the diffusivity b. The conclusion is clear: for increasingly high order the pre- and post-solve run-time is lower by about 50% at $p = 5$. The linear solve time compared to mixed HDG with BR2 is unaffected, but is longer compared to the mixed form with constant stabilization due to the higher constant on the condition number with BR2 (4.42). The effect of the higher constant on the condition number is shown in Figure 4.6 where the primal version HDG-P-BR2 requires more GMRES iterations to converge to the solution of the linear system.

4.4 Post-Processing

An attractive aspect of mixed HDG is the ability to post-process the solution element-by-element to obtain a faster converging solution compared to primal DG. The main idea is that, since the finite element is already conservative, by obtaining optimal convergence of the dual variable the accuracy in that quantity can be transferred to the solution itself. There have been a number of post-processing projections proposed in the literature, but they all operate on this principle. This section outlines the past formulations, which were successful for diffusion-dominated systems, and introduces a new convective formulation for mixed HDG that finally allows post-processing to be extended to the convection-dominated regime.

4.4.1 Prior Formulation

The original formulation for post-processing was created for methods that approximate the dual in a div-conforming space, *e.g.* those based on the RT and BDM spaces [39]. In fact approximations to such spaces have long been used with DG methods to improve the accuracy of velocity fields [40].

The $\vec{H}(\text{div}, \Omega)$ space is defined as the space of vector L^2 functions that additionally have square-integrable divergence:

$$\vec{H}(\text{div}, \Omega) := \left\{ \vec{v} \in [L^2(\Omega)]^d \, : \, \nabla \cdot \vec{v} \in L^2(\Omega) \right\}. \tag{4.48}$$

The RT and BDM spaces are both approximations to this space that use slightly different definitions. The RT space is defined as

$$\begin{aligned} RT^p(\Omega_e) &:= [\mathcal{P}^p(\Omega_e)]^d + \vec{x}\mathcal{P}^p(\Omega_e) \\ RT^p(\Omega) &:= \{\vec{v} \, : \, \vec{v} \in RT^p(\Omega_e) \; \forall \Omega_e \in T_h \; s.t. \; [\![\vec{v}]\!] = 0 \text{ on } f \; \forall f \in \partial\Omega_e\}. \end{aligned}$$

where $\vec{x}$ is the d-dimensional coordinate vector, such that globally the space exists of those functions that have continuous normal component across every interior face of the mesh.

This concept is remarkably similar to the idea of conservation of the flux in finite element methods, and can even be used to enforce the divergence-free condition in incompressible flows. Similarly, since the conservativity condition in HDG enforces that the interface flux $\hat{H}$ is continuous across the face, if the normal component of $\vec{H} \cdot \vec{n} = \hat{H}$ on all faces, it would be globally div-conforming, and

$$\vec{H} \in [\mathcal{P}^p(\Omega_e)]^d,$$

not quite $RT^p(\Omega)$. This flux would be instead in $BDM^p(\Omega)$:

$$\begin{aligned} BDM^p(\Omega_e) &:= [\mathcal{P}^p(\Omega_e)]^d \\ BDM^p(\Omega) &:= \{\vec{v} \, : \, \vec{v} \in BDM^p(\Omega_e) \; \forall \Omega_e \in T_h \; s.t. \; [\![\vec{v}]\!] = 0 \text{ on } f \; \forall f \in \partial\Omega_e\}. \end{aligned}$$

From this definition it is clear that if $\vec{H} \in BDM^p(\Omega)$, then $\nabla \cdot \vec{H} \in [\mathcal{P}^{p-1}(\Omega)]^d$ as expected.

If the flux could instead be placed in $RT^p(\Omega)$, then its divergence $\nabla \cdot \vec{H} \in \mathcal{P}^p(\Omega)$ making it possible to obtain a more accurate solution from the local solver. Cockburn *et. al.* [39] showed that the hybridized RT and BDM admit a similar form, and

therefore solutions from the RT method could be obtained from those of the BDM method by post-processing.

Therefore, the first step of the existing post-processing methods is to project the flux $\vec{H}$ from $BDM^p(\Omega_e)$ to $RT^p(\Omega_e)$. The Raviart-Thomas space has $(p+1)(p+3)$ DOF for a $2d$ simplex, so the projection seeks $\vec{H}^* \in RT^p(\Omega_e)$ such that

$$\begin{aligned}\left(\vec{H}^* - \vec{H}, \vec{v}\right)_{\Omega_e} &= 0, \quad \forall \vec{v} \in [\mathcal{P}^{p-1}(\Omega_e)]^d \\ \left\langle \left(\vec{H}^* - \vec{H}\right) \cdot \vec{n}, \mu \right\rangle_f &= 0, \quad \forall \mu \in \mathcal{P}^p(f) \;\; \forall f \in \partial\Omega_e.\end{aligned}$$

These are $(p+1)(p+3)$ independent equations for a $2d$ simplex, so the projection is unique. Next the method matches the flux computed with $(\vec{q}_h^*, u_h^*) \in [\mathcal{P}^{p+1}(\Omega_e)]^d \times \mathcal{P}^{p+1}(\Omega_e)$ on the element to $\vec{H}^*$ in the Raviart-Thomas space, solving

$$\begin{gathered}\nabla \cdot (H(\vec{q}_h^*, u_h^*)) = \nabla \cdot \vec{H}^* \quad \text{in } \Omega_e, \qquad \vec{H}(\vec{q}_h^*, u_h^*) \cdot \vec{n} = \vec{H}^* \cdot \vec{n} \quad \text{on } \partial\Omega_e, \\ \int_{\Omega_e} (u_h^* - u_h)\, d\Omega = 0\end{gathered} \tag{4.49}$$

Computationally, this solves, *e.g.* for the scalar advection diffusion equation [27]

$$(\vec{q}_h^*, \vec{v})_{\Omega_e} + (u_h^*, \nabla \cdot \vec{v})_{\Omega_e} - \langle \hat{u}^*, \vec{v} \cdot \vec{n} \rangle_{\partial\Omega_e} = 0 \qquad \forall \vec{v} \in [\mathcal{P}^{p+1}(\Omega_e)]^d \tag{4.50}$$

$$-\left(\vec{a}u_h^* - b\vec{q}_h^*, \nabla w\right)_{\Omega_e} + \left\langle \widehat{\vec{a}u_h^*} - \widehat{b\vec{q}_h^*}, w \right\rangle_{\partial\Omega_e} = \left(\nabla \cdot \vec{H}^*, w\right)_{\Omega_e} \quad \forall w \in \mathcal{P}^{p+1}(\Omega_e) \tag{4.51}$$

$$\left\langle \widehat{\vec{a}u_h^*} - \widehat{b\vec{q}_h^*}, \mu \right\rangle_F = \left\langle \vec{H}^* \cdot \vec{n}, \mu \right\rangle_F \quad \forall F \in \partial\Omega_e \qquad \forall \mu \in \mathcal{P}^{p+1}(F) \tag{4.52}$$

$$(u_h^* - u_h, 1)_{\Omega_e} = 0. \tag{4.53}$$

The resulting post-processed solution u_h^* converges at a rate one order higher than the original HDG solution u_h.

Another variant of post-processing uses the observation that $\hat{G}$ is single-valued when diffusive stabilization $\tau^{G,+} = \tau^{G,-}$, and applies the RT projection only to the diffusive flux, finding a $\vec{G}^* \in H(\mathrm{div}, \Omega)$. To compute u_h^* the procedure solves the local Neumann problem [41]

$$\begin{aligned}\left(b\nabla u_h^* - \vec{G}^*, \nabla w\right)_{\Omega_e} &= 0, \quad \forall w \in \mathcal{P}^{p+1}(\Omega_e) \\ (u_h^* - u_h, 1)_{\Omega_e} &= 0.\end{aligned}$$

The statement about conserving the average is correct because the original method is conservative.

Both of the methods described above rely on an optimally converging non-zero diffusive flux, thus making it difficult, or in certain cases impossible, for general systems of equations such as Navier-Stokes for to obtain higher rates for each variable with this method. If the Peclet number is too high, the diffusive flux or gradients fail to optimally converge. An improved formulation for post-processing first relies on obtaining optimal convergence of the dual variable for convection-dominated regime, which is discussed below.

4.4.2 Convective Flux

HDG methods commonly use the Roe-type formulation for the one-sided convective flux, using the stabilization in (4.21). This is a natural extension of the centered method for linear scalar convection proposed in [27]. The problem with this formulation, which has already been alluded to, is that the resulting trace is not properly upwinded in the limit of vanishing viscosity. This introduces a new aspect of designing one-sided fluxes for HDG methods intended for convection-dominated problems: both the flux should be properly upwind-biased and the resulting trace must be set to the upwind state. An alternative formulation of the flux presented below will better approximate the linearized version of the Riemann problem, recovering optimal convergence of the gradients.

It is not immediately obvious that improperly upwinding the trace results in non-optimal convergence of the gradients, but this can be verified numerically. Table 4.7 shows the error convergence of the linear scalar convection-diffusion manufactured solution in Section 4.1.2. Computing with the upwind flux stabilization used in that section, the gradients optimally converge at $p+1$ in the L^2-error norm. Using the Roe-type stabilization for scalar linear convection

$$\tau^F = |\vec{a} \cdot \vec{n}|,$$

the optimal convergence is lost. If the viscosity b were lowered further, the rate would lower toward p using this method, while the upwind stabilization maintains an optimal $p+1$ convergence rate. The central question, therefore, becomes how to emulate this upwind stabilization for cases where the wave-speed is a function of the state and in the presence of multiple waves.

In order to come up with a method to properly upwind the flux and trace, first consider how this is done in DG, for the Riemann problem depicted in Fig. 4.8. A Riemann problem occurs at a discontinuity in the solution; precisely what happens

		Centered stabilization				Upwind stabilization			
		error	rate	error	rate	error	rate	error	rate
p	N_e	$\|\vec{q}_h - \nabla u\|_{L^2(\Omega)}$		$\|\hat{u} - u\|_{L^2(\partial T_h)}$		$\|\vec{q}_h - \nabla u\|_{L^2(\Omega)}$		$\|\hat{u} - u\|_{L^2(\partial T_h)}$	
0	64	0.209e1		5.656e-2		9.558e-1		3.408e-2	
	256	0.196e1	0.10	1.811e-2	1.64	7.913e-1	0.27	1.130e-2	1.59
	1024	0.114e1	0.77	4.622e-3	1.97	5.714e-1	0.47	3.301e-3	1.78
1	64	1.724e-1		1.348e-3		6.555e-2		1.378e-3	
	256	6.497e-2	1.41	1.385e-4	3.28	9.052e-3	2.86	1.557e-4	3.15
	1024	2.187e-2	1.57	1.422e-5	3.28	2.048e-3	2.14	1.862e-5	3.06
2	64	5.094e-3		4.118e-5		9.379e-4		3.411e-5	
	256	1.042e-3	2.29	2.100e-6	4.29	1.188e-4	2.98	2.185e-6	3.96
	1024	1.771e-4	2.56	1.093e-7	4.26	1.554e-5	2.93	1.382e-7	3.98
3	64	1.986e-4		8.023e-7		2.778e-5		9.746e-7	
	256	1.627e-5	3.61	2.079e-8	5.27	1.592e-6	4.13	2.917e-8	5.06
	1024	1.217e-6	3.74	6.014e-10	5.11	9.847e-8	4.02	8.828e-10	5.05

Table 4.7: Error convergence of the gradient and trace resulting from both the centered and upwind HDG convective flux variants for the manufactured solution with $\vec{a} = (\cos(\frac{\pi}{6}), \sin(\frac{\pi}{6}))$, $b = 10^{-2}$.

at interior faces in DG. The central idea of Godunov's scheme is to use the flux based on the state at the interface $\hat{\mathbf{u}}$ from which the waves emanate, a small time later. This in general requires the solution of a nonlinear system of equations, but Roe's method [11] bases the wave speeds off an interface state obtained by linearizing the equations. After rotating into the frame of reference of the interface, the linearization replaces the flux Jacobian $\mathbf{A} = d\vec{\mathbf{F}}(\mathbf{u})/d\mathbf{u} \cdot \vec{n}$ by a constant matrix $\tilde{\mathbf{A}}(\mathbf{u}^+, \mathbf{u}^-)$, satisfying hyperbolicity, consistency, and conservation, resulting in an approximate system. Since hyperbolicity of the flux Jacobian is enforced, it has a set of real eigenvalues $\{\tilde{\lambda}_i\}$, and linearly independent right and left eigenvectors $\{\tilde{\mathbf{r}}_i\}$ and $\{\tilde{\mathbf{l}}_i\}$, respectively. Assuming the eigenvalues are ordered from low to high according to Figure 4.8, the linearized interface state is

$$\hat{\mathbf{u}} = \mathbf{u}_h^+ - \sum_{i=1}^{I} \tilde{\mathbf{r}}_i \tilde{\mathbf{l}}_i \left(\mathbf{u}_h^+ - \mathbf{u}_h^-\right) = \mathbf{u}_h^+ + \sum_{i=I+1}^{s} \tilde{\mathbf{r}}_i \tilde{\mathbf{l}}_i \left(\mathbf{u}_h^+ - \mathbf{u}_h^-\right).$$

Adding these equations together yields a useful expression that shows the deviation from the average

$$\hat{\mathbf{u}} = \{\mathbf{u}_h\} + \frac{1}{2}\tilde{\mathbf{R}} \operatorname{sign}(\tilde{\mathbf{\Lambda}}) \tilde{\mathbf{L}} \left(\mathbf{u}_h^+ - \mathbf{u}_h^-\right) \tag{4.54}$$

where $\tilde{\mathbf{L}}$ and $\tilde{\mathbf{R}}$ are matrices of the left and right eigenvectors, and $\tilde{\mathbf{\Lambda}}$ is the diagonal

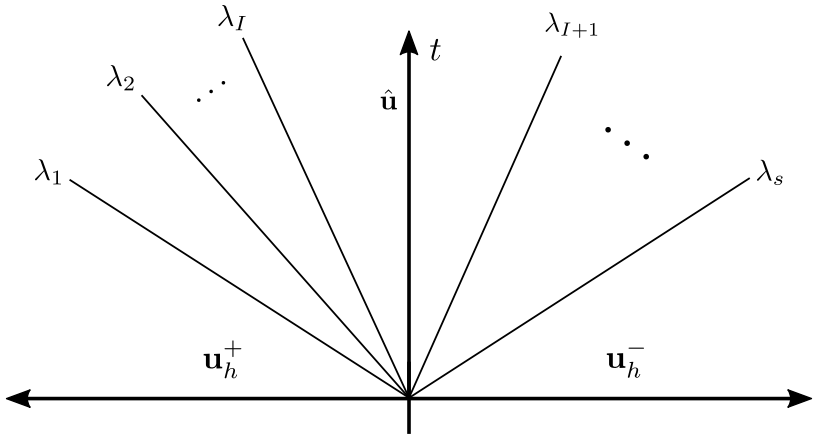


Figure 4.8: The Riemann problem resulting from a general system of equations. When both states are known the correct interface state $\hat{\mathbf{u}}$, and correct flux can be found.

matrix of eigenvalues. Following a similar procedure [12]

$$\hat{\mathbf{F}} = \frac{1}{2}\left(\mathbf{F}(\mathbf{u}_h^+) + \mathbf{F}(\mathbf{u}_h^-)\right)\cdot\vec{n} + \frac{1}{2}\tilde{\mathbf{R}}|\tilde{\mathbf{\Lambda}}|\tilde{\mathbf{L}}\left(\mathbf{u}_h^+ - \mathbf{u}_h^-\right). \tag{4.55}$$

The one-sided flux in HDG only has access to $\mathbf{u}_h^+$ and a trace state $\boldsymbol{\lambda}_h$, so the proper upwind state $\hat{\mathbf{u}}$ and flux $\hat{\mathbf{F}}$ cannot be directly computed as above as the wave structure for the system cannot in general be determined from that data. To remedy this, two alternative formulations are considered which address the proper upwinding.

The first formulation computes the linearized flux inexactly by analogy to the first-order expansion

$$\hat{\mathbf{F}}^+ = \vec{\mathbf{F}}(\mathbf{u}_h^+)\cdot\vec{n}^+ + \frac{\partial\vec{\mathbf{F}}(\mathbf{u}_h^+)}{\partial\mathbf{u}_h^+}\cdot\vec{n}^+\left(\mathbf{u}_h^+ - \boldsymbol{\lambda}_h\right). \tag{4.56}$$

The conservativity condition then, by equating the flux (4.56) from both sides, enforces that

$$\boldsymbol{\lambda}_h = \left(\mathbf{A}^+ + \mathbf{A}^-\right)^{-1}\left(\Delta\vec{\mathbf{F}}\cdot\vec{n}^+ + \mathbf{A}^+\mathbf{u}_h^+ + \mathbf{A}^-\mathbf{u}_h^-\right), \tag{4.57}$$

where $\Delta\vec{\mathbf{F}} = \vec{\mathbf{F}}(\mathbf{u}_h^+) - \vec{\mathbf{F}}(\mathbf{u}_h^-)$, and $\mathbf{A}^\pm$ are the flux Jacobians based on each of the states $\mathbf{u}_h^\pm$. The trace is no longer the simple state average as desired, but is not clearly upwinded. If $\mathbf{A}^+ = \mathbf{A}^-$, then the above formula clearly upwinds the solution; this, however, usually occurs when the states are equal. This also relies on a small jump in the state $|\mathbf{u}_h - \boldsymbol{\lambda}_h|$. If this is too large, the purely one-sided linearization fails to represent the interface flux accurately and the system becomes unstable.

The second formulation takes as ansatz the form of the Roe flux from DG, replac-

ing $\mathbf{u}_h^-$ with $\boldsymbol{\lambda}_h$

$$\hat{\mathbf{F}}^+ = \frac{1}{2}\left(\vec{\mathbf{F}}(\mathbf{u}_h^+) + \vec{\mathbf{F}}(\boldsymbol{\lambda}_h)\right)\cdot\vec{n}^+ + \frac{1}{2}\mathbf{R}|\boldsymbol{\Lambda}|\mathbf{L}\left(\mathbf{u}_h^+ - \boldsymbol{\lambda}_h\right) \tag{4.58}$$

where the flux Jacobian is based on the $\boldsymbol{\lambda}_h$ state. The conservativity condition then, by equating the flux from both sides, enforces that

$$\boldsymbol{\lambda}_h = \{\mathbf{u}_h\} + \frac{1}{2}|\mathbf{A}|^{-1}\Delta\vec{\mathbf{F}}\cdot\vec{n} = \{\mathbf{u}_h\} + \frac{1}{2}\mathbf{R}|\boldsymbol{\Lambda}|^{-1}\mathbf{L}\,\Delta\vec{\mathbf{F}}\cdot\vec{n}. \tag{4.59}$$

By using series expansions it can be shown that

$$\Delta\vec{\mathbf{F}}\cdot\vec{n}^+ = \frac{1}{2}\left(\frac{\partial\vec{\mathbf{F}}}{\partial\mathbf{u}_h^+} + \frac{\partial\vec{\mathbf{F}}}{\partial\mathbf{u}_h^-}\right)\cdot\vec{n}^+\,\Delta\mathbf{u}_h = \frac{\partial\vec{\mathbf{F}}}{\partial\boldsymbol{\lambda}_h}\cdot\vec{n}\,\Delta\mathbf{u}_h + \mathcal{O}(\Delta\mathbf{u}_h^2).$$

Inserting this expression into (4.59) yields the correct linearized upwind state in the form (4.54) to second order.

The flux (4.58) is therefore the version that should be implemented to both correctly upwind the flux and the resulting trace, even in the case of a system of equations.

4.4.3 Current Formulation

The previous methods for post-processing relied on a nonzero diffusive flux and tailor-made Raviart-Thomas spaces for every element, thus these methods do not extend to general systems with curved geometry necessary for application to problems of engineering interest. The RT spaces for curved elements require more DOF, so the state-to-flux mapping defined by the projection, which was unique for linear elements, is no longer one-to-one. The formulation for the convective flux in Section 4.4.2 improves the convergence of the gradient, so assuming the general formulation following Section 4.2 is solved, post-processing of the solution is now possible using only the $\vec{\mathbf{q}}_h$, without the use of $H(\mathrm{div},\Omega)$ approximations. The post-processing seeks for each element Ω_e the unique $\mathbf{u}_h^* \in \boldsymbol{\mathcal{P}}^{p+1}(\Omega_e)$ solving

$$\begin{aligned}(\nabla\mathbf{u}_h^* - \vec{\mathbf{q}}_h, \nabla\mathbf{w})_{\Omega_e} &= \mathbf{0}, \quad \forall\mathbf{w}\in\boldsymbol{\mathcal{P}}^{p+1}(\Omega_e)\\ (\mathbf{u}_h^* - \mathbf{u}_h, \mathbf{1})_{\Omega_e} &= \mathbf{0}.\end{aligned} \tag{4.60}$$

This is a Neumann problem conserving the average, since the original method was conservative. The following result summarizes the accuracy of the method.

Theorem 4.2. *Assume the HDG discretization is applied as stated in Section 4.2, so*

$$\|\vec{\mathbf{q}}_h - \nabla \mathbf{u}\|_{L^2(\Omega_e)} \leq C_1 h^{p+1},$$

implying that $\|\mathbf{u}_h - \mathbf{u}\|_{L^2(\Omega_e)} \leq C_2 h^{p+1}$ *by the error estimates* (4.41)*. The post-processed solution* $\mathbf{u}_h^* \in \mathcal{P}^{p+1}(\Omega_e)$ *determined by* (4.60) *has the property*

$$\|\mathbf{u}_h^* - \mathbf{u}\|_{L^2(\Omega_e)} \leq C_3 h^{p+2}.$$

Note that the constants absorb the regularity of $\mathbf{u}$*.*

Proof. The Poincaré inequality states that for $f \in W^{1,p}(\Omega_e)$,

$$\left\|f - \bar{f}\right\|_{L^p(\Omega_e)} \leq Ch \|\nabla f\|^2_{L^p(\Omega_e)}$$

where C is a constant depending on Ω_e and the integrability p, h is the maximum diameter of Ω_e, and the notation $\bar{f}$ indicates the volume average over Ω_e. This result requires that the domain Ω_e is convex, but this is already required for the finite element method itself.

By the triangle inequality,

$$\|\mathbf{u}_h^* - \mathbf{u}\|_{L^2(\Omega_e)} \leq \left\|\mathbf{u}_h^* - \mathbf{u} - \overline{(\mathbf{u}_h^* - \mathbf{u})}\right\|_{L^2(\Omega_e)} + \left\|\overline{(\mathbf{u}_h^* - \mathbf{u})}\right\|_{L^2(\Omega_e)}$$

The second of these terms is automatically zero by the conservation property of the projection (4.60). Applying Poincaré with $p = 2$ and $f = \mathbf{u}_h^* - \mathbf{u}$ bounds the first term as

$$\|\mathbf{u}_h^* - \mathbf{u}\|_{L^2\Omega_e} \leq Ch \|\nabla \mathbf{u}_h^* - \nabla \mathbf{u}\|_{L^2(\Omega_e)}.$$

The first set of equations in the projection enforces that $\nabla \mathbf{u}_h^* = \vec{\mathbf{q}}_h$, because both the trial and test functions are p order, so

$$\|\mathbf{u}_h^* - \mathbf{u}\|_{L^2\Omega_e} \leq C_3 h^{p+2},$$

finishing the result. □

When solving a Neumann problem instead of using the HDG local solver itself for obtaining a solution, occasionally the resulting solution $\mathbf{u}_h^*$ can become non-physical on large elements. For instance, Navier-Stokes requires positive density and pressure to evaluate wave speeds. This constraint may be violated for u_h^*, but these situations

can be simply handled when post-processing by setting $\mathbf{u}_h^* = \mathbf{u}_h$ when the post-processed solution becomes non-physical anywhere on the element, without affecting the resulting global convergence rate. In practice, the physicality is checked at the quadrature points of the numerical method.

Consider application to a compressible Euler manufactured solution on $[0,1]^2$ with sinusoidally varying density, velocity, and pressure fields given by [42]

$$\begin{aligned} \rho^{\text{MS}} &= a_\rho + b_\rho \sin\left(c_\rho x + d_\rho y\right) \\ v_0^{\text{MS}} &= a_{v_0} + b_{v_0} \cos\left(c_{v_0} x + d_{v_0} y\right) \\ &\vdots \\ p^{\text{MS}} &= a_p + b_p \sin\left(c_p x + d_p y\right). \end{aligned} \tag{4.61}$$

The resulting Euler system with a source term is both nonlinear and displays non-trivial waves at the element intersections, so it is ideal for testing the post-processing. Parameters used in defining the solution $\mathbf{u}^{\text{MS}}$ are shown in Table 4.8. Expected rates

Quantity	a	b	c	d
ρ^{MS}	1	0.5	4	3
v_0^{MS}	0.1	-0.1	2	4
v_1^{MS}	0.05	0.02	3	6
p^{MS}	1.0	0.04	5	-7

Table 4.8: Parameters used for the computation of the manufactured solution (4.61). Additional fluid parameters: $\gamma = 1.4$ and $R = 1.0$.

of convergence in the continuous L^2 norm for each variable are obtained for each variable, as well as the overall system in Table 4.10. Also listed in the table are the post-processed approximate solution, which converges at a higher rate since the results are obtained with the new convective flux formulation. The full additional order is not recovered, but that is not expected given the nonlinear nature of the equations using the linearized upwinding theory. If instead the original HDG Roe flux formulation from (4.22) were used, convergence rates of the post-processed solution, listed for all solution components combined in Table 4.9 remain at $p+1$.

Similarly, the discrete adjoint can also be post-processed, for instance, to obtain a fine-space adjoint for error estimation. However, due to the form of the adjoint equations, this only works under a small subset of circumstances. The post-processing as described in (4.60) cannot be applied to the adjoint, because the definition of the dual variable $\mathbf{\Psi}^{\vec{v}}$ differs from $\vec{\mathbf{q}}$, as described in appendix A for the linear convection-

	$p=1$		$p=2$		$p=3$		$p=4$	
N_e	error	rate	error	rate	error	rate	error	rate
64	2.425e-3		1.215e-4		8.235e-6		2.543e-7	
256	6.044e-4	2.00	1.382e-5	3.14	5.272e-7	3.97	6.874e-9	5.21
1024	1.506e-4	2.00	1.710e-6	3.01	3.340e-8	3.98	2.038e-10	5.08

Table 4.9: Solution L^2 error norm and associated convergence rates after post-processing the sinusoidal Euler manufactured solution using the original centered Roe-like convective flux stabilization formulation. Table 4.10 shows the convergence with the upwind formulation.

diffusion equation. There it is shown that when the dual variable approximates the gradients, the associated adjoint variable approximates the adjoint diffusive flux, and similarly when the dual variable approximates the diffusive flux, the adjoint variable approximates the negative adjoint gradient. Therefore, the post-processing system for the adjoint, given the dual approximates the gradient, seeks $\boldsymbol{\Psi}^{\mathbf{w},*} \in \boldsymbol{\mathcal{P}}^{p+1}(\Omega_e)$ solving

$$\begin{aligned} \left(\mathbf{K}_{ij}\partial_j \boldsymbol{\Psi}^{\mathbf{w},*} - \boldsymbol{\Psi}_i^{\vec{\mathbf{v}}}, \partial_i \mathbf{w}\right)_{\Omega_e} &= \mathbf{0}, \quad \forall \mathbf{w} \in \boldsymbol{\mathcal{P}}^{p+1}(\Omega_e) \\ \left(\boldsymbol{\Psi}^{\mathbf{w},*} - \boldsymbol{\Psi}^{\mathbf{w}}, \mathbf{1}\right)_{\Omega_e} &= \mathbf{0}. \end{aligned} \tag{4.62}$$

These equations become singular when any equation in the system has zero diffusivity. If instead the formulation of the HDG method used the diffusive flux as the dual variable, it would become possible to post-process the adjoint variable.

p	N_e	$\|\mathbf{u}_h - \mathbf{u}\|_{L^2(\Omega)}$		$\|u_h^{\rho} - \rho\|_{L^2(\Omega)}$		$\|u_h^{\rho v_0} - \rho v_0\|_{L^2(\Omega)}$		$\|u_h^{\rho v_1} - \rho v_1\|_{L^2(\Omega)}$		$\|u_h^{\rho E} - \rho E\|_{L^2(\Omega)}$	
		error	rate	error	rate	error	rate	error	rate	error	rate
1	64	3.137e-3		1.138e-3		1.022e-3		9.667e-4		2.563e-3	
	256	7.794e-4	2.01	2.627e-4	2.12	2.439e-4	2.07	2.313e-4	2.06	6.523e-4	1.97
	1024	1.953e-4	2.00	6.418e-5	2.03	6.304e-5	1.95	5.667e-5	2.03	1.638e-4	1.99
2	64	2.010e-4		6.384e-5		5.384e-5		6.113e-5		1.723e-4	
	256	2.494e-5	3.01	7.656e-6	3.06	5.835e-6	3.21	7.605e-6	3.01	2.172e-5	2.99
	1024	3.117e-6	3.00	9.387e-7	3.03	7.256e-7	3.01	9.503e-7	3.00	2.721e-6	3.00
3	64	1.031e-5		3.313e-6		2.089e-6		3.328e-6		8.939e-6	
	256	6.335e-7	4.02	1.884e-7	4.14	9.735e-8	4.42	1.986e-7	4.07	5.630e-7	3.99
	1024	3.949e-8	4.00	1.147e-8	4.04	5.733e-9	4.09	1.232e-8	4.01	3.526e-8	4.00
4	64	4.419e-7		1.451e-7		6.883e-8		1.396e-7		3.873e-7	
	256	1.361e-8	5.02	3.991e-9	5.18	1.658e-9	5.38	4.332e-9	5.01	1.216e-8	4.99
	1024	4.222e-10	5.01	1.180e-10	5.08	4.307e-11	5.27	1.326e-10	5.03	3.806e-10	5.00

(a) Solution with upwind Roe formulation

p	N_e	$\|\mathbf{u}_h^* - \mathbf{u}\|_{L^2(\Omega)}$		$\|u_h^{*,\rho} - \rho\|_{L^2(\Omega)}$		$\|u_h^{*,\rho v_0} - \rho v_0\|_{L^2(\Omega)}$		$\|u_h^{*,\rho v_1} - \rho v_1\|_{L^2(\Omega)}$		$\|u_h^{*,\rho E} - \rho E\|_{L^2(\Omega)}$	
		error	rate	error	rate	error	rate	error	rate	error	rate
1	64	1.120e-3		7.150e-4		6.414e-4		4.261e-4		3.876e-4	
	256	1.630e-4	2.78	1.010e-4	2.82	9.793e-5	2.71	5.987e-5	2.83	5.644e-5	2.78
	1024	2.216e-5	2.88	1.315e-5	2.94	1.357e-5	2.85	7.909e-6	2.92	8.453e-6	2.74
2	64	5.308e-5		2.595e-5		3.979e-5		1.352e-5		1.943e-5	
	256	3.587e-6	3.89	1.799e-6	3.85	2.650e-6	3.91	7.879e-7	4.10	1.409e-6	3.79
	1024	2.582e-7	3.80	1.352e-7	3.73	1.798e-7	3.88	6.534e-8	3.59	1.087e-7	3.70
3	64	2.898e-6		1.671e-6		1.813e-6		1.258e-6		8.557e-7	
	256	8.931e-8	5.02	5.459e-8	4.94	5.505e-8	5.04	3.152e-8	5.32	3.118e-8	4.78
	1024	3.286e-9	4.76	2.159e-9	4.66	1.951e-9	4.82	9.074e-10	5.12	1.227e-9	4.67
4	64	1.125e-7		8.074e-8		6.142e-8		3.693e-8		3.155e-8	
	256	2.059e-9	5.77	1.260e-9	6.00	1.174e-9	5.71	9.522e-10	5.28	6.071e-10	5.70
	1024	2.646e-11	6.28	1.597e-11	6.30	1.618e-11	6.18	6.134e-12	7.28	1.208e-11	5.65

(b) Post-processed solution

Table 4.10: Solution L^2 error norm and associated convergence rates after post-processing the sinusoidal Euler manufactured solution using the new upwind Roe convective flux formulation. Table 4.9 shows the convergence with the original formulation.

4.5 Numerical Comparisons

This section analyzes a set of test cases that compare DG and the HDG methods developed above for a system of equations on both the Euler and Navier-Stokes equations. Because the objective is to compare the performance of these methods, and this can become obscured if adaptation were to be included, this section focuses on a set of canonical test cases without adaptation.

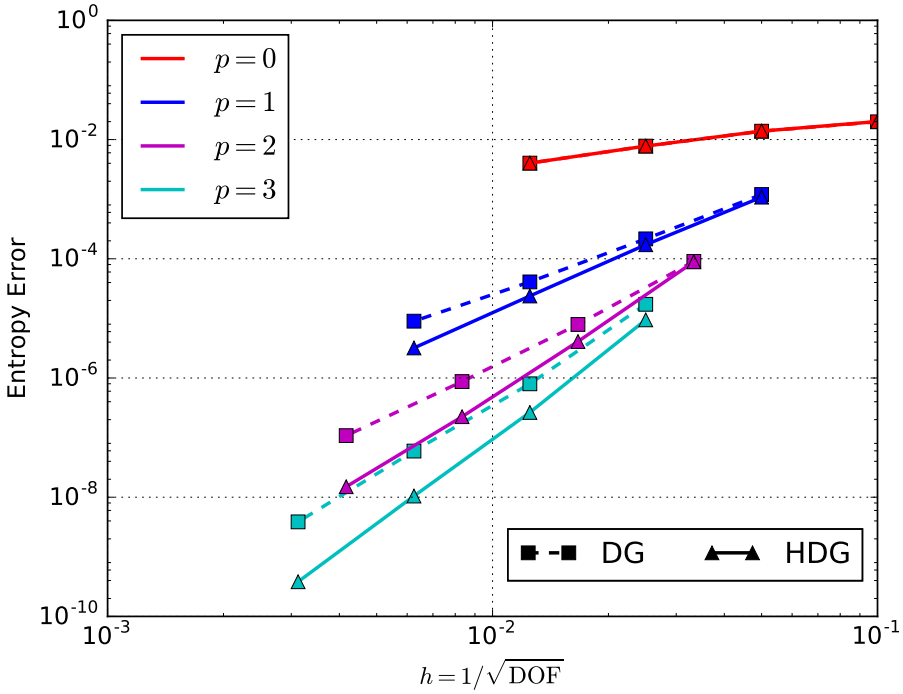
4.5.1 Inviscid Bump

This test case, from the first high-order CFD workshop [3], computes the solution to the Euler equations over a channel with a smooth Gaussian bump geometry, and illustrates that the faster convergence obtained by post-processing HDG is not limited to merely pmanufactured solutions. Since there is no viscosity, a slip condition is imposed at the wall, total temperature and pressure are prescribed ahead of the bump, and the static pressure is imposed behind the bump. Both the entropy in the domain and the drag error calculated on the bottom wall are compared.

The bump is curved, so to obtain smooth high-order solutions the elements themselves need to be curved, usually to at least the order of the solution, called an *isoparametric* mapping. These test cases use quartically curved elements, so the mapping from the element reference space to global coordinates becomes nonlinear. Although the procedure to obtain curved meshes is outside the scope of this comparison, it is a topic of interest in the field. These meshes are created by starting from a linear mesh and solving for a displacement field of the sub-vertices according to linear elasticity [43].

Entropy should be constant for this subsonic flow at $M = 0.5$ with smooth bump, so the L^2 norm of the entropy error indicates the solution accuracy. Figure 4.9 compares the convergence of the entropy error norm for both DG and element-wise post-processed HDG results. The adjoint solution for the entropy output is a singular quantity, so the rate of convergence for the output is limited to $p + 1$. This rate of convergence is obtained by the DG results, and post-processed HDG uncovers nearly an extra order of accuracy for every order. One possible reason that the $p = 3$ post-processed result for HDG may not be converging as well as other orders could be because the elements are only curved to fourth order, and the fifth order expected post-processed rate of convergence exceeds this.

It is also interesting to note that although on the coarsest meshes the entropy error is not reduced much by the post-processing, the resulting flowfield is much



(a) DG (dashed) and post-processed HDG (solid) convergence

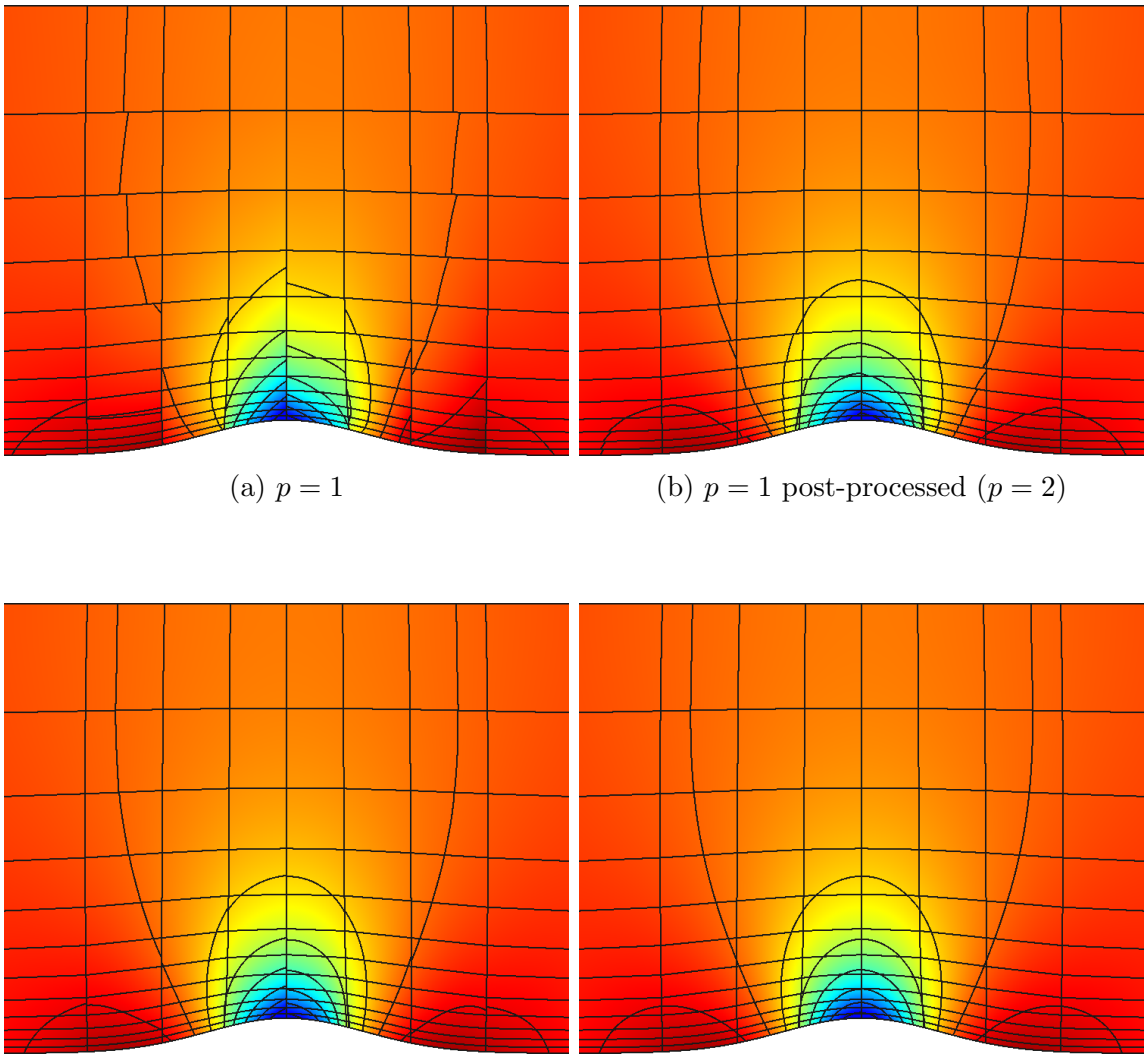
Order	0	1	2	3
DG	0.95	2.19	3.01	3.95
HDG	0.95	2.90	3.90	4.78

(b) Rate of convergence on finest meshes for DG and post-processed HDG

Figure 4.9: Entropy error convergence for the subsonic bump with the Euler equations.

better approximated. Figure 4.10 shows contours of the pressure distribution over the smooth bump. The $p = 1$ post-processed pressure contours look nearly as smooth as the global solution obtained with $p = 2$.

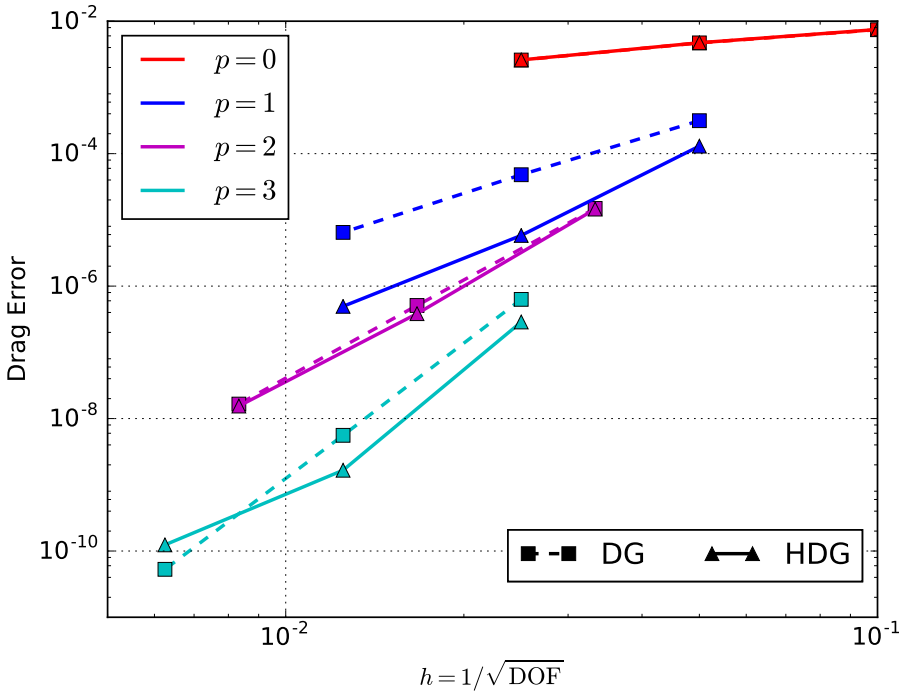
Drag error, on the other hand, should optimally converge at the rate $2p+1$, since the adjoint problem is well defined and there are no singularities. Post-processing HDG will not help, since the output is defined as the integral of the flux itself along the bottom wall and therefore uses the boundary state $\mathbf{u}^B(\mathbf{u}_h^+)$. DG nearly obtains these rates for all orders in the results shown in Figure 4.11, and again the HDG results again uncover slightly higher rates, though not quite a full order of accuracy. The "exact" drag is computed from the solution obtained by using a $p = 5$ solution after uniformly refining the mesh once beyond the finest shown on the plot.



(a) $p = 1$ (b) $p = 1$ post-processed ($p = 2$)

(c) $p = 2$ (d) $p = 2$ post-processed ($p = 3$)

Figure 4.10: Pressure contours for the subsonic bump with the Euler equations.



(a) DG (dashed) and HDG (solid) convergence

Order	0	1	2	3
DG	0.66	2.71	4.87	6.82
HDG	0.66	4.48	5.25	7.44

(b) Rate of convergence on coarsest meshes for DG and HDG

Figure 4.11: Drag convergence for the subsonic bump with the Euler equations.

4.5.2 NACA 0012 Test Case

The NACA 0012 geometry is a classic symmetric airfoil, and computing the flow around it with the Navier-Stokes equations has become the archetypal test case for computational fluid dynamics. This series of cases computes the flow at a set of conditions at different Reynolds numbers. These cases are selected because they are robust enough that differences in the nonlinear convergence is negligible, so the results can focus on accuracy and run time using the same DG and HDG framework.

4.5.2.1 Inviscid Solution

This test case compares the solution of the Euler equations over a NACA 0012 airfoil at $\alpha = 2^\circ$ angle of attack and $M_\infty = 0.5$. The geometry used here has a sharp trailing edge, so this causes a singularity in the solution, which affects the rate of convergence of boundary outputs on the airfoil. Again, since there is no viscosity, a slip condition is imposed at the airfoil surface, total temperature and pressure are imposed at the farfield ahead of the airfoil, and static pressure is imposed behind the airfoil.

The output of interest is the drag force calculated on the airfoil. If the domain were infinite, and the solution were exact, there would be zero drag, but for a fixed size mesh at 500 chord lengths from the airfoil, a calculable level drag exists on the airfoil. The "exact" drag from which the error is computed is obtained by using an adaptive computation with $p = 5$ using 397224 DOFs. Obviously uniform refinement is then not an effective strategy for this case, but is nonetheless used here to compare the methods. Figure 4.12 shows the initial coarse mesh and compares convergence with both the number of entries in the Jacobian matrix, a measure of the memory usage, and total run time of the computation. As explained in the bump case HDG post-processing has negligible effect on the drag calculation, so it is skipped, and the same rate of convergence is expected from both methods. After refining the mesh, the solution is "restarted" for the next solve by injecting the previous solution onto the uniformly refined mesh. Therefore, the total run time includes the time from the previous calculations at the same order.

For this computation there does not seem to be a benefit, in terms of memory, to using to higher orders for a given drag error in DG, but there is an immense savings for increasing p for HDG. In fact, by using $p = 4$ instead of $p = 1$ for HDG, the drag error is reduced by two orders of magnitude for the same amount of memory and only a slightly larger run time. Overall, the run times for HDG are faster, especially

for $p > 2$, and there is a benefit to using higher orders. Not only is the drag error reduced, but the slope on the error versus run time plot is steepened by increasing the order in HDG. The exception is at $p = 4$, which took longer to compute than $p = 3$, suggesting that at this order the computational efficiency of the code or linear solver is decreasing.

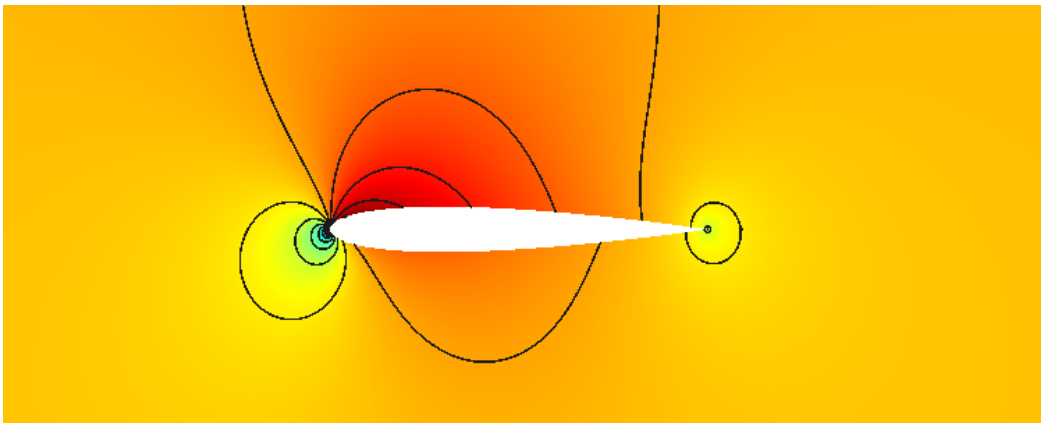
The run time separated into each component of a single linear solve is shown in Fig. 4.19a. In this plot the run time for a single Newton iteration is longer for DG than HDG, but on the convergence against overall timing plot, the run time is actually slower for HDG, due to an extra Newton iteration taken with this method.

4.5.2.2 Viscous Solution

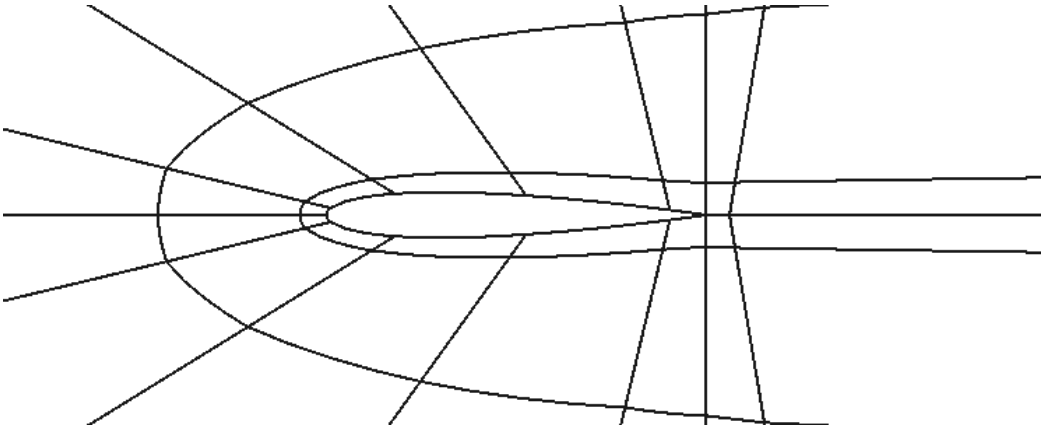
This test solves the full Navier-Stokes equations, with a non-zero viscosity, over the same NACA 0012 geometry. For this case the Reynolds number is $\mathrm{Re} = 5 \times 10^3$ and $\alpha = 1^\circ$ instead of $\alpha = 2^\circ$ to reduce the effect of the singularity at the sharp trailing edge. A no-slip boundary condition is imposed on the airfoil surface, due to the presence of the viscosity term, and doing so requires a mesh that resolves the resulting boundary layer at the airfoil surface. Therefore the initial mesh used here, shown in Figure 4.13b, resolves this boundary layer by adding elements bringing the total to 442 quadrilateral elements.

With the viscosity term active in the system, both the mixed HDG, using the constant-type viscous stabilization, and primal HDG method developed here, removing the dual variable $\vec{\mathbf{q}}_h$ and using BR2-type stabilization are compared against DG. Although post-processing will have little effect on the boundary output, using the mixed form with the new convective flux formulation developed in Section 4.4.2 can allow the gradients represented by the dual variable to optimally converge. If this occurs, the lift and drag outputs, which now are functions of both the boundary state and the boundary gradient, will have improved convergence. As the output contains both the convective and diffusive fluxes, the difference in convergence may not be a full order of accuracy. The "exact" values of lift and drag are computed using adaptive refinement at $p = 5$ with 829440 DOFs.

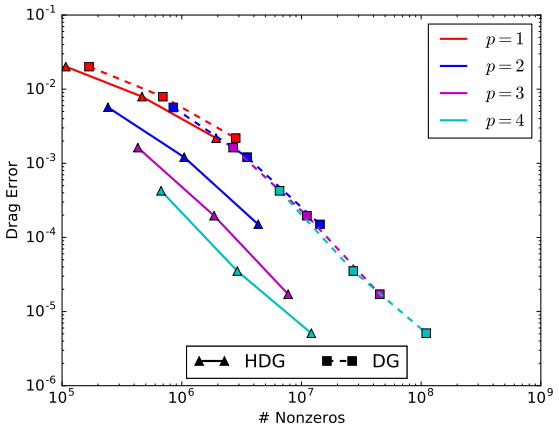
Before considering the outputs, if qualitative accuracy of the field quantities is desired from the simulation, post-processing is useful. Figures 4.17 and 4.18 show contours of Mach number with the same limits computed with DG and post-processed HDG. For the low orders considered here, the run times are similar, so comparing, for example, the $p = 1$ DG results to $p = 2$ (post-processed $p = 1$) results from HDG is a fair comparison of the field accuracy for a given computational effort. Especially



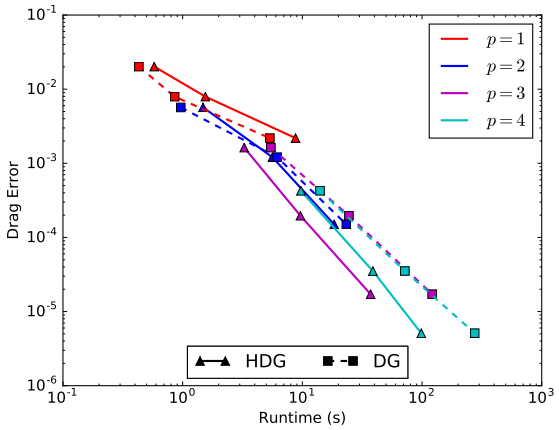
(a) Mach contours over the airfoil between [0, 0.722].



(b) Quartically curved initial mesh (140 elements) used for uniform refinement study.



(c) Convergence with Jacobian entries



(d) Convergence with total run time

Figure 4.12: Comparison of numerical drag convergence with DG and HDG for inviscid flow with the Euler equations at $M_\infty = 0.5$ over a NACA 0012 airfoil at $\alpha = 2°$.

for $p = 1$ the post-processed HDG Mach number gives a better qualitative image of the field. It is interesting to note that the post-processing method introduced in Section 4.4.3 works where the original post-processing methods using RT spaces, had they been extended to systems, would not work due to the curved geometry.

Figure 4.14 compares lift and drag convergence for DG and three HDG variants with different formulations and stabilizations:

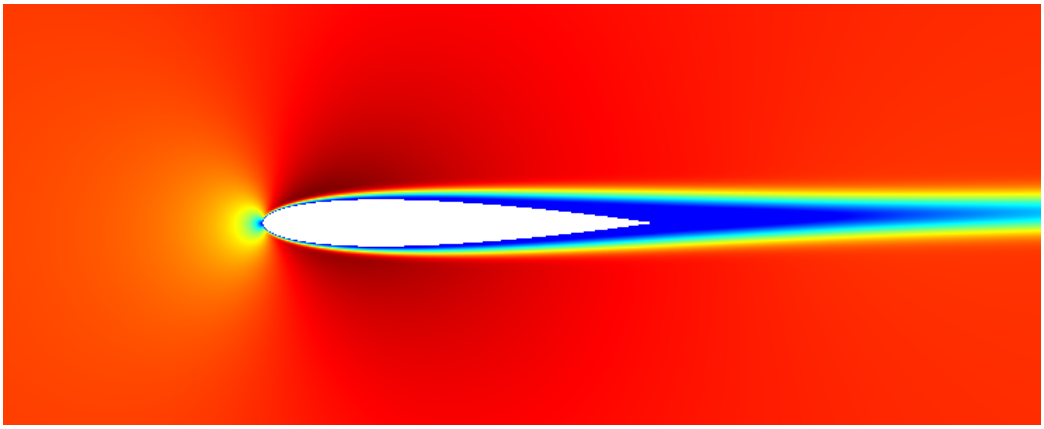
- HDG-M-BR2: Mixed form using one-sided BR2
- HDG-M-BR2H: Mixed form using mesh-size independent one-sided BR2, and
- HDG-P-BR2: Primal form using one-sided BR2.

It appears from this result that the more accurate gradient calculation in mixed HDG is only improving the lift convergence, while slightly negatively impacting drag calculation, compared to the calculation from the primal HDG solution. While it is difficult to determine exactly why this occurs, one reason could be that the gradient near the airfoil could also be no better approximated by the dual variable in this situation. The drag calculation is more affected by the normal gradient at the wall than lift, leading to the negatively impacted convergence. As lift seems better approximated by the mixed HDG, and drag by primal HDG, the remaining Figures 4.15 and 4.16 compare mixed HDG to DG for lift, and primal HDG to DG for drag. There is a large memory savings when computing with HDG toward a given error level for both outputs, and it is especially interesting to note that the error versus non-zeros slope is generally steeper for HDG than DG in the case of lift, due to the improved accuracy on the same mesh.

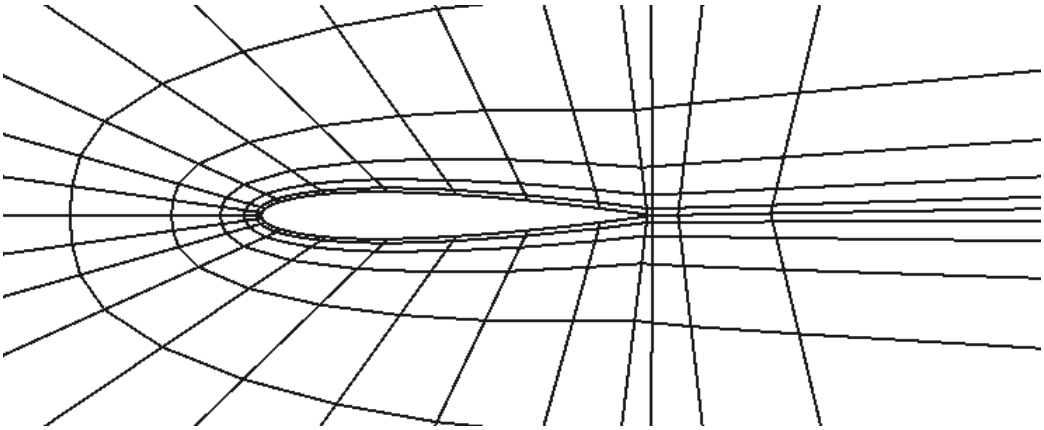
Noticable savings also exists in terms of total run time when computing with HDG except for the lift output with $p = 4$. Figure 4.19b compares DG to HDG-M, mixed HDG using the mesh-size independent BR2 stabilization, and HDG-P, primal HDG with BR2 stabilization, showing that compute time per Newton iteration is increasingly faster when using high orders p with both HDG methods.

4.5.3 Delta Wing

The final case solves the Navier-Stokes equations over a delta wing, thereby demonstrating and comparing the DG and primal and mixed HDG performance on a three-dimensional case for increasing order on a fixed mesh. Recall that the DOF per mesh vertex required for each discretization depends on the element type and therefore also very strongly on the dimension of the problem, so it is important that the new HDG methods perform well for this regime as well. This specific case solves

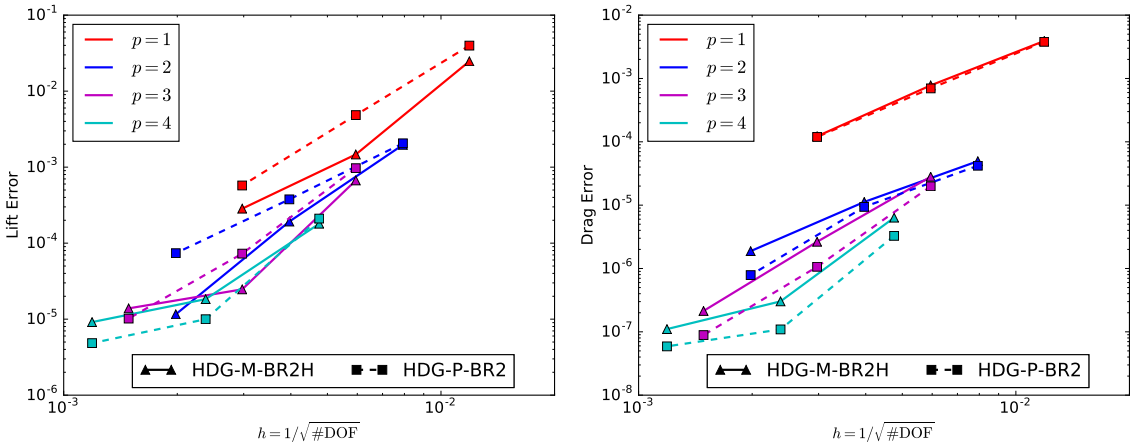


(a) Mach contours over the airfoil between [0, 0.6].



(b) Quartically curved initial mesh (442 elements) used for uniform refinement study.

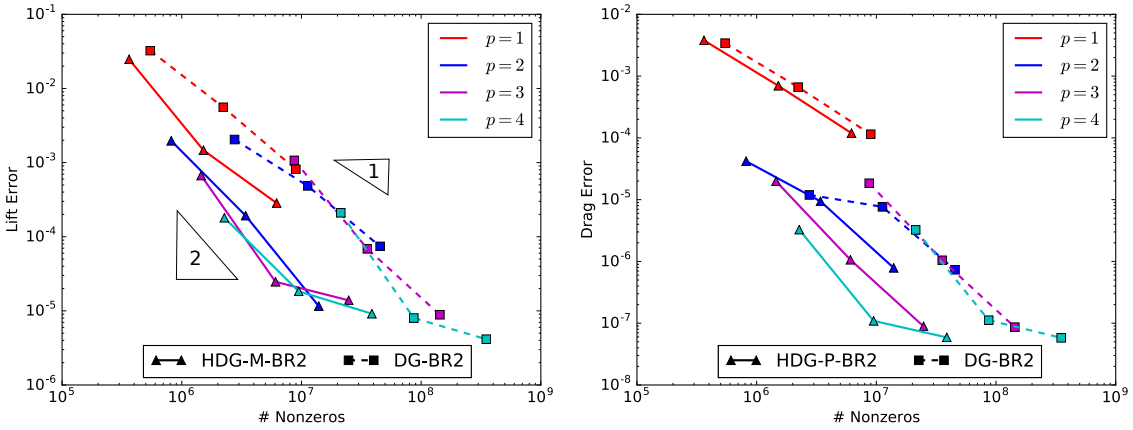
Figure 4.13: Setup for the comparison test case with Navier-Stokes at $\mathrm{Re} = 5 \times 10^3$ and $M_\infty = 0.5$ over a NACA 0012 airfoil at $\alpha = 1°$.



(a) Lift convergence

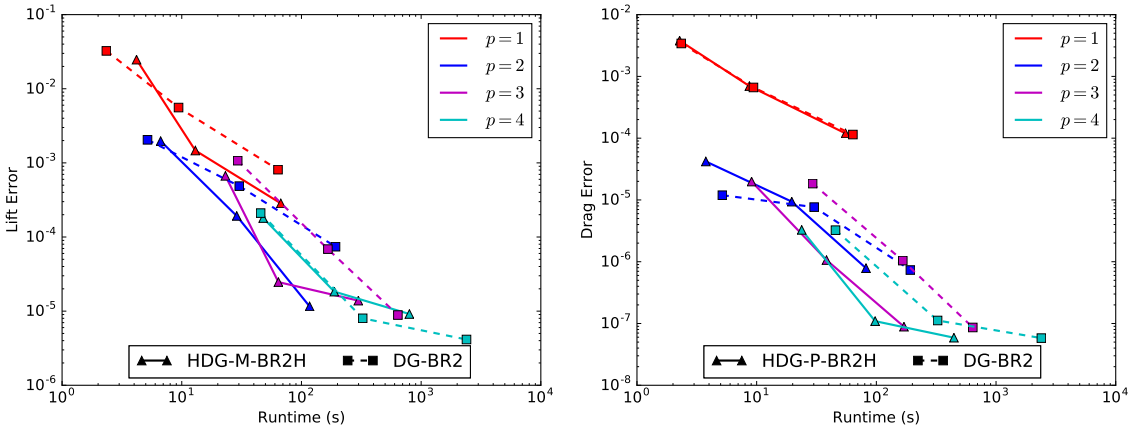
(b) Drag convergence

Figure 4.14: Convergence with h for the viscous NACA 0012 test.



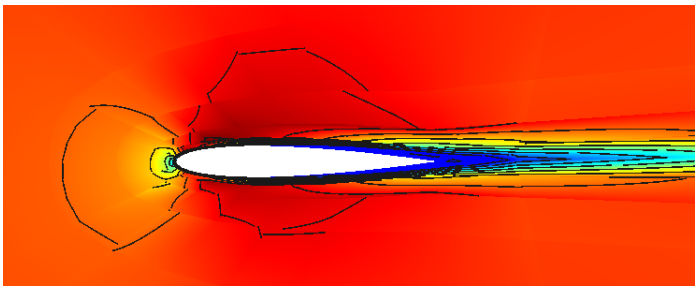
(a) Lift convergence (b) Drag convergence

Figure 4.15: Convergence with Jacobian non-zeros for the viscous NACA 0012 test.

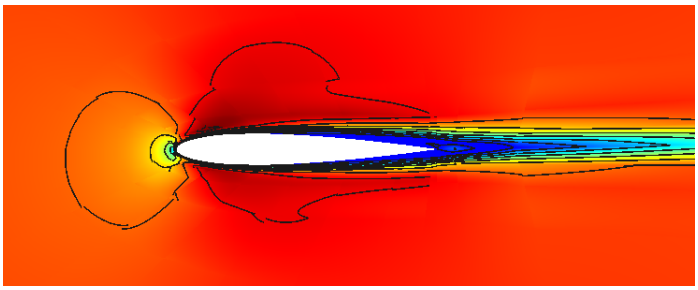


(a) Lift convergence (b) Drag convergence

Figure 4.16: Convergence with total run time for the viscous NACA 0012 test.

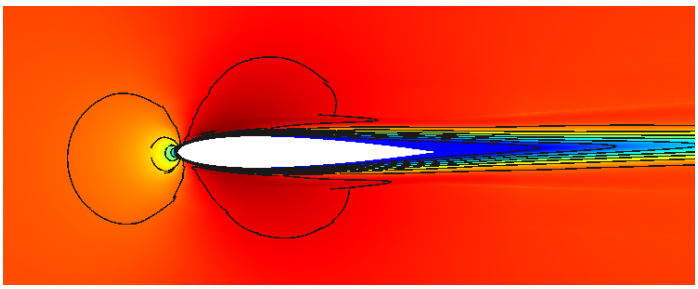


(a) DG $p = 1$

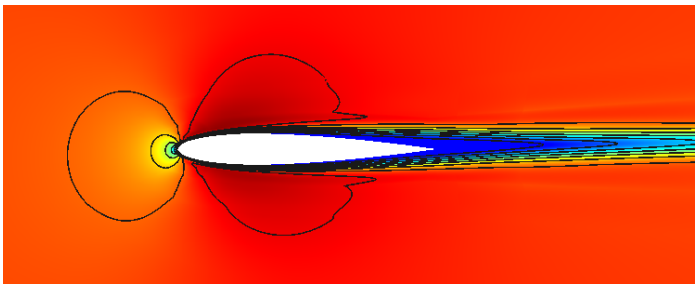


(b) HDG post-processed $p = 1$ ($p = 2$)

Figure 4.17: Mach contours [0,0.61] for DG and post-processed HDG $p = 1$ solutions on the initial mesh.

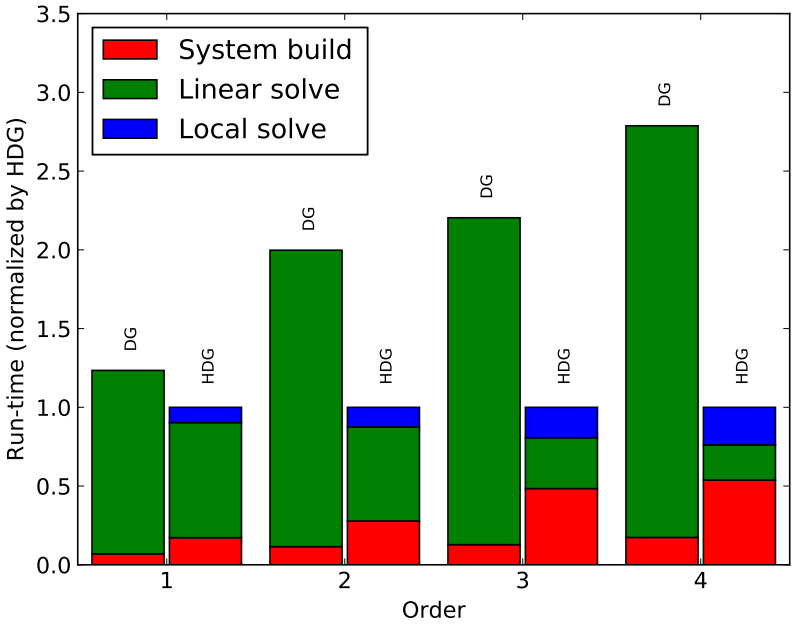


(a) DG $p = 2$

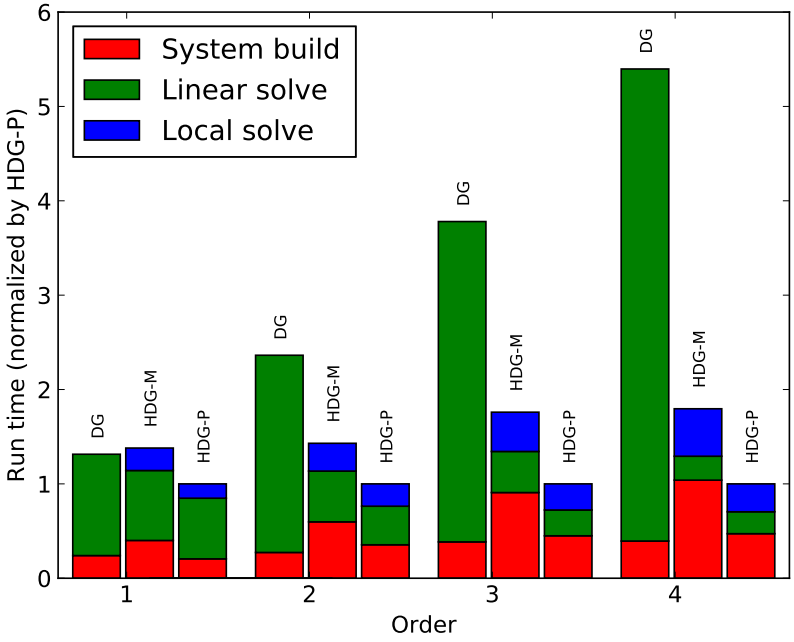


(b) HDG post-processed $p = 2$ ($p = 3$)

Figure 4.18: Mach contours [0,0.61] for DG and post-processed HDG $p = 2$ solutions on the initial mesh.



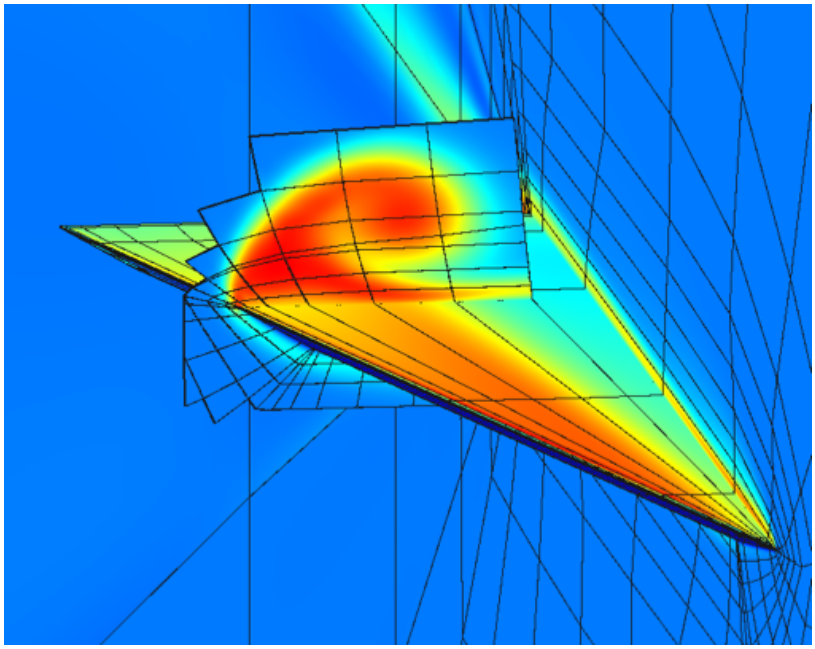
(a) Inviscid NACA 0012 test



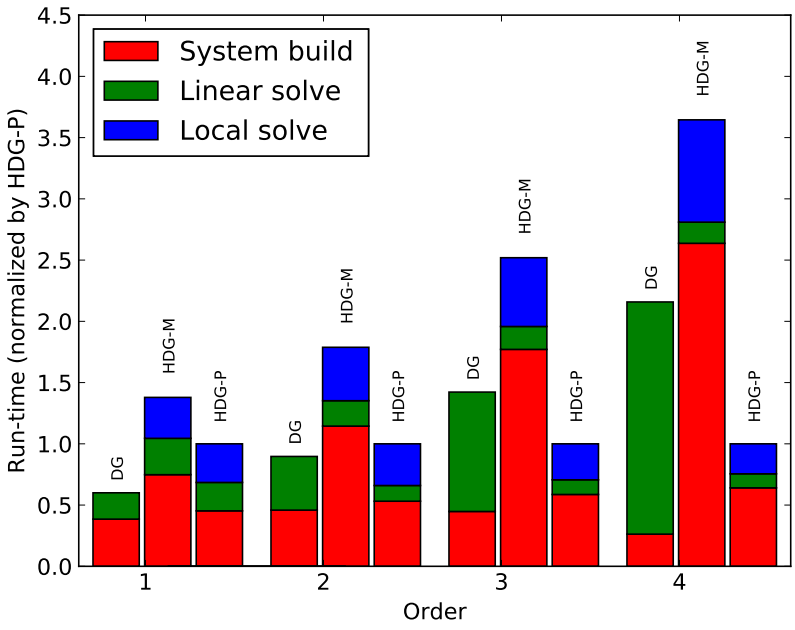
(b) Viscous NACA 0012 test

Figure 4.19: Run time per Newton iteration separated into each component of the solution procedure, and normalized by the total average runtime per Newton iteration for HDG.

the equations at $M_\infty = 0.3$, $\alpha = 12.5°$, and $\mathrm{Re} = 4000$ with the canonical hexahedral tensor product Lagrange basis. Entropy contours on the wing boundary and cut plane and timing results are shown in Fig. 4.20. In the timing results, HDG-M again refers to the mixed form using the mesh-size independent BR2 stabilization, and HDG-P to the primal form using BR2 stabilization. Here the HDG mixed formulation is slower than DG, which the figure indicates is due to the very large cost of the system build. This is due to the large cost of the local matrix assembly in the static condensation, since it is dramatically reduced by using the primal formulation. The HDG primal formulation is increasingly faster than DG for high orders.



(a) Entropy field on boundaries and cut plane



(b) Timing comparison

Figure 4.20: Visualization and timing results from the delta wing test.

CHAPTER 5

Combined Mesh and Order (*hp*) Adaptation

This chapter introduces two algorithms for combined mesh and order adaptation applicable to the discontinuous Galerkin methods previously discussed. Error in the resulting output from these methods is chiefly a function of the approximation space for the solution, and thereby depends on the mesh and order of the computation. Higher computational efficiency is obtained by refining only the regions that most affect the error, in the way that best captures the output. This is especially important for boundary outputs from CFD, where usually only a portion of the domain significantly affects the output. The solution in different regions of the domain will often be better approximated by either refining the mesh or by raising the approximation order, so by combining these approaches the computation can more efficiently represent a solution than when it is limited to only one of these approaches. Section 5.1 gives background on prior adaptation methods in general and defines terminology that is used throughout the chapter. Sections 5.2 and 5.3 then present the algorithms developed in this work for enabling combined *hp*-adaptation. The first of these algorithms refines the mesh and order locally, while the second algorithm replaces the mesh and changes the approximation orders on all elements globally.

5.1 Overview of Existing Methods

In the context of this work, adaptation refers to any algorithm that changes the solution approximation space by altering the mesh or the solution approximation order, with the goal to more efficiently represent the output of interest from the computation. This definition encompasses many types of algorithms, from refining or coarsening regions, to globally redistributing resolution. Numerous adaptation strategies have already been presented that take the local error indicator from Section 3.2 and modify the approximation space to more accurately compute an output.

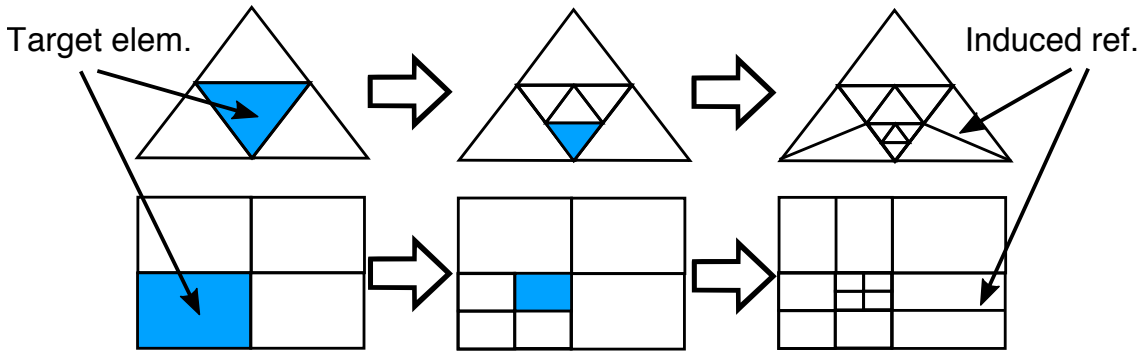


Figure 5.1: Isotropic hanging-node adaptation on triangles and quadrilaterals. Only a single cut is allowed, so the second refinement iteration induces refinement in the neighboring elements.

The most straightforward of these algorithms, and traditionally the most popular for CFD, is to refine the mesh, leaving the solution order constant for all elements.

Refining the mesh is convenient because discontinuous Galerkin methods, together with finite volume methods traditionally used in CFD, allow for the mesh to be non-conforming, in that faces of the mesh do not have to correspond one-to-one. The mesh can therefore be refined by local operations known as *hanging-node* refinement, whereby elements are split by introducing additional nodes and faces [44, 45, 46, 47, 6, 29]. Figure 5.1 shows isotropic refinement, which involves splitting the target element isotropically by halving the length of each face. Note that by only allowing a single level of refinement between elements, refinement may be induced in the neighboring elements. This process can be repeated many times for quadrilaterals, but on triangles, the anisotropic induced refinements in neighboring elements reduce the acute angles in the triangle, raising the aspect ratio. After many refinements in nearby elements, the Jacobian mapping from the element reference space to global coordinates can become nearly singular, leading to poor conditioning of the overall system. Refinement, while tedious to implement, can be efficiently executed. Coarsening, on the other hand, is difficult to implement efficiently without storing the entire history of refinement in a tree-like structure, since it requires looping over sets of elements and checking if coarsening is possible. Hanging-node-based mesh adaptation is therefore usually limited to increasing the number of elements, and therefore also the DOF, without coarsening.

A potentially more efficient mesh adaptation strategy for lowering error in the output is to fix a certain number of elements and move, add, or remove vertices to match the target. Performing this operation locally or globally is often called r-adaptation [48] or mesh optimization. Optimization methods considered here globally

re-mesh the domain by specifying the desired characteristics of the next mesh in the adaptive sequence using a Riemannian metric field. The metric $\mathcal{M}$ is a tensor quantity in the form of a symmetric positive-definite matrix at each point in the domain that locally warps the space, such that the set of points that are a unit distance from $\mathbf{x}$ form an ellipse in Cartesian space. The length of an infinitesimal segment $\delta\mathbf{x} \in \mathbb{R}^d$ is $\delta\Gamma$ under the metric $\mathcal{M}$ such that

$$\delta\Gamma^2 = \delta\mathbf{x}^T \mathcal{M} \delta\mathbf{x}. \tag{5.1}$$

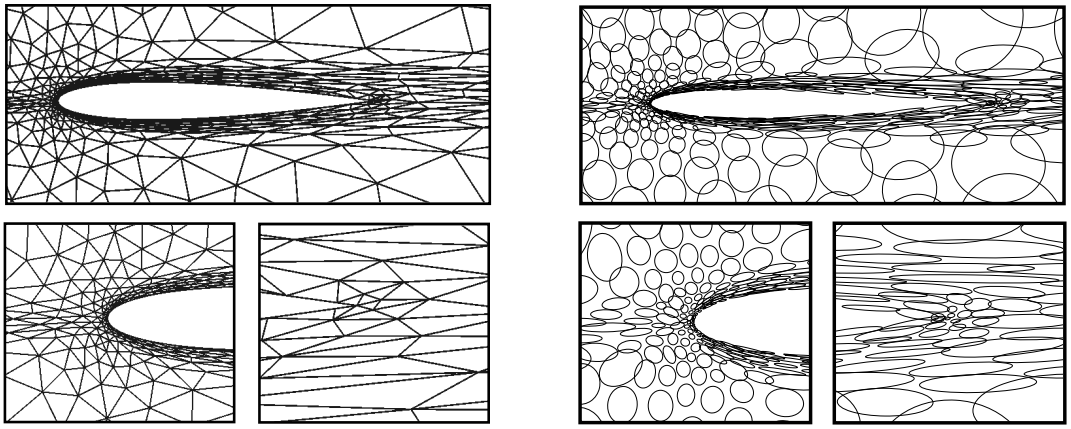
A meshing algorithm can then take this encoded information about the size and anisotropy of the desired elements at every point of an existing, or background, mesh and create a new mesh. In two dimensions, the mesh that exactly conforms to the metric field will have each edge of unit length under the metric [49],

$$\mathbf{x}^T \mathcal{M} \mathbf{x} = 1, \tag{5.2}$$

where $\mathbf{x}$ is the vector defining the edge. This idea is illustrated in Figure 5.2, which shows one such metric represented by an ellipsoid, and one such adapted mesh with anisotropic elements in the boundary layer around the airfoil. The eigenvectors $\mathbf{e}_i$ of $\mathcal{M}$ are the principal axes of the ellipsoid, and the corresponding eigenvalues $\lambda_i = 1/h_i^2$ are related to the size of each axis h_i, so the anisotropy in the element is

$$\frac{h_2}{h_1} = \sqrt{\frac{\lambda_1}{\lambda_2}}. \tag{5.3}$$

Variable-order finite element methods have also been a topic of interest, since for smooth solutions these can lead to faster convergence than mesh refinement. Interestingly, even around certain types of singularities, raising the order can be more efficient in representing the solution than refining the mesh [50]. Such p-adaptation methods that keep the mesh the same have even been considered for use in CFD [20]. It has been observed that the success of these methods is dependent on the mesh [51], implying that the best possible performance is obtained by combining mesh and order adaptation. Therefore, hp-adaptation, combining the best of both strategies, has the most promise to automatically reduce the error fastest.



(a) Mesh obtained by global re-meshing

(b) Metric field for mesh on left

Figure 5.2: An example mesh (left) obtained by anisotropic adaptation using metric-based global re-meshing for a viscous Navier-Stokes test case. The ellipses representing the mesh-implied metric (right) are stretched in the boundary layer, consistent with the anisotroic stretching of the elements.

5.1.1 Anisotropy

An important property of an h-adaptation method for CFD, and many other applications, is to generate stretched elements in areas where the solution changes disproportionately in different directions. Such regions of anisotropy should be reflected in the solution representation, usually in the mesh itself, to optimally capture the solution without unnecessarily adding elements and thereby increasing the computational cost. Mesh anisotropy can be introduced to a limited extent by local operations such as hanging-node refinement, but global mesh optimization offers the most flexibility to create stretched elements without introducing unnecessary cost.

A number of different approaches have been introduced for incorporating anisotropy into the solution, since the error indicator itself does not provide this information. The dominant approach is to enforce that the error estimate of a scalar quantity be the same in any chosen spatial direction [17, 18, 52]. For systems of equations, an appropriate scalar needs to be chosen; *e.g.* the Mach number in compressible Navier-Stokes simulations. For a second-order method, the Hessian matrix of second derivatives stores the needed information, so the resulting metric field is proportional to the Hessian in the scalar scaled by the error indicator. Various other approaches have been developed which extend these methods to high order by utilizing the $(p+1)$-st derivative information [1, 53]. For higher-order methods, the $p+1^{\text{st}}$ derivatives of

the approximate solution indicate the inability of the basis functions to interpolate the exact solution [54]. Hall [46] develops a different method based on solving local problems with different refinements to determine the most efficient option.

The MOESS framework, introduced by Yano and Darmofal [55] determines unknowns in a convergence rate tensor by locally sampling the element, then solving an optimization problem to determine an optimal next mesh. The unknowns in the convergence rate tensor encode the anisotropy automatically, removing the need to consider derivatives of a representative scalar. This work was later extended by Fidkowski [56], simplifying the procedure to compute the samples. This is the approach that has been built on in the present work to create an anisotropic *hp*-optimization algorithm.

5.1.2 *hp*-Adaptation

Combining mesh and order adaptation adds an additional level of complexity beyond anisotropic adaptation, in that the algorithm needs to decide how to distribute the relative mesh resolution as well as the approximation order p. In the past, cases were studied for which theoretically optimal distributions of mesh and order resolution to reduce the interpolation error could be derived. Using these heuristics in general is not an option, as many different types of singularities are encountered in engineering applications and the problem quickly becomes intractable. Instead, a general algorithm must be followed which makes choices based on *a posteriori* information.

To illustrate the need for *hp*-adaptation, consider the one-dimensional error estimate on a given element Ω_e of size h [57]

$$\|u_h - u\|_{H^1(\Omega_e)} \leq C h^{\min(p,m-1)} p^{1-m} \|u\|_{H^m(\Omega)}, \tag{5.4}$$

where C is a constant independent of the element size and order p, and $H^m(\Omega)$ is the m-th order Sobolev space. The convergence rate will be algebraic if the solution regularity m is limited, and will be nearly unlimited for increasing p if the solution is very smooth (*i.e.* $m \gg 1$) [58]. Although this argument is for the interpolation error instead of the error in an output functional, which is the target of this work, it shows the main concept behind *hp*-adaptation, namely that the order should be increased in regions where the solution is smooth, while near singularities where m is small a careful balance between mesh refinement and order refinement should be maintained.

For this reason, many *hp*-adaptation algorithms use regularity estimates that check the underlying smoothness to guide the refinement. Methods based on such estimates

have been shown to be very successful in guiding the adaptation, even for Navier-Stokes computations [37]. There are various methods to achieve this for resolving outputs in DG methods. Recent examples include:

- Woopen and May [37] use an element-based indicator that integrates the jump between the approximate solution and a lower-order projection
- Burgess and Mavriplis [59], and Shi and Wang [60] use a jump-based indicator that integrates the normalized jump between elemental approximate solutions
- Hall, Georgoulis, and Houston [46] determine the smoothness from an expansion in terms of Legendre polynomials.

There are, however, drawbacks to using such methods. Firstly, these methods require an additional parameter to either define when the local solution is smooth enough, or a set of parameters to designate the target number of elements and overall DOF of the approximate solution, thereby setting the amount of resolution due to order refinement. In this sense, the methods can only seek to be "optimal" up to this parameter, and if chosen incorrectly the efficiency of the adaptation will be decreased. The other drawback with such methods is that they usually require a sufficiently high order p to be able to access the regularity.

The aim of this work is both to develop the methods described above, which do not select between order and mesh refinement based on a smoothness indicator, and to determine the extent to which such a parameter-free hp-adaptation algorithm makes the correct refinement decision. These methods are not the first of their kind, as there have been several sampling-based approaches in recent years that depart from using regularity estimates, as summarized in Table 5.1. The method in the table based on hanging-node (HN) mesh mechanics, by Ceze and Fidkowski [47], selects the refinement option – any of a set of isotropic and anisotropic local refinements or order refinement – that maximizes a merit function. Another recent method by Balan, Woopen, and May [61] uses global re-meshing (GR) and determines the elemental order by minimizing an *a priori* interpolation error estimate. This methodology is based on the anisotropic mesh framework presented by Vít Dolejší [62].

This work introduces two algorithms for hp-adaptation, one for each type of mesh mechanics. The first of the algorithms presented here is similar to the approach developed by Ceze and Fidkowski [47], in that it selects a local refinement option based on maximizing or minimizing an objective function. The present method, however, differs in the how the error estimates are computed and has been generalized

		Mesh / Order Decision	
		Regularity estimate-based	Sampling-based
Mesh Adaptation	HN	• Hall, Georgoulis, Houston, 2007 [46] • Burgess and Mavriplis, 2011 [59] • Shi and Wang, 2013 [60]	• Ceze and Fidkowski, 2012 [47] • *Dahm, Fidkowski*
	GR	• Woopen and May, 2015 [37]	• Balan, Woopen, May, 2016 [61] • *Dahm, Fidkowski*

Table 5.1: Recent DG *hp*-adaptation algorithms targeting output functionals known to this author. Rows denote the mesh refinement strategies: HN (hanging-node) or GR (global re-meshing); columns denote how the choice between mesh and order refinement is ultimately decided.

to support both DG and HDG methods, requiring a vastly generalized adaptation framework to be implemented. The second approach, for simplex optimization, uses a global metric-based re-meshing for simplex elements, extending the algorithm by Yano and Darmofal [55], described below in Section 5.1.3. This approach has similarities to that developed by Balan, Woopen, and May [61] in that it differentiates between refinement options based on a measure of the error instead of a regularity estimate, but utilizes an optimization-based procedure based on error and cost models instead of an interpolation error estimate.

5.1.3 Mesh Optimization

This section reviews the metric-based re-meshing method with a focus on the optimization algorithm proposed by Yano [55], and is intended to introduce the concepts that will be necessary to understand the *hp*-optimization algorithm presented in Section 5.3. The MOESS algorithm for mesh optimization seeks the mesh T_h^* that minimizes the error in the output, namely

$$T_h^* = \arg\inf_{T_h} \delta J \quad \text{such that } C(T_h) \approx C_{\text{tgt}}.$$

To solve this problem in an exact sense would be nearly impossible, since it would require knowing the high-dimensional relationship between the mesh and output error. Instead, continuous surrogate models of the error and cost in DOF are used, from which the algorithm can choose the appropriate metric.

5.1.3.1 Metric Field

Symmetric positive-definiteness of the matrix $\mathcal{M}$ is necessary for it to act as a metric for the Cartesian space, since this implies that all the eigenvalues are positive, and thereby the size of each of the principal axes is non-negative and real. This however leads to at least two problems. Firstly that the metrics themselves do not form a complete vector space, since $a\mathcal{M}$ is not necessarily symmetric positive-definite (SPD), and secondly that the "swelling effect" can result, in which the determinant, and thus size of the metric, is not preserved under the average [63]. Therefore it is not easily possible to define a continuously changing metric based on $\mathcal{M}$ alone that can be represented in a coordinate-independent way. Fortunately however, the exponential map $\mathcal{S} \to \mathcal{M}$ on the tangent space about an initial metric $\mathcal{M}_0$ such that

$$\mathcal{M} = \mathcal{M}_0^{1/2} \exp(\mathcal{S}) \mathcal{M}_0^{1/2} \tag{5.5}$$

is a bijection with a well-defined inverse logarithm map

$$\mathcal{S} = \log\left(\mathcal{M}_0^{-1/2} \mathcal{M} \mathcal{M}_0^{-1/2}\right). \tag{5.6}$$

This means that for every *step matrix* $\mathcal{S}$, there exists a unique metric $\mathcal{M}$ according to (5.5), and for every metric there also exists a unique step matrix about a given $\mathcal{M}_0$. The step matrix is the variable used to represent changes to the initial metric in the optimization. When using $\mathcal{S}$ to represent changes to the metric, the average of a set of metrics $\{\mathcal{M}\}$ is defined by taking the arithmetric average of the corresponding step matrices $\{\mathcal{S}\}$ in (5.5). Each step matrix, as given in (5.6), is defined in the tangent space about the mean metric, so taking the average requires the solution of a nonlinear equation [64].

The metric field of an existing simplex mesh is determined by solving a set of $d(d+1)/2$ equations for the unknowns in the metric $\mathcal{M}_0^e$ on each of the simplex elements, enforcing (5.2) for each edge. Note that this procedure already contains an approximation, since in the continuous sense the edges are unit length under the metric defined at that point, requiring an integral over the metric field. This process defines the metric element-wise, but the mesher – for two-dimensions BAMG [65] – requires the metric defined at the vertices. The neighboring elemental metrics are averaged, as above, to obtain these.

A greater understanding of how the step matrix affects the metric can be created

by decomposing the step matrix into isotropic and anisotropic components as

$$\mathcal{S} = s\mathcal{I} + \tilde{\mathcal{S}} \tag{5.7}$$

where $s = \text{tr}(\mathcal{S})/d$. It can be easily shown that s scales the metric while preserving the shape, and that changes to $\tilde{\mathcal{S}}$ preserve the volume, namely the determinant, of $\mathcal{M}$ [55].

5.1.3.2 Error and Cost Models

In order to determine how changing the shape of the element, via the elemental step matrix $\mathcal{S}_e$, affects the error in the output, a form of the error is assumed, with parameters which will be specific to the element. The idea employed by MOESS is that the error model parameters may be determined purely locally by a "sampling" procedure that solves a local system, which can be performed element-wise and then globally optimized. Changes in the size and shape of the element also affect the overall cost of the computation in terms of DOF since, assuming all other elements stay the same size, elements have to fill the space.

Motivation for the error model can be found in the result for an isotropic shape change from an element of size H to h

$$\frac{E}{E_0} = \left(\frac{h}{H}\right)^r = \exp\left(r \log\left(\frac{h}{H}\right)\right) \tag{5.8}$$

where r is the local rate parameter. If the entire step matrix were to be used, the rate may also vary when changing different components, so an appropriate form for each element instead, utilizing different rates, is

$$E_e = \epsilon_e \exp(\text{tr}(\mathcal{R}_e \mathcal{S}_e)), \tag{5.9}$$

where ϵ_e is the original error estimate based on the unprojected fine-space adjoint. In this form $\mathcal{R}_e$ is a symmetric matrix of unknown components and $\mathcal{E}$ is a measure of the error. Yano [55] shows that when decomposing the rate matrix similarly, $\mathcal{R}_e = r\mathcal{I} + \tilde{\mathcal{R}}_e$, both the expressions rs and $\text{tr}(\tilde{\mathcal{R}}_e \tilde{\mathcal{S}}_e)$ control the error magnitude, but that the off-diagonal terms have no effect in the error model.

The coefficients of the rate matrix $\mathcal{R}_e$ are determined by fitting the error model to a list of error and step matrix pairs. The samples for mesh optimization consist of anisotropic refinements, as shown in Figure 5.3. For each refinement the step matrix

is found by taking an affine-invariant mean of the mesh-induced metrics for each of the sub-elements $\mathcal{M}_{ik}$, then computing the associated step for element e, namely $\mathcal{S}_{ei}$.

To find the error associated with each of these samples, the computation and error estimation could be re-run for every element and every sample, but this is intractable, even for a small number of elements. Instead the fine-space adjoint $\mathbf{\Psi}_{h,p+1}$, usually from the order $p+1$ space, is L^2-projected onto the space of each of the mesh refinement samples – at order p – then back to the fine-space, for every element of the mesh. After this projection to the local refinement sample h_i, the adjoint variable is denoted by $\mathbf{\Psi}_{h_i,p}^{h,p+1}$, and represented back on the fine space (shown in the super-script) by $\tilde{\mathbf{\Psi}}_{h_i,p}^{h,p+1}$. If the fine-space adjoint is in the space $\mathcal{V}_{h,p+1}$ and the samples are represented by $\mathcal{V}_{h_i,p}$, then this is

$$\mathcal{V}_{h,p+1} \to \mathcal{V}_{h_i,p} \to \mathcal{V}_{h,p+1}, \tag{5.10}$$

where the arrows denote L^2 projections. This series of projections removes some of the information in the fine-space adjoint. If the adjoint-weighted error estimate based on $\tilde{\mathbf{\Psi}}_{h_i,p}^{h,p+1}$ were used alone in the error model, the goal of the optimization should be to maximize the error, since that would lead the algorithm to choose the option that best represents the error. Instead, with the objective of minimizing an error, the model is fit to the remaining error after subtracting off this error estimate, $\Delta\epsilon_{ei} = (\tilde{\mathbf{\Psi}}_{h_i,p}^{h,p+1})^T \mathbf{R}_{h,p+1}(\mathbf{U}_{h,p})$, from the error estimate based on the unprojected fine-space, as

$$E_e(\mathcal{S}_{ei}) = \epsilon_e - \Delta\epsilon_{e,i} \quad i = 0 \dots n_{\text{samp}}, \tag{5.11}$$

where ϵ_e is the usual error estimate based on the unprojected fine-space adjoint. This yields a set of samples $\{E_e(\mathcal{S}_{e,i}), \mathcal{S}_i\}$ that can be used to find the unknowns in the rate matrix. The coefficients of $\mathcal{R}_e$ in the error model are fit by a modified least-squares form

$$\mathcal{R}_e = \arg\min_Q \sum_{i=0}^{n_{\text{samp}}} \left(\log \frac{\epsilon_{e,i}}{\epsilon_e} - \text{tr}(Q\mathcal{S}_{e,i}) \right)^2. \tag{5.12}$$

The cost model $C(\mathcal{S}_e)$ can be computed without the use of sampling, instead integrating a cost density $c(\mathcal{M}(x), x)$ over the element. If the metric acts to shrink the element, the cost density increases so the overall cost increases. The integral of the density is explicitly computable if the order p is held constant, and takes the form

$$C(\mathcal{S}_e) = N_p \exp\left(\frac{1}{2} \text{tr}\left(\mathcal{S}_e\right) \right). \tag{5.13}$$

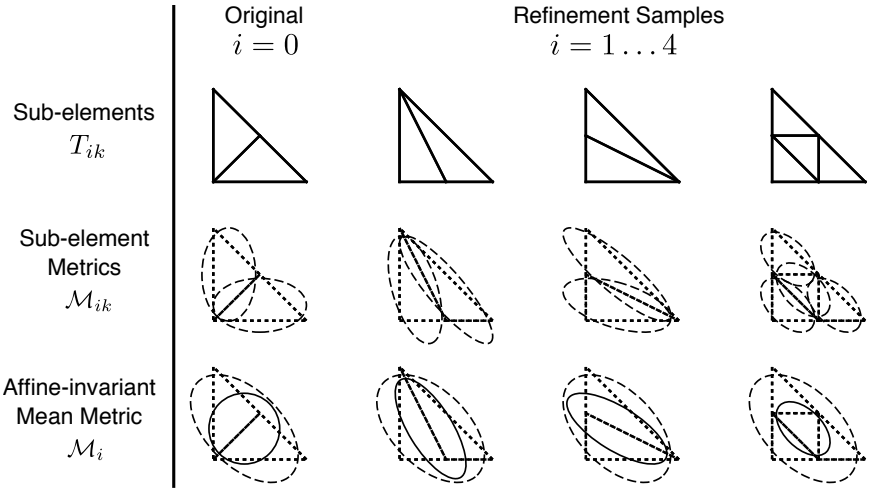


Figure 5.3: Procedure to obtain each of the triangular refinement sample mean metrics.

The multiplicative factor N_p is the number of DOF of the original approximate solution on element Ω_e, so that when $\mathcal{S}_e = 0$ the original cost is recovered.

5.1.3.3 Algorithm

The goal is to find a set of step matrices $\mathcal{S}_v$, defined on the vertices of the mesh, that minimize the error at a target cost, using the continuous error and cost models above. More specifically, the algorithm determines $\mathcal{S}_v$ so that the error is minimized for a given cost, providing the proper trace s_v, and the error is stationary with respect to shape changes, providing the trace-free part $\tilde{\mathcal{S}}_v$.

The conditions above require the derivatives of the error and cost models, defined element-wise with respect to $\mathcal{S}_e$. For these derivatives, the step matrices at the vertices are converted to element step matrices by an arithmetic average over those at the neighboring vertices. If the set of vertices around an element is denoted by V_e, and similarly the set of elements around a vertex is E_v, the relevant formulas are

$$\mathcal{S}_e = \frac{1}{|V_e|} \sum_{v \in V_e} \mathcal{S}_v, \qquad \frac{\partial Q}{\partial \mathcal{S}_v} = \underbrace{\frac{1}{|V_e|}}_{\partial \mathcal{S}_e / \partial \mathcal{S}_v} \sum_{e \in E_v} \frac{\partial Q_e}{\partial \mathcal{S}_e}, \tag{5.14}$$

where $|V_e|$ denotes the number of vertices around element Ω_e. In these formulas, Q is either the error E or the cost C. From these, the linearizations with respect to the

trace s_v and trace-free $\tilde{\mathcal{S}}_v$ are then

$$\frac{\partial Q}{\partial s_v} = \operatorname{tr}\left(\frac{\partial Q}{\partial \mathcal{S}_v}\right), \qquad \frac{\partial Q}{\partial \tilde{\mathcal{S}}_v} = \frac{\partial Q}{\partial \mathcal{S}_v} - \frac{\partial Q}{\partial s_v}\frac{\mathcal{I}}{d}. \tag{5.15}$$

The MOESS algorithm for mesh optimization is described in detail in [56], and a similar procedure will be described later for *hp*-optimization, so only the relevant concepts will be introduced here. Since the linearizations computed as given above are only correct for the current value of the step matrix, the optimization algorithm consists primarily of a main loop that recomputes these linearizations after changes to the step matrices. In order to minimize the overall error and set the trace, MOESS considers the marginal error-to-cost ratio

$$\lambda_v = \frac{\partial E}{\partial s_v} \Big/ \frac{\partial C}{\partial s_v}. \tag{5.16}$$

When this is equally distributed, it implies there is an equal investment in refining any given node. On each iteration of the main loop, δs is added to or subtracted from the trace to reduce the range of λ_v values, then the trace of all step matrices is rescaled so that the total cost matches the target. Finally, the algorithm sets the anisotropy of the element through the trace-free part of $\mathcal{S}_v$ toward the critical point of $\partial E/\partial \tilde{\mathcal{S}}_v$, as

$$\mathcal{S}_v = \mathcal{S}_v + \delta s \frac{\partial E}{\partial \tilde{\mathcal{S}}_v} \Big/ \frac{\partial E}{\partial s_v}.$$

5.2 Output-Based Hanging-Node *hp*-Adaptation

The first approach developed herein for combined mesh and order adaptation draws from the hanging-node-based framework introduced by Ceze and Fidkowski [47], in that it uses hanging-node local mesh mechanics with localized order refinement. In this way the algorithm is fully discrete, rather than fitting parameters in continuous models as done in MOESS. It is also fully decoupled, in the sense that the chosen local refinement option minimizes a locally-defined objective function at every step. The idea is that by locally selecting the optimal refinement for every element, the overall configuration will also tend toward optimality.

5.2.1 Algorithm

The primary loop in the algorithm is over the elements targeted by the element-wise output error indicator given in (3.18), each time deciding whether or not a finer

representation of the approximation space – either by refining the mesh or the order – will lead to a better approximation of the output. The refinement decision for each targeted element is determined through a set of projections to each of the refinement options. More specifically, a value is assigned to each of the options that involves a measure of the error estimate and the additional cost, in terms of the increase in DOFs, incurred by refining the approximation space in that way.

The first step is to determine what additional cost will be acceptable to the computation. As this adaptation algorithm primarily contains a loop over elements, the error indicators are first ranked from high to low error, and processed from this list until one of a set of specified conditions becomes invalidated. Three such conditions are used in this work, namely that:

- The number of elements considered for adaptation must not exceed a fraction f_m^{adapt} of the total number of elements N_e;
- The sum of the error indicators considered must not exceed a fraction f_e^{adapt} of the sum of the elemental error estimates $|\epsilon_e|$ for all elements; and
- The ratio of the new total number of DOF after applying the refinement options to the former number of DOF must not exceed a fraction f_c^{adapt}.

Note that any of these conditions can be effectively removed by increasing the associated f^{adapt}, so that one of the other two conditions will be triggered first.

The projections that define the error estimate for each of the refinement options follow a procedure similar to that used in the MOESS framework. Specifically, the fine-space adjoint is projected to each of the refinement options, and a remaining error is calculated back on the fine space. However, for *hp*-adaptation this reveals a fundamental issue with such a procedure, namely that the fine space for the adjoint calculation is usually the space with order $p+1$, which is itself one of the refinement options. The remaining error calculated on the fine space with the adjoint projected to this option is identically zero due to Galerkin orthogonality. Instead, the method needs to separate the idea of the fine space used for the error estimation and localization, usually $(h, p+1)$, from the even finer space for the *hp* adaptation, denoted for now generically by $(\tilde{h}, \tilde{p})$. This idea is illustrated in Figure 5.4, where the fine space is shown as $(h/2, p+1)$.

To fix the problem of Galerkin orthogonality with the order refinement option, the method requires information about the adjoint from the finer space; simply representing the adjoint on the space is insufficient. The algorithm could solve the adjoint

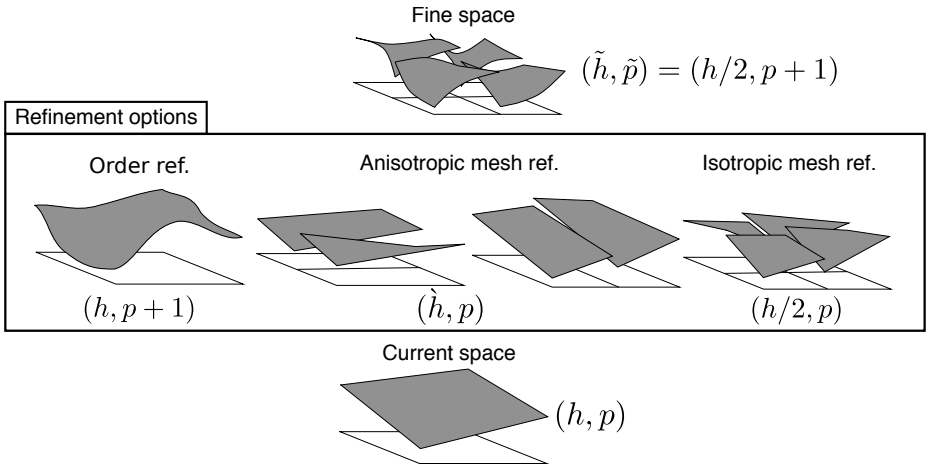


Figure 5.4: Definition of the local refinement options and fine space on each element shown for the *hp*-adaptation algorithm. The fine space shown is only one such option; it could also be defined as $(h, p+2)$.

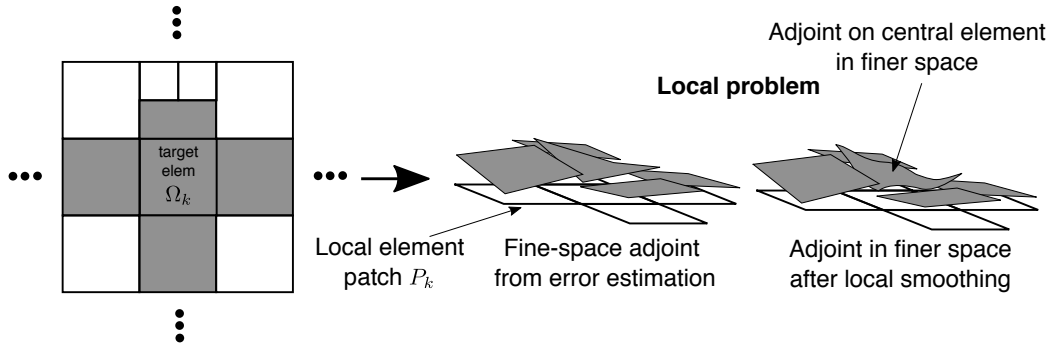


Figure 5.5: Illustration of the local problem and adjoint correction procedure in preparation for the evaluation of the refinement options.

problem globally on the $(\tilde{h}, \tilde{p})$ space, which would be accurate, but often prohibitively expensive. To reduce the cost of this procedure, the method instead restricts itself to only incorporating information from the *local problem* consisting of the target element with its nearest neighboring elements, and defines an *adjoint correction procedure* on these local patches to enhance the accuracy. Figure 5.5 shows the steps involved with such a local problem from a targeted element during the adaptation. In this illustration the $p+1$ fine-space adjoint is projected to the finer space, shown here with order $p+2$, on the targeted element before the adjoint correction procedure. Only the finer-space adjoint on the central element must be projected to the finer-space as shown in Figure 5.5.

The first adjoint correction procedure, a least-squares local reconstruction procedure, has been used in the past for obtaining the fine-space adjoint itself [1]. This

method determines the finer-space adjoint on the target element k, written in a functional sense as $\boldsymbol{\psi}_{\tilde{h},\tilde{p}}|_k$, from the fine-space adjoint on the local patch $\boldsymbol{\psi}_{h,p+1}$ such that the function $E(\boldsymbol{\psi}_{\tilde{h},\tilde{p}}|_k, \boldsymbol{\psi}_{h,p+1})$ is minimized, where

$$E(\mathbf{v}_k, \mathbf{u}) = \sum_{\Omega_e \in P_k} \left(\int_{\Omega_e} (\mathbf{v}_k - \mathbf{u})^2 \, d\Omega + \sum_{i=1}^{d} c_e \int_{\Omega_e} (\partial_i \mathbf{v}_k - \partial_i \mathbf{u})^2 \, d\Omega \right). \tag{5.17}$$

The weights c_e by dimensional reasoning should be $\mathcal{O}(h_e^2)$, where h_e is the diameter of element Ω_e and they are chosen as the edge lengths of the element bounding box. This creates a smooth representation of the function using the additional DOF on element Ω_k to match both the value and the slope of the neighboring fine-space adjoint solutions. When this method is combined with an order-refined finer space, the resulting adjoint converges with an extra order of accuracy for smooth regions. However, this method could perform poorly near singularities in the adjoint, and these are almost certainly some of the locations to be targeted for refinement.

The second adjoint correction procedure developed instead solves for a defect in the adjoint equations as seen on the finer space, and uses this to correct the function. The existing fine-space adjoint on the local patch of elements does not satisfy the discrete finer-space equations

$$\mathbf{R}^{\psi} = \left(\frac{\partial \mathbf{R}_{\tilde{h},\tilde{p}}}{\partial \mathbf{U}^{h,p}_{\tilde{h},\tilde{p}}} \right)^T \boldsymbol{\Psi}^{h,p+1}_{\tilde{h},\tilde{p}} + \left(\frac{\partial J_{\tilde{h},\tilde{p}}}{\partial \mathbf{U}^{h,p}_{\tilde{h},\tilde{p}}} \right)^T \neq \mathbf{0}. \tag{5.18}$$

Note that the output linearization is computed globally, so for example with a boundary output this term is zero everywhere unless the local patch neighbors the boundary. A correction to the adjoint can then be solved so that the resulting adjoint satisfies the finer-space equations on the patch. To do so, the Jacobian matrix resulting from the finer-space equations and the defect $\mathbf{R}^{\psi}$ are restricted to the target element Ω_k. This results in a square system for the DOF on the central element

$$\left(\left. \frac{\partial \mathbf{R}_{\tilde{h},\tilde{p}}}{\partial \mathbf{U}^{h,p}_{\tilde{h},\tilde{p}}} \right|_k \right)^T \Delta \boldsymbol{\Psi} + \mathbf{R}^{\psi}\big|_k = \mathbf{0}. \tag{5.19}$$

The defect $\Delta \boldsymbol{\Psi}$ is then added to the adjoint so that

$$\boldsymbol{\Psi}_{\tilde{h},\tilde{p}} = \boldsymbol{\Psi}^{h,p+1}_{\tilde{h},\tilde{p}} + \Delta \boldsymbol{\Psi}. \tag{5.20}$$

This method incorporates information both from the neighboring fine-space adjoint and the finer-space equations locally. Section 5.2.2 compares this method to a least-squares projection above.

When the resulting finer-space adjoint $\mathbf{\Psi}_{\tilde{h},\tilde{p}}$ is least-squares projected to each of the refinement options i, and then projected back to the finer space as $\tilde{\mathbf{\Psi}}^{\tilde{h},\tilde{p}}_{h_i,p_i}$, some of the added fidelity in the adjoint is lost, leading to an effective adjoint error

$$\mathbf{e}_{i,\psi} = \mathbf{\Psi}_{\tilde{h},\tilde{p}}\Big|_k - \tilde{\mathbf{\Psi}}^{\tilde{h},\tilde{p}}_{h_i,p_i}\Big|_k . \tag{5.21}$$

Note that for consistency the error is only calculated on the target element k and the sub-elements, if any, that are created by the refinement option. While a norm of $\mathbf{e}_\psi$ is an indicator of how well the adjoint is represented by the refinement option, what is really sought is the best-possible representation of the adjoint-weighted residual. Therefore the indicator used here is

$$e_i = \left| \left(\mathbf{\Psi}_{\tilde{h},\tilde{p}}\Big|_k - \tilde{\mathbf{\Psi}}^{\tilde{h},\tilde{p}}_{h_i,p_i}\Big|_k \right)^T \mathbf{R}_{\tilde{h},\tilde{p}}(\mathbf{U}^{h,p}_{\tilde{h},\tilde{p}})\Big|_k \right| . \tag{5.22}$$

However, if this were the only input used to select the refinement option then the method would not work as intended. For example, the method would never select an anisotropic refinement because $\mathbf{e}_i$ would be at least as low for uniform refinement, since the anisotropic refinements are contained in that space. Instead, the method should seek to minimize the error multiplied by a function of the cost, in terms of the increase in DOF, that would result from selecting the option. Recall that the MOESS method used a cost model that was directly proportional to the DOF. To verify that this is a good assumption, consider Figures 4.15 and 4.16 from the HDG comparisons. If the true cost of a computation is the run time, then since both methods have roughly constant slope in the asymptotic region of these figures the output error E is a power of both the cost C and the number of non-zeros N_z, namely $E \sim C^t$ and $E \sim N_z^n$. Then since the degrees of freedom $N \sim N_z$, this gives $C \sim N^{t/n}$ where t and n are the slopes in the figures referenced above. For both methods the slopes t and n are roughly equal, so $C \sim N$ is a good approximation to the cost. Therefore the added cost,

$$c_i = N_i - N_0 = \Delta\text{DOF}, \tag{5.23}$$

is a positive function for each of the refinement options.

This work considers several variants of the objective function, which synthesizes the error and cost calculations as in (5.22) and (5.23). The first objective function

is $g_{1,i} = c_i e_i$, with the goal being to minimize this combination of error and cost. Low-error e_i options, which imply good representation of the adjoint-weighted error, and low-cost c_i options reduce this positive function toward zero. A second option for the objective function is to instead define b_i as the absolute adjoint-weighted residual,

$$b_i = \left(\tilde{\mathbf{\Psi}}^{\tilde{h},\tilde{p}}_{h_i,p_i}\right)^T \mathbf{R}_{\tilde{h},\tilde{p}}(\mathbf{U}^{h,p}_{\tilde{h},\tilde{p}}),$$

and then maximize $g_{2,i} = b_i/c_i$. The idea behind this objective function is that the error e_i might be misleading if in practice the finer-space adjoint is no more accurate. In this case, simply maximizing the error "uncovered" by the error estimate could yield a better result.

The algorithm presented above varies from that developed by Ceze and Fidkowski [47] in a few key ways. Firstly, the present algorithm defines the defect correction procedure for the finer-space adjoint, whereas the algorithm from Ceze and Fidkowski only uses a reconstruction. Secondly, the method computes the error estimates and objective function on the finer-space rather than the coarser space, thereby providing a more consistent framework and defines the error estimate defect e_i that captures more information about effectiveness of the refinement option. Lastly and most importantly, the framework in the code was rewritten and generalized to support both DG and HDG discretizations.

The algorithm in practice defines several heuristic constraints on the refinement, both so that the discretized problem itself remains solvable and because the information given to the refinement algorithm is itself inexact. These are summarized below and addressed where appropriate in the results.

1. Order refinement should be avoided if the solution is under-resolved. When important features in the solution are missed, the algorithm will not be able to correctly distinguish between the order refinement and mesh refinement options, and so mesh refinement should be favored. An example of this occurs initially around discontinuities, where increasing order makes the solution more oscillatory. This work uses an indicator that has previously been used to place artificial diffusion for shock capturing [6]. The smoothness is determined in a scalar $s(\mathbf{u}_h)$ relative to the $p-1$ projection $s_1(\mathbf{u}_h)$ with the function

$$f = \frac{\int_{\Omega_e} (s(\mathbf{u}_h) - s_1(\mathbf{u}_h))^2 \, d\Omega}{\int_{\Omega_e} (s(\mathbf{u}_h))^2 \, d\Omega}.$$

The indicator then detects smoothness, and accordingly allows order refinement, when

$$\frac{f}{f+f_0}\frac{f}{f_0} < \frac{1}{2}, \qquad f_0 = \frac{1}{2}10^{-p}. \tag{5.24}$$

2. If all of the anisotropic mesh refinement options have the objective function within $t^{iso}\%$ of each other, then isotropic refinement should be favored over any of the anisotropic options. This occurs when the adjoint-weighted error is locally isotropic, but the error-to-cost ratio is not well scaled locally.

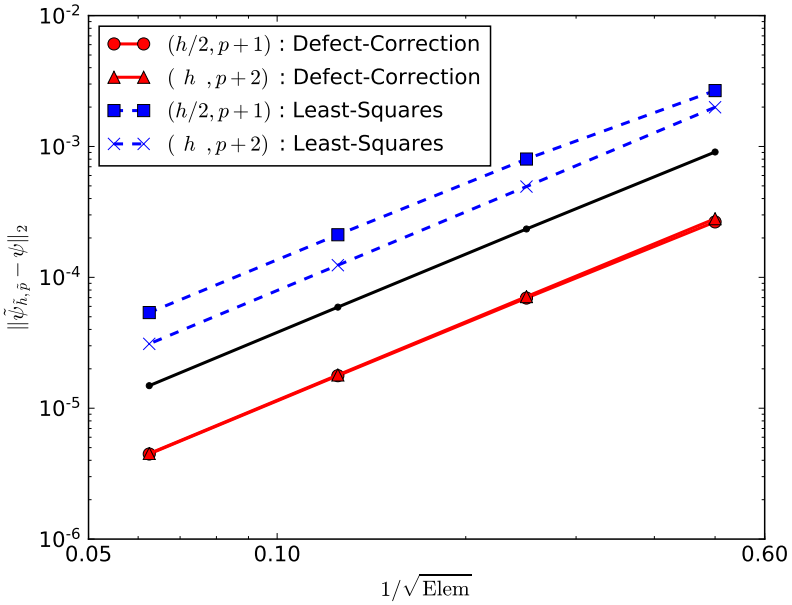
5.2.2 Comparison of Finer-Space Correction Methods

This section compares the different finer space and adjoint correction methods to determine which combination reduces the error most effectively, so that the resulting finer-space adjoint is best approximated. In order to compare these, a scalar linear advection-reaction equation is constructed with a known adjoint solution. Assuming the equation and output $J(u)$ are

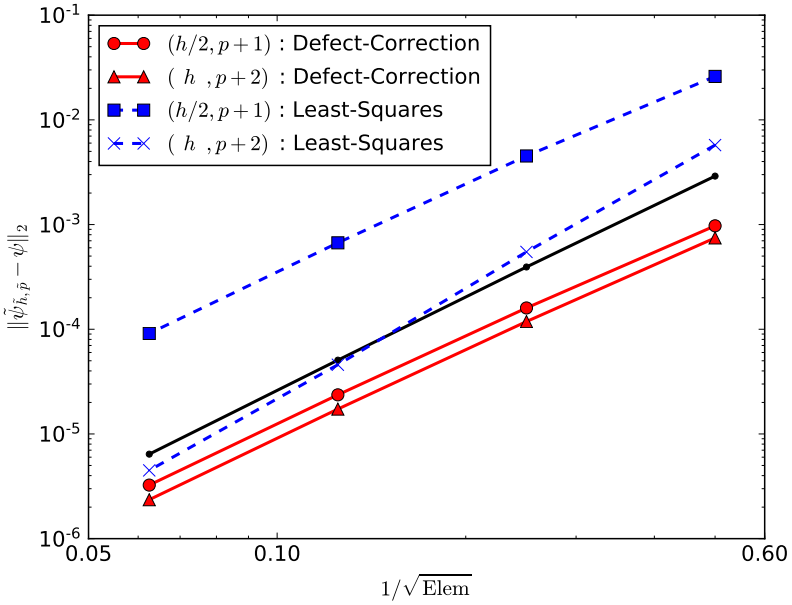
$$\vec{a}\cdot\nabla u + bu = 0 \quad \text{in } [0,1]^2, \qquad J(u) = \int_{\{x=1\}\cup\{y=1\}} j_\Gamma\, u\, ds, \tag{5.25}$$

with $\vec{a} = (1,1)$, then the adjoint equation is $-\vec{a}\cdot\nabla\psi + b\psi = 0$ with the boundary condition $j_\Gamma = -\psi$ on the boundary where the output is defined. If the form for the adjoint $\psi = \exp(r^n)$ is assumed, where $r = \sqrt{x^2+y^2}$, then $b = nr^{n-2}(x+y)$. This gives a convenient platform for defining both a regular solution with $n = 2$, and a solution with a singularity in the equations with $n = 1$.

Figure 5.6 compares the L^2 error of the adjoint corrected using different methods on this set of test cases. For each case $n = 1, 2$, there are two different finer-space definitions: uniformly refining the target element so the sub-elements have diameter $h/2$, or projecting the $p+1$ fine-space adjoint to $p+2$ prior to the correction. For each of these finer-space definitions there are two correction methods to test: defect-correction following (5.20), and least-squares fitting minimizing (5.17). The results show that when the equations are non-singular ($n = 2$), the least-squares fitting increases the order of accuracy, but does not reduce the error significantly, while the defect correction converges at the same rate with lower error. For the singular case ($n = 1$), the least-squares fitting increases the error, possibly due to over-fitting, while the defect-correction, especially with a $p+2$ finer space, reduces the error.



(a) $n = 1$



(b) $n = 2$

Figure 5.6: Test showing the L^2 error of the adjoint after the finer-space correction for the adjoint manufactured solution. The original fine-space adjoint error is shown in black.

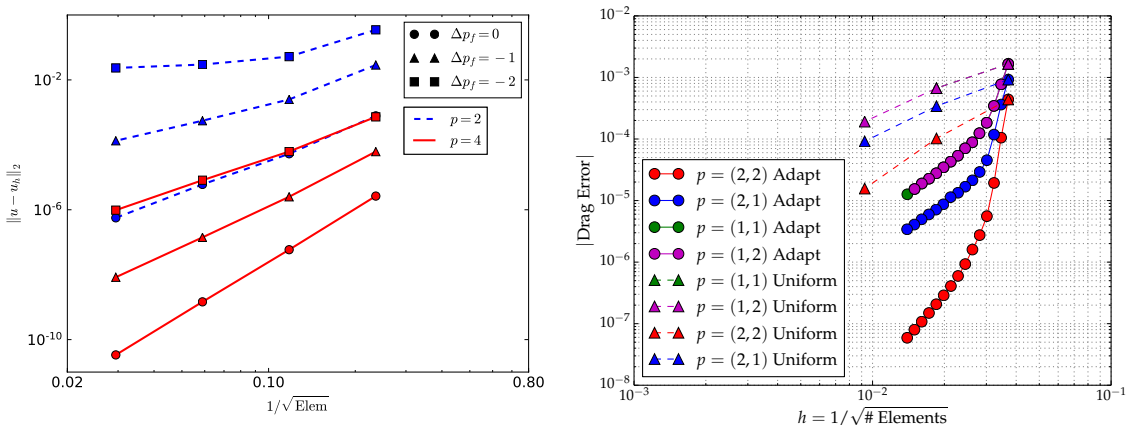
5.2.3 Mesh Mechanics and Implementation Considerations

Certain implementation hurdles and details will be described in this section before transitioning to discuss examples of the method. The method described above in Section 5.2.1 may seem elegant and general, but as with any method, successful algorithmic implementation requires correctly managing even the difficult situations. Additionally, this section will describe aspects of the algorithm that help increase its efficiency.

1. *On hanging-node refinement:* When defining hanging-node refinement such that no face may be cut more than once, it is algorithmically nearly impossible to perform the refinement entirely in parallel. This then affects this *hp*-adaptation algorithm as well, since when applying the mesh refinement options to the local patches, the induced refinement may propagate more than one neighbor away from the target element. Although this refinement could be ignored in the cost calculation, algorithmically this work set out to use only a single hanging-node refinement function. Therefore, creating the patch with each refinement actually consists of the following steps:

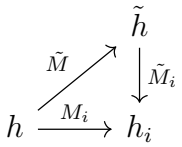
1. Recursively add elements into the local patch $\tilde{P}_k$ until no outer element is on the fine side of a hanging face;
2. Mark the target element k and its nearest neighbors;
3. Refine the target element, possibly inducing refinement throughout $\tilde{P}_k$, and propagate the marker;
4. Remove unmarked elements from the patch $\tilde{P}_k \to P_k$.

2. *On least-squares projections:* When performing the hanging-node refinement, the algorithm keeps track of a map of which element each of the children originated from, and what the element reference-space transformation is from these to the original element. Such a map is exceedingly useful when defining the projections between the finer-space and each of the refinement options, since it removes the need to convert the quadrature point locations to global coordinates, locate the element in which they fall in the other refinement, and convert back to reference space. Instead, the maps from the original patch of elements to both finer space ($\tilde{M}$) and the refinement option (M_i) can be reduced to a single map $\tilde{M}_i$ through a set-theoretical-like operation.



(a) Linear advection-diffusion, domain norm (b) Inviscid NACA 0012, drag output

Figure 5.7: Error convergence with HDG when varying the face order from the order used for the element approximation. The orders are specified as (p_e, p_f) where p_e denotes the element orders, and p_f denotes the face orders.



3. *On HDG compatibility:* The description of the *hp*-adaptation algorithm has largely been discussed in terms of elements, so the question of how to extend such methods to HDG naturally arises. There are at least two points of interest, namely whether the order of approximation for the trace should be allowed to vary from the order used for the neighboring element states, and how the local correction methods should be defined for HDG.

Insight into the first of these questions can be obtained by appealing to error estimates, but it is difficult to get a sense of exactly how his would affect, for example, boundary outputs in nonlinear systems. If the accuracy in the face representation decreases no faster than the efficiency increases, then it would be beneficial in practice. However, if an order of accuracy or more is lost in the output error convergence, then it is not effective. To check this assertion a set of two test cases was run: the first testing the L^2 domain error norm, and the second testing the accuracy of a boundary output on a NACA 0012 airfoil with the Euler equations. The results presented in Figure 5.7 show that both the order of accuracy and the error level suffer when reducing the approximation order. Furthermore, the rate of convergence is essentially determined by $\min(p_e, p_f)$. With this result in mind, for optimal convergence the order of the

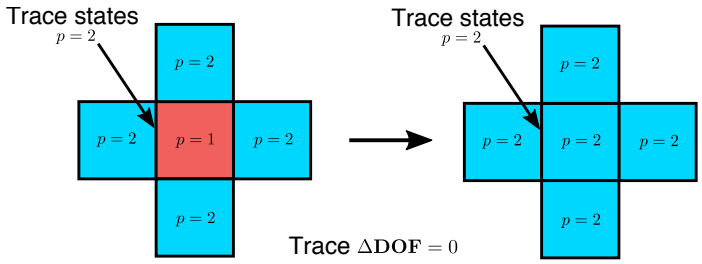


Figure 5.8: A possible local problem for hanging-node anisotropic *hp*-adaption. HDG computes the cost of order refinement here to be zero.

faces should be at least as high as those of the neighboring elements, which simplifies the adaptation process. Therefore, the face orders should be set after determining the orders of the elements.

The error indicators involving the adjoint-weighted error for each refinement option, e_i and b_i, are evaluated element-wise, so this itself is discretization-independent. The least-squares method for incorporating information from the neighboring elements also works in a discretization-independent way, but the defect-correction needs to be redefined for HDG, since the Jacobian matrix is face-based. For this reason, the defect-correction used in this work updates the fine-space adjoint using a DG discretization for the discrete adjoint problem.

A remaining question is how should the cost c_i should be measured for each refinement. The most fair approach to a cost measure for HDG is to count the increase in the number of DOFs for the *global solve*; *i.e.*, in the trace state. However, this can cause the cost to be identically zero for the order option of an island, as for example in the case illustrated in Figure 5.8. This behavior should only occur for cases of order refinement, since any additional faces will always increase the DOFs in the trace. It can be argued that this case should not occur, but this is algorithmically difficult to ensure with order smoothing in the current framework. The approach taken in these cases is therefore to instead automatically select this refinement option, avoiding the need to compute the ill-defined objective function.

5.2.4 Results

The focus of the examples included here is on application of the output-based hanging-node *hp*-adaptation algorithm with DG, where the adaptation serves to reduce the cost of uniformly high-order solutions to DG that are unnecessarily computationally expensive, especially for boundary outputs. Although the focus is on

DG solutions, the frameworks are developed to support HDG as well, and selective comparisons are performed. The intent of these examples is to determine whether such a framework can be trusted to result in a nearly optimal refinement strategy. Therefore, these examples begin with a set of test cases where such an optimal refinement strategy is clear, and then progress toward difficult cases. For each test case, the *hp*-adaptation – and associated anistropic *h*-adaptation by eliminating the option to raise the order – will be compared to isotropic mesh and order refinement.

Past studies presenting hanging-node *hp*-adaptation have not compared against pure order-adaptation, since at some point the solution will almost always become oscillatory and the mesh will need to be refined. However, until this occurs, it provides a bound for the adaptation algorithm, and in some cases the resulting error reduction is the highest possible for a given number of added DOF from the starting mesh. Therefore, if the hanging-node *hp*-adaptation performs nearly as well as a static mesh with *p*-adaptation, then the hanging-node adaptation is nearly optimal.

5.2.4.1 2D Scalar Advection

The first test case uses the first-order scalar advection equation

$$\frac{\vec{a}}{|\vec{a}|} \cdot \nabla u = 0, \quad \text{in } [0,2] \times [0,1], \qquad u(0,y) = 0,\ u(x,0) = \begin{cases} 1 & \frac{1}{8} < x < \frac{3}{4} \\ 0 & \text{otherwise,} \end{cases} \tag{5.26}$$

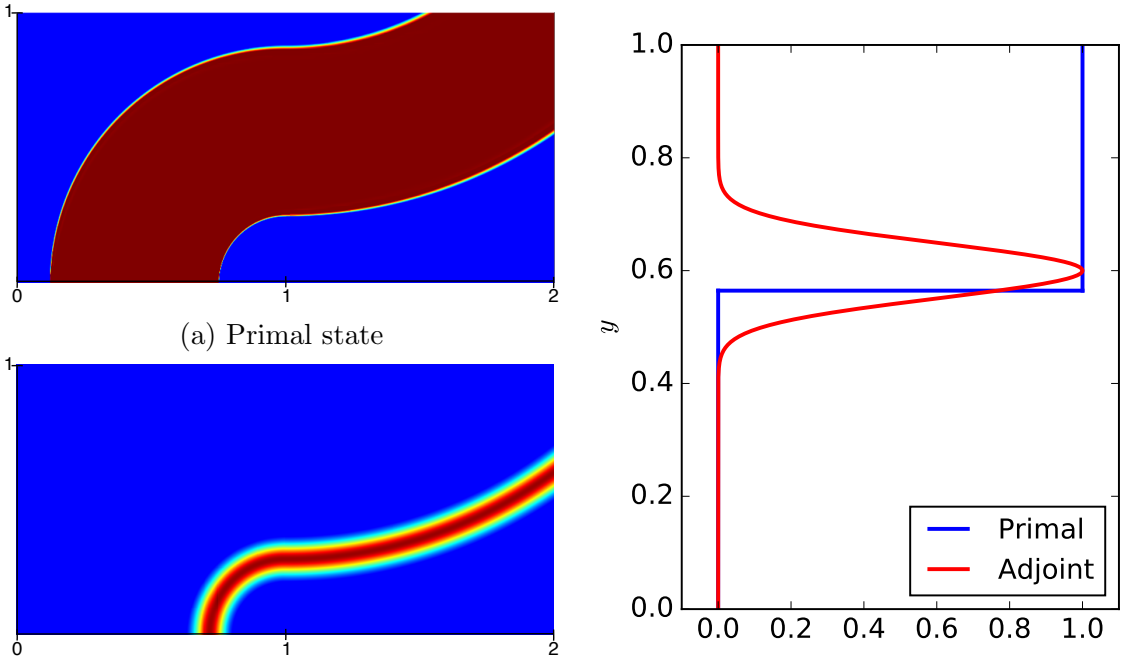
where the advection velocity field is defined piecewise as

$$\vec{a} = \begin{cases} (y, 1-x) & x < 1 \\ (2-y, x-1) & \text{otherwise.} \end{cases} \tag{5.27}$$

A variant of this test case was proposed by Hartmann [66] in 2002. In this equation the step function specified by the bottom boundary condition is advected up and to the right with a velocity that is a varying function of the position in the domain. Although theoretically simple, this is an interesting case for anisotropic refinement because when combined with a structured grid, the velocity at some locations will be aligned with the grid and at others will be unaligned.

The adaptation algorithm defined in this section is applied for the output

$$J(u) = \int_0^1 j(y)\, u(2,y)\, dy, \quad j(y) = \frac{1}{2} \exp\left(-s\left(y - z - \frac{1}{2}\right)^2\right) (1 - \cos(2\pi(y-z))) \tag{5.28}$$



(a) Primal state

(b) Adjoint state

(c) Boundary profile at $x = 2$

Figure 5.9: Setup for the scalar advection test case. In the primal state the $u = 1$ profile propagates up and to the right, and the adjoint traces back from the location of the output defined on the right boundary with the opposite advection velocity.

with the steepness factor $s = 200$ and offset $z = 0.1$. The adjoint operator reverses the direction of the velocity field, so the adjoint for the boundary output traces back along the velocity field from the right boundary to the bottom where the boundary condition for the primal was set. Note in Figure 5.9 that, by construction, the non-zero values of the adjoint on the right boundary coincide with the sharp interface in the exact primal solution. Therefore, there is a competing need to resolve the features with both mesh resolution and high-order polynomial representation. The adaptive cases begin by solving with DG on a uniform 20×10 quadrilateral mesh with $p = 1$, and on every step of the adaptation the following parameters are used unless otherwise specified:

- Adaptation parameters $f_m^{\text{adapt}} = 0.1,\ f_m^{\text{adapt}} = 0.6,\ f_c^{\text{adapt}} = 1.2$
- Defect-correction for the finer-space solve.

The main results from this set of test cases are presented in Figure 5.10 through Figure 5.14. Figure 5.10 compares the anisotropic-h and anisotropic-hp adaptation using two different objective functions described Section 5.2.1. The trend in the results, which could be specific to this case, shows that using the objective function

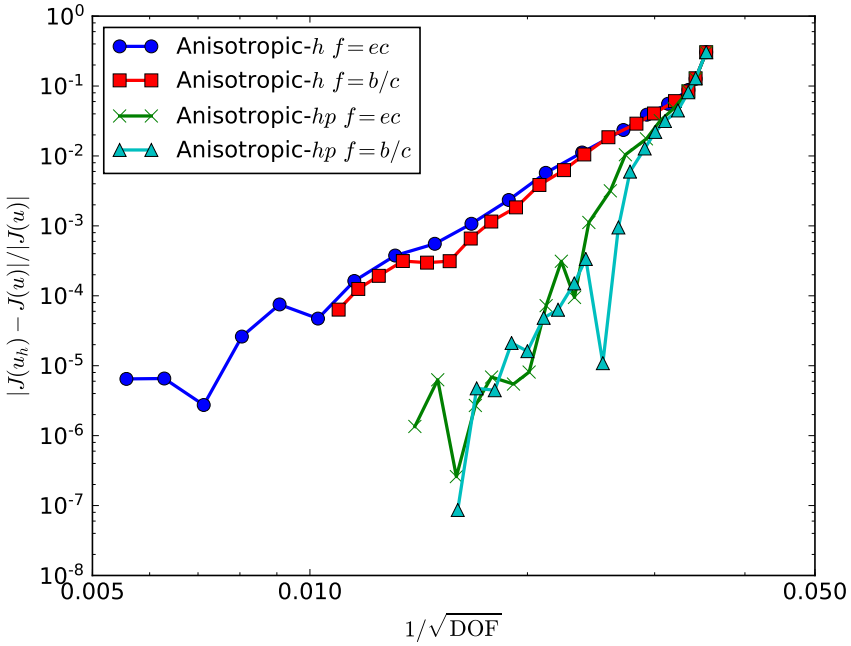
based on the adjoint-weighted residual $f = b/c$, instead of the difference between the finer-space and projected adjoints $f = ec$, yields better refinement patterns. Although the resulting errors are similar, the resulting mesh and order field is qualitatively better for the $f = b/c$ function, since it contains more anisotropic mesh refinements and higher-order elements.

Figure 5.11 shows the convergence of the output using different refinement techniques, where it is apparent that the *hp*-adaptation method, using $f = b/c$, clearly performs best. The order-only adaptation results indicate that this is an insufficient mesh resolution for this type of adaptation. Yet order refinement is very efficient in reducing the error in this test, since the resulting mesh and orders from *hp*-adaptation in Figure 5.10 display considerable order refinement. The key to refining efficiently, which is evident from these results, is to provide enough mesh resolution so the solution is not under-resolved, then increase the order. The final *hp*-adaptation mesh shows high orders, up to $p = 6$, near the peak of the adjoint, and mesh refinement near the tails in the adjoint profile.

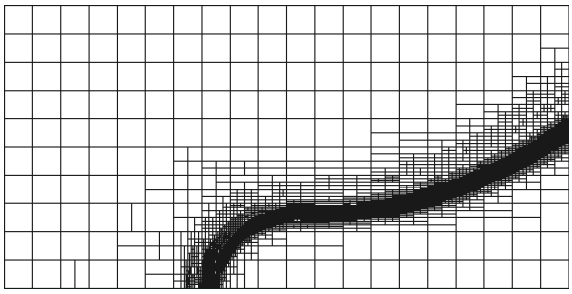
The final mesh from isotropic-h in Figure 5.12 displays horizontal refinement near the bottom boundary, but this is induced refinement from the single-cut restriction imposed by the mesh mechanics. This induced refinement is removed almost entirely in the final anisotropic refinement mesh shown in Figure 5.10, which correctly aligns the anisotropy of the refinement.

The *hp*-adaptation method is performing better, but additional insight can be gained by considering how and when the refinement is occurring. Figure 5.13 shows that the initial refinement is more mesh-based, mostly anisotropic refinement, then after a few adaptive iterations, the majority of the refinement raises the order. Additionally, the element-wise percentage of the different refinement types correlates strongly with the error fraction refinement by each of these types. That is, no single type of refinement more predominantly targets the elements that are contributing most to the error.

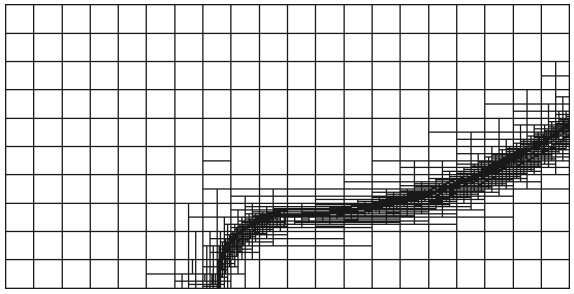
The adaptation results using HDG, shown in Figure 5.14, are notably different from those using DG in Figure 5.10. First and foremost, the results using the objective function form $f = b/c$ display odd refinement behavior that is often perpendicular to the direction of the propagation. This behavior is largely absent when using the other objective function, $f = ec$, but excess mesh adaptation far from the adjoint peak still affects the convergence with DOF. This could be due to the difference in error modes between DG and HDG; specifically, since HDG couples across faces, in effect the stencil affecting the solution at any point is larger.



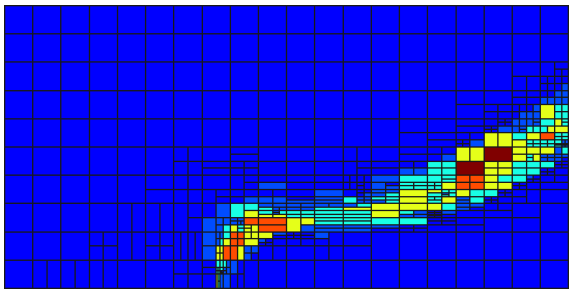
(a) Error convergence with refinement



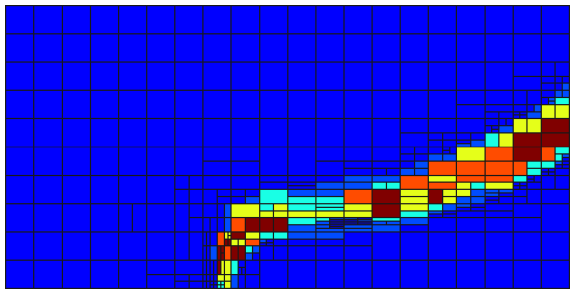
(b) Anisotropic h, $f = ec$



(c) Anisotropic h, $f = b/c$



(d) Anisotropic hp, $f = ec$



(e) Anisotropic hp, $f = b/c$

Figure 5.10: Convergence comparison of the anisotropic hanging-node adaptation using DG with different objective functions for the scalar advection test case. Final meshes are shown with order fields for hp results.

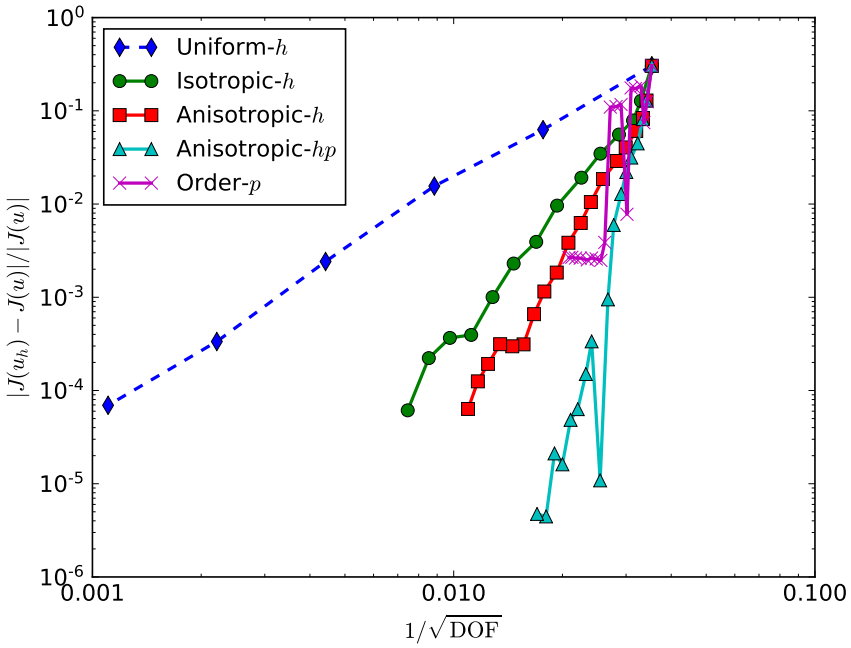
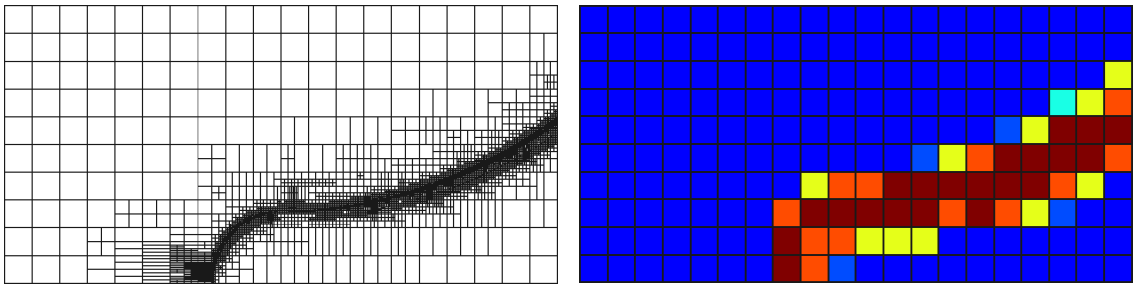
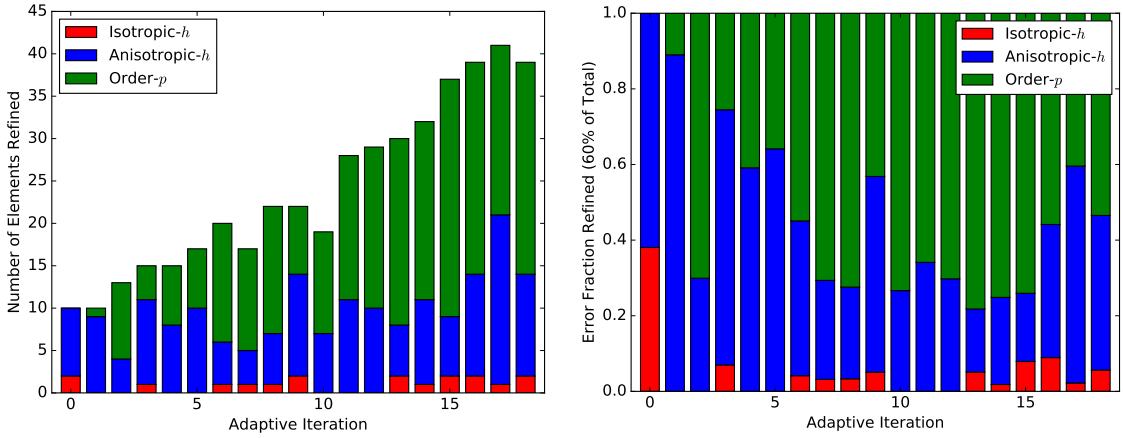


Figure 5.11: Convergence with degrees of freedom using different refinement strategies on the scalar advection test case.



(a) Isotropic-h (b) Order-p ($p = 1$: blue, $\ldots$ 6 : red)

Figure 5.12: Final meshes for isotropic mesh and order refinement for the scalar advection test case.



(a) Number of elements

(b) Fraction of error

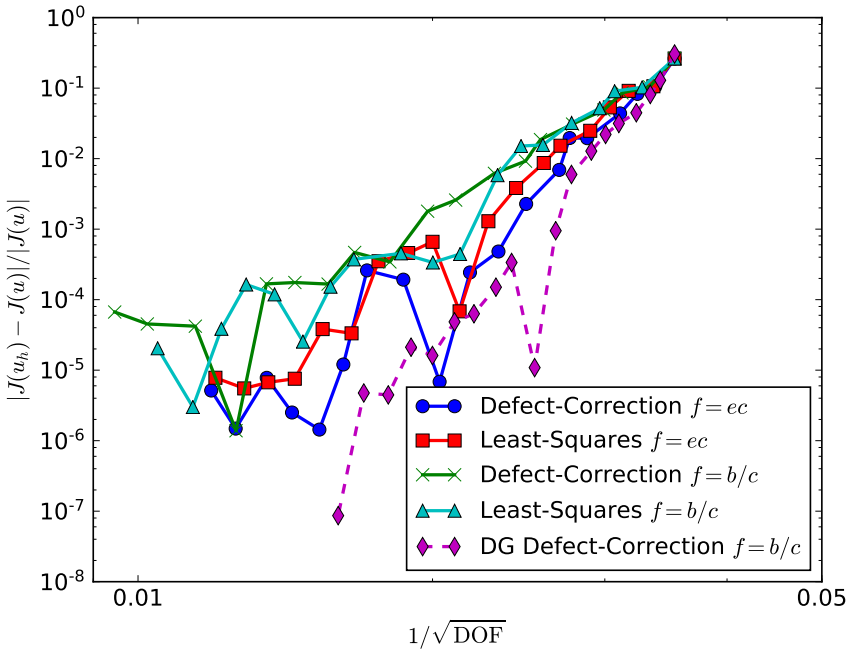
Figure 5.13: Refinement option chosen for each adaptive step of the *hp*-adaptation using $f = b/c$ with the scalar advection test case. The algorithm first refines the mesh, but after only a few refinements switches to refining element orders.

5.2.4.2 NACA 0012 Test Cases

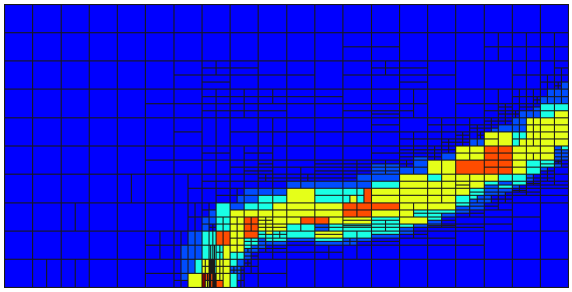
This section applies the output-based hanging-node *hp*-adaptation algorithm to test cases involving the NACA 0012 airfoil, which was also used for tests in the previous chapter. Both inviscid and viscous test cases are considered.

1. Inviscid: The first case applies adaptation to the drag output on the airfoil with the inviscid Navier-Stokes equations and initial mesh defined in Section 4.5.2.1. The main results are presented in Figure 5.15 through Figure 5.18. The solution is smooth except for a singularity at the trailing edge of the airfoil, so order adaptation is again very efficient in reducing the error, as evident from Figure 5.15. This figure shows that, for the same level of error, the anisotropic- h and hp adaptation methods reduce the cost from 12360 DOF for isotropic-h to only 2087 DOF for hp-adaptation. The meshes corresponding to each of the labeled results on the error plot are displayed in Figure 5.16. This highlights an important result. Specifically, despite the anisotropic-hp method refining both the mesh, largely around the trailing edge singularity, and the order, the error reduction in drag relative to initial solution is nearly identical to that with order-only adaptation.

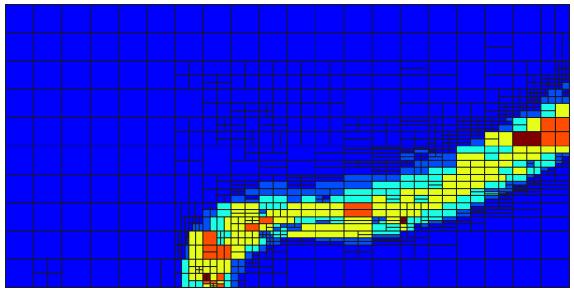
2. Viscous: The next case test considers the NACA 0012 airfoil with viscous Navier-Stokes at the conditions in Section 4.5.2.2. In this case a boundary layer exists on the airfoil, and a small recirculation region is present near the trailing edge. Near both of these flow phenomena the solution should be refined to accurately resolve the drag



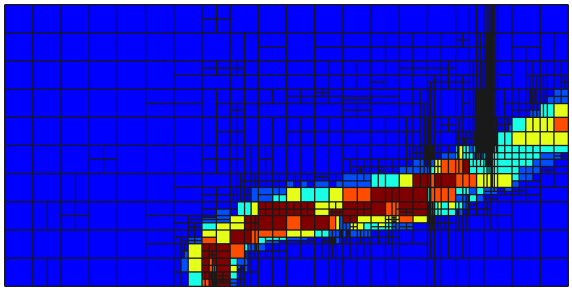
(a) Error convergence with refinement



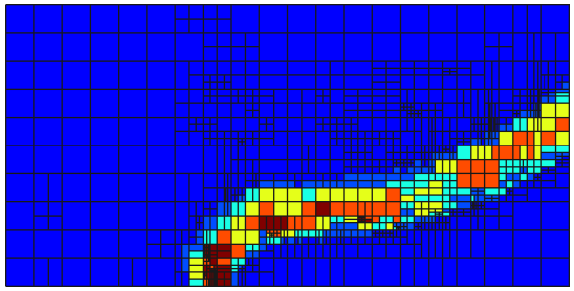
(b) Defect-correction, $f = ec$



(c) Least-squares, $f = ec$



(d) Defect-correction, $f = b/c$



(e) Least-squares, $f = b/c$

Figure 5.14: *hp*-adaptive HDG results for scalar advection test case, using both finer space solution types, and both objective functions. The result is opposite to DG: the refinement is qualitatively better with $f = ec$.

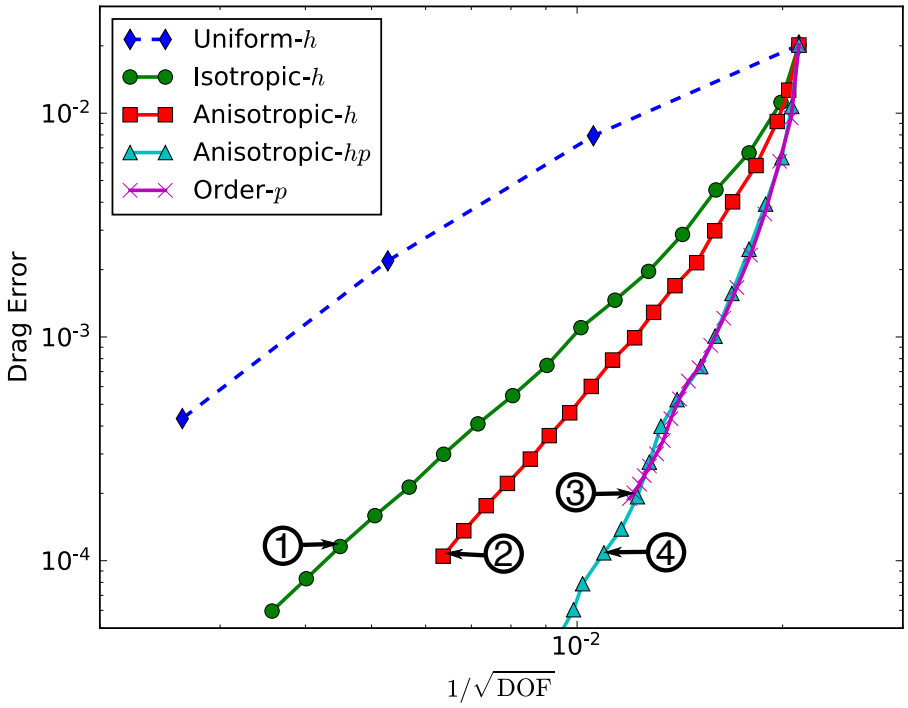
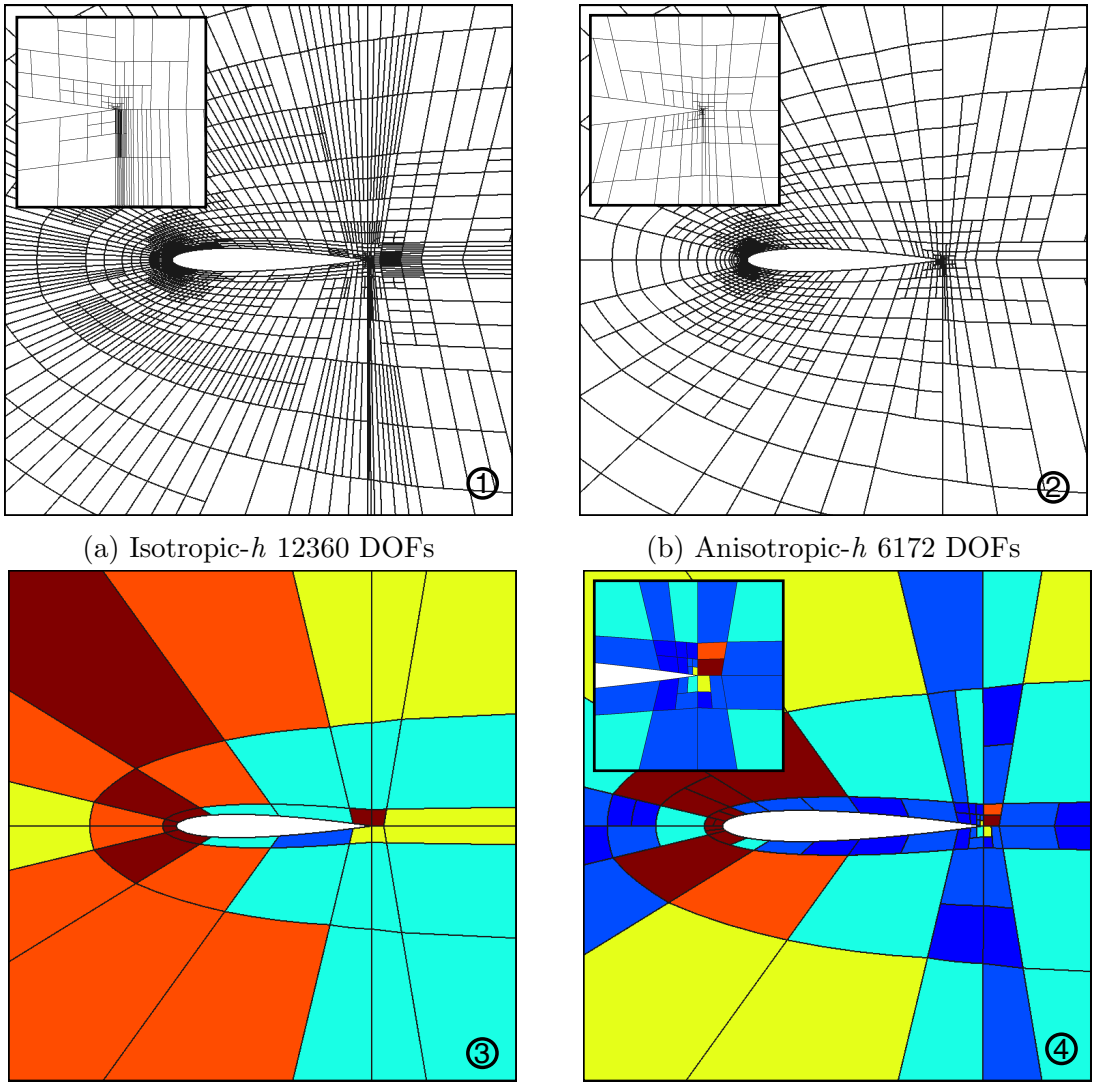


Figure 5.15: Drag convergence with degrees of freedom (DOF) using different refinement strategies on the inviscid NACA 0012 test case. The numbers correspond to meshes in Figure 5.16.



(a) Isotropic-h 12360 DOFs

(b) Anisotropic-h 6172 DOFs

(c) Order-p 1841 DOFs

(d) Anisotropic-hp 2087 DOFs

Figure 5.16: Refined meshes and elemental orders, where applicable, for the inviscid NACA 0012 test case. In Figures (c) and (d), the orders are shown as colors $p = 1$: blue $\ldots$ 6 : red. The numbers correspond to the data locations in Figure 5.15.

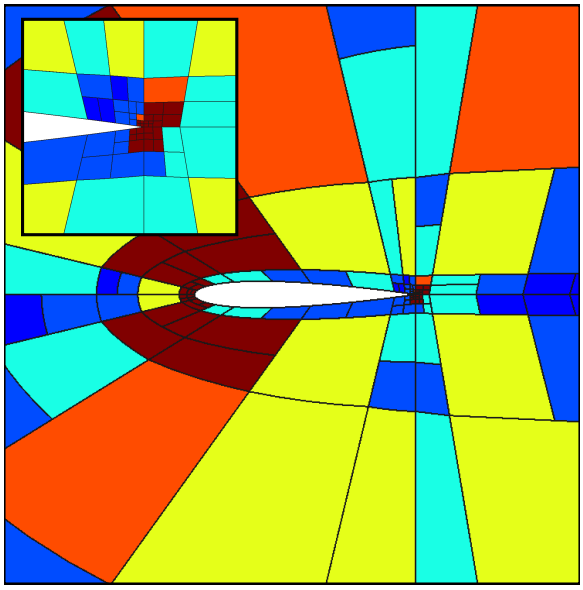
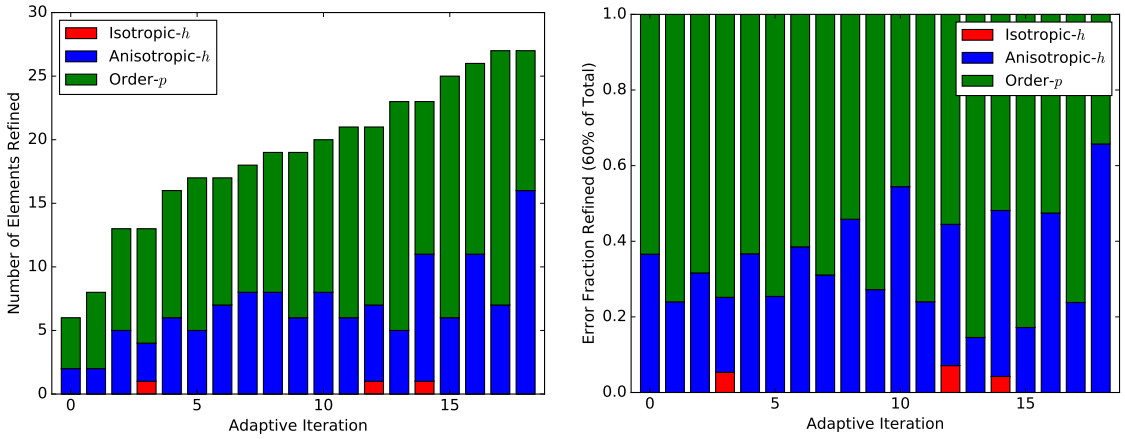


Figure 5.17: Mesh and elemental orders for the *hp*-adaptation algorithm after the full 20 iterations.



(a) Number of elements

(b) Fraction of error

Figure 5.18: Refinement option chosen for each adaptive step of the *hp*-adaptation on the inviscid NACA 0012 test case. The algorithm first refines the mesh, but after only a few refinements switches to refining element orders.

on the airfoil, but the question remains as to what combination of mesh and order refinement best resolves these features.

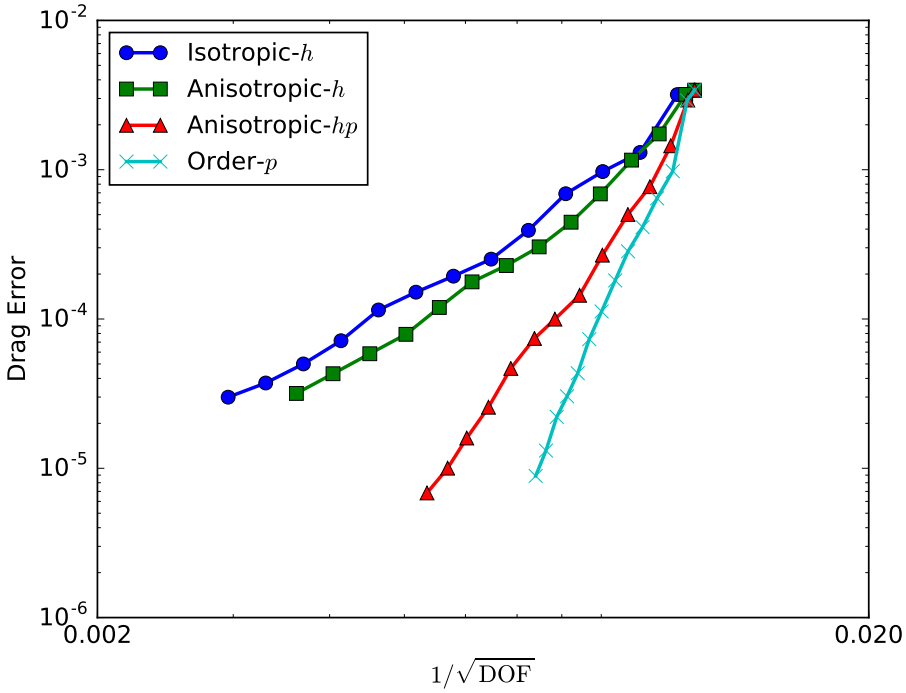
The main results are presented in Figure 5.15 and Figure 5.20. In Figure 5.15, the error convergence with number of degrees of freedom shows that order adaptation just outside the viscous sublayer in the boundary layer appears to perform best, even over a combination of hanging-node mesh refinement and order adaptation, seen in Figure 5.20. Perhaps hanging-node refinement in the boundary layer would be more effective if, for instance, the initial mesh were coarser. However, this level of mesh resolution was required to converge with $p = 1$ initially, and the goal of the test was to check whether, regardless of mesh resolution, the correct refinement would be chosen.

Note also in Figure 5.15 that in this test case the objective function again affects the convergence. Using the form $f = b/c$ without taking the differences of the adjoint reduces the convergence to nearly that of anisotropic mesh adaptation alone. Interestingly, the order refinement in the boundary layer using $f = b/c$ matches more closely to that of order adaptation than with the $f = ec$ objective function, yet the convergence is far poorer. This indicates that there might not be a single universal optimal *hp*-refinement approach in such features, and reaffirms that how the refinement is done in these regions has a large impact on the effectiveness of the method.

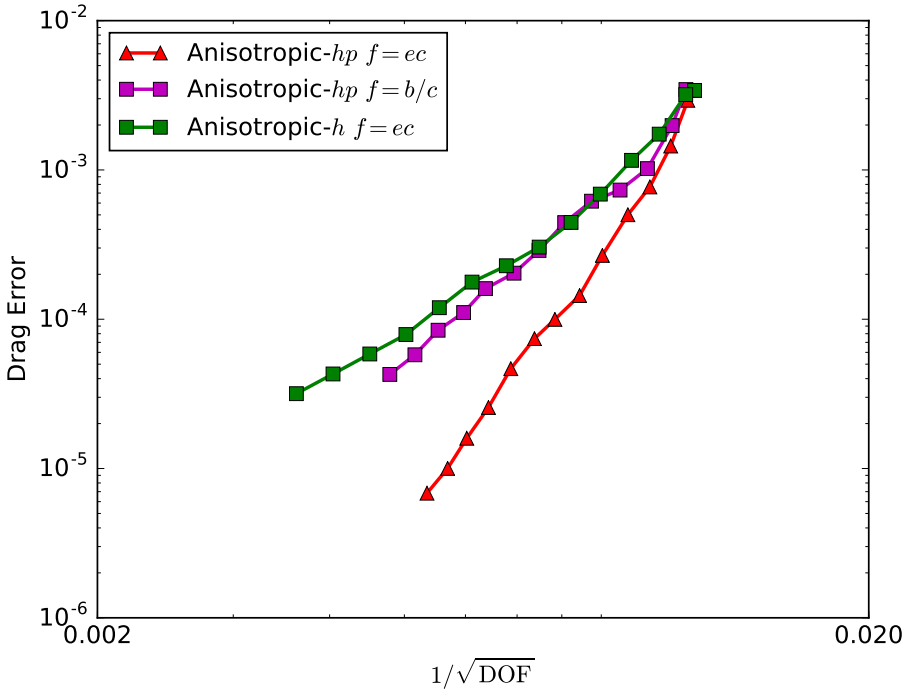
5.2.4.3 Review

The results above show that the anisotropic *hp*-adaptation framework is a robust method that can be widely applied to cases using quadrilateral and hexahedral meshes. The framework performs as well as pure order refinement in the regions of high error for cases with only advection present. Even so, there is some ambiguity in defining the objective function, and the choice can affect results. Overall, minimizing the object function $f = ec$ yields better results, as it incorporates a measure of how well the projected adjoint is represented by the refinement option. Results for the NACA 0012 airfoil with viscosity indicated that the adaptation was too constrained by the initial mesh, so even when mesh refinement was preferred, it could not be placed in a way that reduced the error most effectively.

The *hp*-optimization method discussed next generates unstructured simplex meshes and globally remeshes at each step. Therefore, it has the ability to more effectively place mesh resolution.

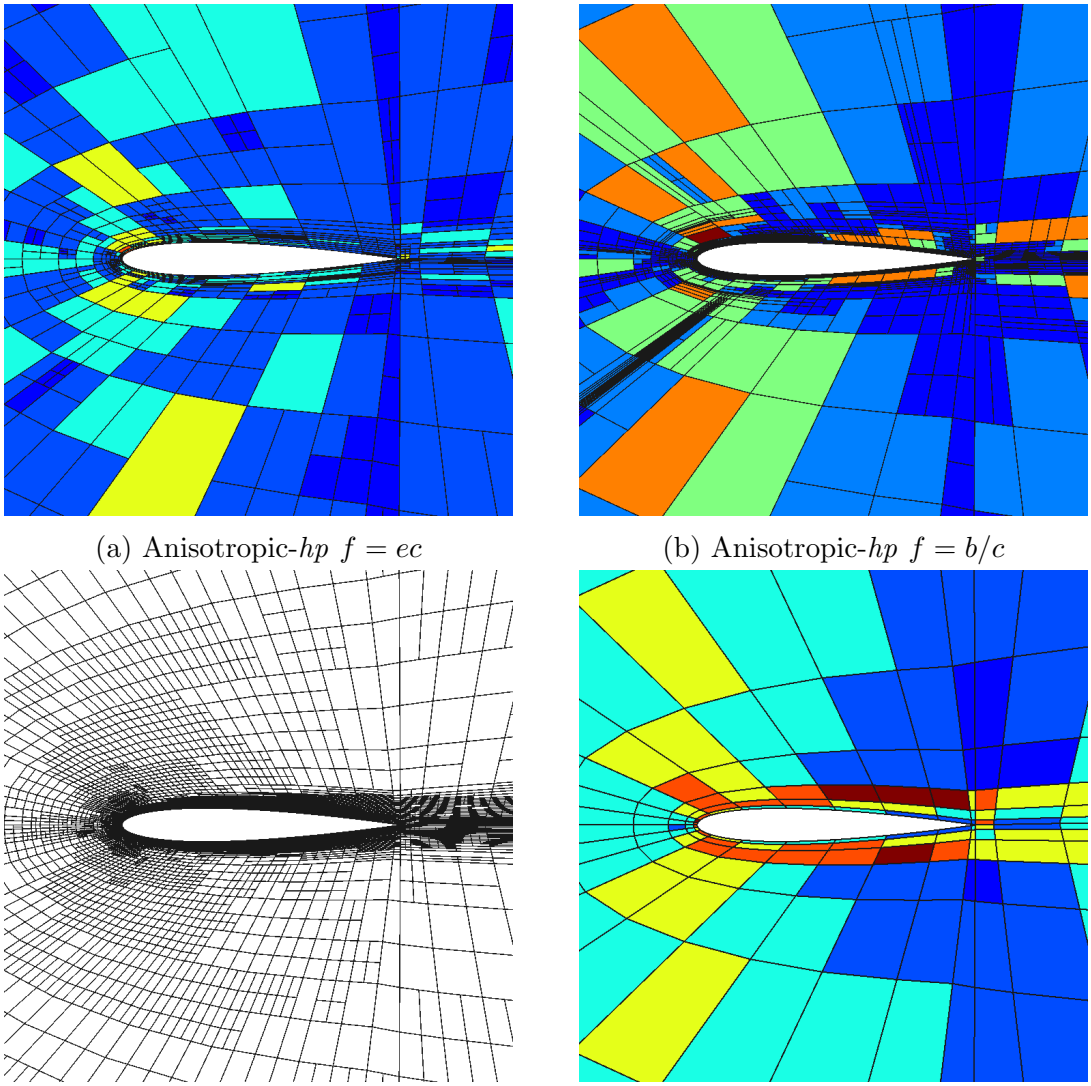


(a) Comparing all strategies



(b) Comparing objective functions

Figure 5.19: Drag convergence with degrees of freedom (DOF) using different refinement strategies on the viscous NACA 0012 test case. In figure (a), the anisotropic strategies use the $f = ec$ objective function.



(a) Anisotropic-hp $f = ec$

(b) Anisotropic-hp $f = b/c$

(c) Anisotropic-h

(d) Order-p

Figure 5.20: Refined meshes and elemental orders, where applicable, for the viscous NACA 0012 test case. In Figures (a) and (c), the orders are shown as colors $p = 1$: blue $\ldots$ 6 : red.

5.3 Output-Based *hp*-Optimization

The method described in this section is an effort to extend the MOESS framework from mesh optimization, described in Section 5.1.3, to full *hp*-optimization. It does this by optimizing both the metric field defining the mesh and a scalar order function over the domain, $p(\vec{x})$, that defines the order at every point. The volume average of $p(\vec{x})$ over elements of the new mesh defines the solution approximation orders for the next iteration of the adaptive process. The error and cost models, in addition to the optimization algorithm itself, need to be changed for this method in order to accommodate this additional field. This section details the necessary development and the rationale behind the choices made to define these methods, and then demonstrates the method on a set of test cases.

The order function is represented in a similar way as the metric field for the mesh, namely defined at vertices and linearly interpolated to the elements as needed, thereby restricting the function to be continuous over the domain. Note that this definition does not, in theory, preclude large order differences between neighboring elements, since $p(\vec{x})$ may rise sharply over an element. Then, similar to how the MOESS algorithm works with changes to the metric field through the metric step, the order field is split as

$$p(\vec{x}) = p_0(\vec{x}) + q(\vec{x}) \tag{5.29}$$

so that the order step q_v, defined on the vertices, will be the working variable. The initial order field, defined at vertices $p_{0,v}$ is determined by averaging the integer orders on neighboring elements p_e as

$$p_{0,v} = \frac{1}{|E_v|} \sum_{e \in E_v} p_e. \tag{5.30}$$

When interpolating back to the elements, a reverse operation is performed, averaging the total order, or order step, at vertices

$$p_e = \frac{1}{|V_e|} \sum_{v \in V_e} p_{0,v} + q_v, \qquad q_e = \frac{1}{|V_e|} \sum_{v \in V_e} q_v. \tag{5.31}$$

Note that this definition does not in general match the initial integer value when $q_v = 0$ since it incorporates orders on neighboring elements.

5.3.1 Cost Model

The MOESS framework for mesh adaptation sets the cost model with the result of an integral over the original element of a cost density, with the result that the cost is the number of DOF for the element times a function of the metric step matrix trace (5.13). Implicitly this result simplifies the calculation, avoiding explicit quadrature, by assuming that the metric step matrix is constant over the element. The integral expression, from Yano [55], is

$$C_e(\mathcal{S}) = \int_{\Omega_e} c_p \sqrt{\det \mathcal{M}(\vec{x})}\, d\Omega.$$

When using the approximation that $\mathcal{S} = \mathcal{S}_e$ is constant over the element, this has the explicitly computable result

$$\begin{aligned} C_e(\mathcal{S}_e) &= \int_{\Omega_e} c_p \sqrt{\det\left(\mathcal{M}_{0,e}^{1/2} \exp \mathcal{S}_e \mathcal{M}_{0,e}^{1/2}\right)}\, d\Omega \\ &= \int_{\Omega_e} c_p \sqrt{\det \mathcal{M}_{0,e}} \sqrt{\exp(\mathrm{tr}(\mathcal{S}_{0,e}))}\, d\Omega \\ &= \underbrace{c_p \sqrt{\det \mathcal{M}_{0,e}}}_{N_p} \exp\left(\frac{1}{2}\mathrm{tr}\left(\mathcal{S}_e\right)\right), \end{aligned}$$

where the cost density c_p times the element area is the number of DOF, denoted by N_p. In the same manner, if the order field $p(\vec{x})$ is assumed to be a constant for the element – equal to p_e determined by (5.31) above – then the integration can again be done explicitly, with the result

$$C_e(\mathcal{S}_e, p_e) = \mathcal{N}(p_e) \exp\left(\frac{1}{2}\mathrm{tr}\left(\mathcal{S}_e\right)\right). \tag{5.32}$$

The function $\mathcal{N} : \mathbb{R} \to \mathbb{R}$ is a real-valued extension of the integer polynomial dimension calculation; *e.g.* , for a two-dimensional full-order basis $\mathcal{N}(p) = 0.5(p+1)(p+2)$.

The algorithm requires the linearization of this model with respect to the MOESS unknowns

$$\begin{aligned} \frac{\partial C_e}{\partial \mathcal{S}_e} &= \frac{1}{2} C_e \mathcal{I} \\ \frac{\partial C_e}{\partial q_e} &= \frac{\partial C_e}{\partial p_e} = \frac{\partial \mathcal{N}}{\partial p_e} \frac{C_e}{\mathcal{N}(p_e)}. \end{aligned} \tag{5.33}$$

5.3.2 Error Model

Recall that the MOESS error model for mesh optimization is motivated by the isotropic estimate between two different element diameters h and H, in the form $(h/H)^r$ with an unknown r. The first step to adding order dependence is determining the correct functional form to use, since the form above for isotropic dependence only included the mesh size. The models considered here add a direct dependence on the order, similar to the general one-dimensional error estimate in (5.4), as

$$\frac{E}{E_0} = \left(\frac{h}{H}\right)^{r(p)} \left(\frac{p}{p_0}\right)^m,$$

where m is an additional unknown. It can be the case that $p_0 = 0$, so to keep the error calculation well-defined, here the error estimates are instead based on the form

$$\frac{E}{E_0} = \left(\frac{h}{H}\right)^{r(p)} \left(1 + \frac{q}{p_0 + 1}\right)^m.$$

In tensor notation, the error models considered in this work both take the form

$$E_e(\mathcal{S}_e, q_e) = \epsilon_e \left(1 + \frac{q_e}{p_{0,e} + 1}\right)^m \exp\left(\operatorname{tr}\left(\left(\mathcal{R}_e + \frac{q_e}{p_{0,e} + 1}\mathcal{L}_e\right)\mathcal{S}_e\right)\right) \tag{5.34}$$

and differ by the definition of $\mathcal{L}_e$. The forms differ by the number of unknown parameters in the rate tensor $\mathcal{L}_e$:

Isotropic	$\mathcal{L}_e^{\text{iso}}$	$=$	$\ell_e \mathcal{I}$
Anisotropic	$\mathcal{L}_e^{\text{aniso}}$	$\in$	Sym_d.

The isotropic model in total adds two additional unknowns, m and ℓ, while the anisotropic model adds $1 + d(d+1)/2$ unknowns, m and the symmetric matrix $\mathcal{L}_e$. The results in this work use the isotropic model, and the comparison to the anisotropic model is left as future work.

Recall that the error model works on the remaining error defined in (5.11). The form of the error model is consistent with this functional form of the remaining error because when $\mathcal{S}_e = 0$, the adjoint-weighted error estimate $\Delta\epsilon_{e,0} = 0$ from Galerkin orthogonality. The models (5.34) ensure this property when both $\mathcal{S}_e$ and q_e are zero.

The algorithm requires the linearization of this model with respect to the MOESS

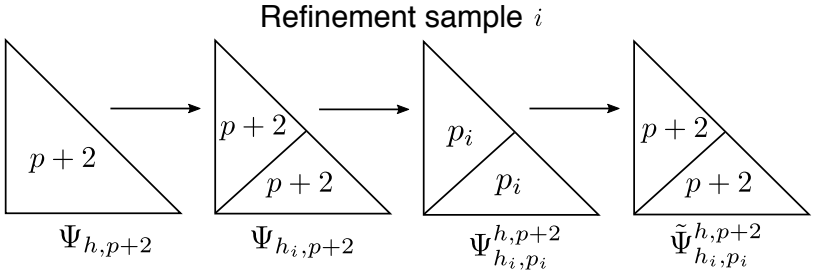


Figure 5.21: Fine-space adjoint projection procedure for a MOESS-P refinement sample.

unknowns

$$\begin{aligned}\frac{\partial E_e}{\partial \mathcal{S}_e} &= E_e\left(\mathcal{R}_e + \frac{q_e}{p_0+1}\mathcal{L}_e\right)\\ \frac{\partial E_e}{\partial q_e} &= E_e\left(\frac{\operatorname{tr}(\mathcal{L}_e\mathcal{S}_e)}{p_0+1} + \frac{m}{p_0+q_e+1}\right).\end{aligned} \tag{5.35}$$

5.3.3 Sampling and Model Fitting

The sampling procedure to create the data points from which to fit the error model is determined similarly to both the hanging-node *hp* method, and in the context of this method follows from the work of Fidkowski for mesh refinement [56]. The same obstacle exists with this method, namely that one of the refinement options corresponds with the usual fine space definition for the adjoint. To overcome this issue, the work presented here uses the $p+2$ fine-space adjoint instead of a local defect-correction or least-squares enrichment of the $p+1$ fine-space adjoint as the hanging-node-based adaptation used. This decision was made to minimize the number of potential sources of error when developing the method.

Figure 5.21 shows each of the steps involved in the L^2 projection procedure for each of the fine-space adjoint samples. For mesh-only, this resulted in $n_{\text{samp}} = 4$ samples, not counting the original coarse-space projection (h,p) that gave $\Delta\epsilon_{e,0} = 0$ due to Galerkin orthogonality. Projecting to order p and $p+1$ creates $n_{\text{samp}} = 9$ samples $\{\mathcal{S}_{ei}, q_{ei}, \Delta\epsilon_{e,i}\}$. Note that for each sample i, $q_{e,i}$ is either 0, for projections to order p, or 1, for projections to order $p+1$, since these are relative to the original order.

A least-squares problem is then solved to determine the unknown error model parameters $\mathcal{R}_e$, and $\mathcal{L}_e$, and m minimizing

$$\sum_i^{n_{\text{samp}}} \left(\log\frac{E_e}{\epsilon_e} - m\log\left(1+\frac{q_{e,i}}{p_{0,e}+1}\right) - \operatorname{tr}\left(\left(\mathcal{R}_e + \frac{q_{e,i}}{p_{0,e}+1}\mathcal{L}_e\right)\mathcal{S}_{e,i}\right)\right)^2.$$

The isotropic and anisotropic p-dependent error models have 5 and 7 parameters, respectively, so these are uniquely defined. The normal form of the resulting least-squares matrix will consist of blocks for each of the unknown parameters, resulting in a 3×3 block system.

5.3.4 Algorithm

When extending the MOESS framework to include an order field, the primary stages of the optimization algorithm remain the same, but how these are formulated differ. In fact, since much of the algorithm at the top-level is general, the set of variables $\{W_v\}$ will be used throughout to refer to all MOESS step variables, both metric and order, collectively. This section details the approach taken, and refers back to the outline of the mesh optimization algorithm in Section 5.1.3.3.

The objective of the optimization algorithm is to enforce a set of optimality conditions that minimize the error subject to a constrained total cost. Mathematically, the goal is to find the set of variables that minimizes the sum of the error model over the elements, namely

$$\begin{aligned} \{W_v\}^* = \arg\inf_{\{W_v\}} E(W_v) \qquad & \textit{Minimization} \\ \text{such that } \textstyle\sum_{\Omega_e \in T_h} C_e(S_e, q_e) = C_{\text{tgt}} \qquad & \textit{Constraint 1} \\ |\mathcal{S}_v|_{ij} \leq \mathcal{S}_{\max} \qquad & \textit{Constraint 2} \\ |q_v| \leq q_{\max} \qquad & \textit{Constraint 3} \end{aligned} \tag{5.36}$$

where the total error E and cost C are sums of the local error and cost models,

$$\begin{aligned} E(W_v) &= \sum_{\Omega_e \in T_h} E_e(\mathcal{S}_e(\mathcal{S}_v), q_e(q_v)) \\ C(W_v) &= \sum_{\Omega_e \in T_h} C_e(\mathcal{S}_e(\mathcal{S}_v), q_e(q_v)). \end{aligned}$$

This is an intuitive extension from the original definition of the method by Yano and Darmofal in [55]. The minimization above involves the marginal error-to-cost ratio, defined in (5.16) for the metric step trace, for both the metric step trace s_v and order step q_v as

$$\lambda_v^s = \frac{\partial E}{\partial s_v} \Big/ \frac{\partial C}{\partial s_v}, \quad \lambda_v^q = \frac{\partial E}{\partial q_v} \Big/ \frac{\partial C}{\partial q_v}. \tag{5.37}$$

At optimality the following conditions are met for all vertices v:

$$\begin{aligned} \lambda_v^s = \lambda_v^q = \lambda \qquad & \textit{Condition 1} \\ \frac{\partial E}{\partial \tilde{\mathcal{S}}_v} = 0 \qquad & \textit{Condition 2.} \end{aligned} \tag{5.38}$$

Condition 1 states that the marginal error-to-cost ratios are equally distributed, and thus all are equal to a constant λ. The equal distribution of marginal error-to-cost ratios, both λ_v^s and λ_v^q, indicates that there is an equal investment by changing the order of any element and by isotropically resizing the element. If order refinement is favored around a vertex v, then q_v will be driven larger than S_v to enforce the equality. Condition 2 states that the total error is stationary with respect to changes in the trace-free part $\tilde{\mathcal{S}}_v$ of the metric step. This condition sets the anisotropy of the elements at element sizes given by Condition 1.

The conditions and constraints above are enforced by the MOESS-P Algorithm 3. Again, at the top-level this is nearly identical to that for mesh adaptation alone, but most stages of the algorithm now operate on both step variables. There are twice as many margin error-to-cost ratios, computed at the beginning of each MOESS iteration with Algorithm 4.

Algorithm 3 Overview of the MOESS-P algorithm.

Compute the error model unknowns for each element e

Set $\delta s = \delta s_{\max}/n_{\text{step}},\ \delta q = \delta q_{\max}/n_{\text{step}}$

For $i = 1 \ldots n_{\text{step}}$

1. Compute marginal error-to-cost ratio $\lambda_v^s,\ \lambda_v^q$ with Algorithm 4
2. Sort $\{|\lambda_v|\} = \{|\lambda_v^s|\} \cup \{|\lambda_v^q|\}$ from high to low
3. (*Cond. 1*) Refine (+) at first 30%, and coarsen (−) at last 30% of $\{|\lambda_v|\}$
$$\mathcal{S}_v = \mathcal{S}_v \pm \delta s \mathcal{I}, \qquad q_v = q_v \pm \delta q$$
4. (*Cond. 2*) Augment trace-free part of $\mathcal{S}_v$
$$\mathcal{S}_v = \mathcal{S}_v + \delta s (\partial E_e / \partial \tilde{\mathcal{S}}_v)(\partial E_e / \partial s_v)$$
5. (*Constr. 1*) Rescale $\mathcal{S}_v$ and q_v to match target cost with Algorithm 5

The rescaling stage contains the majority of the differences from the original MOESS algorithm. The core difficulty here is that, unlike in the original algorithm for mesh adaptation, there is no option to explicitly solve for an update to the metric and order steps that brings the total cost C to the target cost N. The total cost,

Algorithm 4 Steps to compute marginal error-to-cost ratio at vertices for MOESS-P.

1. Convert step variables to element-centered values
$$\mathcal{S}_e = \frac{1}{|V_e|}\sum_{v\in V_e}\mathcal{S}_v, \qquad q_e = \frac{1}{|V_e|}\sum_{v\in V_e} q_v$$
2. Compute element error and cost model linearizations with respect to $\mathcal{S}_e$ and q_e via (5.33) and (5.35)
$$\frac{\partial C_e}{\partial \mathcal{S}_e}, \frac{\partial E_e}{\partial \mathcal{S}_e}, \frac{\partial C_e}{\partial q_e}, \frac{\partial E_e}{\partial q_e}$$
3. Convert error and cost model linearizations to vertex step variables
$$\frac{\partial K}{\partial \mathcal{S}_v} = \frac{1}{|V_e|}\sum_{e\in E_v}\frac{\partial K_e}{\partial \mathcal{S}_e}, \quad \frac{\partial K}{\partial q_v} = \frac{1}{|V_e|}\sum_{e\in E_v}\frac{\partial K_e}{\partial q_e}, \qquad K = E, C$$
4. Compute marginal error-to-cost ratios for all vertices via (5.37).

expressed as a sum of element cost models, can be written in the form

$$\log C = \sum_{\Omega_e\in T_h}\log\mathcal{N}(p_e) + \sum_{\Omega_e\in T_h}\frac{d\,s_e}{2}. \tag{5.39}$$

For mesh adaptation, the order cost $\mathcal{N}(p_e) = N_p$ is a constant for all elements, and thus

$$\log\frac{C_{\text{tgt}}}{C} = \sum_{\Omega_e\in T_h}\frac{d\,(s_e+\beta)}{2} - \sum_{\Omega_e\in T_h}\frac{d\,s_e}{2} = \sum_{\Omega_e\in T_h}\frac{d\,\beta}{2}, \tag{5.40}$$

which can be solved for a β that, when added to the trace of the metric step of each element, brings the total cost to C_{tgt}. When the order is also variable, rescaling instead must solve for both α and β satisfying

$$\log\frac{C_{\text{tgt}}}{C} = \sum_{\Omega_e\in T_h}\mathcal{N}(p_e+\alpha) - \sum_{\Omega_e\in T_h}\mathcal{N}(p_e) + \sum_{\Omega_e\in T_h}\frac{d\,\beta}{2}, \tag{5.41}$$

and another equation that sets the ratio α/β. There is no clear choice for what to set this ratio to, but this work chooses the ratio of marginal error-to-cost,

$$\frac{\sum_{\Omega_e\in T_h} C_e(\mathcal{S}_e+\beta, q_e)}{\sum_{\Omega_e\in T_h} C_e(\mathcal{S}_e, q_e+\alpha)} = \frac{\sum_v |\lambda_v^s|}{\sum_v |\lambda_v^q|} = r_{\alpha,\beta}. \tag{5.42}$$

This ratio indicates potential for improved efficiency in either mesh or order refinement, since the magnitude of the individual values indicate the relative benefit of investing in refinement. Even with this expression to close the system, solving (5.41)

and (5.42) still requires the solution of a nonlinear system, due to the nonlinearity of the cost coefficient function $\mathcal{N}(p)$.

Algorithm 5 Rescaling algorithm for MOESS-P on iteration i of the MOESS loop.

$C_{\text{tgt},i} = C_{\text{tgt}},\ r_{\alpha,\beta} = \dfrac{\sum_v |\lambda_v^s|}{\sum_v |\lambda_v^q|}$
$\alpha = \beta = 0$
While $|C_{\text{tgt},i} - C(\mathcal{S}_e + \beta, q_e + \alpha)| > \text{tol}$

1. Solve the nonlinear system with Newton for α and β

$$\sum_{\Omega_e \in T_h} \mathcal{N}(q_e + \alpha) \exp\left(d(s_e + \beta)/2\right) - C_{\text{tgt,i}} = 0$$

$$\frac{\sum_{\Omega_e \in T_h} C_e(\mathcal{S}_e + \beta, q_e)}{\sum_{\Omega_e \in T_h} C_e(\mathcal{S}_e, q_e + \alpha)} - r_{\alpha,\beta} = 0$$

2. (*Constr. 2, 3*) Check if constraints are active

$$|S_v|_{ij} > S_{\max}, \qquad |q_v| > q_{\max}$$

3. If any constraints were applied

   - Find the actual cost after clipping the step variables as C_{actual}
   - Reset the target cost

$$C_{\text{tgt},i} = \begin{cases} (1+f) \cdot C_{\text{tgt},i} & C_{\text{actual}} > C_{\text{tgt},i} \\ (1-f) \cdot C_{\text{tgt},i} & \text{otherwise.} \end{cases}$$

In fact, the system of equations is a nonlinear system with constraints that limit the magnitude of the resulting q_v and s_v for all vertices v. Algorithm 5 details the procedure for rescaling. Note it is possible for this system of equations with constraints to not have a solution for α and β, in which case either the ratio $r_{\alpha,\beta}$, or the target cost C_{tgt} must be changed to render the system solvable. However the total cost is a monotonic function of the metric step trace and order step, so by changing the target cost as in step (3) of the rescaling algorithm the system has a solution. If the target cost was changed, it is reset on the next MOESS iteration and over the course of many iterations the cost is brought toward the intended target.

5.3.5 Results

This section applies the MOESS-P method developed above to a set of test cases with the goal of assessing whether the algorithm is correctly implemented and whether the existing versions of error and cost models are sufficient to allow the algorithm to, in a sense automatically, select the optimal configuration of mesh and order resolution.

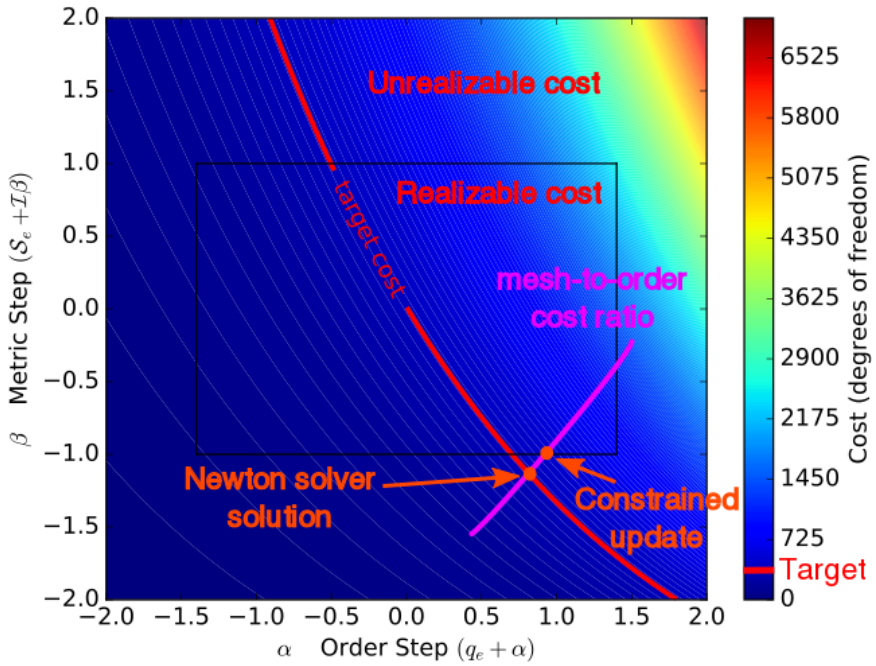


Figure 5.22: Example of rescaling on the first iteration of the scalar boundary layer test case from Section 5.3.5.2. The Newton solver solution for the rescaling parameters is unrealizable due to the constraints (3) and (4) in (5.36).

Therefore, this set of test cases also set out to determine whether the method is robust and accurate enough to be readily applied to a more complicated equations.

As with hanging-node-based *hp*-adaptation, the usual mode of operation is to begin with a uniform mesh and constant order, and iteratively solve the primal solution with and fine-space adjoint with HDG, obtain the error estimates, and adapt. Since the MOESS-P method changes the mesh and orders in place at the cost C_{tgt}, it is applied multiple times, 10 times for each of the tests below, at a fixed cost to ensure that the maximum update constraints (3) an (4) in (5.36) are not limiting the changes, then changes the target. This procedure is depicted for the first test case in Figure 5.24. On an error convergence plot, either the last error or an average of the errors can be used to represent the error at a cost. For each of the iterations, the MOESS-P algorithm itself has $n_{\text{step}} = 20$ sub-iterations performed at the top-level of the Algorithm 3 to smoothly change the step variables.

5.3.5.1 2D Scalar Smooth Solution

The first test case analyzes the algorithm behavior applied to a smooth solution defined by the equation

$$\begin{aligned} \nabla \cdot (\vec{a}u - b\nabla u) &= 0 && \text{in } \Omega := [0,1]^2 \\ u(x,y) &= \exp(-(x-1/2)^2 - (y-1/2)^2) && \text{on } \partial\Omega \end{aligned} \tag{5.43}$$

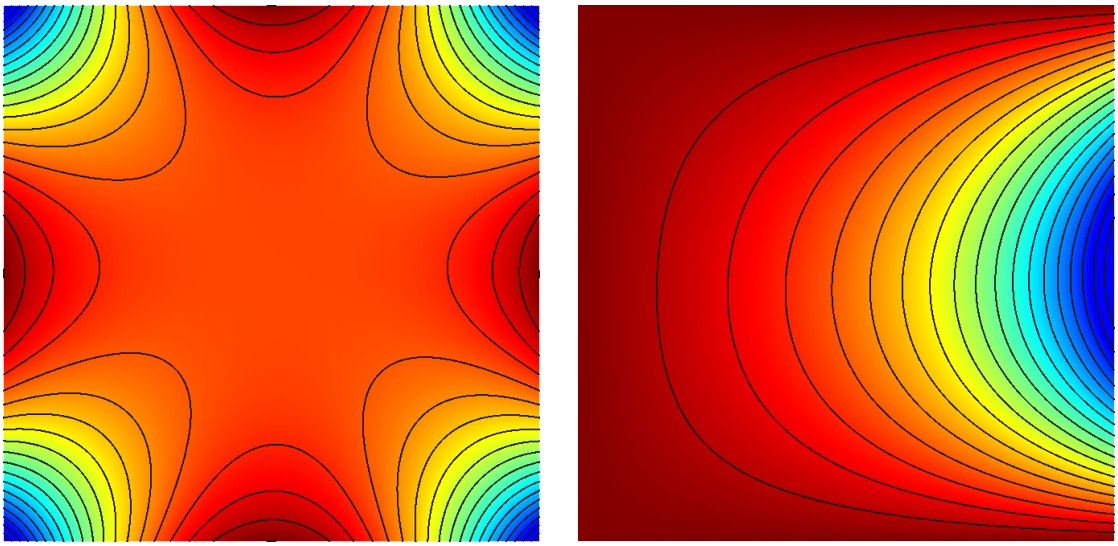
with $\vec{a} = (3/4, 4/5)$ and $b = 1$. The output is the weighted functional on the right boundary

$$J(u) = \int_0^1 \frac{\partial u}{\partial x}(1,y)\sin(\pi y)\, dy.$$

The primal and adjoint solutions for $J(u)$ are shown in Figure 5.23. Note that since the adjoint is computed with a sign change as in (4.37), it is shown as negative on the right boundary. Since the solution and adjoint are both everywhere smooth, the optimal mesh should have relatively few elements but with high order.

The optimization cycle begins with a uniform mesh of 512 triangles at $p = 1$, corresponding to a cost of 1536 DOF. To test if the method selects the correct adaptation choice, the 10 adaptive iterations of the MOESS-P optimization are applied with the original target cost, $C_{\text{tgt}} = 1536$. Figure 5.24 plots the ratio of the marginal error-to-cost ratios $r_{\alpha,\beta}$, the total cost, the average order, and the average trace of the metric step trace for the first MOESS iteration. As expected for this smooth test case, these results show that the algorithm finds raising the order to be more effective than adding mesh resolution, and thus increases the order and reduces the metric step trace. Note also that on the earlier sub-iterations where $r_{\alpha,\beta} < 1$, the next sub-iteration raises the order. On later sub-iterations for which the target cost is nearly matched, the rescaling does not factor in as heavily and the focus is instead on raising the order and increasing the size of the elements. This result provides a level of confidence that the algorithm is working as intended.

While C_{tgt} is usually increased to obtain error convergence, since the intent of this test case is to both qualitatively and quantitatively test whether the resulting mesh has fewer high-order elements, here the target cost is lowered after each set of iterations to 500, and then 100. The final meshes in Figure 5.26 show that indeed the algorithm is operating as intended, and reducing the number of elements. Although the elements may have variable orders, the final meshes show a constant order on all elements. For true optimality, the two higher target cost solutions should instead use the maximum order $p = 5$, but seem to settle with $p = 4$ instead. Future work should



(a) Primal solution u (b) Adjoint solution ψ, -1 : blue $\ldots$ 0 : red

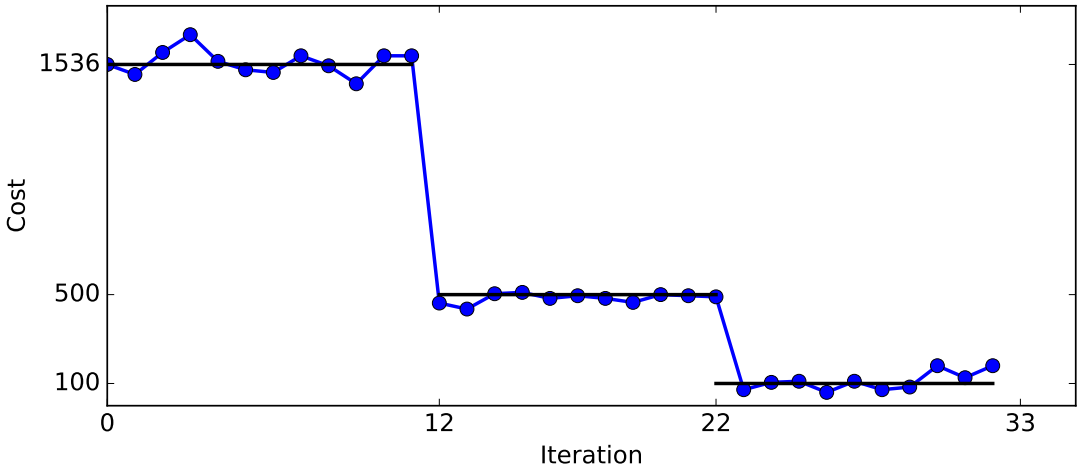
Figure 5.23: Primal and adjoint solution for the two-dimensional scalar smooth solution MOESS-P test case.

try to assess the reason why this occurs.

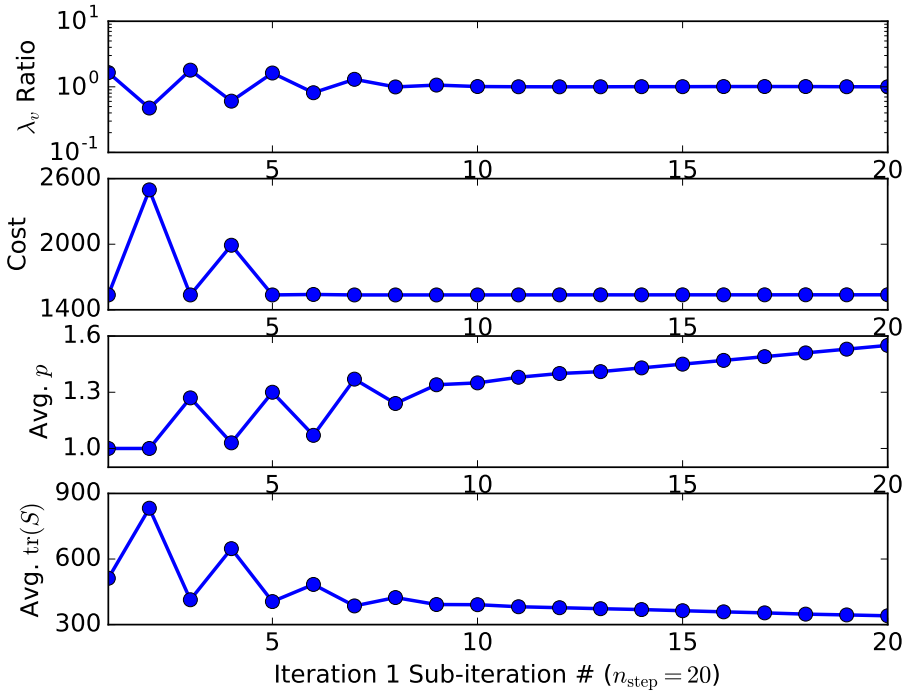
The convergence of the error in Figure 5.25 shows that even though the initial order was $p = 1$, the convergence rate is approximately sixth order. This figure also shows a quantitative comparison to an order-adapted solution on a static 16-element uniform quadrilateral element mesh, using the same process as in the hanging-node adaptation section above to assess the efficiency of this method. While it is difficult to make a direct comparison in this case since these two sets of cases begin with different meshes, the MOESS-P method obtains a similar or lower level of error compared to the order-adapted result.

5.3.5.2 2D Scalar Boundary Layer

The second test case examines the behavior of the MOESS-P method applied to a resolve a boundary layer in the scalar advection-diffusion equation (5.43) with $\vec{a} = (1, 0)$ and $b = 10^{-3}$, and boundary conditions described in Figure 5.27. A similar test case was proposed [55] to test the anisotropic mesh adaptation from MOESS, but used a different boundary condition on the left. This test case begins with a uniform mesh of 64 triangles at $p = 3$, in total 640 DOF, so the goal is to test whether the higher-order elements and mesh resolution will effectively cluster in the boundary



(a) Cost for all iterations



(b) Internal variables for iteration 1

Figure 5.24: Cost for each iteration of the MOESS-P algorithm and several internal variables plotted over sub-iterations of the first adaptive iteration of the scalar smooth solution test case.

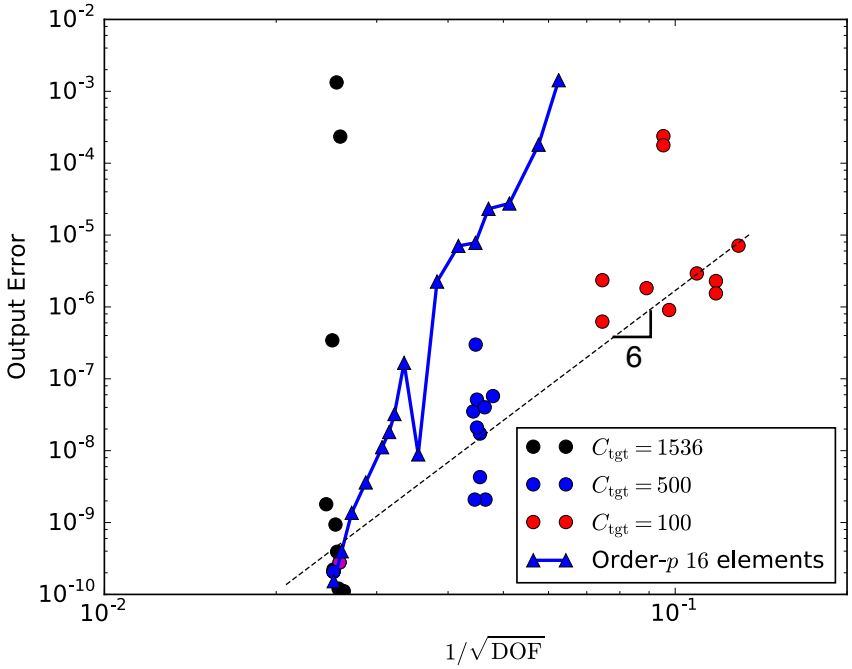


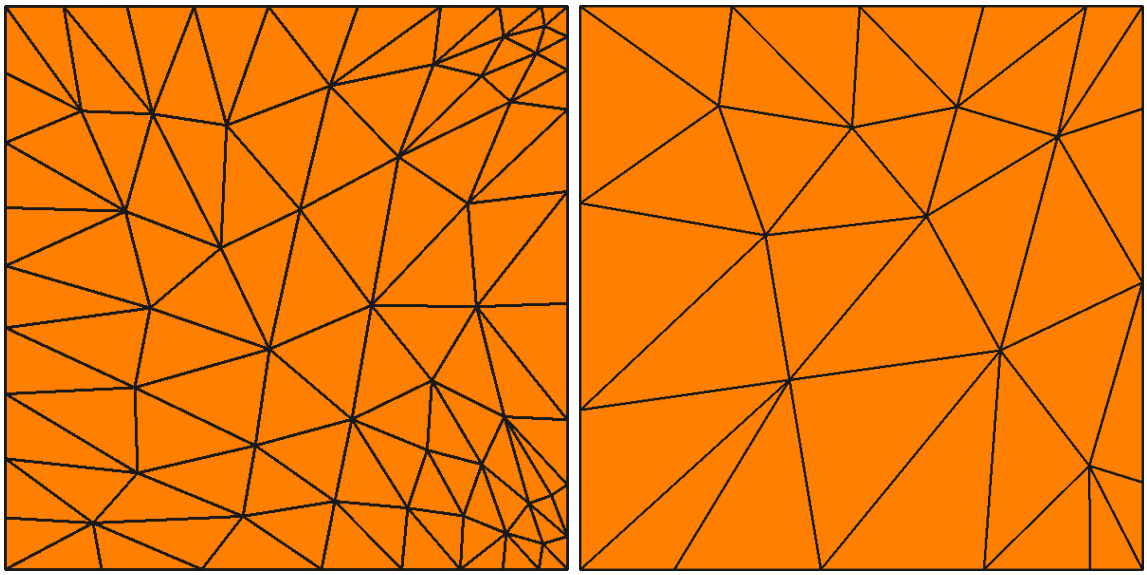
Figure 5.25: Output error convergence versus degrees of freedom for MOESS-P, compared to order-only adaptation with a static mesh beginning from 16 quadrilateral elements.

layer that most affects the output

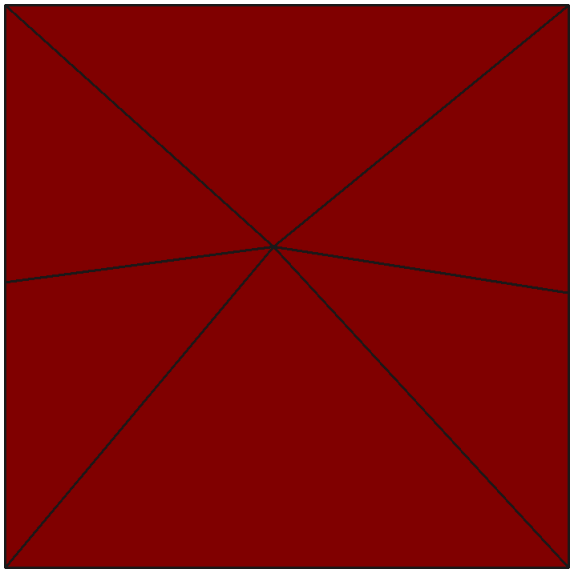
$$J(u) = \int_0^1 \frac{\partial u}{\partial x}(x, 0)\, dx.$$

Note that the adjoint solution for this output has a singularity at the origin, due to the boundary condition at the left, which makes this test case especially challenging. Again, the adjoint boundary layer shown in the setup figure is computed with a sign change, so the adjoint value near the wall is negative.

The adaptation begins with a target cost $C_{\text{tgt}} = 1000$, and then increases this to 2000, 5000, and 10000. Although the error, shown in Figure 5.28, is sub-optimally converging due to the singularity in the adjoint at the origin, the algorithm does indeed seem to be clustering the mesh adaptation and high-order elements in the boundary layer, as desired. The final meshes in Figure 5.29, after the full 10 adaptive iterations at each cost, indicate that the order dependence in the error model may still need to be tested and changed further to keep high-order elements in the boundary layer for large target costs.



(a) $C_{\text{tgt}} = 1536$, 105 elem, $p = 4$ (b) $C_{\text{tgt}} = 500$, 30 elem, $p = 4$



(c) $C_{\text{tgt}} = 100$, 12 elem, $p = 5$

Figure 5.26: Final meshes for the scalar smooth solution test case.

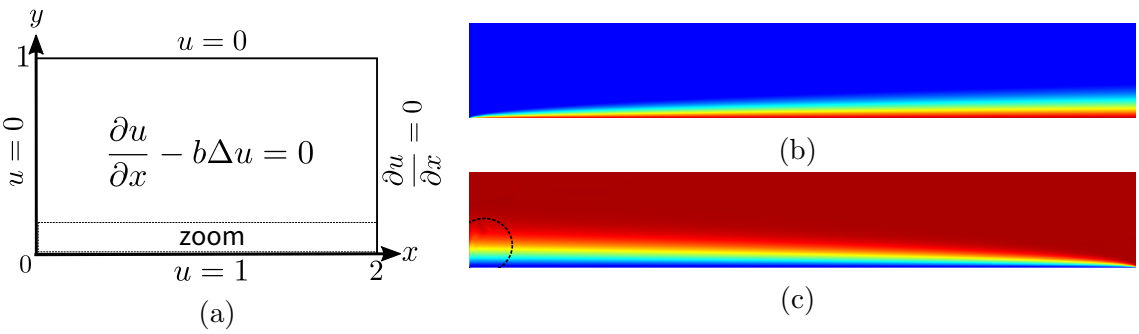


Figure 5.27: Domain and boundary conditions for the scalar advection-diffusion boundary layer test case. The primal and adjoint boundary layers are shown on the right in (b) and (c) -1 : blue . . . 0 : red.

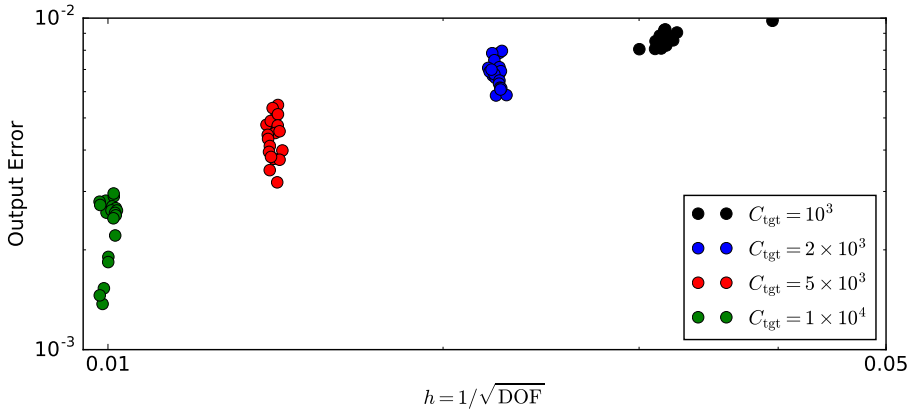
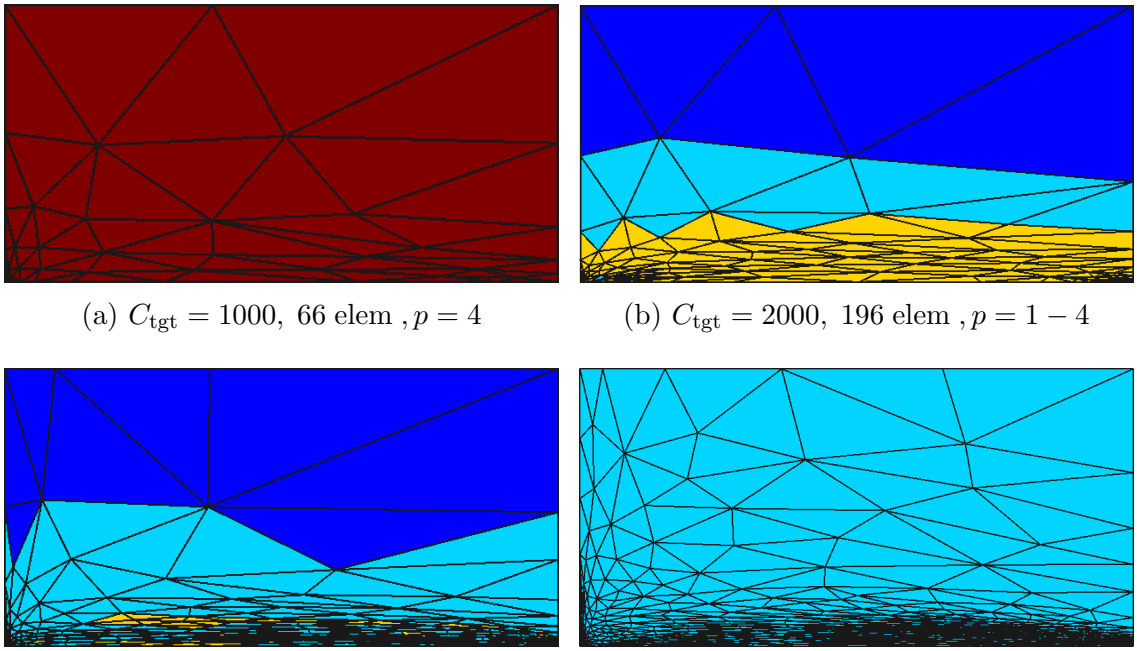


Figure 5.28: Heat flux error convergence for the scalar boundary layer test case.



(a) $C_{\text{tgt}} = 1000,\ 66$ elem $, p = 4$ (b) $C_{\text{tgt}} = 2000,\ 196$ elem $, p = 1 - 4$

(c) $C_{\text{tgt}} = 5000,\ 492$ elem $, p = 1 - 3$ (d) $C_{\text{tgt}} = 10000,\ 196$ elem $, p = 2, 3$

Figure 5.29: Final meshes for the scalar boundary layer test case. For low target costs, the elements stay high order, then as the target cost increases, the high order elements generally cluster in the boundary layer.

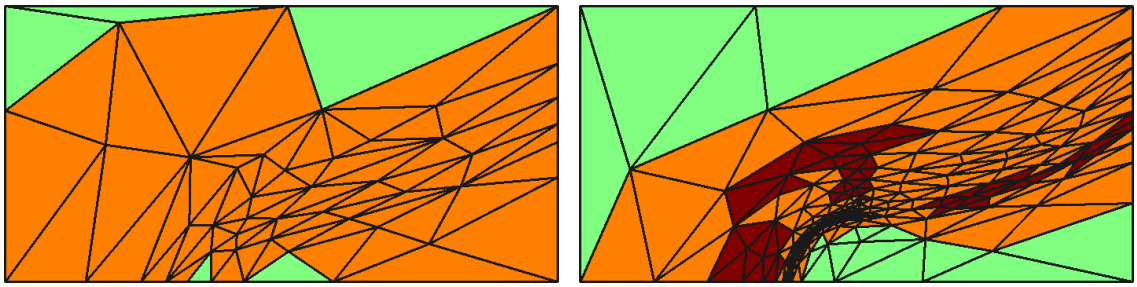
5.3.5.3 2D Scalar Advection Problem

The final test case applies MOESS-P to the scalar advection problem already introduced in the context of hanging-node adaptation in Section 5.2.4.1, except that the output kernel function is

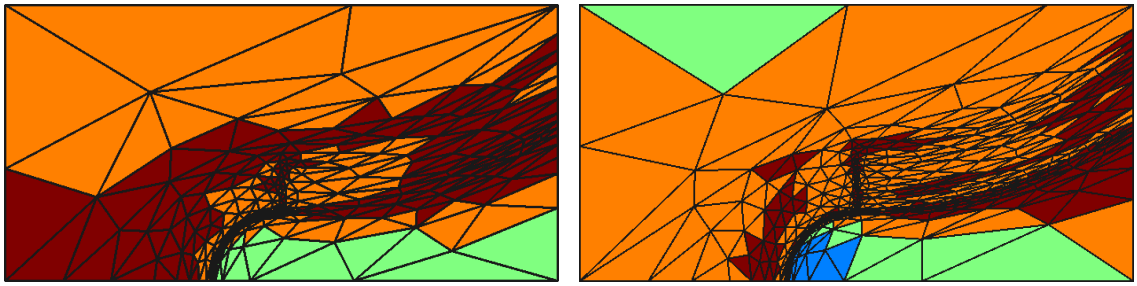
$$j(y) = \exp\left((8/3)^2 - 1/((y - 5/8)^2 - 3/8)^2\right). \tag{5.44}$$

The initial mesh has 240 elements at $p = 1$, and a target cost of 1000, then after 10 adaptive iterations the target cost is raised to 5000, 10000, and then 20000. Although the 2D scalar boundary layer case suggests that the error model or sampling may need to be changed to better indicate the regions where high order accelerates the convergence of the output, it is nonetheless instructive to test the current version of the method on a more difficult test case, allowing for a direct comparison to the *hp*-adaptation results.

The results in Figures 5.30 through 5.32 show, somewhat unsurprisingly given the other two test cases, that the more matured *hp*-adaptation method is outperforming MOESS-P. One reason for this seems to be that the resulting meshes are not well



(a) MOESS-P $C_{\text{tgt}} = 1000,\ p = 3, 4$ (b) MOESS-P $C_{\text{tgt}} = 5000\ , p = 3 - 5$



(c) MOESS-P $C_{\text{tgt}} = 10000,\ p = 3 - 5$ (d) MOESS-P $C_{\text{tgt}} = 20000,\ p = 2 - 5$

Figure 5.30: Final meshes for the scalar advection test case using the MOESS-P algorithm.

clustered along the path of the adjoint and discontinuity in the primal solution, so it essentially wastes mesh resolution in smooth regions. It is interesting to note that although in the previous sections the hanging-node framework was seen as a limitation, in this case it in fact beneficial to help guide the regions for mesh and order resolution. The other reason is that high order elements are kept in regions that have extremely little or no effect at all on the right side output, specifically near the top left of the domain. This could be due to the rescaling algorithm: these elements have q_e lowered by step 2 of Algorithm 3 only to have it raised again by the rescaling that finds order refinement beneficial, so $r_{\alpha,\beta} \ll 1$. Thus, more work needs to be done on the algorithm for rescaling to avoid this back-and-forth behavior.

5.3.5.4 Review

The sections above demonstrate the current version of the MOESS-P method applied to a series of test cases to assess the development progress. Current results show promise for such a method to correctly select the order and mesh resolution, though the current algorithm fails to perform better than existing hanging-node *hp*-adaptation for a similar test case. Although the adaptation does not perform as desired on the final test case, many aspects of the method have been shown to perform

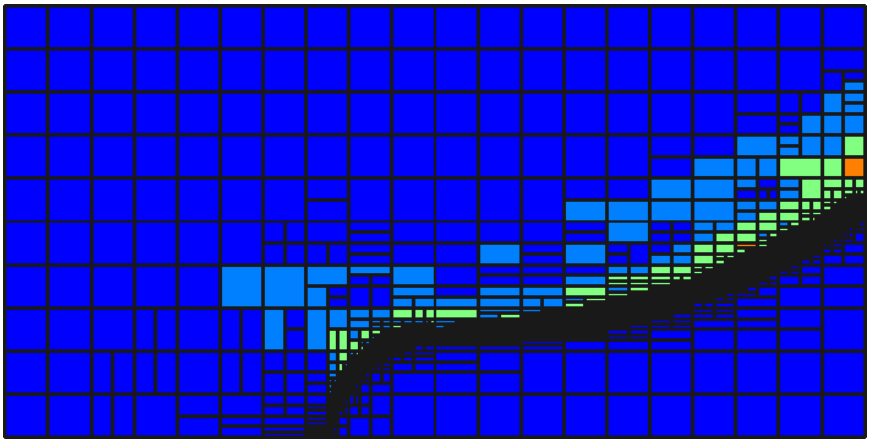


Figure 5.31: Final mesh for the scalar advection test case after applying the hanging-node hp-adaptation. 17210 DOF with $p = 1 - 4$.

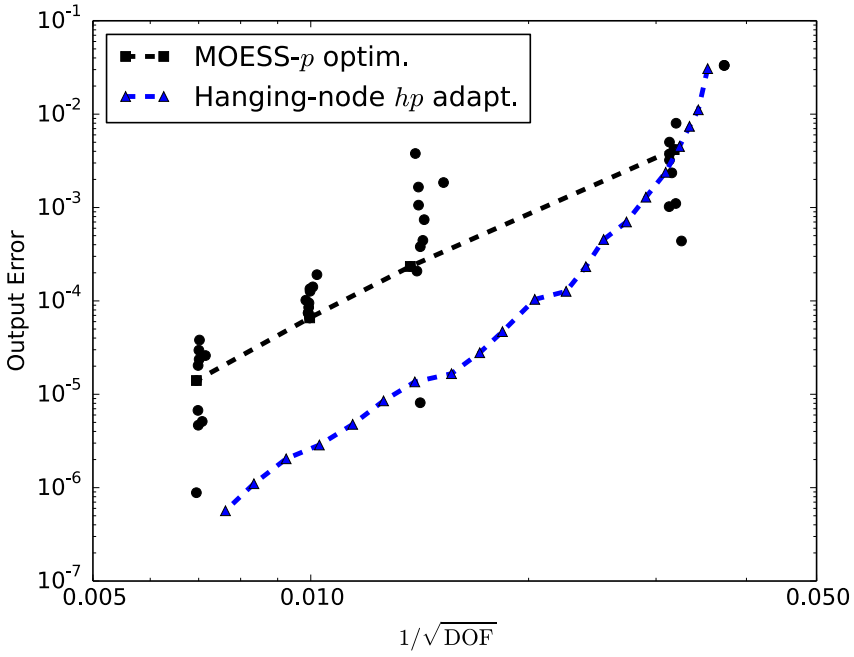


Figure 5.32: Comparison of output error between hanging-node hp-adaptation and the present version of MOESS-P.

well. For instance, at a fixed target cost the final error is approximately equal to that of pure order adaptation with a quadrilateral mesh. The method also correctly clusters anisotropic elements in a scalar boundary layer and at certain target costs also places high-order elements in the layer.

CHAPTER 6

Conclusions

6.1 Summary and Conclusions

This dissertation has presented new results contributing to the development of accurate, efficient, and robust high-order discontinuous Galerkin (DG) methods, and especially hybridized discontinous Galerkin (HDG) methods, for finite-element calculations involving convection and diffusion, with primary application to the compressible Navier-Stokes equations. In particular, this work has addressed development of general HDG discretizations for systems of equations, and output-based methods for simultaneous mesh (h) and order (p) adaptation using adjoint error estimation to reduce the error in a targeted scalar functional of the approximate solution to the discretized equations. It has made contributions in several key aspects of such methods as they apply to systems of equations, addressing scalability and memory requirements, accuracy and computational efficiency of the HDG discretization, and robustness of the hp-adaptation for DG and HDG methods.

In Chapter 4 this work has extended existing HDG methods to general systems of convection-diffusion equations. It has done so and improved the robustness of the methods by introducing a viscous stabilization that depends on the viscous diffusivity tensor, thereby properly stabilizing each equation independently. Along with the development of the mixed formulation for such systems, a new primal formulation has also been developed, together with Woopen and May, that borrows viscous stabilization concepts from DG. This primal formulation has been shown in this work to be more computationally efficient than existing methods and sufficiently robust to apply to general compressible Navier-Stokes computations. The HDG and DG discretizations are implemented in a common framework that in Section 4.5 allows for an objective comparison of their respective run times on a set of test cases.

Additionally, in Section 4.3 this work developed a new formulation for the convective flux stabilization, for which the convective flux is properly upwinded while also enforcing that the resulting trace is similarly upwinded. This reduces the stiffness and recovers optimal convergence of the solution gradient variable, which with the previously existing stabilization formulation was lost. This improved convergence for the gradient allows for an element-by-element post-processing to be applied broadly for general systems, even in the case of pure convection, and which overcomes the difficulties that the Raviart-Thomas-based post-processing faced. The resulting post-processed solution is further shown to positively impact the convergence of typical outputs relevant to computational fluid dynamics.

Chapter 5 detailed the development of two new adaptation approaches for combined mesh and order adaptation that are suitable for *hp*-adaptation with both DG and HDG discretizations. For the first of these, based on hanging-node mesh adaptation for quadrilateral or hexahedral meshes, in Section 5.2.1 this work has developed a novel local approach for evaluating refinement options to determine an optimal refinement combination, based on minimizing or maximizing an objective function involving projections of the adjoint used to compute an error estimate. The resulting algorithm was presented here, along with two procedures for correcting the fine-space adjoint to obtain meaningful measures of the error reduction with each of the refinement options. The second approach has extended the MOESS framework for metric mesh optimization to incorporate order refinement, thereby enabling efficient *hp*-adaptation on simplex meshes. The accompanying error model has introduced order dependence and a novel rescaling approach is designed that incorporates the relative efficiency of resolving the output with order and mesh refinement.

Collectively, the results in this work have addressed a number of the key issues currently at the forefront of developing high-order CFD methods, and have contributed to improvements in output-based mesh and order adaptation for DG and HDG methods. These have lead to the principal major conclusions summarized below.

1. **HDG Discretizations**

   - The mixed and primal formulations of the HDG discretization both improve the efficiency of computing outputs of interest from CFD simulations compared to DG, and the HDG method when applied to systems without diffusion is even more computationally efficient than DG applied to the same system.
   - Both of these new formulations save an order of magnitude in memory to

store the Jacobian matrix for the same error level compared to DG with the same order.

- The ILU-0 preconditioner is more effective than a line-based Jacobi iteration applied as a preconditioner for GMRES when solving HDG systems.
- The element-by-element post-processing method developed for general systems improves the order of accuracy in computing outputs involving domain integrals.

2. *hp*-Adaptation and Optimization

- The hanging-node *hp*-adaptation method developed herein is as effective as pure order adaptation with a static mesh when no diffusion term is present, and almost as effective even when applied to the viscous Navier-Stokes equations.
- When using local adjoint projections, taking the difference of the projected and finer-space adjoints in computing the adjoint-weighted error estimate improves the accuracy of refinement choices.
- Hanging-node *hp*-adaptation is limited in resolving boundary layers by the geometry of the initial mesh, thus only an *hp*-optimization method on simplex meshes is theoretically able to best approximate the output for a given cost.
- The current MOESS-P method is able to make correct adaptation choices, trading high- and low-order elements and mesh resolution, for a set of scalar test cases.
- More development needs to be done on the MOESS-P method to apply it successfully to general applications.

6.2 Recommendations For Future Work

During the course of this research, several areas for potential future work were identified.

1. HDG Discretizations

- **Formulation to allow adjoint post-processing:** If the HDG method is reformulated by instead solving $\vec{\mathbf{q}} - \mathbf{K}\nabla u = 0$, then the resulting adjoint

variable represents the primal adjoint gradient, enabling post-processing. This post-processed adjoint could then be used for error estimation in a "cost-free" approach, without the need for the additional fine-space linear solve or patch-based adjoint reconstruction.

- **Constant-scaling viscous stabilizations approaches:** For coarse or highly anisotropic meshes, constant-scaling viscous stabilization methods often fail to converge. An approach that uses BR2 or a similar method only in these regions, or that corrects the solution using constant-scaling stabilization, could be made both robust and accurate.
- **Extension to embedded discontinuous Galerkin (EDG):** Embedded discontinuous Galerkin methods approximate the trace by continuous functions over the skeleton of the domain, increasing the error, but further reducing the size of the linear system. Even if the error increases slightly by using such methods, the high memory and efficiency gains with HDG could be further improved.
- **Implementation with multithreading:** Much of the added efficiency of HDG comes about from a trade-off between a large global linear system and the solutions of small local problems. If CPU or GPU threads are used, the solution of these local problems can be greatly accelerated, further increasing the efficiency gains from hybridization. This direction has already been taken by Roca *et al.* [67] and others, but with new architectures increasing the bandwidth between the GPU and memory there is more that can be done.

2. *hp*-Adaptation and Optimization

- **Extension to unsteady problems:** The methods for *hp*-adaptation developed herein were formulated and tested on steady-state problems, but the ideas can be extended to unsteady calculations using the unsteady adjoint. The challenge in doing this is to implement the adaptation in an efficient manner, as the unsteady adjoint and the solution of the local problems will add to the run time cost.
- **Improve the behavior of MOESS-P algorithm:** Although the MOESS-P algorithm generally selects the correct refinement, when both mesh and order refinement are required the rescaling algorithm raises the order uniformly faster than it can be reduced in regions that have little effect.

- **Improve error estimation with HDG:** The localized error indicator resulting from HDG is often more spread out than that from DG. A closer look at when exactly this occurs could lead to new insights into how to adapt the HDG state to better approximate the outputs.

APPENDIX A

Adjoints for Mixed and Hybridized Methods

This writeup addresses the adjoint interpretation for mixed hybridized discontinuous Galerkin. A detailed analysis for a mixed hybridized method was already shown in [34], so this work will address the differences necessary for this work. Not only does this analysis help to better understand how to interpret the adjoint variables, but is critical for correctly forming a post-processing projection for them.

Consider the possibly nonlinear convection-diffusion equation on a domain $\Omega \subset \mathbb{R}^d$

$$\begin{aligned} \vec{q} - \nabla u &= 0 \\ \nabla \cdot (f(u) - b\vec{q}) + s(u) &= 0 \end{aligned} \quad \text{in } \Omega, \qquad u = g \ \text{ on } \partial\Omega. \tag{1.1}$$

Note that [34] considered instead $\vec{q} = -b\nabla u$, so this differs in the definition of $\vec{q}$. The output functional of interest will be

$$\mathcal{J}(\vec{q}, u) = \int_\Omega j_\Omega u \, d\Omega + \int_\Gamma j_\Gamma \vec{q} \cdot \vec{n} \, ds. \tag{1.2}$$

The continuous adjoint equations are obtained by first multiplying (1.1) by the adjoint variables $\vec{\psi}^{\vec{v}}$ and ψ^w and integrating by parts. This yields a mixed semilinear form $\mathcal{R}(\vec{q}, u; \vec{\psi}^{\vec{v}}, \psi^w) = \mathcal{R}_1(\vec{q}, u; \vec{\psi}^{\vec{v}}) + \mathcal{R}_2(\vec{q}, u; \psi^w)$ where

$$\mathcal{R}_1(\vec{q}, u; \vec{\psi}^{\vec{v}}) = \left(\vec{q}, \vec{\psi}^{\vec{v}}\right)_\Omega + \left(u, \nabla \cdot \vec{\psi}^{\vec{v}}\right)_\Omega - \left\langle g, \vec{\psi}^{\vec{v}} \cdot \vec{n} \right\rangle_{\partial\Omega} \tag{1.3}$$

$$\mathcal{R}_2(\vec{q}, u; \psi^w) = -\left(f(u) - b\vec{q}, \nabla \psi^w\right)_\Omega + \left\langle f(g) - b\vec{q}, \psi^w \right\rangle_{\partial\Omega} + \left(s(u), \psi^w\right)_\Omega \tag{1.4}$$

The adjoints $\vec{\psi}^{\vec{v}}$ and ψ^w satisfy the adjoint relation

$$\mathcal{R}'[\vec{q}](\delta\vec{q}, 0; \vec{\psi}^{\vec{v}}, \psi^w) + \mathcal{R}'[u](0, \delta u; \vec{\psi}^{\vec{v}}, \psi^w) = \mathcal{J}[\vec{q}](\delta\vec{q}, 0) + J[u](0, \delta u)$$

for all variations $\delta\vec{q}$ and δu compatible with the boundary conditions. Computing

this and separating the variations into different expressions yields,

$$\left(\vec{\psi}^{\vec{v}}, \delta\vec{q}\right)_\Omega + (b\nabla\psi^w, \delta\vec{q})_\Omega - \langle b\psi^w, \delta\vec{q}\cdot\vec{n}\rangle_{\partial\Omega} + \langle j_\Gamma, \delta\vec{q}\cdot\vec{n}\rangle_\Gamma = 0$$
$$\left(\nabla\cdot\vec{\psi}^{\vec{v}}, \delta u\right)_\Omega - \left(f'(u)^T\nabla\psi^w, \delta u\right)_\Omega + (s'(u)\psi^w, \delta u)_\Omega + (j_\Omega, \delta u)_\Omega = 0.$$

The continuous adjoint problem is then given by

$$\begin{aligned} \vec{\psi}^{\vec{v}} + b\nabla\psi^w &= 0 \\ -f'(u)^T\nabla\psi^w + \nabla\cdot\vec{\psi}^{\vec{v}} + s'(u)\psi^w + j_\Omega &= 0 \end{aligned} \quad \text{in } \Omega \qquad b\psi^w = \begin{cases} j_\Gamma & \Gamma \\ 0 & \partial\Omega\backslash\Gamma. \end{cases} \tag{1.5}$$

Thus when the primal dual $\vec{q} = \nabla u$, the adjoint dual $\vec{\psi}^{\vec{v}} = -b\nabla\psi^w$. In contrast, Schütz and May find that $\vec{\psi}^{\vec{v}} = -\nabla\psi^w$ when $\vec{q} = b\nabla u$. There is a duality to these expressions but the result is clear: the diffusivity enters the dual either in the primal or the adjoint variable definition.

BIBLIOGRAPHY

[1] Fidkowski, K. J., *A simplex cut-cell adaptive method for high-order discretizations of the compressible Navier–Stokes equations*, Ph.D. thesis, Massachusetts Institute of Technology, 2007.

[2] Fidkowski, K. J., *A high-order discontinuous Galerkin multigrid solver for aerodynamic applications*, Master's thesis, Massachusetts Institute of Technology, 2004.

[3] Wang, Z. J., Fidkowski, K., Abgrall, R., Bassi, F., Caraeni, D., Cary, A., Deconinck, H., Hartmann, R., Hillewaert, K., Huynh, H. T., 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. 72, No. 8, 2013, pp. 811–845.

[4] Oliver, T. A., *A high-order, adaptive, discontinuous Galerkin finite element method for the Reynolds-averaged Navier-Stokes equations*, Ph.D. thesis, Massachusetts Institute of Technology, 2008.

[5] Allmaras, S. R. and Johnson, F. T., "Modifications and clarifications for the implementation of the Spalart-Allmaras turbulence model," *Seventh International Conference on Computational Fluid Dynamics (ICCFD7)*, 2012, pp. 1–11.

[6] Ceze, M., *A robust hp-adaptation method for discontinuous Galerkin discretizations applied to aerodynamic flows*, Ph.D. thesis, University of Michigan, 2013.

[7] 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.

[8] Douglas, J. and Dupont, T., "Interior penalty procedures for elliptic and parabolic Galerkin methods," *Computing Methods in Applied Sciences: Second*

International Symposium, edited by R. Glowinski and J. L. Lions, Springer Berlin Heidelberg, Berlin, Heidelberg, December 1976, pp. 207–216.

[9] Bassi, F. and Rebay, S., *GMRES discontinuous Galerkin solution of the compressible Navier-Stokes equations*, chap. 1, Springer Berlin Heidelberg, Berlin, Heidelberg, 2000, pp. 197–208.

[10] Shahbazi, K., "An explicit expression for the penalty parameter of the interior penalty method," *Journal of Computational Physics*, Vol. 205, No. 2, 2005, pp. 401 – 407.

[11] Roe, P., "Approximate Riemann solvers, parameter vectors, and difference schemes," *Journal of Computational Physics*, Vol. 43, No. 2, 1981, pp. 357 – 372.

[12] Toro, E. F., *Riemann solvers and numerical methods for fluid dynamics: a practical introduction*, Springer Science & Business Media, 2013.

[13] Pierce, N. A. and Giles, M. B., "Adjoint recovery of superconvergent functionals from PDE approximations," *SIAM Review*, Vol. 42, No. 2, 2000, pp. 247–264.

[14] Becker, R. and Rannacher, R., "An optimal control approach to a posteriori error estimation in finite element methods," *Acta Numerica*, Vol. 10, 2001, pp. 1–102.

[15] Giles, M. B. and Süli, E., "Adjoint methods for PDEs: a posteriori error analysis and postprocessing by duality," *Acta Numerica*, Vol. 11, 2002, pp. 145–236.

[16] 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.

[17] Formaggia, L., Perotto, S., and Zunino, P., "An anisotropic a-posteriori error estimate for a convection-diffusion problem," *Computing and Visualization in Science*, Vol. 4, No. 2, 2001, pp. 99–104.

[18] 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.

[19] Fidkowski, K. J. and Darmofal, D. L., "Review of output-based error estimation and mesh adaptation in computational fluid dynamics," *AIAA Journal*, Vol. 49, No. 4, April 2011, pp. 673–694.

[20] Lu, J. C.-C., *An a posteriori error control framework for adaptive precision optimization using discontinuous Galerkin finite element method*, Ph.D. thesis, Massachusetts Institute of Technology, 2005.

[21] Oliver, T. A. and Darmofal, D. L., "Analysis of dual consistency for discontinuous Galerkin discretizations of source terms," *SIAM Journal on Numerical Analysis*, Vol. 47, No. 5, 2009, pp. 3507–3525.

[22] De Veubeque, B. F., "Displacement and equilibrium models in the finite element method," *Stress Analysis*, Vol. 9, 1965, pp. 145–197.

[23] Cockburn, B., Guzmán, J., and Wang, H., "Superconvergent discontinuous Galerkin methods for second-order elliptic problems," *Math. Comp.*, Vol. 78, No. 265, 2009, pp. 1–24.

[24] Cockburn, B., Gopalakrishnan, J., and Lazarov, R., "Unified hybridization of discontinuous Galerkin, mixed, and continuous Galerkin methods for second order elliptic problems," *SIAM Journal on Numerical Analysis*, Vol. 47, No. 2, 2009, pp. 1319–1365.

[25] Cockburn, B., Dong, B., and Guzmán, J., "A superconvergent LDG-hybridizable Galerkin method for second-order elliptic problems," *Mathematics of Computation*, Vol. 77, No. 264, 2008, pp. 1887–1916.

[26] Cockburn, B., Dong, B., Guzmán, J., Restelli, M., and Sacco, R., "A hybridizable discontinuous Galerkin method for steady-state convection-diffusion-reaction problems," *SIAM Journal on Scientific Computing*, Vol. 31, No. 5, 2009, pp. 3827–3846.

[27] 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, No. 9, 2009, pp. 3232 – 3254.

[28] Peraire, J., Nguyen, N., and Cockburn, B., "A hybridizable discontinuous Galerkin method for the compressible Euler and Navier-Stokes equations," *48th AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition*, Aerospace Sciences Meetings, American Institute of Aeronautics and Astronautics, January 2010.

[29] Dahm, J. P. and Fidkowski, K., "Error Estimation and Adaptation in Hybridized Discontinous Galerkin Methods," *52nd Aerospace Sciences Meeting*,

AIAA SciTech Forum, American Institute of Aeronautics and Astronautics, January 2014.

[30] Qiu, W. and Shi, K., "An HDG method for convection diffusion equation," *Journal of Scientific Computing*, Vol. 66, No. 1, 2016, pp. 346–357.

[31] Persson, P.-O. and Peraire, J., "Newton-GMRES preconditioning for discontinuous Galerkin discretizations of the Navier–Stokes equations," *SIAM Journal on Scientific Computing*, Vol. 30, No. 6, 2008, pp. 2709–2733.

[32] Cockburn, B., Dubois, O., Gopalakrishnan, J., and Tan, S., "Multigrid for an HDG method," *IMA Journal of Numerical Analysis*, 2013.

[33] Karypis, G. and Kumar, V., "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM Journal on Scientific Computing*, Vol. 20, No. 1, 1998, pp. 359–392.

[34] Schütz, J. and May, G., "An adjoint consistency analysis for a class of hybrid mixed methods," *IMA Journal of Numerical Analysis*, 2013.

[35] Cockburn, B., Gopalakrishnan, J., and Sayas, F.-J., "A projection-based error analysis of HDG methods," *Mathematics of Computation*, Vol. 79, No. 271, 2010, pp. 1351–1367.

[36] Fidkowski, K. J., "A hybridized discontinuous Galerkin method on mapped deforming domains," *Computers & Fluids*, Vol. 139, November 2016, pp. 80–91.

[37] Woopen, M. and May, G., "An anisotropic adjoint-based hp-adaptive HDG method for compressible turbulent flow," *53rd AIAA Aerospace Sciences Meeting*, AIAA SciTech, American Institute of Aeronautics and Astronautics, January 2015.

[38] Hartmann, R., "Adjoint consistency analysis of discontinuous Galerkin discretizations," *SIAM Journal on Numerical Analysis*, Vol. 45, No. 6, 2007, pp. 2671–2696.

[39] Cockburn, B. and Gopalakrishnan, J., "A characterization of hybridized mixed methods for second order elliptic problems," *SIAM Journal on Numerical Analysis*, Vol. 42, No. 1, 2004, pp. 283–301.

[40] Bastian, P. and Rivière, B., "Superconvergence and H(div) projection for discontinuous Galerkin methods," *International Journal for Numerical Methods in Fluids*, Vol. 42, No. 10, 2003, pp. 1043–1057.

[41] Nguyen, N., Peraire, J., and Cockburn, B., "An implicit high-order hybridizable discontinuous Galerkin method for nonlinear convection–diffusion equations," *Journal of Computational Physics*, Vol. 228, No. 23, 2009, pp. 8841 – 8855.

[42] Fidkowski, K., "High-order output-based adaptive methods for steady and unsteady aerodynamics," *37th Advanced VKI CFD Lecture Series: Recent Developments in Higher Order Methods and Industrial Application in Aeronautic*, The von Karman Institute for Fluid Dynamics, 2013.

[43] Persson, P.-O. and Peraire, J., "Curved mesh generation and mesh refinement using Lagrangian solid mechanics," *47th AIAA Aerospace Sciences Meeting including The New Horizons Forum and Aerospace Exposition*, Aerospace Sciences Meetings, American Institute of Aeronautics and Astronautics, January 2009.

[44] Peraire, J. and Patera, A., "Bounds for linear-functional outputs of coercive partial differential equations: local indicators and adaptive refinement," *Studies in Applied Mechanics*, Vol. 47, 1998, pp. 199–216.

[45] Demkowicz, L., Rachowicz, W., and Devloo, P., "A fully automatic hp-adaptivity," *Journal of Scientific Computing*, Vol. 17, No. 1, 2002, pp. 117–142.

[46] Hall, E. J. C., *Anisotropic adaptive refinement for discontinuous Galerkin methods*, Ph.D. thesis, University of Leicester, Department of Mathematics, 2007.

[47] Ceze, M. and Fidkowski, K. J., "Anisotropic hp-adaptation framework for functional prediction," *AIAA journal*, Vol. 51, No. 2, 2012, pp. 492–509.

[48] McRae, D., "r-Refinement grid adaptation algorithms and issues," *Computer Methods in Applied Mechanics and Engineering*, Vol. 189, No. 4, 2000, pp. 1161 – 1182, Adaptive Methods for Compressible CFD.

[49] Castro-Diaz, M., Hecht, F., Mohammadi, B., and Pironneau, O., "Anisotropic unstructured mesh adaption for flow simulations," *International Journal for Numerical Methods in Fluids*, Vol. 25, No. 4, 1997, pp. 475–491.

[50] Babuska, I., Szabo, B. A., and Katz, I. N., "The p-version of the finite element method," *SIAM Journal on Numerical Analysis*, Vol. 18, No. 3, 1981, pp. 515–545.

[51] Szabo, B. A., "Mesh design for the p-version of the finite element method," *Computer Methods in Applied Mechanics and Engineering*, Vol. 55, No. 1–2, 1986, pp. 181 – 197.

[52] Huang, W. and Li, X., "An anisotropic mesh adaptation method for the finite element solution of variational problems," *Finite Elements in Analysis and Design*, Vol. 46, No. 1–2, 2010, pp. 61 – 73, Mesh Generation - Applications and Adaptation.

[53] Yano, M., Modisette, J. M., and Darmofal, D. L., "The importance of mesh adaptation for higher-order discretizations of aerodynamic flows," *20th AIAA CFD Conference, AIAA-2011*, Vol. 3852, 2011.

[54] 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, No. 2, 2007, pp. 1653 – 1672.

[55] Yano, M. and Darmofal, D. L., "An optimization-based framework for anisotropic simplex mesh adaptation," *Journal of Computational Physics*, Vol. 231, No. 22, 2012, pp. 7626–7649.

[56] Fidkowski, K., "A local sampling approach to anisotropic metric-based mesh optimization," *54th AIAA Aerospace Sciences Meeting*, 2016, p. 0835.

[57] Babuska, I. and Suri, M., "The p- and h-p versions of the finite element method, an overview," *Computer Methods in Applied Mechanics and Engineering*, Vol. 80, No. 1, 1990, pp. 5 – 26.

[58] Ainsworth, M. and Senior, B., "An adaptive refinement strategy for hp-finite element computations," *Applied Numerical Mathematics*, Vol. 26, No. 1–2, 1998, pp. 165 – 178.

[59] Burgess, N. and Mavriplis, D., "An hp-adaptive discontinuous Galerkin solver for aerodynamic flows on mixed-element meshes," *49th AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition*, Aerospace Sciences Meetings, American Institute of Aeronautics and Astronautics, January 2011.

[60] Shi, L. and Wang, Z., "Adjoint based error estimation and hp-adaptation for the high-order CPR method," *51st AIAA Aerospace Sciences Meeting including the*

New Horizons Forum and Aerospace Exposition, Aerospace Sciences Meetings, American Institute of Aeronautics and Astronautics, January 2013.

[61] Balan, A., Woopen, M., and May, G., "Adjoint-based hp-adaptivity on anisotropic meshes for high-order compressible flow simulations," *Computers & Fluids*, November 2016, pp. 47–67.

[62] Dolejší, V., "Anisotropic hp-adaptive method based on interpolation error estimates in the Lq-norm," *Applied Numerical Mathematics*, Vol. 82, 2014, pp. 80 – 114.

[63] Jung, S., Schwartzman, A., and Groisser, D., "Scaling-rotation distance and interpolation of symmetric positive-definite matrices," *SIAM Journal on Matrix Analysis and Applications*, Vol. 36, No. 3, 2015, pp. 1180–1201.

[64] Pennec, X., Fillard, P., and Ayache, N., "A Riemannian framework for tensor computing," *International Journal of Computer Vision*, Vol. 66, No. 1, 2006, pp. 41–66.

[65] Borouchaki, H., George, P.-L., Hecht, F., Laug, P., and Saltel, E., "Mailleur bidimensionnel de Delaunay gouverné par une carte de métriques. Partie I: Algorithmes," Tech. Rep. 2741, INRIA-Rocquencourt, France, 1995.

[66] Hartmann, R., *Adaptive finite element methods for the compressible Euler equations*, Ph.D. thesis, Ruprecht-Karls-Universität Heidelberg, 2002.

[67] Roca, X., Nguyen, N. C., and Peraire, J., "GPU-accelerated sparse matrix-vector product for a hybridizable discontinuous Galerkin method," *49th AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition*, Aerospace Sciences Meetings, American Institute of Aeronautics and Astronautics, January 2011.