Methods for Optimal Output Prediction in Computational Fluid Dynamics

by

Steven Michael Kast

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Aerospace Engineering)
in The University of Michigan
2016

Doctoral Committee:

Associate Professor Krzysztof J. Fidkowski, Chair
Assistant Professor Tan Bui-Thanh, University of Texas at Austin
Professor Robert Krasny
Professor Philip L. Roe

"So this is it," said Arthur, "We are going to die."

"Yes," said Ford, "except... no! Wait a minute!" He suddenly lunged across the chamber at something behind Arthur's line of vision."What's this switch?" he cried.

"What? Where?" cried Arthur, twisting round.

"No, I was only fooling," said Ford, "we are going to die after all."

DOUGLAS ADAMS
The Hitchhiker's Guide to the Galaxy

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my advisor, Professor Krzysztof Fidkowksi, for his support and guidance over the past few years. I am incredibly lucky to have had such an insightful and understanding mentor, and this work would not have been possible without him. His love of teaching and ability to explain almost anything are things that I will always admire. Thank you for making my time here truly enjoyable.

I would also like to thank my committee members: Professor Phil Roe, Professor Robert Krasny, and Professor Tan Bui-Thanh for taking the time to review this thesis and for providing helpful suggestions. To Professor Roe for his philosophical insight and historical perspective, to Professor Krasny for serving as cognate on short notice, and to Professor Bui-Thanh for his long-distance insight from Texas. I am also grateful for the support of Professor Smadar Karni.

I am indebted to Professor Leslie Olsen and Professor Pete Washabaugh, who taught the first Aerospace Engineering class I ever took and who have been continual sources of inspiration and help over the years. Finally, I would like to thank Professor Luis Bernal for providing me with my first taste of (experimental) research, even if I did switch over to the “dark side” of computation in the end.

The Aerospace Engineering staff also deserves a huge thanks. Denise Phelps has been a life-saver and has made my time here much more enjoyable. Chris Chartier has also been a frequent source of help and humor.

I would also like to thank my research group members, both past (Isaac Asher and Marco Ceze) and present (Guodong Chen, Johann Dahm, Kyle Ding, Devina Sanjaya, and Yukiko Shimizu). Grad school would have been much less fun without you. Special thanks is owed to Marco for showing me the ropes when I was new to the group and for being a constant source of both wisdom and humor. Thanks also to Johann for his friendship over the years, as well as for many interesting discussions and help related to both research and programming. Last but not least, thanks to Yuki for her encouragement and support, and for helping me see things from a new perspective.

My officemates, both past (Tim Eymann, Paul Giuliano, and Ashley Verhoff) and present (Sam Chen, Horatiu Dragnea, and Kyle Hanquist) also deserve thanks for sharing the same space peacefully and for being good friends. Thanks especially to Ashley for many good discussions, and to Kyle for his Nebraskan sense of humor, as well as for serving as intramural sports captain for several years running. To all members of the Aerospace intramural basketball, softball, soccer, and flag football teams, as well as to the frisbee and tennis groups – thank you; it's been fun.

To other friends over the years – Erin and Tony D'Amato, Luke Hansen, Lauren Mackey, Brandon Smith, Jon and Michelle Wiebenga, as well as Maria Choi (and Max), Doreen Fan, Tyler Lung, Chris Ngigi, Lu-Yin Wang, Daniel Zaide and others – thank you for all the good times. For those I have not mentioned, thank you as well – you know who you are.

Finally, I would like to thank my family. To my parents, Doug and Lynn, for their endless support and encouragement in too many ways to name. And to my brother John and sister Jenna, for leading the way in life and for always providing me with a good example.

Funding for this work was provided by a National Defense Science and Engineering Graduate (NDSEG) Fellowship sponsored by the Department of Defense High Performance Computing Modernization Program, as well as by a Rackham Predoctoral Fellowship.

TABLE OF CONTENTS

CHAPTER

LIST OF FIGURES

Figure

LIST OF TABLES

Table

LIST OF APPENDICES

Appendix

ABSTRACT

Methods for Optimal Output Prediction in Computational Fluid Dynamics
by
Steven M. Kast

Chair: Krzysztof J. Fidkowski

In a Computational Fluid Dynamics (CFD) simulation, not all data is of equal importance. Instead, the goal of the user is often to compute certain critical *outputs* – such as lift and drag – accurately. While in recent years CFD simulations have become routine, ensuring accuracy in these outputs is still surprisingly difficult. Unacceptable levels of output error arise even in industry-standard simulations, such as the steady flow around commercial aircraft. This problem is only exacerbated when simulating more complex, unsteady flows.

In this thesis, we present a mesh adaptation strategy for unsteady problems that can automatically reduce errors in outputs of interest. This strategy applies to problems in which the computational domain deforms in time – such as flapping-flight simulations – and relies on an unsteady adjoint to identify regions of the mesh contributing most to the output error. This error is then driven down via refinement of the critical regions in both space and time. Here, we demonstrate this strategy on a series of flapping-wing problems in two and three dimensions, using high-order discontinuous Galerkin (DG) methods for both spatial and temporal discretizations. Compared to other methods, results indicate that this strategy can deliver a desired level of output accuracy with significant reductions in computational cost.

After concluding our work on mesh adaptation, we take a step back and investigate another idea for obtaining output accuracy: adapting the numerical method itself. In particular, we show how the test space of discontinuous finite element methods can be "optimized" to achieve accuracy in certain outputs or regions. In this work, we compute test functions that ensure accuracy specifically along domain *boundaries*. These regions – which are vital to both scalar outputs (such as lift and drag) and distributions (such as pressure and skin friction) – are often the most important from an engineering standpoint.

CHAPTER I

Introduction

1.1 Motivation

With the growth in computational power over the past few decades, many of the experiments traditionally used in aircraft design are being replaced by Computational Fluid Dynamics (CFD) simulations. While CFD simulations can be faster, cheaper, and more flexible than experiments, their approximate nature inevitably prompts the question: are they *accurate*? More importantly, are they accurate where it matters most – in their prediction of critical quantities (or "outputs") such as lift and drag?

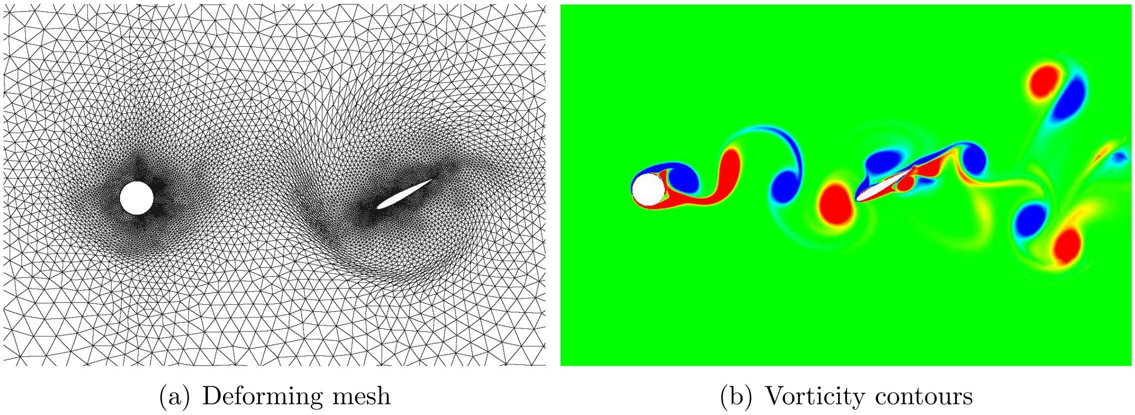
Unfortunately, for many practical problems, the answer is no. Addressing this issue is the motivating theme throughout this thesis.

Before attempting to improve output accuracy, we should first ask: what accuracy do we actually need? And how far are we from achieving it? Here, an example from aircraft design is illustrative.

Every three years, the American Institute of Aeronautics and Astronautics (AIAA) asks researchers from around the world to simulate the steady-state flow around a commercial aircraft. The groups then meet and compare the drag values predicted from their CFD simulations. Ideally, these values should agree. In reality, errors associated with the simulations lead to significant variations in the predicted drag. For example, at the 2006 workshop, variations in the drag of approximately 30 counts were observed (where one drag count corresponds to 10^{-4} of the drag coefficient) [40]. From a design perspective, a variation of just 1 drag count can translate into a difference of 4-8 passengers for a commercial aircraft with a fixed range and fuel capacity [94]. With the precarious state of modern airlines, this is clearly an unacceptable level of error.

Since 2006, the accuracy for these types of steady simulations has improved gradually. The latest DPW V results from 2012 show that the spread in drag values has been reduced to approximately 12 counts [93], though achieving even this level of accuracy can require significant supercomputing resources.

As computing power grows, however, another issue arises: the complexity of the simulations themselves is increasing proportionally. Engineers are expanding the envelope of simulations from steady-state problems to more complicated unsteady problems. Thus, even as the ability to compute accurate outputs improves for steady cases, the same issues arise for unsteady problems, often with even greater severity. These problems range from aircraft maneuver and flutter scenarios to low-Reynolds-number flapping-flight simulations. The latter simulations – which fall in the regime of bird or insect flight – are especially important due to their application in the emerging field of Micro Aerial Vehicles (MAVs) [90, 100, 89, 28, 76]. For these cases, obtaining accuracy in outputs such as the time-averaged lift-to-drag ratio or the peak power expenditure is critical for determining design feasibility.


(a) Deforming mesh

(b) Vorticity contours

Figure 1.1: A cylinder and airfoil pitching and plunging in series at a Reynolds number of 1000 and a Strouhal number of 0.2. ($Str = fc/U$, where f is the stroke frequency, c is the airfoil chord, and U is the freestream velocity.) The mesh must deform to capture the motion of the bodies, and the resulting complexity of the flow field makes accurate output prediction difficult.

Simulations of these problems – an example of which is shown in Fig. 1.1 – require the computational domain to deform in order to capture the motion of the objects in question. For this reason, they are referred to as ***deforming domain*** problems. Developing a strategy to reduce and control output errors for these problems is one of the primary goals of this thesis.

After completing our work on deforming domain problems, we then take a step back to investigate an alternative method for reducing output errors, which in theory could be applied to both steady and unsteady problems alike.

1.2 Numerical Errors

Before we can reduce output errors, we must first understand why they occur. One potential source of output error is *modeling* error, which arises if the equations solved on the computer do not faithfully represent reality. While modeling errors can be significant for certain simulations – particularly those involving turbulence[1] – for the problems considered here the Navier-Stokes equations provide an accurate description of the fluid flow. Instead, a more prevalent source of output error – and the focus of this work – is ***discretization*** error.

Discretization error, which is the difference between the exact and numerical solution to a differential equation, arises due to the discrete nature of both the computational mesh and the numerical method itself. During a numerical simulation, "truncation" errors are generated locally wherever the problem is under-resolved, leading to discretization errors that then propagate throughout the domain, eventually corrupting the values of relevant outputs.

One way to reduce discretization errors – and hence output errors – is to uniformly refine the mesh, either through h-refinement (i.e. subdividing the elements) or p-refinement (i.e. increasing the solution approximation order)[2]. Theoretically, in the limit of infinite mesh refinement, discretization errors would vanish and the outputs would approach their true values.

However, due to the large computational expense, for most problems this type of uniform mesh refinement is infeasible. Instead, for a given output of interest, a better strategy is to identify the regions of the mesh contributing most to the output error, and to then target only those regions for refinement. Performing this type of **output-based mesh adaptation** for deforming domain problems will be one of our primary goals in this work.

There is, however, a second option for reducing the error in certain outputs, which does not involve refining the mesh. Rather than attempting to reduce the *magnitude* of discretization errors (as with mesh refinement), we could instead attempt to *redistribute* the discretization errors in such a way that they have minimal influence on the outputs of interest. This idea forms the basis for the latter half of the thesis, and – in the context of finite element methods – will involve **optimizing** the finite element ***test space***.

[1] Note that some of the variation in the DPW results discussed earlier is due to the use of different turbulence models. However, a further study in [69] indicates that discretization errors contribute a significant amount to the total drag error in these cases.

[2] Assuming the numerical method supports variable orders of approximation.

Since finite element methods play a central role in this thesis, let us briefly comment on their properties. We will then return to a discussion of how the above options (mesh adaptation and test space optimization) can be used to reduce output errors in this context.

1.3 High-order Finite Element Methods

In this work, we employ high-order finite element methods of a discontinuous Galerkin (DG) type. To approximate the weak form of a partial differential equation (PDE), these methods first weight the PDE by a set of "test" functions on each element. They then define a set of "trial" functions on each element that are used to represent the numerical solution. These test and trial functions are taken to be identical, and are typically chosen such that the solution can be represented as a polynomial of arbitrarily high order, p. Between elements, the solution is allowed to be discontinuous and a numerical flux is defined on element faces. This provides stability for convection-dominated problems and makes these methods suitable for aerospace applications.

While second-order finite volume schemes remain the dominant methods within the aerospace industry, methods of a discontinuous Galerkin type have recently been gaining in popularity [80, 8, 22, 3, 50, 87]. Some of their relevant advantages include the fact that:

1. Their variational nature provides a rigorous setting for output error estimation.

2. Their outputs are typically superconvergent (at a rate of $2p$ for viscous problems and $2p + 1$ for inviscid problems) [1, 23, 2, 62].

3. Their ability to handle different solution orders on each element makes p-adaptation possible.

4. For smooth problems (such as low-Reynolds-number flows), increasing p yields an exponential rate of convergence with respect to degrees of freedom.

5. Their flexibility in the choice of test and trial functions means that we can attempt to optimize these functions to obtain improved accuracy.[3]

[3] Note that if the test and trial space are no longer identical after optimization, we have technically shifted to a Petrov-Galerkin method.

For these reasons and others, we employ discontinuous Galerkin methods (and related schemes) in this work.

We now discuss how the two strategies for reducing output error mentioned above – i.e. mesh adaptation and test space optimization – can be carried out in this context.

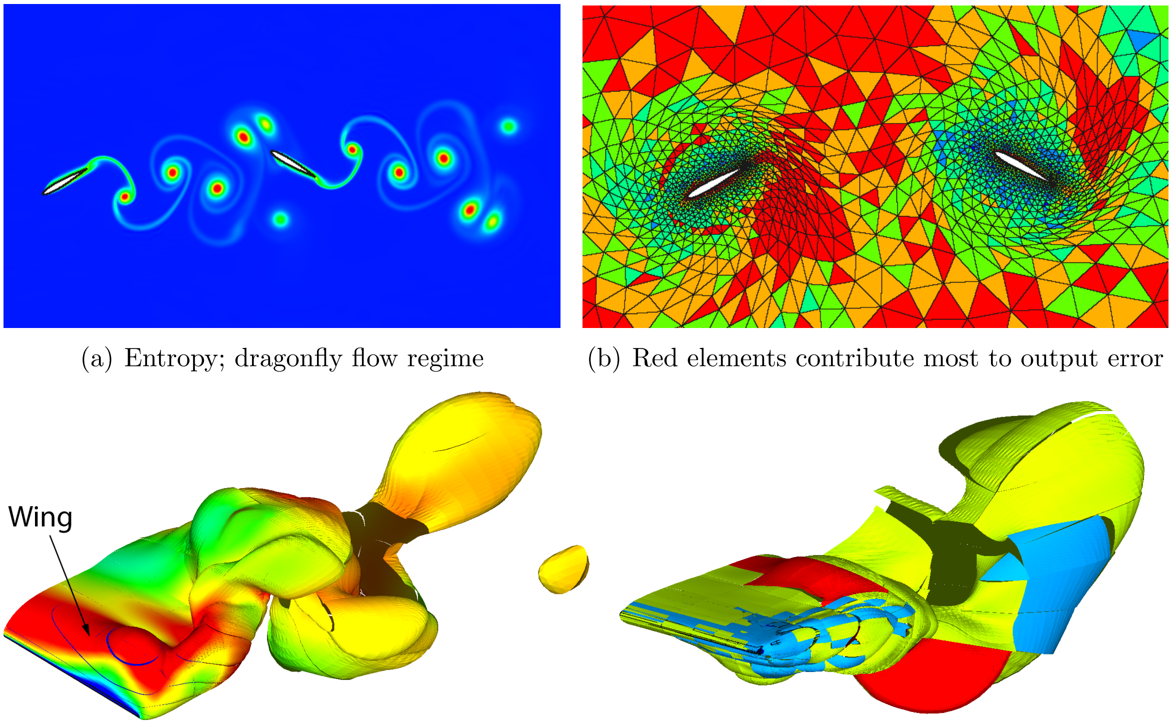
1.4 Output Error Estimation and Mesh Adapatation

The first option for achieving output accuracy – adapting the mesh – borrows concepts from control theory in an effort to reduce output errors. Given some output of interest, the idea is to identify the regions of the mesh contributing most to the output error. Once identified, these regions are refined (reducing their contribution to the error), and the problem is solved again. New output errors are then estimated, new regions of the mesh are refined, and the cycle repeats until the output error drops below a specified tolerance. Thus, the output error is effectively "controlled" through an automated, closed-loop process.

The key ingredient to this process is a sensitivity analysis – i.e. an analysis that identifies how sensitive the output is to perturbations made in different regions of the domain. This information can be obtained by solving an ***adjoint*** problem. The adjoint quantifies how perturbations in the local **residuals** (i.e., the continuous form of the governing equations) propagate to influence the output of interest [57, 43, 29]. In the context of error estimation, residual perturbations occur due to the presence of truncation errors. If the output happens to be sensitive to a region where the truncation errors / residuals are large, then this region should be refined, since it is likely contributing a significant amount of error to the output.

Use of an **adjoint-weighted residual** to identify sources of output error and drive mesh adaptation is not new, and has been pursued in a fluid dynamics context for over fifteen years. Notable works in this area incude those by Giles and Pierce [79], Giles and Suli [42], Becker and Rannacher [10], Hartmann and Houston [49], and Venditti and Darmofal [95], among others [6, 42, 36].

While most previous works have focused on steady-state problems [79, 10, 49, 95, 88, 74, 36, 18, 104, 19, 24, 101], recently, output-based strategies for unsteady problems have also been investigated. In a finite element context, output error estimation for scalar parabolic problems was studied in [70] and [86], with a high-order reconstructed adjoint used to drive dynamic space-time mesh adaptation. In addition, spatial-only [7] and combined space-time [12] adaptation have been performed for two-dimensional Navier-Stokes simulations on static domains. Within a finite volume framework, temporal-only adaptation has been shown for the Euler equations on deforming domains [67, 68], while on static domains a spatial-only [11] and preliminary space-time adaptation [38] have been demonstrated. Finally, in recent work [37, 66, 35, 103], combined space-time adaptation strategies have been presented for Euler and Navier-Stokes simulations on static domains. In each of the above works, improvements in output convergence were obtained through the use of unsteady output-based adaptation.



(a) Entropy; dragonfly flow regime

(b) Red elements contribute most to output error

(c) Entropy surf./Mach colors; housefly regime

(d) Red regions contribute most to output error

Figure 1.2: Snapshots of CFD simulations for (a) two-dimensional and (c) three-dimensional wings in flapping flight, taken from Chapter V. Parts (b) and (d) show the solution order distribution for meshes that have been adapted to achieve accurate lift outputs near the final time. (For (b), the output is the lift on the right airfoil.) Red corresponds to $p = 5$ while blue corresponds to $p = 1$. Initially, the red regions in (b) and (d) contribute the most to the output error, so they are refined to drive the output error down.

In this work, our goal is to extend these output-based techniques to problems with deforming domains, some examples of which are shown in Fig. 1.2. In particular, we wish to demonstrate a combined space-time adaptation strategy in which the spatial order distribution (p) varies dynamically in time, while the time step sizes are refined or coarsened as needed. This type of adaptation – which will be guided by an unsteady adjoint solution – will allow us to resolve features critical to the output accuracy as they propagate throughout the domain.

In addition, we will investigate several other aspects of these problems related to error estimation and mesh adaptation. For these cases, a so-called Geometric Conservation Law (GCL) must be satisfied in order to enforce conservation as the mesh deforms. For error estimation purposes, this GCL equation then requires a corresponding GCL adjoint. In this work, we investigate the relevance of both the GCL and its adjoint with regard to output accuracy, and test several approximations that can be made to the adjoint problem to improve computational efficiency. In the end, we show that the output-based technique can lead to significant computational savings – in terms of both mesh size and run time – compared to more standard refinement techniques.

1.5 Test Space Optimization

The strategy described above employs a standard numerical method (in this case a DG method) and attempts to improve output accuracy through "optimization" of the mesh alone. However, as mentioned, an alternative strategy is to try to optimize the numerical method *itself*. By doing so, we can attempt to redistribute the discretization errors in a way that is favorable to output accuracy. In the second part of this thesis, we take a step back and investigate this alternative option.

What benefit is there to optimizing the method rather than the mesh? The first answer is that these ideas are not mutually exclusive: the ideal strategy may involve a combination of both. However, theoretically, "method optimization" alone could provide an advantage over mesh adaptation. Since mesh adaptation generally requires refining the mesh to reduce output errors, the computational expense increases as the adaptation proceeds. On the other hand, if we were able to optimize the method itself, output accuracy could be obtained even on a coarse (i.e. inexpensive) mesh.

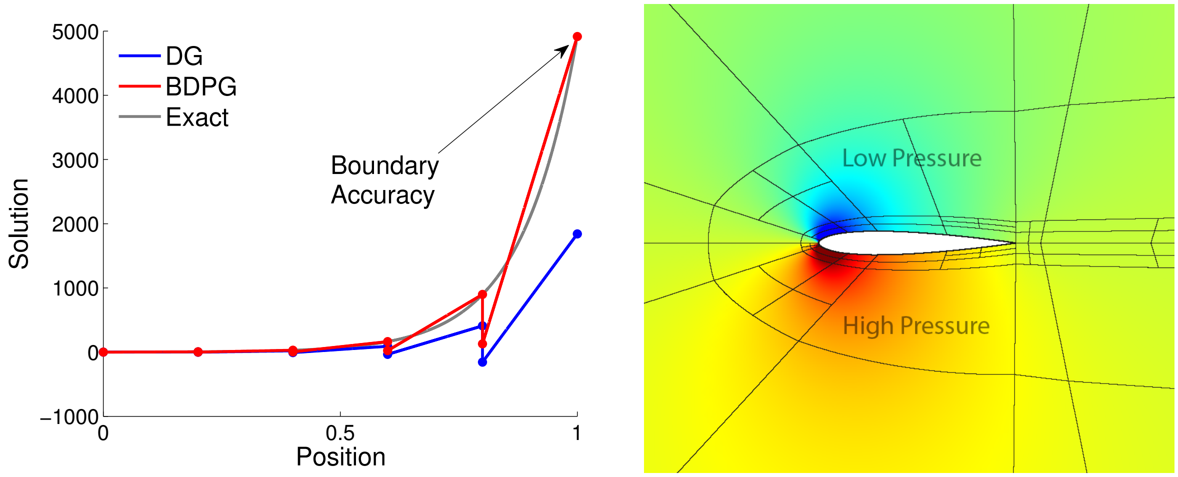
In the context of finite element methods, we can be more specific about what "method optimization" means. As mentioned, finite element methods approximate the weak form of a PDE and consist of two main components: (1) the *trial* space, which determines how the solution is represented within each element, and (2) the *test* space, which defines the functions that weight the PDE in the weak form.

While standard Galerkin methods choose the test and trial functions to be identical, this choice is not necessarily optimal, since the the role of these functions is different. If we think of our numerical solution as attempting to provide a good "fit" of the true solution, then while the trial functions determine the type of curve used to perform the fit (e.g. a polynomial of order p), the test functions determine the *goal* of the fit itself. In other words, they determine the *type* of fit obtained, which could be (say) a least-squares fit or a fit based on some other norm. Thus, if our goal is to achieve accuracy in certain outputs or regions within the domain, we can attempt to adjust how the numerical solution fits the exact solution to emphasize accuracy in these regions. Since this fit is ultimately controlled by the test space, it is the test space that should be optimized.

In general, we can seek to optimize the test space such that the numerical solution achieves accuracy in a certain *norm* of interest. This norm can be defined to weight certain regions or quantities more heavily than others, and the optimal test functions can be defined as those that render the numerical solution the best approximation in the desired norm.

The idea of optimizing the test space has surfaced several times over the last few decades, in various contexts. In a continuous finite element context, this idea has been pursued by several authors [5, 27, 44, 17, 51, 4], dating back to the work of Barrett and Morton in 1984 [5]. In addition, the test functions of stabilized schemes such as the Streamline Upwind Petrov-Galerkin (SUPG) method (which achieves H^1 optimality for certain problems [55]) can be viewed as optimal in a similar sense. More recently, Demkowicz and Gopalakrishnan have introduced discontinuous Petrov-Galerkin (DPG) methods, which employ optimal test functions within a more general, discontinuous framework [25, 26, 105, 20, 16]. These methods, developed initially within an "ultra-weak" [26] context and adapted to hybrid methods in [71], have interior least-squares (L^2) optimality as their primary goal.

In the present work, we pursue a different goal. We note that while domain-interior accuracy is important, from an engineering standpoint the regions of greatest interest are often the domain *boundaries*. Indeed, obtaining the forces, fluxes, and distributions of quantities along the boundaries is often the principal goal of a simulation. We therefore make this our aim, and we pursue this aim within the context of standard DG and hybrid DG (HDG) methods [75, 87]. When the optimal test functions are employed within this context, we call the resulting method the boundary discontinuous Petrov-Galerkin (BDPG) method. A preview of this BDPG method is provided in Fig. 1.3, which illustrates the potential boundary accuracy that can be achieved.



(a) $p = 1$ DG and BDPG solutons for a one-dimensional advection-reaction problem

(b) Linearized Euler pressure field around airfoil, computed with BDPG method

Figure 1.3: (a) The goal of BDPG (with optimal test functions) is to achieve greater boundary accuracy than a standard DG method, as can be seen for this one-dimensional problem. (b) We also extend these ideas to more practical problems, such as for flow around airfoils.

1.6 Thesis Objectives

Here, we summarize the primary goals of this thesis, broken into two main parts:

1. **Unsteady Output-based Mesh Adaptation on Deforming Domains**

   (a) Extend output-based error estimation and mesh adaptation techniques to unsteady problems on deforming domains.

   (b) Perform dynamic-p adaptation in space combined with time step refinement/coarsening.

   (c) Evaluate the performance of the output-based strategy on low-Reynolds-number flapping-flight problems, using DG methods in both space and time. Compare to residual-based and uniform-refinement strategies.

   (d) Investigate the need for satisfying a Geometric Conservation Law (GCL), which helps ensure conservation on deforming domains.

   (e) Investigate means of efficient adjoint computation for error estimation, including use of inexact smoothing, reconstruction, and computing (or not computing) an adjoint for the GCL.

2. **Test Space Optimization for Finite Element Methods**

(a) Derive optimal test functions for finite element methods that reduce the error in outputs and distributions (such as pressure distributions) defined along the *boundaries* of the computational domain.

(b) Draw connections between optimal test functions and other ideas, such as adjoints, error estimation, multiscale methods, and numerical upwinding.

(c) Show how these test functions can be employed within both DG and HDG discretizations, resulting in a new boundary discontinuous Petrov Galerkin (BDPG) method.

(d) Investigate the ability of this BDPG method to provide accuracy in boundary outputs and distributions for both linear and nonlinear steady-state problems.

(e) Compare the effectiveness of BDPG to standard DG/HDG methods and discuss benefits, limitations, and future work.

As we will show, the connection between optimal test functions and output-based mesh adaptation runs deeper than the fact that both strategies attempt to improve output accuracy. Indeed, it turns out that both output-based adaptation ***and*** optimal test functions depend in a fundamental way on the concept of an adjoint vector. For this reason, the following chapter (Chapter II) provides a relatively comprehensive review of adjoint vectors in both discrete and continuous contexts.

In Chapter III, we discuss the extension of the discrete adjoint to unsteady problems. Chapter IV then gives a brief overview of the spatial and temporal discretizations used in this work, while Chapter V presents the output-based mesh adaptation strategy for deformable domains. In Chapter VI, we return to steady problems and discuss the theory and implementation of optimal test functions. Finally, conclusions and suggestions for future work are given in Chapter VII.

CHAPTER II

Adjoint and Error Estimation Review

In this chapter, we provide an overview of adjoints and their relationship to error estimation within the context of partial differential equations. While these ideas have been discussed extensively in previous works (see e.g. [10, 49, 43, 29]), a review of the concepts here will serve as the foundation for much of the work in this thesis.

2.1 Discrete Adjoints

Imagine we have a differential equation of the form

$$Lu = f\,, \tag{2.1}$$

where L is some linear differential operator (e.g. $L \equiv \frac{\partial}{\partial x}$ or $L \equiv \nabla^2$), f is a prescribed source term, and u is the unknown solution, defined on some domain Ω. For most operators L (combined with appropriate boundary conditions), this equation would be difficult to solve analytically, and we must instead approximate the solution numerically.

After discretizing the above equation with an appropriate numerical method (such as a finite difference or finite element method), we will arrive at a system of equations of the form

$$\mathbf{A}\mathbf{U} = \mathbf{F}\,, \tag{2.2}$$

where $\mathbf{A} \in \mathbb{R}^{N\times N}$ is a matrix representing the operator L, $\mathbf{U} \in \mathbb{R}^N$ is a vector of unknowns representing u, and $\mathbf{F} \in \mathbb{R}^N$ is a vector of source terms and boundary data.

Now, if we are interested in computing the *entire* solution $\mathbf{U}$ – i.e. in finding all components of the $\mathbf{U}$ vector – then we will effectively need to know the entire $\mathbf{A}^{-1}$

matrix, since we would then compute $\mathbf{U}$ from

$$\mathbf{U} = \mathbf{A}^{-1}\mathbf{F}\,. \tag{2.3}$$

However, what if, instead of the entire solution, we are interested in just a single *component* of $\mathbf{U}$? In that case, would we still need to know the entire $\mathbf{A}^{-1}$ matrix, or would less information suffice?

For example, imagine that we are interested in computing (say) U_N, the last entry in the $\mathbf{U}$ vector. This situation is depicted in the diagram below.

$$\underbrace{\begin{bmatrix} U_1 \\ U_2 \\ \vdots \\ \boxed{U_N} \end{bmatrix}}_{\mathbf{U}} = \underbrace{\begin{bmatrix} \bullet & \bullet & \cdots & \bullet \\ \bullet & \bullet & \cdots & \bullet \\ \vdots & \vdots & & \\ \boxed{\bullet \quad \bullet \quad \cdots \quad \bullet} \end{bmatrix}}_{\mathbf{A}^{-1}} \underbrace{\begin{bmatrix} F_1 \\ F_2 \\ \vdots \\ F_N \end{bmatrix}}_{\mathbf{F}} \tag{2.4}$$

By the properties of matrix multiplication, it is clear that to compute U_N we require knowledge of only a single *row* of $\mathbf{A}^{-1}$. (In this case, the last row – as highlighted above.) This row, when combined with the source term $\mathbf{F}$, contains all the information we need to compute our desired “output” U_N.

Now, for our purposes, there is another property of this row that is even more important. Imagine, for example, that the first component of the highlighted row were zero. Then if we were to change the source term F_1, the value of U_N would not change at all (since F_1 would just be multiplied by 0 during the computation of U_N). In other words, we could say that our output U_N is *insensitive* to changes in F_1. In a similar manner, this logic could be applied to all entries in the highlighted row: the smaller a given entry, the less sensitive U_N is to changes in the corresponding F component, and likewise, the larger a given entry, the more sensitive it is. In summary then, not only does the highlighted row provide the information required to compute U_N, it also provides the ***sensitivity*** of U_N to perturbations in the source terms F.

This row then – this sensitivity vector – is what is commonly referred to in the literature as the “**dual vector**,” the “**output adjoint vector**,” or simply, the “**adjoint**.” For a given output of interest (in this case U_N), it provides the sensitivity of that output to perturbations in the source terms of the governing equations.

In this work, we will denote the adjoint vector by the symbol $\boldsymbol{\Psi}$ and the corresponding “**output of interest**” by the symbol J, as depicted in the diagram below.

$$\text{Output } J \quad \underbrace{\begin{bmatrix} U_1 \\ U_2 \\ \vdots \\ \boxed{U_N} \end{bmatrix}}_{\mathbf{U}} = \underbrace{\begin{bmatrix} \bullet & \bullet & \dots & \bullet \\ \bullet & \bullet & \dots & \bullet \\ \vdots & \vdots & & \\ \multicolumn{4}{c}{\boxed{\bullet \quad \bullet \quad \dots \quad \bullet}} \end{bmatrix}}_{\mathbf{A}^{-1}} \underbrace{\begin{bmatrix} F_1 \\ F_2 \\ \vdots \\ F_N \end{bmatrix}}_{\mathbf{F}} \quad \text{Adjoint } \boldsymbol{\Psi}^T \tag{2.5}$$

Note that we will use $\boldsymbol{\Psi}^T$ to refer to the adjoint in row-vector form, whereas $\boldsymbol{\Psi}$ alone will denote the adjoint in column form.

Using the above notation, an important point is that the output J can be written as the inner product between the adjoint and the source vector, i.e. as

$$\boxed{J = \boldsymbol{\Psi}^T \mathbf{F}} \; . \tag{2.6}$$

This is known as the "**dual form**" of the output, and is just a mathematical restatement of our earlier claim that the only information required to compute the output is the highlighted row and the source vector $\mathbf{F}$.

The next question is: how should we define the adjoint itself? So far, we have just labeled it as the last row of $\mathbf{A}^{-1}$, but is there a way to define it mathematically?

It turns out that, in order to define the adjoint formally, it will be helpful to first compute the derivative of the output J with respect to $\mathbf{U}$. In this case, since $J = U_N$, we can write its derivative with respect to $\mathbf{U}$ as

$$\frac{\partial J}{\partial \mathbf{U}} \equiv \left[\frac{\partial J}{\partial U_1} \quad \frac{\partial J}{\partial U_2} \; \dots \; \frac{\partial J}{\partial U_N} \right] = [0 \;\; 0 \; \dots \; 1] \; . \tag{2.7}$$

This is just the Cartesian row vector with the last entry nonzero. In the present case, we see that the adjoint $\boldsymbol{\Psi}^T$ can then be defined as

$$\boxed{\boldsymbol{\Psi}^T = \frac{\partial J}{\partial \mathbf{U}} \mathbf{A}^{-1}} \; . \tag{2.8}$$

Here, multiplying $\mathbf{A}^{-1}$ by the Cartesian row vector simply picks off the last row of $\mathbf{A}^{-1}$ and calls it the adjoint, $\boldsymbol{\Psi}^T$.

So far, we do not seem to have gained much from the above derivations. However, an important fact is that, while this $J = U_N$ example may seem trivial, both Eqn. 2.8 (the so-called "**adjoint equation**") and Eqn. 2.6 (the "**dual form**") hold *regardless* of what the desired output is.[1] For example, rather than a "single-component" output like $J = U_N$, we might instead be interested in computing an average or sum of certain components of $\mathbf{U}$. One possibility would be the average of all components, i.e.

$$J = \frac{1}{N}\sum_{i=1}^{N} U_i \ . \tag{2.9}$$

In that case, we would have

$$\frac{\partial J}{\partial \mathbf{U}} \ = \ \frac{1}{N}\,[1 \ \ 1 \ ... \ 1] \ , \tag{2.10}$$

and from Eqn. 2.8, applying this vector to $\mathbf{A}^{-1}$ would then tell us that the adjoint of J (i.e. its sensitivity to perturbations in the source terms) is just the corresponding average of all rows of $\mathbf{A}^{-1}$.

In the same way, the sensitivity of *any* scalar output can be represented as a weighted average of the rows of $\mathbf{A}^{-1}$, in accordance with Eqn. 2.8. As we will see, in a CFD simulation where $\mathbf{U}$ may represent the density or momentum of a fluid, these scalar outputs will tend to be quantities of physical importance, such as lift, drag, moment, or heat flux.

2.1.1 Alternative (CFD) Notation

Before moving on, let us rewrite the adjoint equation (Eqn. 2.8) in a form typically seen in Computational Fluid Dynamics. In CFD, the governing equations

$$\mathbf{A}\mathbf{U} = \mathbf{F}$$

(i.e. Eqn. 2.2), would typically be written as a set of discrete "residuals," where the residual vector $\mathbf{R} \in \mathbb{R}^N$ is defined as:

$$\mathbf{R}(\mathbf{U}) \equiv \mathbf{A}\mathbf{U} - \mathbf{F} \, . \tag{2.11}$$

The governing equations are therefore satisfied when the residual vector is zero.

[1] Strictly speaking, Eqn. 2.6 (the dual form) holds only if J is a *linear* combination of the components of $\mathbf{U}$, though Eqn. 2.8 (the adjoint equation) holds even in the nonlinear case, since the adjoint is always defined to be a linear sensitivity vector.

Taking the derivative of $\mathbf{R}$ with respect to $\mathbf{U}$, we see that

$$\frac{\partial \mathbf{R}}{\partial \mathbf{U}} = \mathbf{A}\,. \tag{2.12}$$

Thus, we can say that the $\mathbf{A}$ matrix corresponds to the "residual Jacobian" matrix, $\partial \mathbf{R}/\partial \mathbf{U}$. Inserting this Jacobian matrix into the adjoint equation (Eqn. 2.8) then gives

$$\boxed{\boldsymbol{\Psi}^T = \frac{\partial J}{\partial \mathbf{U}} \frac{\partial \mathbf{R}}{\partial \mathbf{U}}^{-1}}\,. \tag{2.13}$$

To obtain a more common form of this equation, we transpose both sides to get

$$\boldsymbol{\Psi} = \frac{\partial \mathbf{R}}{\partial \mathbf{U}}^{-T} \frac{\partial J}{\partial \mathbf{U}}^{T}\,. \tag{2.14}$$

Finally, multiplying both sides by the Jacobian transpose gives

$$\boxed{\frac{\partial \mathbf{R}}{\partial \mathbf{U}}^{T} \boldsymbol{\Psi} = \frac{\partial J}{\partial \mathbf{U}}^{T}}\,. \tag{2.15}$$

This is the adjoint equation often seen in the literature.[2]

By comparing both sides of this equation, we see that in order for the left-hand side to have dimensions of $\partial J/\partial \mathbf{U}$, the adjoint must behave like

$$\boldsymbol{\Psi} \approx \frac{\partial J}{\partial \mathbf{R}}\,. \tag{2.16}$$

Thus, in general, we can say that the adjoint represents the **sensitivity** of an output to perturbations in the **residuals**. (Note that previously we described the adjoint in terms of perturbations to $\mathbf{F}$, but since perturbations in $\mathbf{F}$ lead directly to perturbations in the residuals, these are different ways of saying the same thing.[3])

In the end, Eqn. 2.15 is same as that in the previous section (Eqn. 2.8), but with one additional benefit. By writing the adjoint equation in terms of a residual Jacobian matrix as opposed to an $\mathbf{A}$ matrix, we have effectively extended its definition to problems with *nonlinear* residuals as well. In that case, although we could not write the residual itself as $\mathbf{R} = \mathbf{A}\mathbf{U} - \mathbf{F}$, we *could* still compute its derivative $\partial \mathbf{R}/\partial \mathbf{U}$

[2] Note that the adjoint equation is sometimes defined with a negative sign on $\frac{\partial J}{\partial \mathbf{U}}^T$. This is a convention often used within the field of optimization. In that case, the adjoint would represent the output sensitivity to perturbations on the left-hand rather than right-hand side of the residuals.

[3] Except for a difference in sign, since $\mathbf{F}$ enters the residual with a negative sign.

about a given location in state space (just as we could compute the derivative of any nonlinear function – say, a quadratic – at a given point in space). From Eqn. 2.15, this first derivative – this Jacobian matrix – is all that is required to define the adjoint.

Thus, for **nonlinear** problems, we would write the adjoint equation as

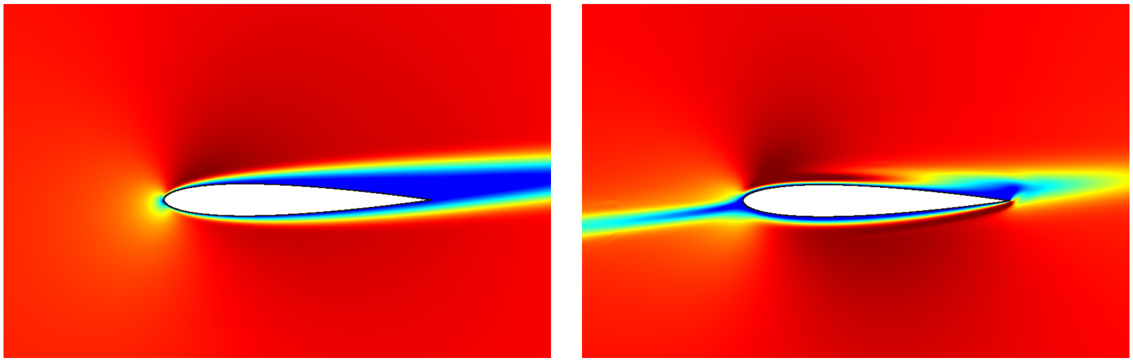
$$\boxed{\left.\frac{\partial \mathbf{R}}{\partial \mathbf{U}}^T\right|_{\mathbf{U}} \mathbf{\Psi} = \left.\frac{\partial J}{\partial \mathbf{U}}^T\right|_{\mathbf{U}}} \quad . \tag{2.17}$$

where the Jacobian and output linearization are evaluated at a particular state, $\mathbf{U}$.

Finally, we note that, in general, there are several means by which $\frac{\partial \mathbf{R}}{\partial \mathbf{U}}$ and $\frac{\partial J}{\partial \mathbf{U}}$ can be computed. These include finite differencing of the discrete residuals and output, automatic code differentiation, and analytical differentiation. In this work, we rely primarily on analytical differentiation, which – despite the effort involved – typically results in a more accurate and efficient implementation.

2.1.2 Adjoint Example

At this point, it is worth giving a practical example. Here, we show results from a simulation of the Navier-Stokes equations around an airfoil. In this case, the flow moves upward and to the right at an angle of attack of 5° and a Reynolds number of 5000, with contours of the x-direction momentum shown in Fig. 2.1a. A low-velocity wake forms behind the airfoil, as indicated by the blue region in the figure.



(a) x-momentum of fluid

(b) cons. of x-momentum drag adjoint

Figure 2.1: Contours of (a) x-momentum ([blue, dark red] = [-0.035, 0.55]) and (b) the conservation of x-momentum component of the drag adjoint ([blue, dark red] = [-1.05, 0.15]) for Re = 5000 flow around an airfoil. The drag is most sensitive to x-momentum residual perturbations made in the dark blue/red regions of (b).

If we are interested in the drag on the airfoil, we can define this drag to be our "output" J and can compute a corresponding adjoint from Eqn. 2.17. Looking back at the equations above (e.g. Eqn. 2.5) it is clear that the adjoint is always a vector of the same dimension as the state vector, $\mathbf{U}$. Thus, if we can plot contours of $\mathbf{U}$, we can plot contours of $\boldsymbol{\Psi}$ as well.[4] These contours are shown in Fig. 2.1b. (Specifically, we plot the "conservation of x-momentum" component of the adjoint, which represents the sensitivity of the drag to perturbations in the x-momentum residuals. The adjoint also has conservation of mass, y-momentum, and energy components.)

The dark blue and dark red regions in Fig. 2.1b are those of strong positive and negative output sensitivity, respectively. Any residual perturbations made in these regions would therefore have a large effect on the final drag value, whereas perturbations made in the lighter red regions (where the adjoint is near zero) would have a small influence on the drag.

One interesting feature of the adjoint is the blue region extending leftward from the airfoil leading edge. This adjoint "wake" travels in the opposite direction to the fluid wake, and is representative of a general feature of adjoints: namely, that information in the adjoint problem flows in the ***opposite*** direction as information in the original ("primal") problem. While we will discuss this in more detail later, here, the existence of an adjoint wake on the left makes sense, since any source perturbations made in this region will flow rightward and collide with the airfoil, ultimately having a strong influence on the drag value.

Finally, one potentially suprising fact about the adjoint is that it is ***smooth***. It is not obvious that this should be the case. Upon plotting its contours, we may have expected to see more or less random "spikes" in sensitivity. However, this smoothness can be explained by the fact that, in the end, perturbations in the residuals propagate via physical mechanisms (e.g. acoustic waves or convective motion) on their way to influencing the output. Since most physical modes of propagation are by nature "smooth," this smoothness is reflected in the adjoint. Indeed, as we will discuss later, it turns out that just as the discrete system $\mathbf{AU} = \mathbf{F}$ approximates a continuous (i.e. smooth) primal PDE, the discrete adjoint equations – if properly posed – approximate a **continuous *adjoint* PDE**.

[4] Note that we are using a finite element formulation here, so plotting $\boldsymbol{\Psi}$ requires first multiplying it by the set of finite element basis functions. For non-variational methods the components of $\boldsymbol{\Psi}$ may have to be scaled in other ways before plotting.

2.1.3 Applications of Adjoints

While the idea of an adjoint vector is relatively straightforward, this concept lies at the heart of several active fields of research, including design optimization, mesh adaptation, error estimation, data assimilation, and uncertainty quantification. In this section, we discuss briefly the role of adjoints in the context of design optimization and mesh adaptation.

2.1.3.1 Design Optimization

In the field of optimization, the goal is typically to minimize or maximize some output of interest by systematically modifying (or “optimizing”) certain parameters of the problem at hand. For example, in aircraft design, the goal may be to minimize the drag of an aircraft by finding the optimal fuselage and/or wing shape at a given angle of attack.

For these problems, where “**gradient-based**” techniques are often employed, the adjoint plays a critical role. This is because any changes in the design parameters (e.g. the aircraft geometry) would result in corresponding changes to the residuals of the governing equations. But since the adjoint provides the sensitivity of an output to perturbations in the residuals, it therefore indicates whether a given design change would cause the output value to increase or decrease. (Most importantly, it provides this sensitivity without any additional solves of the primal problem, which would otherwise be required if a “forward” sensitivity method were used.) Thus, if the goal is to e.g. minimize drag, design changes can be made that – according to the adjoint – lead to a reduction in drag. By iterating this process until the design parameters converge, the “optimal” design that minimizes drag[5] can be found.

2.1.3.2 Error Estimation and Mesh Adaptation

Just as design modifications can be treated as perturbations to the residuals, so can numerical errors. In a CFD simulation, the solution on a given mesh will not satisfy the differential (i.e. “continuous”) form of the governing equations exactly. Instead, small **truncation errors** will exist throughout the domain, which can be viewed as source-term perturbations to the local (continuous) **residual**. For a given output of interest, the adjoint then indicates how these residual perturbations translate into errors in the final output value.

[5] At least in a local sense – a global minimum is not guaranteed.

Furthermore, in addition to providing an estimate for the *total* output error, the adjoint also indicates *where* in the domain this error originates from. Roughly speaking, if truncation errors (i.e. residual perturbations) occur in a region where the adjoint is large, then these perturbations are likely contributing a significant amount to the output error. To reduce the output error, we should therefore attempt to reduce these residual perturbations, which can be done via local mesh refinement. Thus, the adjoint – or more accurately, the adjoint multiplied by the local residuals – can serve as an effective **mesh adaptation indicator**. This "**adjoint-weighted residual**" method forms the basis for much of the work in this thesis.

2.2 Continuous Adjoints

In the above section, we have given an overview of the discrete adjoint and its applications within Computational Fluid Dynamics. As mentioned, the discrete adjoint – while useful in its own right – turns out to be an approximation to a more fundamental concept: the continuous adjoint. In this section, we provide an overview of continuous adjoints and their associated operators, which will play a prominent role in the latter half of the thesis.

To introduce the relevant ideas, let us return to the differential equation from the previous section:

$$Lu = f\,, \tag{2.18}$$

where L is a linear differential operator, u is the unknown solution, f is a source term, and suitable boundary conditions are imposed.

To make the discussion more concrete, we will initially take $L \equiv a\frac{\partial()}{\partial x}$ and consider the following advection equation,

$$a\frac{\partial u}{\partial x} = f(x) \qquad\qquad x \in \Omega \tag{2.19}$$

$$u = 0 \qquad\qquad x = x_L\,, \tag{2.20}$$

where a is a positive scalar and a homogeneous Dirichlet condition is imposed on the left side of the domain, defined as $\Omega \equiv [x_L, x_R]$. Here, since the advection speed a is positive, information propagates from left to right, making the specification of a boundary condition on the left well-posed.

Now, imagine that, as in the previous section, we are interested not in solving for the *entire* solution u, but instead in computing a certain scalar **output** (or "functional") of the form:

$$\boxed{J = \int_{\Omega} g(x)\,u(x)\,dx \;\equiv\; (g,u)} \;. \tag{2.21}$$

For example, if – as in the discrete adjoint section – we are interested in computing the value of u at a single point (say, x_p) then we would choose $g(x) \equiv \delta(x - x_p)$. By the properties of the Dirac delta function, our output of interest would then be

$$J = \int_{\Omega} g(x)\,u(x)\,dx \;=\; \int_{\Omega} \delta(x - x_p)\,u(x)\,dx \;=\; u(x_p)\,. \tag{2.22}$$

Alternatively, if we were interested in computing (e.g.) the average of the solution over the domain, we would simply choose $g(x) \equiv 1$.

For any desired choice of the output J (and hence g)[6] the continuous adjoint[7] for that output would satisfy the following differential equation:

$$-a\frac{\partial \psi}{\partial x} = g(x) \qquad\qquad x \in \Omega \tag{2.23}$$

$$\psi = 0 \qquad\qquad x = x_R\,. \tag{2.24}$$

Here, we have labeled the **continuous adjoint** 'ψ', similar to the notation for the discrete adjoint, $\boldsymbol{\Psi}$.

Looking at the above **adjoint equation** (Eqn. 2.23), we see that it is very similar to the primal equation (Eqn. 2.19), with the only differences being the presence of $g(x)$ rather than $f(x)$, the inclusion of a negative sign in front of a, and the imposition of a boundary condition on the *right* rather than the left side of the domain. (Note that these last two differences indicate a **reversal** in the direction of **information flow** compared to the primal problem – as mentioned in Sec. 2.1.2 above.)

Now, what is the significance of this equation, and why should we call ψ an "adjoint"? Consider again the definition of our desired output, from which – in light

[6] Except $g = \delta(x - x_R)$, which lies on the boundary.

[7] Note that the continuous adjoint is sometimes referred to as a ***generalized Green's function*** due to its similarity to standard Green's functions [30].

of Eqn. 2.23 – we can perform the following manipulations:

$$\begin{aligned}
J &= \int\limits_{\Omega} g(x)\, u\, dx && \text{(output definition)} \\
&= \int\limits_{\Omega} -a \frac{\partial \psi}{\partial x}\, u\, dx && \text{(by adjoint equation, Eqn. 2.23)} \\
&= \int\limits_{\Omega} \psi\, a \frac{\partial u}{\partial x}\, dx - a\psi u \Big|_{x_L}^{x_R} && \text{(integrate by parts)} \\
&= \int\limits_{\Omega} \psi\, a \frac{\partial u}{\partial x}\, dx && \text{(ψ and u BCs $= 0$)} \\
&= \int\limits_{\Omega} \psi\, f(x)\, dx\,. && \text{(by primal equation, Eqn. 2.19)}
\end{aligned} \tag{2.25}$$

Or, using bracket notation to denote the L^2 inner product, we can write simply

$$J = (\psi, f) \ . \tag{2.26}$$

This is the so-called ***dual form*** of the output, which is analogous to the one derived in the discrete case (i.e. Eqn. 2.6). As in the discrete case, it says that the only information required to compute the desired output is the adjoint ψ and the source term f. Furthermore, just as the discrete adjoint $\mathbf{\Psi}$ represents the sensitivity of an output to perturbations in the discrete source term $\mathbf{F}$, the continuous adjoint ψ represents the sensitivity of an output to perturbations in the *continuous* source term f.

For the current problem, the continuous adjoint associated with the output $J = u(x_p)$ is plotted in Fig. 2.2. For this output, we have $g = \delta(x - x_p)$, so the solution to the adjoint equation (Eqn. 2.23) is just the step function.[8] From the figure, we see that ψ is nonzero only in the region upstream of x_p, since any perturbations made downstream of x_p cannot influence the output value. This confirms our earlier statement that ψ should represent the sensitivity of the output to perturbations made in various regions of the domain.

2.2.1 Generalization of the Continuous Adjoint

Now, let us generalize the continuous adjoint to cases beyond advection. Looking back at the dual form derivation (Eqn. 2.25), we see that it hinges on the action

[8] Note that the "continuous" adjoint is not necessarily a continuous function itself.

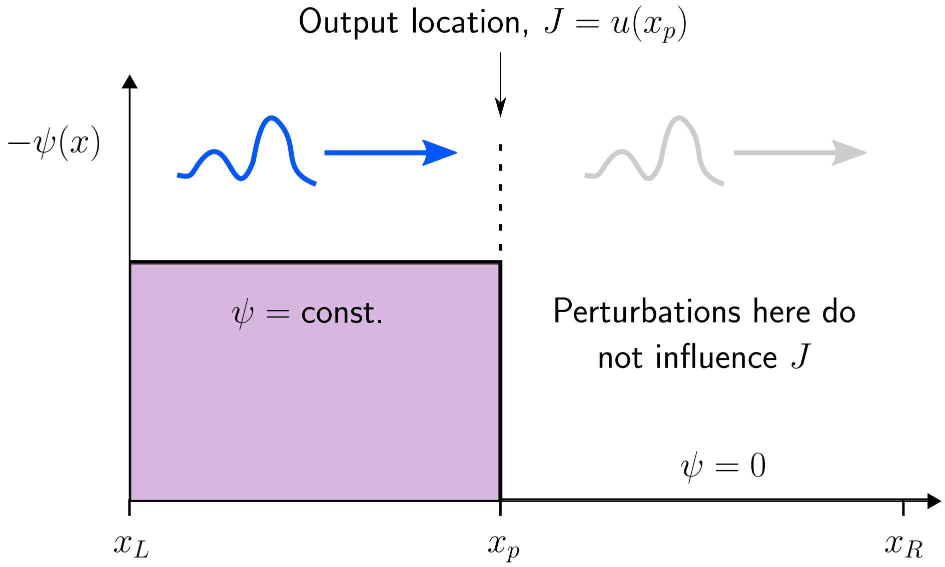

Output location, $J = u(x_p)$
$-\psi(x)$
$\psi =$ const.
Perturbations here do not influence J
$\psi = 0$
x_L
x_p
x_R

Figure 2.2: 1D advection: The adjoint associated with the output $J = u(x_p)$. Here, primal source perturbations (indicated by the squiggle) travel rightward, so only perturbations made to the left of x_p can influence the output value. Thus, for $x > x_p$, $\psi = 0$. Furthermore, since all perturbations made in $x_L < x < x_p$ will propagate to x_p eventually, the output is equally sensitive to all of them, meaning $\psi =$ constant in this region.

performed between steps 2 and 3 – i.e. the integration by parts. The key point is that the operator acting on the adjoint ψ should, after integration by parts, recover the primal differential operator. (E.g., in this case, the $-a\frac{\partial \psi}{\partial x}$ term yields the primal term $a\frac{\partial u}{\partial x}$ after integration by parts.)

In general, let us denote the operator acting on ψ as L^*. Then, for a given primal operator L, we need this L^* operator to satisfy the following identity:

$$\int_\Omega (Lu)\,\psi\,dx \;=\; \int_\Omega u\,(L^*\psi)\;dx \qquad\qquad \forall\,\text{suitable } u, \psi \tag{2.27}$$

This says that, if we have a differential term Lu integrated against a function ψ, then L^* is the operator that acts on ψ after we integrate by parts enough times to remove all derivatives from u.[9] (As suggested above, the integration by parts can also be thought of as occuring in the reverse direction, from L^* to L. However, since we typically know L from the primal equation, in practice we integrate the left-hand side

[9] Note that all boundary terms vanish in the integration by parts since we assume homogeneous boundary conditions on u and ψ.

of Eqn. 2.27 by parts in order to determine L^*.)

For example, if $L \equiv \frac{\partial()}{\partial x}$, only one integration by parts in Eqn. 2.27 is required to remove the derivative on u. Since this integration by parts introduces a negative sign, the L^* in Eqn. 2.27 then turns out to be $L^* = -L \equiv -\frac{\partial()}{\partial x}$. However, if we instead had the operator $L \equiv \frac{\partial^2()}{\partial x^2}$, then two integrations by parts would be required, and the two subsequent negative signs would cancel, yielding $L^* = L \equiv \frac{\partial^2()}{\partial x^2}$. This trend holds in general: for any **odd-derivative** terms in L, L^* will contain the ***negative*** of these terms, and for any **even-derivative** terms in L, L^* will contain the ***same*** (unmodified) terms.[10]

To generalize a bit further, we can write the above relation (Eqn. 2.27) in inner product notation as

$$(Lu, v) \;=\; (u, L^*v) \qquad\qquad \forall u, v \in V \;, \tag{2.28}$$

where V is a suitable function space over which the above inner product (which could be, e.g., the L^2 inner product) is defined. In functional analysis, this relation is known as the **adjoint identity**. For a given operator L, it serves as the definition of the so-called ***adjoint operator***, L^*. As suggested, this L^* is exactly the operator we need to define the continuous adjoint ψ.

We can now summarize the discussion. For a **primal** differential equation

$$Lu = f \qquad\qquad x \in \Omega \tag{2.29}$$

$$\text{primal b.c.} \qquad\qquad x \in \partial\Omega \tag{2.30}$$

and an output of interest $J = (g, u)$, the **continuous adjoint** equation is defined as

$$L^*\psi = g \qquad\qquad x \in \Omega \tag{2.31}$$

$$\text{adjoint b.c.} \qquad\qquad x \in \partial\Omega \tag{2.32}$$

where L^* is the formal adjoint operator defined by Eqn. 2.28, and the adjoint boundary conditions are chosen such that Eqn. 2.28 is satisfied.[11]

With the adjoint problem defined in this way, the following derivation can then

[10] These even-derivative terms are the so-called "self-adjoint" terms.

[11] We will discuss this more later – it is not always *strictly* true when the adjoint boundary conditions are non-homogeneous.

be performed:

$$\begin{aligned} J &= (g, u) && \text{(output definition)} \\ &= (L^* \psi, u) && \text{(by adjoint equation, Eqn. 2.31)} \\ &= (\psi, Lu) && \text{(by adjoint identity, Eqn. 2.28)} \\ &= (\psi, f)\,. && \text{(by primal equation, Eqn. 2.29)} \end{aligned} \tag{2.33}$$

Thus, the above definition of the adjoint problem allows us to write the output in **dual form**, from which it follows that ψ represents the **sensitivity** of the output to perturbations in f.

2.2.2 Further Generalization of the Continuous Adjoint

So far, we have avoided a detailed discussion of boundary conditions for both the primal and adjoint problems. Our choice of boundary condition for ψ in Eqn. 2.24 happened to work, but we have given little motivation for this choice. In addition, we have assumed homogeneous boundary conditions for the primal problem, which will not always be the case. Finally, we have assumed that the output itself can be represented as an "interior" integral of the form (g, u). However, in practice, the most important outputs – such as lift and drag – are often those defined on the **domain *boundaries***.

In order to address these issues, we need to further generalize our definition of the continuous adjoint. Recall that, in the discrete adjoint section, we initially treated the adjoint as the sensitivity of an output with respect to $\mathbf{F}$, then later generalized it to represent the sensitivity of an output with respect to the residual, $\mathbf{R}$. By following this same path in the definition of the continuous adjoint, we can address the aforementioned issues.

First, as in the discrete case, we can define a **continuous "residual"** $r(u)$ to be the function that is zero when the primal differential equation is satisfied, i.e.

$$r(u) \equiv Lu - f\,. \tag{2.34}$$

Now, if the adjoint ψ is to represent the sensitivity of an output J with respect to perturbations in $r(u)$, we require that it satisfy the following equation:

$$J'(\delta u) = \int_{\Omega} \psi \, r'(\delta u) \, d\Omega \qquad \forall \, (\text{permissible}) \, \delta u \quad . \tag{2.35}$$

This is the generalized form of the **continuous adjoint equation**, which serves as the definition of ψ regardless of output type, boundary conditions, and even problem nonlinearity. Conceptually, this equation just says that for a given change in the residual, $r'(\delta u)$, the adjoint dictates how this change leads to a change in the output, $J'(\delta u)$. Here, a prime denotes taking the "**first variation**" (or **Fréchet linearization**) of a quantity with respect to u. From variational calculus, taking the first variation of e.g. J means considering how some change in the state, δu (which is defined over all of Ω – see Fig. 2.3), would change the corresponding value of J.[12]

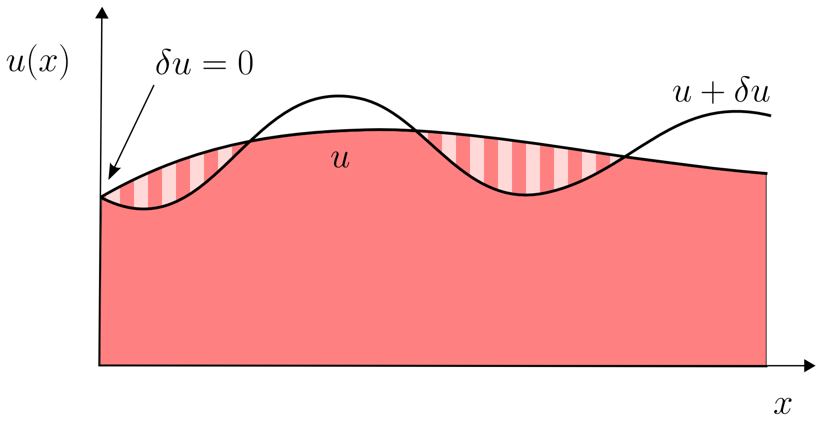


Figure 2.3: Illustration of the variation, δu. The variation is a function of x and represents a perturbation to the state, u. If the value of u is fixed at a boundary by a Dirichlet condition (as it is on the left here) the variation is constrained to be 0 at that boundary.

A similar idea holds for the variation of the residual, $r'(\delta u)$. In practice, for linear problems, computing the variation of the residual simply means inserting a δu term in place of the original u, and eliminating any terms that have no dependence on u. For example, for the advection problem above, we have

$$r(u) \equiv a\frac{\partial u}{\partial x} - f \tag{2.36}$$

[12] Thus, rather than a *rate* of change, $J'(\delta u)$ actually represents a direct change in J, which could be written in shorthand as simply δJ.

and would compute its first variation as

$$r'(\delta u) = a\frac{\partial(\delta u)}{\partial x}\,. \tag{2.37}$$

Here, since f does not depend on u, it vanishes when the variation is taken.

Finally, an important point is that the variation δu must be a "permissible" one, which – in particular – means that it must satisfy any boundary conditions imposed on the primal problem. For example, if a Dirichlet condition is imposed on a given boundary, then the value of u is fixed there, meaning no variation is allowed and hence $\delta u = 0$ there. Figure 2.3 shows an example of what a permissible δu might look like in this situation.

Remark 1. (**Systems of Equations**) Note that Eqn. 2.35 defines the adjoint in the case of a scalar equaton $r(u)$. If we instead had a system of N_s equations with state components $\mathbf{u} = [\,u_1 \;\; u_2 \;\; \ldots \;\; u_{N_s}\,]^T$ and residual components $\mathbf{r}(\mathbf{u}) = [\,r_1(\mathbf{u}) \;\; r_2(\mathbf{u}) \;\; \ldots \;\; r_{N_s}(\mathbf{u})\,]^T$, then the adjoint would be written as $\boldsymbol{\psi} = [\,\psi_1 \;\; \psi_2 \;\; \ldots \;\; \psi_{N_s}\,]^T$ and the adjoint equation would become:

$$\boxed{J'(\delta\mathbf{u}) = \int\limits_{\Omega} \boldsymbol{\psi}^T \mathbf{r}'(\delta\mathbf{u})\, d\Omega \qquad \forall\,(\text{permissible})\,\delta\mathbf{u}}\;. \tag{2.38}$$

Here, we use boldface to indicate a vector of N_s state components. Furthermore, the variation is now defined as e.g. $J'(\delta\mathbf{u}) \equiv J'(\delta u_1) + J'(\delta u_2) + \ldots + J'(\delta u_{N_s})$ – in other words, it is the sum of the variations with respect to each component of $\mathbf{u}$. Note that we are being a little loose with notation here: for example, $J'(\delta u_1)$ should be formally written as $J'_{u_1}([\delta u_1\, 0 \ldots 0])$. Here, the u_1 subscript means that we are taking the variation of J with respect to u_1, which is then a function of δu_1 only. However, when it is clear from context, we will leave the subscript off and will ignore the zero-valued inputs.

For the sake of simplicity, let us continue with our scalar advection example to show how Eqn. 2.35 is used in practice. We will then provide some further examples.

2.2.2.1 Example: Continuous Adjoint for Steady Advection

Assume a primal advection problem of the form

$$a\frac{\partial u}{\partial x} = f \qquad\qquad x \in \Omega \tag{2.39}$$

$$u = u_L \qquad\qquad x = x_L \tag{2.40}$$

where u_L is a Dirichlet boundary condition imposed on the left side of the domain. As before, we imagine that we are interested in some output $J(u)$, which will be left unspecified for now.

The variation of the residual, $r'(\delta u)$, is given by Eqn. 2.37. Inserting this into the general adjoint equation (Eqn. 2.35) gives:

$$J'(\delta u) = \int\limits_{\Omega} \psi\, a\frac{\partial(\delta u)}{\partial x}\, dx \qquad \forall\, \delta u\,. \tag{2.41}$$

To determine the continuous adjoint PDE and boundary conditions, we now integrate by parts, giving

$$J'(\delta u) = \int\limits_{\Omega} \underbrace{\left(-a\frac{\partial \psi}{\partial x}\right)}_{L^*\psi} \delta u\, dx + \psi\, a\, \delta u\Big|_{x_L}^{x_R} \qquad \forall\, \delta u\,, \tag{2.42}$$

or

$$J'(\delta u) = \int\limits_{\Omega} (L^*\psi)\, \delta u\, dx + \psi\, a\, \delta u\Big|_{x_L}^{x_R} \qquad \forall\, \delta u\,. \tag{2.43}$$

Next, since the Dirichlet boundary condition is imposed at $x = x_L$, the variation δu must be zero there. Eliminating this term in the above equation then leaves

$$J'(\delta u) = \int\limits_{\Omega} (L^*\psi)\, \delta u\, dx + \psi\, a\, \delta u\Big|_{x_R} \qquad \forall\, \delta u\,. \tag{2.44}$$

Now, if, as before, we assume that our output has the form $J = (g, u)$, then its variaton $J'(\delta u)$ would just be $J'(\delta u) = (g, \delta u)$. Inserting this expression into the left-hand side of the above equation then gives

$$\int\limits_{\Omega} g\,\delta u\,dx = \int\limits_{\Omega} (L^{*}\psi)\,\delta u\,dx + \psi\,a\,\delta u\Big|_{x_R} \qquad \forall\,\delta u\,. \tag{2.45}$$

We now see that, if the left- and right-hand sides of this equation are to be equal for all δu, we need ψ to satisfy the following conditions:

$$\boxed{L^{*}\psi = g \quad \text{and} \quad \psi\big|_{x_R} = 0}\;. \tag{2.46}$$

Inserting these expressions into the right-hand side of Eqn. 2.45 would then leave us with

$$\int\limits_{\Omega} g\,\delta u\,dx = \int\limits_{\Omega} g\,\delta u\,dx \qquad \forall\,\delta u\,, \tag{2.47}$$

which (clearly) holds for all δu.

Thus, for the output $J = (g, u)$, the constraints in Eqn. 2.46 represent the differential equation and boundary condition that must be satisfied by ψ in order to fulfill the general adjoint equation (Eqn. 2.35). And indeed, looking back at our original example in Eqn. 2.23, this is exactly how we defined the adjoint in that case.

At this point, we have recovered our previous definition of the adjoint for outputs of the form $J = (g, u)$. However, as mentioned, we may also be interested in an output defined along the *boundary* of the domain. It turns out that the specific form of the adjoint equation – in this case Eqn. 2.44 – determines the type of boundary terms that can be included in the output. Whichever output we choose, we need its variation to look similar to the right-hand side of Eqn. 2.44. Otherwise, it would not be possible to choose ψ in such a way that Eqn. 2.44 remains valid for all δu. In standard terminology, we would say that the output must be ***compatible*** with Eqn. 2.44.

In the present case, since Eqn. 2.44 contains only a right-boundary term (as opposed to a left-boundary term), this means that our output can likewise only involve the state on the right boundary. This makes sense from a logical perspective as well: since we are imposing a Dirichlet condition on the left boundary, we already know the value of u there, so there is no sense in treating it as part of an "unknown" output. This idea holds true in general: ***compatible*** **outputs** are those whose components have **not already been fixed** by the boundary conditions, and hence, they are the outputs we would naturally be interested in anyway.

For our current problem, the general form of a compatible output is

$$\boxed{J(u) = (g, u) + g_R \, u\big|_{x_R}} \quad , \tag{2.48}$$

where g_R is an arbitrary weight on the right-boundary state (i.e the outgoing flux). The variation of this output, $J'(\delta u)$, is then

$$J'(\delta u) = (g, \delta u) + g_R \, \delta u|_{x_R} \, . \tag{2.49}$$

To find the corresponding adjoint equation, we insert this $J'(\delta u)$ into Eqn. 2.44, giving

$$\int_\Omega g \, \delta u \, dx + g_R \, \delta u\Big|_{x_R} = \int_\Omega (L^* \psi) \, \delta u \, dx + \psi \, a \, \delta u\Big|_{x_R} \qquad \forall \, \delta u \, . \tag{2.50}$$

We then see that for this equation to hold for all δu, ψ must satisfy

$$\boxed{L^* \psi = g \quad \text{and} \quad \psi\big|_{x_R} = \frac{g_R}{a}} \; . \tag{2.51}$$

The conclusion is thus that, when the output includes a boundary term, the corresponding boundary condition for the adjoint problem becomes nonhomogeneous.

Remark 2. (**Dual Form**) In this section, in addition to the adjoint boundary condition, we have also allowed the ***primal* boundary condition** ($u = u_L$) to be **nonzero**. So far, this has had no influence on the derivation of the adjoint, since this derivation involves only the variation δu on the boundary, rather than the actual value there. However, while the primal boundary condition does not influence the adjoint problem itself, it does change how we would express the output in dual form.

To derive the dual form of the output, we start from the output definition in Eqn. 2.48 and perform the following steps:

$$\begin{aligned}
J(u) &= (g, u) \, + \, g_R \, u\big|_{x_R} && \text{(output definition)} \\
&= (L^* \psi, u) \, + \, \psi \, a u\big|_{x_R} && \text{(adjoint equation and BC, Eqn. 2.51)} \\
&= (\psi, Lu) \, - \, \psi \, a u\big|_{x_L}^{x_R} \, + \, \psi \, a u\big|_{x_R} && \text{(integration by parts)} \\
&= (\psi, Lu) \, + \, \psi \, a u\big|_{x_L} && \text{(right-boundary term cancellation)} \\
&= (\psi, f) \, + \, \psi \, a u_L\big|_{x_L} && \text{(primal equation and BC, Eqn. 2.39)}
\end{aligned} \tag{2.52}$$

Thus, when the primal boundary conditions are nonzero, the dual form of the output is given by

$$\boxed{J(u) = (\psi, f) + \psi \tilde{f}\big|_{x_L}} \; . \tag{2.53}$$

where $\tilde{f} \equiv a u_L$ is the prescribed boundary flux.

Note that although we now have a boundary term in the dual form, the underlying concept remains the same: the **dual form** involves only the **adjoint** and the ***prescribed data*** associated with the primal problem. In this case, that prescribed data includes both the source term f and the boundary data $\tilde{f} = a u_L$. Note that this is still analogous to the dual form in the discrete case. There, the discrete $\mathbf{F}$ – which we referred to as the "source" vector – in practice contains boundary data as well.

2.2.2.2 Example: Continuous Adjoint for Steady Diffusion

To solidfy the above ideas, we provide another example – a steady diffusion problem. Assume we have the following primal equation

$$-\nu \frac{\partial^2 u}{\partial x^2} = f \qquad\qquad x \in \Omega \tag{2.54}$$

$$u = u_L \qquad\qquad x = x_L \tag{2.55}$$

$$\frac{\partial u}{\partial x} = u_{x,R} \qquad\qquad x = x_R \tag{2.56}$$

where ν is a positive diffusion coefficient, u_L is a prescribed Dirichlet condition on the left, and $u_{x,R}$ is a prescribed Neumann condition on the right.

The residual is then defined as

$$r(u) \equiv -\nu \frac{\partial^2 u}{\partial x^2} - f, \tag{2.57}$$

and its variation is

$$r'(\delta u) = -\nu \frac{\partial^2 (\delta u)}{\partial x^2} \, . \tag{2.58}$$

We again assume that we are interested in some output $J(u)$, which we will leave unspecified for the moment. Substituting the above residual variation into the general adjoint equation (Eqn. 2.35) gives

$$J'(\delta u) = \int\limits_{\Omega} \psi \left(-\nu \frac{\partial^2 (\delta u)}{\partial x^2} \right) dx. \tag{2.59}$$

Now integrating by parts (twice) to move all derivatives onto ψ yields:

$$J'(\delta u) \;=\; \int\limits_{\Omega} \underbrace{\left(-\nu \frac{\partial^2 \psi}{\partial x^2}\right)}_{L^*\psi} \delta u \, dx \;+\; \nu \frac{\partial \psi}{\partial x} \delta u \Bigg|_{x_L}^{x_R} \;-\; \nu \psi \frac{\partial(\delta u)}{\partial x} \Bigg|_{x_L}^{x_R} . \tag{2.60}$$

Next, since the Dirichlet condition constrains u at the left boundary, the variation δu must be zero there. Likewise, since the Neumann condition constrains $\partial u / \partial x$ at the right boundary, the variation of the *derivative* must be zero there. (Note that mathematically, $\delta \left(\partial u / \partial x\right) = \partial \left(\delta u\right) / \partial x$, so we require $\partial \left(\delta u\right) / \partial x = 0$ at $x = x_R$.) Applying these constraints then leaves

$$J'(\delta u) \;=\; \int\limits_{\Omega} L^* \psi \, \delta u \, dx \;+\; \nu \frac{\partial \psi}{\partial x} \delta u \Bigg|_{x_R} \;+\; \nu \psi \frac{\partial(\delta u)}{\partial x} \Bigg|_{x_L} . \tag{2.61}$$

We see now that to be compatible with this equation, our output can include either a "u"-type contribution on the right boundary or a "$\partial u / \partial x$"-type contribution on the left boundary. In other words, our output can be of the form

$$\boxed{J(u) = (g, u) \;+\; g_R \, u \Bigg|_{x_R} \;+\; g_L \, \frac{\partial u}{\partial x} \Bigg|_{x_L}} \; . \tag{2.62}$$

Once again, this makes sense from a logical perspective as well: the output includes only quantities that are *not* already prescribed from the boundary conditions. By taking the variation of this $J(u)$ and comparing it to Eqn. 2.61, we see that the adjoint must satisfy the following conditions:

$$\boxed{L^* \psi = g \, , \qquad \frac{\partial \psi}{\partial x} \Bigg|_{x_R} = \frac{g_R}{\nu} \, , \qquad \text{and} \qquad \psi \big|_{x_L} = \frac{g_L}{\nu}} \; . \tag{2.63}$$

2.2.2.3 Example: Continuous Adjoint for Nonlinear Burgers Equation

Finally, we note that the adjoint equation (Eqn. 2.35) applies to **nonlinear** problems as well. For these problems, the adjoint represents the sensitivity of an output to residual perturbations *about* some particular state u, in the same way that we could compute the slope of a standard nonlinear function (say $f(x) = x^2$) about a particular value of x.

In general, both the residual and the desired output may be nonlinear expressions (as would be the case with, e.g. a Navier-Stokes simulation and a lift output), and the variations of these expressions would then represent **linearizations** about some particular state – say, u_0. Thus, while for linear problems we just label the variations $J'(\delta u)$ and $r'(\delta u)$, for nonlinear problems we must also indicate the state about which these variations are performed. We can do this by including that state in square brackets. For example, when linearizing about u_0, we would write $J'[u_0](\delta u)$ and $r'[u_0](\delta u)$. This notation is common in the literature.

As a simple example, consider the following one-dimensional Burgers equation (with a u^3 source term):

$$u\frac{\partial u}{\partial x} + u^3 = f \qquad\qquad x \in \Omega \tag{2.64}$$

$$u = u_L \qquad\qquad x = x_L \tag{2.65}$$

The residual is defined as

$$r(u) \equiv u\frac{\partial u}{\partial x} + u^3 - f\,, \tag{2.66}$$

and its variation about a particular state $u_0(x)$ is then given by

$$r'[u_0](\delta u) = u_0\frac{\partial(\delta u)}{\partial x} + \delta u\frac{\partial u_0}{\partial x} + 3u_0^2\,\delta u\,. \tag{2.67}$$

Here, the first two terms arise from applying the product rule[13] to the $u\,\partial u/\partial x$ term in Eqn. 2.66, while the last term represents the variation of u^3.

Similarly, if our output were some nonlinear quantity, such as the right-boundary flux

$$J(u) = \frac{1}{2}u^2\bigg|_{x_R}\,, \tag{2.68}$$

then its variation could be computed as

$$J'[u_0](\delta u) = u_0\,\delta u\bigg|_{x_R}\,. \tag{2.69}$$

Since u_0 is treated as a "frozen" state, both $r'[u_0](\delta u)$ and $J'[u_0](\delta u)$ are now linear in δu, and the adjoint equations and boundary conditions can be derived as usual from Eqn. 2.35.

[13] The variation obeys the product rule in the same way that a standard derivative does.

2.3 Summary: Discrete and Continuous Adjoints

With discrete and continuous adjoints discussed, we conclude with a brief discussion of the connections between these concepts. We have already mentioned some of the similarities between discrete and continuous adjoints, such as how they lead to a "dual form" of an output, and hence represent an output sensitivity to residual perturbations. However, we can draw some further parallels between the discrete and continuous adjoint equations themselves.

We start with a simple side-by-side comparison of these equations. Recall that the discrete adjoint equation is given by Eqn. 2.15 as

$$\frac{\partial \mathbf{R}}{\partial \mathbf{U}}^T \boldsymbol{\Psi} = \frac{\partial J}{\partial \mathbf{U}}^T .$$

Transposing both sides of this equation and swapping left- and right-hand sides then gives:

$$\frac{\partial J}{\partial \mathbf{U}} = \boldsymbol{\Psi}^T \frac{\partial \mathbf{R}}{\partial \mathbf{U}} .$$

On the other hand, the continuous adjoint equation (Eqn. 2.38) is defined as

$$J'(\delta \mathbf{u}) = \int\limits_{\Omega} \boldsymbol{\psi}^T \mathbf{r}'(\delta \mathbf{u}) \, d\Omega .$$

A clear resemblance is seen between these equations – namely, they both involve an output linearization on the left and a residual linearization on the right, related via an adjoint vector.

2.3.1 The Discrete Adjoint Operator and Adjoint Consistency

To draw some further connections, let us look more closely at the discrete adjoint equation

$$\frac{\partial \mathbf{R}}{\partial \mathbf{U}}^T \boldsymbol{\Psi} = \frac{\partial J}{\partial \mathbf{U}}^T .$$

If, for simplicity, we call the residual Jacobian matrix $\mathbf{A}$ and label the output linearization $\mathbf{G}$ (such that, e.g., for a linear output $J \equiv \mathbf{G}^T \mathbf{U}$), then this equation can be written as

$$\mathbf{A}^T\boldsymbol{\Psi} = \mathbf{G} \ . \tag{2.70}$$

On the other hand, from Eqn. 2.31, the differential form of the continuous adjoint equation is written as

$$L^*\psi = g.$$

Recall that the adjoint operator in this equation, L^*, was defined by the adjoint identity

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

A question which then arises is: if the continuous adjoint operator L^* satisfies a so-called "adjoint identity," can the same be said for the operator in the *discrete* adjoint equation – i.e. $\mathbf{A}^T$?

To answer this question, note that we can formulate a discrete version of the above adjoint identity by: (1) replacing L by a discrete operator $\mathbf{A}$, (2) replacing the inner product with a discrete dot product, and (3) replacing u and v by discrete vectors $\mathbf{U}$ and $\mathbf{V}$. We then obtain the following identity

$$\mathbf{A}\mathbf{U}\cdot\mathbf{V} = \mathbf{U}\cdot\mathbf{A}^*\mathbf{V} \qquad\qquad \forall\,\mathbf{U},\mathbf{V}\in\mathbb{R}^N\,, \tag{2.71}$$

which defines a certain matrix $\mathbf{A}^*$ as the **discrete adjoint operator**. To determine $\mathbf{A}^*$, we rewrite the above relation using the formal definition of the dot product (i.e. $\mathbf{u}\cdot\mathbf{v} \equiv \mathbf{u}^T\mathbf{v}$):

$$(\mathbf{A}\mathbf{U})^T\mathbf{V} = \mathbf{U}^T(\mathbf{A}^*\mathbf{V}) \qquad\qquad \forall\,\mathbf{U},\mathbf{V}\in\mathbb{R}^N \tag{2.72}$$

$$\mathbf{U}^T\mathbf{A}^T\mathbf{V} = \mathbf{U}^T\mathbf{A}^*\mathbf{V} \qquad\qquad \forall\,\mathbf{U},\mathbf{V}\in\mathbb{R}^N \tag{2.73}$$

$$\Longrightarrow \boxed{\mathbf{A}^T = \mathbf{A}^*} \ . \tag{2.74}$$

The formal adjoint operator associated with a matrix $\mathbf{A}$ is therefore just its **transpose**, $\mathbf{A}^T$. Thus, the presence of $\mathbf{A}^T$ in the discrete adjoint equation does in fact "parallel" the continuous adjoint equation, in the sense that both equations involve formal adjoint operators on the left-hand side and output linearizations on the right.

This connection between $\mathbf{A}^T$ and L^* is more than superficial. In fact, just as the discrete primal equation $\mathbf{AU} = \mathbf{F}$ should (ideally) represent a consistent discretization of the continuous primal problem,

$$\boxed{\mathbf{AU} = \mathbf{F} \quad \Longleftrightarrow \quad Lu = f} \; ,$$

the discrete *adjoint* equation $\mathbf{A}^T\mathbf{\Psi} = \mathbf{G}$ should ideally represent a consistent discretization of the continuous *adjoint* problem,

$$\boxed{\mathbf{A}^T\mathbf{\Psi} = \mathbf{G} \quad \Longleftrightarrow \quad L^*\psi = g} \; .$$

A discretization which satisfies this property – i.e. whose matrix **A** has a transpose that is also a consistent discretization of L^* – is known as ***adjoint consistent***.[14]

While not every primal discretization is adjoint-consistent, a lack of adjoint-consistency can lead to suboptimal convergence rates in outputs of interest. Thus, even if the adjoint equation is never explicitly solved, adjoint-consistent discretizations are often preferable to adjoint-inconsistent ones. Furthermore, if the discrete adjoint equation *is* actually solved using an adjoint-inconsistent discretization, spurious oscillations will appear in the adjoint, limiting its usefulness for applications such as optimization and error estimation. In order to avoid these issues, adjoint-inconsistent discretizations can often be modified to recover adjoint-consistency. A discussion of these issues is provided in e.g. [65, 48, 46].

2.3.2 Summary of Adjoint Properties

1. For a **scalar output** of interest, both discrete and continuous adjoints can be used to represent the output in an equivalent "**dual form**" (provided the problem is linear).

2. For both linear and nonlinear problems, the discrete and continuous adjoints represent the **sensitivity** of a given output to perturbations in the governing equations (residuals).

3. The discrete adjoint is just a weighted average of the rows of the inverse Jacobian matrix (i.e. $\mathbf{A}^{-1}$ or $\partial\mathbf{R}/\partial\mathbf{U}^{-1}$).

[14] Technically, both the adjoint operator (along with its corresponding boundary conditions) and the output linearization must be represented in a consistent manner for a method to be considered adjoint-consistent.

4. The continuous adjoint is the solution of a linear "adjoint" PDE involving the adjoint operator L^*.

5. Any odd-order derivatives in L^* have the opposite sign as in the primal operator, leading to a **reversal of information flow** in the adjoint problem.

6. The adjoint equations – both discrete and continuous – are **linear** even when the primal problem is nonlinear. For nonlinear problems, the adjoint equations represent a local linearization about a given primal state.

2.4 Output-based Error Estimation

With the discussion of adjoints complete, we now turn to one of the primary uses of adjoints: the estimation of numerical errors. The concept of adjoint-based *a posteriori* error estimaton (or simply "**output-based**" **error estimation**) can be treated in either a continuous or discrete context. In this section, we begin with a simplified discussion in a continuous context before moving on to a more general discrete formulation.

2.4.1 Continuous Error Estimation

Assume that we are dealing with a linear differential equation of the form

$$Lu = f \qquad x \in \Omega \tag{2.75}$$

$$\text{primal b.c.} \qquad x \in \partial\Omega \tag{2.76}$$

and, for simplicity, that our desired output $J(u)$ can be represented as

$$J(u) = (u, g)\,. \tag{2.77}$$

Now imagine that somehow (e.g. through a numerical method) we have arrived at an approximate solution, u_H.[15] If we were to compute our output using this u_H, its value would be given by

$$J(u_H) = (u_H, g)\,. \tag{2.78}$$

[15] Assume that u_H is sufficiently smooth to evaluate $r(u_H)$. If u_H were nonsmooth, similar arguments would hold, but the residual would have to be treated in a distributional sense.

A question we might now ask is: how much ***error*** is present in this output? In other words, what is the difference between the exact and approximate outputs, i.e. $\delta J \equiv J(u) - J(u_H)$?

It turns out that the adjoint is helpful in answering this question. Starting from the definition of δJ and using the relevant adjoint identities, we find:

$$\begin{aligned}
\delta J &= J(u) - J(u_H) & \\
&= (u, g) - (u_H, g) & \text{(output definitions)} \\
&= (u - u_H, g) & \text{(linearity of inner product)} \\
&= (u - u_H, L^*\psi) & \text{(adjoint equation, Eqn. 2.31)} \\
&= (L(u - u_H), \psi) & \text{(adjoint identity, Eqn. 2.28)} \\
&= (Lu, \psi) - (Lu_H, \psi) & \text{(linearity of inner product)} \\
&= (f, \psi) - (Lu_H, \psi) & \text{(primal equation, Eqn. 2.75)} \\
&= -(Lu_H - f, \psi) & \text{(linearity of inner product)}
\end{aligned} \tag{2.79}$$

Now, recall that the residual is defined as

$$r(u) \equiv Lu - f\,. \tag{2.80}$$

Thus, the quantity $Lu_H - f$ is just the residual evaluated with the approximate solution, i.e. $r(u_H)$. Since u_H does not satisfy the primal differential equation exactly, this $r(u_H)$ will in general be nonzero, and represents the local truncation error at a given region of the domain.

Replacing $Lu_H - f$ with $r(u_H)$ in Eqn. 2.79 then gives the following expression for the output error:

$$\delta J = -(r(u_H), \psi). \tag{2.81}$$

Or, in integral form, we have

$$\delta J = -\int_{\Omega} \psi\, r(u_H)\, d\Omega \quad . \tag{2.82}$$

Thus, we see that the amount of error in an output is given by an ***adjoint-weighted*** (or ***"dual-weighted"***) ***residual***. This expression indicates that if nonzero residuals occur in regions where the adjoint is large, then these residuals will contribute a relatively large amount to the total output error.[16] Thus, not only does it provide the total output error, it also indicates the regions of the domain that are *responsible* for this output error. For this reason, the above expression will play a critical role in both output error estimation and **adaptive mesh refinement**.

Remark 3. (**Dual Form of Error Estimate**) Note that just as we can write the output itself in both primal and dual forms, the output error can be written in dual form as well. For a discussion of this topic, see Appendix A (Sec. A.1).

2.4.1.1 Continuous Error Estimation: Approximate Adjoint

In the current derivation, we have made the assumption of a linear primal problem and a linear interior output. In this context, the above expression for the output error is exact, provided the exact adjoint ψ is used. In practice, we will not have access to this exact adjoint and must instead settle for a numerical approximation, ψ_h. The error δJ above would then become an ***error estimate*** δJ_{est}, which could be written as

$$\boxed{\delta J_{\text{est}} = -\int_{\Omega} \psi_h \, r(u_H) \, d\Omega \;\approx\; \delta J} \; . \tag{2.83}$$

The closer ψ_h is to ψ, the closer δJ_{est} will be to the true output error.

2.4.1.2 Continuous Error Estimation: Finite Element Methods

So far, we have not needed to specify how ψ_h or u_H are obtained. However, since finite element methods play a central role in this work, it is useful to derive a form of the error estimate particular to these methods.

In general, a finite element method weights the continuous residual $r(u_H)$ by "test functions" $v \in \mathcal{V}_H$, where $\mathcal{V}_H$ may be, e.g., the space of polynomial functions of a certain order p. It then enforces orthogonality of the residual with respect to all functions in $\mathcal{V}_H$, so that

$$\int_{\Omega} v \, r(u_H) \, d\Omega \;=\; 0 \qquad\qquad \forall v \in \mathcal{V}_H \, . \tag{2.84}$$

[16] Assuming that minimal error cancellation occurs between different regions within the integral.

Now, if we were to approximate the adjoint *within* $\mathcal{V}_H$ itself, this "ψ_H" would necessarily satisfy

$$\int_{\Omega} \psi_H \, r(u_H) \, d\Omega \; = \; 0 \, , \tag{2.85}$$

since $\psi_H \in \mathcal{V}_H$. We can thus add this term to the error estimate in Eqn. 2.83 with no effect, so that for a finite element method, the following form of the error estimate holds:

$$\boxed{\delta J_{\text{est}} = - \int_{\Omega} (\psi_h - \psi_H) \, r(u_H) \, d\Omega} \quad . \tag{2.86}$$

From this expression, we see that in order to obtain a useful error estimate for a finite element method, we need to approximate the adjoint ψ_h in a different (typically ***finer***) space than $\mathcal{V}_H$ itself. Otherwise, if we were to take $\psi_h = \psi_H$ in the above formula, we would always obtain an error estimate of zero. In this work, we will compute ψ_h in an **order-enriched** space, though it could also be computed on e.g. a uniformly h-refined mesh.

Remark 4. (**Output Convergence Rate**) An additional point of interest is that Eqn. 2.86 can be used to predict the output convergence rate associated with a finite element method. First, if we assume that $\psi_h = \psi$ for simplicity, then this expression represents the exact output error. It then says that the output error involves the product of two terms: $(\psi - \psi_H)$ and $r(u_H)$. Therefore, the convergence rate of this output error should (at least) correspond to the sum of the convergence rates of these individual terms. Now, if $\mathcal{V}_H$ is an order-p space, then the quantity $(\psi - \psi_H)$ will typically converge at order $p + 1$. Furthermore, for (e.g.) a first-order operator such as advection, the residual $r(u_H)$ will converge at order p, since it involves taking one derivative of u_H. Summing these respective adjoint and residual rates, we predict that the output error should converge at a rate of

$$\underbrace{(p+1)}_{\text{adjoint}} \; + \underbrace{p}_{\text{residual}} \; = \underbrace{2p+1}_{\text{output}} \quad . \tag{2.87}$$

This is indeed the rate typically observed for the discontinuous Galerkin methods employed in this work.[17]

[17] Note that although the term $(\psi - \psi_H)$ will only converge at order $p + 1$ when ψ is *smooth*, in practice, $2p + 1$ output rates are often obtained when ψ is nonsmooth as well. Furthermore, note

2.4.1.3 Continuous Error Estimation: Nonlinear Problems

While we have focused on linear problems so far, the above error estimates carry over to nonlinear problems virtually unmodified, with the caveat that they are no longer exact due to the presence of a linearization error. See Appendix A (Sec. A.3) for a discussion of this topic. For now, we will move on and treat the nonlinear case in the discrete section to follow.

2.4.2 Discrete Error Estimation

We now turn to the primary method of error estimation used in this work – that of discrete adjoint-based error estimation. The ideas described in this section hold for general problems – linear or nonlinear, steady or unsteady – and for arbitrary output types and numerical discretizations.

To start, imagine that we are interested in solving a set of nonlinear equations, which may be written in discrete form as

$$\mathbf{R}_H(\mathbf{U}_H) = \mathbf{0}\,. \tag{2.88}$$

Here, the subscript H denotes a discretization on a given mesh, while $\mathbf{R}_H(\mathbf{U}_H)$ could represent the residuals associated with, for example, the Navier-Stokes equations.

Assume now that we are interested in a particular output, $J_H(\mathbf{U}_H)$, which could represent (say) lift or drag. After solving for $\mathbf{U}_H$ and computing this $J_H(\mathbf{U}_H)$, we may again like to know: how much ***error*** is present in this output?

Ideally, we would like to compute the true output error

$$\delta J = J(\mathbf{U}) - J_H(\mathbf{U}_H)\,. \tag{2.89}$$

However, without knowledge of the exact solution $\mathbf{U}$, this is infeasible. Instead, as a surrogate for $\mathbf{U}$, let us consider a so-called "fine-space" solution $\mathbf{U}_h$, and attempt to compute the **error** ***estimate***

$$\boxed{\delta J_{\text{est}} = J_h(\mathbf{U}_h) - J_H(\mathbf{U}_H)}\,. \tag{2.90}$$

Here, the fine space (which we will denote by $\mathcal{V}_h$) could be a uniformly refined or order-incremented version of the original space, $\mathcal{V}_H$. Thus, the fine space will typically

that for diffusion problems the output error is expected to converge at a rate of just $2p$, since $r(u)$ then contains a second-derivative operator and thus converges at order $p-1$.

contain the coarse space, i.e. $\mathcal{V}_H \subset \mathcal{V}_h$.

The fine-space solution $\mathbf{U}_h$ would then satisfy a corresponding set of fine-space equations, written as

$$\mathbf{R}_h(\mathbf{U}_h) = \mathbf{0}\,. \tag{2.91}$$

While we could attempt to solve these equations directly, this would be impractical. Instead, we would like to compute δJ_{est} *without* actually solving for $\mathbf{U}_h$.

In order to achieve this, we first define an **injection** of the coarse solution $\mathbf{U}_H$ into the fine space as

$$\mathbf{U}_h^H = \mathbf{I}_h^H \mathbf{U}_H\,. \tag{2.92}$$

Here, $\mathbf{I}_h^H$ is a lossless injection operator which effectively "samples" $\mathbf{U}_H$ on the fine space, such that the resulting $\mathbf{U}_h^H$ has the same dimension as $\mathbf{U}_h$ but contains only coarse-space information.[18] With this $\mathbf{U}_h^H$, we can then define the state perturbation, $\delta\mathbf{U}$, as the difference between the fine and (injected) coarse states:

$$\delta\mathbf{U} = \mathbf{U}_h - \mathbf{U}_h^H\,. \tag{2.93}$$

With these definitions in hand, we now turn to computing δJ_{est} (Eqn. 2.90). First, we note that the fine-space output, $J_h(\mathbf{U}_h)$, (which is at this point unknown) can be Taylor-expanded about the coarse-space state $\mathbf{U}_h^H$ as follows:

$$J_h(\mathbf{U}_h) \;=\; J_h(\mathbf{U}_h^H) \;+\; \left.\frac{\partial J_h}{\partial \mathbf{U}_h}\right|_{\mathbf{U}_h^H} \delta\mathbf{U} \;+\; \mathcal{O}(\delta\mathbf{U}^2)\;. \tag{2.94}$$

Dropping the $\mathcal{O}(\delta\mathbf{U}^2)$ remainder then leaves

$$\boxed{J_h(\mathbf{U}_h) \;\approx\; J_h(\mathbf{U}_h^H) \;+\; \left.\frac{\partial J_h}{\partial \mathbf{U}_h}\right|_{\mathbf{U}_h^H} \delta\mathbf{U}}\;. \tag{2.95}$$

Here, the perturbation $\delta\mathbf{U}$ is unknown. In order to determine it, we perform a similar expansion of the fine-space residuals, $\mathbf{R}_h(\mathbf{U}_h) = \mathbf{0}$, about the coarse solution:

$$\mathbf{R}_h(\mathbf{U}_h) \;\approx\; \boxed{\mathbf{R}_h(\mathbf{U}_h^H) \;+\; \left.\frac{\partial \mathbf{R}_h}{\partial \mathbf{U}_h}\right|_{\mathbf{U}_h^H} \delta\mathbf{U} \;=\; \mathbf{0}}\;. \tag{2.96}$$

[18] In a finite element context, this injection would just mean representing $\mathbf{U}_H$ with the fine-space basis functions rather than the coarse-space bases.

From this equation, we can solve for $\delta\mathbf{U}$, giving

$$\delta\mathbf{U} \;=\; -\frac{\partial\mathbf{R}_h}{\partial\mathbf{U}_h}\bigg|_{\mathbf{U}_h^H}^{-1}\mathbf{R}_h(\mathbf{U}_h^H) \tag{2.97}$$

Inserting this $\delta\mathbf{U}$ back into Eqn. 2.95 then gives

$$J_h(\mathbf{U}_h) \;\approx\; J_h(\mathbf{U}_h^H) \;-\; \underbrace{\frac{\partial J_h}{\partial\mathbf{U}_h}\bigg|_{\mathbf{U}_h^H}\frac{\partial\mathbf{R}_h}{\partial\mathbf{U}_h}\bigg|_{\mathbf{U}_h^H}^{-1}}_{\mathbf{\Psi}_h^T}\mathbf{R}_h(\mathbf{U}_h^H)\,. \tag{2.98}$$

The terms multiplying the residual in this equation now look familiar. Indeed, looking back at the discrete adjoint equation (Eqn. 2.13), we see that they correspond exactly to the *adjoint* of J. Specifically, they correspond to a **fine-space adjoint**, which we can denote by $\mathbf{\Psi}_h^T$. By the above definition, this adjoint is computed on the fine-space, $\mathcal{V}_h$, using a linearization about the injected coarse-space solution, $\mathbf{U}_h^H$.

Writing Eqn. 2.98 in terms of $\mathbf{\Psi}_h^T$ then gives

$$J_h(\mathbf{U}_h) \;\approx\; J_h(\mathbf{U}_h^H) \;-\; \mathbf{\Psi}_h^T\,\mathbf{R}_h(\mathbf{U}_h^H)\,. \tag{2.99}$$

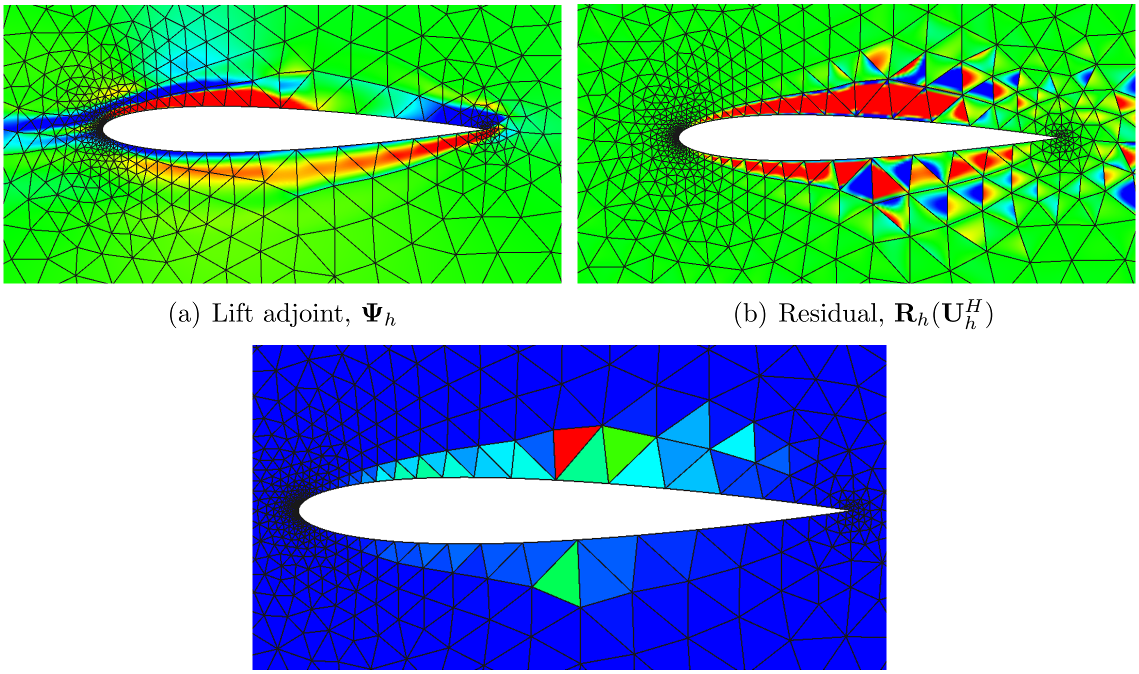
Finally, since $J_h(\mathbf{U}_h^H) = J_H(\mathbf{U}_H)$ as long as the mesh geometry does not change between the coarse and fine spaces, this equation can be rewritten as

$$J_h(\mathbf{U}_h) - J_H(\mathbf{U}_H) \;\approx\; -\mathbf{\Psi}_h^T\,\mathbf{R}_h(\mathbf{U}_h^H)\,. \tag{2.100}$$

The left-hand side of this equation is exactly the error estimate we wished to compute, δJ_{est} (Eqn. 2.90). Thus, we have

$$\boxed{\delta J_{\text{est}} \;\approx\; -\mathbf{\Psi}_h^T\,\mathbf{R}_h(\mathbf{U}_h^H)}\,. \tag{2.101}$$

We see that, as in the continuous case (Eqn. 2.83), the output error estimate involves the product of a “fine-space” adjoint and the residuals evaluated with the coarse-space solution. This $\mathbf{R}_h(\mathbf{U}_h^H)$ will in general be nonzero, and indicates regions of the domain where local truncation errors are generated. Weighting this term by the adjoint then determines to what extent each of these local truncations errors contributes to the final output error.



(a) Lift adjoint, $\boldsymbol{\Psi}_h$

(b) Residual, $\mathbf{R}_h(\mathbf{U}_h^H)$

(c) Error indicator

Figure 2.4: (a) Fine-space adjoint for the lift on the airfoil. (b) Fine-space residual evaluated with the injected coarse state (a measure of local truncation error). (c) Error indicators, representing the amount of error each element contributes to the lift. This is given by the product of the adjoint and residual, i.e. (a)·(b). (Figures reproduced from [31].)

2.4.2.1 Error Localization and Mesh Adaptation

If an element-based method (such as a finite element or finite volume method) is used to solve for $\mathbf{U}_H$, the above error estimate can be localized to individual elements in the mesh. If there are N_e^h total elements on the fine space, it can be rewritten as simply

$$\delta J_{\text{est}} \approx \sum_{e=1}^{N_e^h} -\boldsymbol{\Psi}_{h,e}^T \mathbf{R}_{h,e}(\mathbf{U}_h^H) \quad , \tag{2.102}$$

where $\boldsymbol{\Psi}_{h,e}^T$ and $\mathbf{R}_{h,e}(\mathbf{U}_h^H)$ are the components of the adjoint and residual associated with (i.e. restricted to) a given element e. A local **error indicator** can then be defined as the absolute value of this product on each fine-space element, i.e. as

$$\varepsilon_e = \left|\boldsymbol{\Psi}_{h,e}^T \mathbf{R}_{h,e}(\mathbf{U}_h^H)\right| \quad . \tag{2.103}$$

If order enrichment is used to obtain the fine space, then $N_e^H = N_e^h$, and ε_e directly indicates how much error a given coarse-space element contributes to the output. Alternatively, if uniform h-refinement is used to obtain the fine space, then ε_e can be summed over all fine-space elements within a given coarse-space element to arrive at a final coarse-space indicator. Figure 2.4 shows an example of the error indicator computation around an airfoil, where order enrichment was used for the fine space.

Once an error indicator is computed for each element in the coarse-space mesh, the elements with the highest indicators (or a related "figure of merit") can be selected for refinement. Refining these elements then drives down their local residuals, leading to a reduction in the amount of output error generated. This error estimation and adaptation procedure can be performed in an iterative fashion to efficiently drive the output error toward zero.

2.4.2.2 Finite Element Methods

If a finite element method is used to compute $\mathbf{U}_H$, a coarse-space adjoint can be subtracted from the above δJ_{est} (and corresponding error indicators) with no effect. This is due to the orthogonality property of finite element methods, as discussed previously in the continuous context (Eqn. 2.86).

If we call the coarse-space adjoint $\mathbf{\Psi}_H$ and denote its injection into the fine space as $\mathbf{\Psi}_h^H$, then for finite element methods, Eqns. 2.101, 2.102, and 2.103 become:

$$\delta J_{\text{est}} \approx -\left(\mathbf{\Psi}_h^T - \left(\mathbf{\Psi}_h^H\right)^T\right) \mathbf{R}_h(\mathbf{U}_h^H) \; , \tag{2.104}$$

$$\delta J_{\text{est}} \approx \sum_{e=1}^{N_e^h} -\left(\mathbf{\Psi}_{h,e}^T - \left(\mathbf{\Psi}_{h,e}^H\right)^T\right) \mathbf{R}_{h,e}(\mathbf{U}_h^H) \; , \tag{2.105}$$

and

$$\varepsilon_e = \left|\left(\mathbf{\Psi}_{h,e}^T - \left(\mathbf{\Psi}_{h,e}^H\right)^T\right) \mathbf{R}_{h,e}(\mathbf{U}_h^H)\right| \; . \tag{2.106}$$

In practice, it is not necessary to subtract off a coarse-space adjoint, but these expressions illustrate that for a finite element method, output errors will be generated not where the adjoint *values* are large, but instead where the adjoint is not *well-approximated*. In other words, the important regions are those where the difference between fine- and coarse-space adjoints, $\left(\mathbf{\Psi}_{h,e}^T - \left(\mathbf{\Psi}_{h,e}^H\right)^T\right)$, is large. Thus, if the fine-space adjoint were (e.g.) large but constant in a given region, then since this constant could also be represented in the coarse space (assuming it includes the $p = 0$ mode), no output errors would be generated in this region.

A related point to emphasize is that, while a coarse-space adjoint $\mathbf{\Psi}_H$ may be useful for optimization applications, for error estimation some type of fine-space adjoint is required. If we were to simply inject $\mathbf{\Psi}_H$ to the fine space and use that as our effective "$\mathbf{\Psi}_h$," then the above formulas would reduce to zero, providing no useful information. In practice, however, we do not need to solve the fine-space adjoint equations exactly. Oftentimes, injecting $\mathbf{\Psi}_H$ to the fine space and performing a few smoothing iterations is enough to provide meaningful error estimates.

2.4.2.3 Linearization Error

Finally, we note that for nonlinear problems – where the residuals $\mathbf{R}(\mathbf{U})$ and/or the output $J(\mathbf{U})$ are nonlinear – there is a so-called ***linearization* error** associated with the above error estimate, δJ_{est}.

Looking back at the derivation of δJ_{est} (Eqn. 2.101), we see that when performing the Taylor expansion of $J_h(\mathbf{U}_h)$ in Eqn. 2.94, we ignored the $\mathcal{O}(\delta\mathbf{U}^2)$ remainder term. Likewise, we ignored a similar $\mathcal{O}(\delta\mathbf{U}^2)$ term in the Taylor expansion of $\mathbf{R}_h(\mathbf{U}_h)$ in Eqn. 2.96. If we had carried these terms throughout the derivation, they would have appeared in the final error estimate, giving:

$$\boxed{\delta J_{\text{est}} \;\approx\; -\mathbf{\Psi}_h^T\,\mathbf{R}_h(\mathbf{U}_h^H) \;+\; \mathcal{O}(\delta\mathbf{U}^2)}\;. \tag{2.107}$$

Thus, for nonlinear problems, even if the adjoint $\mathbf{\Psi}_h$ were computed on an infinitely refined mesh, there would still be an $\mathcal{O}(\delta\mathbf{U}^2)$ *error* in the error estimate, due to the fact that the adjoint problem represents a linearization about the coarse-space state, $\mathbf{U}_h^H$. Since $\delta\mathbf{U}$ typically converges at order $p_H + 1$ (where p_H is the coarse-space approximation order), this $\mathcal{O}(\delta\mathbf{U}^2)$ term is of order $2(p_H + 1) = 2p_H + 2$. Thus, for nonlinear problems, the accuracy of the output error estimate overall is limited to $\mathcal{O}(h^{2p+2})$. (Though we note that, if desired, the accuracy of this error estimate can be improved to $\mathcal{O}(\delta\mathbf{U}^3) = \mathcal{O}(h^{3p+3})$ by averaging Eqn. 2.107 with a dual form of the error estimate. See Appendix A (Sec. A.3.2) for a discussion of this topic.)

2.4.2.4 Summary: Discrete Error Estimation and Mesh Adaptation

Here, we summarize the steps involved in the discrete error estimation and mesh adaptation process:

1. Solve $\mathbf{R}_H(\mathbf{U}_H) = 0$ on the coarse space, $\mathcal{V}_H$, to obtain $\mathbf{U}_H$.

2. Evaluate the output of interest, $J(\mathbf{U}_H)$.

3. Inject $\mathbf{U}_H$ to an order-incremented or uniformly refined space, $\mathcal{V}_h$. (Compute $\mathbf{U}_h^H$.)

4. Evaluate the fine-space residuals with the injected solution. (Compute $\mathbf{R}_h(\mathbf{U}_h^H)$.)

5. Solve (or approximate) the linear adjoint equation

$$\left.\frac{\partial \mathbf{R}_h}{\partial \mathbf{U}_h}^T\right|_{\mathbf{U}_h^H} \boldsymbol{\Psi}_h = \left.\frac{\partial J_h}{\partial \mathbf{U}_h}^T\right|_{\mathbf{U}_h^H} \tag{2.108}$$

   to find $\boldsymbol{\Psi}_h$ on the fine space.

6. Compute the error estimate $\delta J_{\text{est}} \approx -\boldsymbol{\Psi}_h^T \mathbf{R}_h(\mathbf{U}_h^H)$.

7. Correct the original $J(\mathbf{U}_H)$ with this error estimate. (Compute $J_{\text{corrected}} = J(\mathbf{U}_H) + \delta J_{\text{est}}$.) This corrected output is more accurate than $J(\mathbf{U}_H)$.

8. Localize δJ_{est} to individual elements in the mesh. (Compute the indicators ε_e from Eqn. 2.103.)

9. Select a certain percentage of elements with the highest error indicators (or a related "figure of merit") and refine them.

10. Solve the primal problem on the new mesh and repeat steps 2-10 until the output error is driven below a desired tolerance.

CHAPTER III

Unsteady Problems

While adjoint-based error estimation and mesh adaptation have been investigated previously for steady problems, one of the goals of this work was to extend these techniques to unsteady problems. From a mathematical perspective, the extension of adjoints to unsteady problems is straightforward. Conceptually, the time dimension can be treated as another "space-like" dimension, and the adjoint can be defined in a similar manner as before. The differences lie primarily in the details of the adjoint, error estimation, and mesh adaptation procedures, as well as in the fact that – unlike space – information propagates in a specific *direction* in time.

3.1 Unsteady Adjoints

As in the previous chapter, let us start by assuming we have a linear problem. While in the steady section we considered the system $\mathbf{A}\mathbf{U} = \mathbf{F}$, here we will include a time derivative, so that the governing equations become

$$\mathbf{M}\frac{d\mathbf{U}}{dt} + \mathbf{A}\mathbf{U} = \mathbf{F}\,. \tag{3.1}$$

Here, $\mathbf{M}$ is a "mass matrix," which is often just the identity matrix, but may in general differ (e.g. for finite element methods). As before, $\mathbf{A}$ represents a discrete spatial operator, while $\mathbf{U}$ represents the state and $\mathbf{F}$ is a prescribed source term. Many physical phenomena satisfy this form of equation, including heat diffusion, acoustic propagation, and the linearized Euler equations, to name a few.

To make the discussion more concrete, let us assume that the temporal derivative is discretized using a backward Euler method, such that

$$\mathbf{M}\,\frac{\mathbf{U}^m - \mathbf{U}^{m-1}}{\Delta t} + \mathbf{A}\mathbf{U}^m = \mathbf{F}^m\,. \tag{3.2}$$

Here, a superscript on a vector denotes a time index, so that e.g. $\mathbf{U}^m$ represents the state at time level m. As before, the number of spatial degrees of freedom is assumed to be N, while the temporal index m ranges from 1 to N_t, with N_t being the total number of temporal degrees of freedom. (In addition, from now on, when we write a variable such as $\mathbf{U}$ or $\mathbf{F}$ *without* a time index, we are referring to the column vector containing all space-time components of that variable, so that e.g. $\mathbf{U} \equiv \{\mathbf{U}^m\}_{\forall m} \in \mathbb{R}^{N \times N_t}$.)

Returning to the problem, we see that since Eqn. 3.2 is linear, we can rewrite it as simply

$$\bar{\mathbf{A}}\mathbf{U} = \mathbf{F}\,, \tag{3.3}$$

where $\bar{\mathbf{A}}$ is the matrix representing the full space-time operator, i.e.

$$\left(\bar{\mathbf{A}}\mathbf{U}\right)^m \;\equiv\; \mathbf{M}\,\frac{\mathbf{U}^m - \mathbf{U}^{m-1}}{\Delta t} + \mathbf{A}\mathbf{U}^m\,. \tag{3.4}$$

Here, the bar is used to distinguish the space-time operator $\bar{\mathbf{A}}$ from the spatial operator, $\mathbf{A}$.

Written out in matrix form, Eqn. 3.3 looks like

$$\underbrace{\begin{bmatrix} \bullet & & & & \\ \bullet & \bullet & & & \\ & \bullet & \bullet & & \\ & & \ddots & \ddots & \\ & & & \bullet & \\ & & & \bullet & \bullet \end{bmatrix}}_{\bar{\mathbf{A}}} \underbrace{\begin{bmatrix} \mathbf{U}^1 \\ \mathbf{U}^2 \\ \mathbf{U}^3 \\ \vdots \\ \mathbf{U}^{N_t-1} \\ \mathbf{U}^{N_t} \end{bmatrix}}_{\mathbf{U}} = \underbrace{\begin{bmatrix} \mathbf{F}^1 \\ \mathbf{F}^2 \\ \mathbf{F}^3 \\ \vdots \\ \mathbf{F}^{N_t-1} \\ \mathbf{F}^{N_t} \end{bmatrix}}_{\mathbf{F}} \tag{3.5}$$

where each dot in the matrix has the dimension of the spatial Jacobian, $\mathbf{A}$:

$$\bullet = \boxed{\swarrow \; N \times N \; \nearrow} \;=\; \text{Size of spatial } \mathbf{A} \text{ matrix} \tag{3.6}$$

From Eqn. 3.5 we see that, due to the nature of the backward Euler discretization, the $\bar{\mathbf{A}}$ matrix has a block diagonal structure in which the equations at a given time level (i.e. row of $\bar{\mathbf{A}}$) depend on the states at both the current and previous time levels. This structure arises due to the forward-in-time propagation of physical information, and would be similar for other temporal discretizations as well.

Now, if we were interested in solving for the entire $\mathbf{U}$ vector, this could be accomplished by performing a forward time march, which amounts to inverting the entire $\bar{\mathbf{A}}$ matrix:

$$\mathbf{U} = \bar{\mathbf{A}}^{-1}\mathbf{F}\,. \tag{3.7}$$

However, as in the previous chapter (Sec. 2.1), we can again ask: what if, rather than the entire solution $\mathbf{U}$, we are interested in just a single component of $\mathbf{U}$? Let us again take as an example the last component of $\mathbf{U}$, i.e. $U_N^{N_t}$, which represents the last spatial unknown at the final time. Physically, this could represent a "point" output at a certain location in space, evaluated at the end of the simulation.

The solution vector (given by Eqn. 3.7) can be written out explicitly as

$$\mathbf{U}^{N_t}\left\{\underbrace{\begin{bmatrix} \mathbf{U}^1 \\ \mathbf{U}^2 \\ \mathbf{U}^3 \\ \vdots \\ \mathbf{U}^{N_t-1} \\ \hdashline U_1^{N_t} \\ \vdots \\ \boxed{U_N^{N_t}} \end{bmatrix}}_{\mathbf{U}}\right. = \underbrace{\begin{bmatrix} \bullet & & & & & & & \\ \bullet & \bullet & & & & & & \\ \bullet & \bullet & \bullet & & & & & \\ \vdots & \vdots & \vdots & & & & & \\ \bullet & \bullet & \bullet & \dots & \bullet & & & \\ \circ & \circ & \circ & \dots & \circ & \circ & \dots & \circ \\ \vdots & \vdots & \vdots & & \vdots & \vdots & & \vdots \\ \circ & \circ & \circ & \dots & \circ & \circ & \dots & \circ \end{bmatrix}}_{\bar{\mathbf{A}}^{-1}} \underbrace{\begin{bmatrix} \mathbf{F}^1 \\ \mathbf{F}^2 \\ \mathbf{F}^3 \\ \vdots \\ \mathbf{F}^{N_t-1} \\ \hdashline F_1^{N_t} \\ \vdots \\ F_N^{N_t} \end{bmatrix}}_{\mathbf{F}} \tag{3.8}$$

$$\text{Output } J \qquad\qquad\qquad \text{Adjoint } \boldsymbol{\Psi}^T \tag{3.9}$$

Here, we have expanded the state at the last timestep, $\mathbf{U}^{N_t}$, into its spatial components in order to reveal the desired output, $J = U_N^{N_t}$. Also, note that $\bar{\mathbf{A}}^{-1}$ has a lower block-triangular structure due to the fact that the solution at a given timestep depends only on the source terms associated with earlier times.

From this equation, we see that (as in the steady case) the only information required to compute $U_N^{N_t}$ is the highlighted row of $\bar{\mathbf{A}}^{-1}$. Likewise, from our earlier argument, this row also represents the *sensitivity* of the output J to perturbations in the source term $\mathbf{F}$, since it multiplies the components of $\mathbf{F}$ during the computation of

$U_N^{N_t}$. Thus, as before, this row is exactly the **adjoint** of J, which we can again denote by $\boldsymbol{\Psi}^T$. Finally, note that for unsteady problems, $\boldsymbol{\Psi}^T$ is now a vector spanning the entire space-time domain, meaning it can be thought of as a quantity that evolves in time (similar to the state).

As before, J need not be a point output – it could be any linear combination of the components of $\mathbf{U}$, and could then be represented in **dual form** as

$$\boxed{J = \boldsymbol{\Psi}^T \mathbf{F}}\ , \tag{3.10}$$

where the adjoint $\boldsymbol{\Psi}^T$ is defined to be an output-specific weighted average of the rows of $\bar{\mathbf{A}}^{-1}$:

$$\boxed{\boldsymbol{\Psi}^T = \frac{\partial J}{\partial \mathbf{U}} \bar{\mathbf{A}}^{-1}}\ . \tag{3.11}$$

Finally, to write this equation in a more common form, we can define the **space-time residual** as

$$\bar{\mathbf{R}} = \bar{\mathbf{A}}\mathbf{U} - \mathbf{F} = \mathbf{0}\,, \tag{3.12}$$

so that

$$\frac{\partial \bar{\mathbf{R}}}{\partial \mathbf{U}} = \bar{\mathbf{A}}\,. \tag{3.13}$$

Making this replacement in Eqn. 3.11 and rearranging then gives the following form of the **adjoint equation**:

$$\boxed{\frac{\partial \bar{\mathbf{R}}}{\partial \mathbf{U}}^T \boldsymbol{\Psi} = \frac{\partial J}{\partial \mathbf{U}}^T}\ . \tag{3.14}$$

This is virtually identical to the adjoint equation derived in the steady case (Eqn. 2.15). Comparing the two, we see that the extension of the adjoint to unsteady problems consists primarily in drawing a bar over the residual. Indeed, the true differences lie not in the theory but in how this adjoint equation is *solved*. We will discuss this solution procedure later on. First, let us generalize to nonlinear problems.

3.1.1 Nonlinear Unsteady Problems

In practice, we are often interested in problems where both the governing equations and output are nonlinear. For example, we may wish to solve the unsteady Navier-Stokes equations around e.g. an airfoil and compute a time-averaged lift or drag output.

In that case, instead of the equation

$$\mathbf{M}\frac{d\mathbf{U}}{dt} + \underbrace{\mathbf{A}\mathbf{U} - \mathbf{F}}_{\mathbf{R}} = \mathbf{0}\,, \tag{3.15}$$

where the spatial residual $\mathbf{R}$ is linear, we would instead write the governing equations as

$$\mathbf{M}\frac{d\mathbf{U}}{dt} + \mathbf{R}(\mathbf{U}) = \mathbf{0}\,, \tag{3.16}$$

where $\mathbf{R}(\mathbf{U})$ is a general nonlinear function of $\mathbf{U}$.

Next, if we assume for simplicity that a backward Euler method is used, the unsteady residual associated with the mth temporal degree of freedom can be written as

$$\underbrace{\mathbf{M}\,\frac{\mathbf{U}^m - \mathbf{U}^{m-1}}{\Delta t} + \mathbf{R}(\mathbf{U}^m)}_{\overline{\mathbf{R}}^m(\mathbf{U})} = \mathbf{0}\,. \tag{3.17}$$

Here, the bar over the space-time residual $\overline{\mathbf{R}}^m(\mathbf{U})$ is used to distinguish it from the spatial residual, $\mathbf{R}(\mathbf{U})$.

Stepping back one level further, the entire set of space-time residuals can then be written:

$$\overline{\mathbf{R}}(\mathbf{U}) = \mathbf{0}\,. \tag{3.18}$$

Now, recall that our goal in the end is to define the adjoint equation, which requires the derivative of the residual with respect to $\mathbf{U}$. While in the above section this derivative was just the constant operator $\bar{\mathbf{A}}$ (from Eqn. 3.12), for a nonlinear $\overline{\mathbf{R}}(\mathbf{U})$, this derivative will be non-constant. Instead, it will be a function of the particular state about which it is computed. We can write this derivative as the space-time Jacobian

$$\left.\frac{\partial \overline{\mathbf{R}}}{\partial \mathbf{U}}\right|_{\mathbf{U}}, \tag{3.19}$$

which, as shown, is evaluated at a given state $\mathbf{U}$. Note that while the individual entries in this matrix will depend on $\mathbf{U}$, its sparsity pattern is identical to that of $\bar{\mathbf{A}}$ in Eqn. 3.5.

Likewise, as with the residual, if the output $J(\mathbf{U})$ is nonlinear, its derivative $\partial J/\partial \mathbf{U}$ will also depend on the state $\mathbf{U}$, and can be written as

$$\left.\frac{\partial J}{\partial \mathbf{U}}\right|_{\mathbf{U}}. \tag{3.20}$$

By substituting these residual and output linearizations into Eqn. 3.14, we then obtain the definition of the adjoint for nonlinear problems:

$$\boxed{\left.\frac{\partial \overline{\mathbf{R}}}{\partial \mathbf{U}}^T\right|_{\mathbf{U}} \mathbf{\Psi} = \left.\frac{\partial J}{\partial \mathbf{U}}^T\right|_{\mathbf{U}}}. \tag{3.21}$$

For a given output of interest, this is a linear system of space-time equations that can be solved for $\mathbf{\Psi}$. If the problem were steady, we would use a standard iterative method to find the solution. However, for unsteady problems, we can solve this problem more efficiently by taking advantage of the direction of information flow. We discuss this below.

3.1.2 Solution of the Unsteady Adjoint Equation

As mentioned, regardless of whether the residual is linear or nonlinear, the primal Jacobian $\partial \overline{\mathbf{R}}/\partial \mathbf{U}$ will have the sparsity pattern of the $\bar{\mathbf{A}}$ matrix in Eqn. 3.5. (Assuming a two-step temporal discretization is used.) This structure arises due to the fact that primal information propagates only forward in time. Consequently, the primal equation is most efficiently solved by performing a forward time march.

Just like the primal equation, the adjoint equation (Eqn. 3.21) also spans the entire space-time domain, and can therefore be thought of as representing the evolution of adjoint information in time. This raises the question: can we also solve the *adjoint* equation by using a forward time march? To answer this question, we take a closer look at the sparsity pattern of the discrete primal and adjoint operators.

$$\begin{array}{cc}
\textbf{Primal Unsteady Jacobian} & \textbf{Adjoint Unsteady Jacobian} \\
\underbrace{\begin{bmatrix}
\bullet & & & & & \\
\bullet & \bullet & & & & \\
& \bullet & \bullet & & & \\
& & \ddots & \ddots & & \\
& & & & \bullet & \\
& & & & \bullet & \bullet
\end{bmatrix}}_{\dfrac{\partial \overline{\mathbf{R}}}{\partial \mathbf{U}}}
&
\underbrace{\begin{bmatrix}
\bullet & \bullet & & & & \\
& \bullet & \bullet & & & \\
& & \bullet & \ddots & & \\
& & & \ddots & & \\
& & & & \bullet & \bullet \\
& & & & & \bullet
\end{bmatrix}}_{\dfrac{\partial \overline{\mathbf{R}}}{\partial \mathbf{U}}^{T}}
\end{array} \tag{3.22}$$

The above diagram shows the primal Jacobian on the left and its transpose on the right. From Eqn. 3.21, this transpose operator is exactly the one that appears in the adjoint equation. As before, each "dot" in these matrices represents a spatial Jacobian matrix (plus any temporal discretization terms), i.e.

$$\bullet = \boxed{N \times N} = \text{Size of spatial } \frac{\partial \mathbf{R}}{\partial \mathbf{U}} \text{ matrix} \tag{3.23}$$

In the primal Jacobian, since there is a single unknown (vector) in the first row, it is clear that we should first solve for this unknown, then perform forward-substitution (i.e. a forward time march) to obtain the remaining unknowns. On the other hand, due to the transposed nature of the adjoint operator, we see that it has a single unknown in the *last* row, which corresponds to the final time in the simulation. Therefore, unlike the primal problem, the most efficient way to solve the **adjoint problem** is to perform a *back*-substitution – in other words, a **backward time march**.

To see this more clearly, we can write out the adjoint equation (Eqn. 3.21) explicitly as

$$\underbrace{\begin{bmatrix} \bullet & \bullet & & & & \\ & \bullet & \bullet & & & \\ & & \bullet & \ddots & & \\ & & & \ddots & & \\ & & & & \bullet & \bullet \\ & & & & & \bullet \end{bmatrix}}_{\frac{\partial \bar{\mathbf{R}}}{\partial \mathbf{U}}^T} \underbrace{\begin{bmatrix} \boldsymbol{\Psi}^1 \\ \boldsymbol{\Psi}^2 \\ \boldsymbol{\Psi}^3 \\ \vdots \\ \boldsymbol{\Psi}^{N_t-1} \\ \boldsymbol{\Psi}^{N_t} \end{bmatrix}}_{\boldsymbol{\Psi}} = \underbrace{\begin{bmatrix} (\partial J/\partial \mathbf{U}^1)^T \\ (\partial J/\partial \mathbf{U}^2)^T \\ (\partial J/\partial \mathbf{U}^3)^T \\ \vdots \\ \left(\partial J/\partial \mathbf{U}^{N_t-1}\right)^T \\ \left(\partial J/\partial \mathbf{U}^{N_t}\right)^T \end{bmatrix}}_{\frac{\partial J}{\partial \mathbf{U}}^T} \tag{3.24}$$

The right-hand side of this equation is the output linearization, which is a known quantity that depends on the particular output of interest. (For example, for a final-time output, all terms on the right would be zero except for $\partial J/\partial \mathbf{U}^{N_t}$.) On the left-hand side, we have the Jacobian transpose weighting the adjoint vector $\boldsymbol{\Psi}$. As suggested, it is clear from this diagram that to find $\boldsymbol{\Psi}$, we should first compute $\boldsymbol{\Psi}^{N_t}$, then perform a back-substitution to find the remaining adjoint values. Since the backward Euler method is adjoint-consistent[1], this is equivalent to performing a backward time march of the adjoint problem, starting from a final rather than initial condition.

3.1.2.1 State-Dependence of the Adjoint Equation

Note that for nonlinear problems, both $\frac{\partial \bar{\mathbf{R}}}{\partial \mathbf{U}}^T$ and $\frac{\partial J}{\partial \mathbf{U}}^T$ in the above equation must be evaluated at a particular primal state, $\mathbf{U}$. Thus, for these problems, the adjoint equation cannot be solved until some approximation of the primal solution is obtained. For this reason, the procedure (as depicted in Fig. 3.1) is typically to:

1. March the primal problem forward in time to compute the state $\mathbf{U}$.

2. Save $\mathbf{U}$ to disk.

3. March the adjoint equation backward in time, evaluating $\frac{\partial \bar{\mathbf{R}}}{\partial \mathbf{U}}^T\big|_{\mathbf{U}}$ and $\frac{\partial J}{\partial \mathbf{U}}^T\big|_{\mathbf{U}}$ with the corresponding $\mathbf{U}$ values at each time level.

[1] Note that not all temporal discretizations are adjoint-consistent. For example, the second-order backward difference (BDF2) method is adjoint-inconsistent if non-uniform time steps are used [85]. However, Runge-Kutta methods [84] – as well as the DG-in-time method employed in this work – are adjoint-consistent regardless of time-step size.

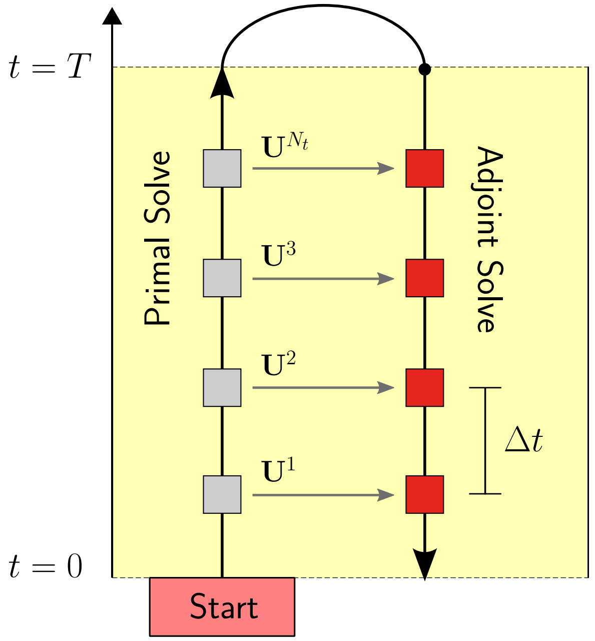


Figure 3.1: Unsteady primal and adjoint solution procedure. For nonlinear problems, the state **U** is first computed via a forward time march. It is then stored and used in the adjoint problem, which is solved by marching backward in time.

Remark 5. (**Solution Checkpointing**) In certain cases, storing the entire space-time state **U** may be prohibitive in terms of memory. In that case, a procedure known as "solution checkpointing" can be performed. This procedure consists of saving the solution at a relatively small number of "checkpoints" in time, and then re-solving for the solution between the checkpoints as the backward-in-time adjoint solve progresses. While this results in additional computational expense, it alleviates the memory requirements associated with storing the entire **U** vector.

3.1.2.2 Continuous Unsteady Adjoint

Note that the backward-in-time propagation of adjoint information can also be revealed by studying the continuous adjoint equation for unsteady problems. In fact, we have already discussed a continuous unsteady adjoint problem without recognizing it. In our original steady advection example (i.e. Eqn. 2.39), information flows in only one direction (to the right), just as in an unsteady problem it would flow only forward in time. Thus, a steady advection problem is actually "time-like," so that we could replace all instances of x with t to no effect. The primal problem would then become an ordinary differential equation in time, and the adjoint equation given by

Eqn. 2.46 would become:

$$\boxed{L^*\psi = -a\frac{d\psi}{dt} = g(t) \quad \text{and} \quad \psi\big|_T = 0} \, , \tag{3.25}$$

where T denotes the final time. We see then that we have a *final-time* condition on ψ (instead of an initial condition) as well as a negative sign in front of the $a\,d\psi/dt$ term – both of which indicate a backward flow of information in time.

3.2 Unsteady Error Estimation and Mesh Adaptation

A primary goal of this thesis is to use the unsteady adjoint to perform error estimation and mesh adaptation for problems of engineering interest. While the following chapters discuss the implementation of these procedures in a discontinuous Galerkin (DG) context, in this section we give a summary of the relevant ideas.

3.2.1 Unsteady Error Estimation

Assume that we have computed an unsteady output $J(\mathbf{U}_H)$ on a coarse space $\mathcal{V}_H$, and would like to estimate the amount of error in this output. As in the steady case, we can estimate the error with respect to a "fine space," $\mathcal{V}_h$. For unsteady problems, we can choose $\mathcal{V}_h$ to be a uniformly refined version of $\mathcal{V}_H$ in both space and time. For example, the space-time grid could be uniformly h-refined, or (if the numerical method allows) the solution order could be uniformly incremented in both space and time.

In either case, we would then derive the output error estimate

$$\boxed{\delta J_{\text{est}} = J_h(\mathbf{U}_h) - J_H(\mathbf{U}_H)} \tag{3.26}$$

in the same manner as for steady problems, by performing Taylor expansions of both the fine-space output and residuals about the coarse-space state. Indeed, looking back at Sec. 2.4.2, we see that we made no assumptions about whether the problem was steady or unsteady. Thus, the results from that section carry over directly, with the only difference being that we must now place a bar over the residual to denote that it is unsteady. For unsteady problems, the error estimate given by Eqn. 2.107 then becomes

$$\boxed{\delta J_{\text{est}} \;\approx\; -\mathbf{\Psi}_h^T\,\overline{\mathbf{R}}_h(\mathbf{U}_h^H) \;+\; \mathcal{O}(\delta\mathbf{U}^2)} \, . \tag{3.27}$$

This is just the product of the adjoint (computed on the fine space) and the fine-space residuals, $\overline{\mathbf{R}}_h(\mathbf{U}_h^H)$.

3.2.1.1 Fine-Space Unsteady Adjoint Solve

For unsteady problems, the cost of a full fine-space adjoint solve can be prohibitive, due to the fact that it involves a full backward time-march. In practice, rather than computing the fine-space adjoint $\boldsymbol{\Psi}_h$ exactly, we can either (1) compute a coarse space adjoint $\boldsymbol{\Psi}_H$ and perform several (inexact) smoothing iterations on the fine space, or (2) compute $\boldsymbol{\Psi}_H$ and perform a nearest-neighbors reconstruction in space and time to obtain an approximation for $\boldsymbol{\Psi}_h$. Since these procedures depend upon the numerical discretization, they will be discussed in more detail later on.

3.2.2 Unsteady Mesh Adaptation

In a similar manner as for steady problems, the error estimate in Eqn. 3.27 can be localized to individual space-time "elements" in the mesh. This enables us to determine which regions of the mesh are contributing most to the output error, and to then selectively adapt those regions.

For example, for a backward Euler discretization in time (combined with, e.g., a finite volume or finite element method in space), the error estimate can be written as a sum over all space-time elements in $\mathcal{V}_h$ as follows:

$$\delta J_{\mathrm{est}} \;\approx\; \sum_{k=1}^{N_t^h} \sum_{e=1}^{N_e^h} -\left(\boldsymbol{\Psi}_{h,e}^k\right)^T \overline{\mathbf{R}}_{h,e}^k(\mathbf{U}_h^H) \quad . \tag{3.28}$$

Here, N_t^h is the number of time steps on the fine space $\mathcal{V}_h$, while N_e^h is the number of spatial elements (which could theoretically depend on time). A pair of indices (e, k) then corresponds to a single space-time element, which in this case would mean a spatial element e over a particular time step, $[k-1, k]$. If $\mathcal{V}_H \subset \mathcal{V}_h$, the output error generated on each element (e, k) in $\mathcal{V}_h$ could then be summed over the "parent" element in $\mathcal{V}_H$ to obtain a coarse-space error indicator.

3.2.2.1 Space-Time Anisotropy

Unlike steady problems, having an estimate for the total amount of output error generated on each space-time element does not give us enough information to adapt the mesh. This is because, for unsteady problems, not only do we need to determine *which* space-time elements to adapt, but also whether to adapt them in space, time, or both. Thus, we need to determine how much of the output error generated on a given space-time element is due to the spatial vs. temporal discretization. In other words, we need a measure of **space-time *anisotropy***.

While the definition of this anisotropy indicator will be discussed later, once computed, it provides us with the fraction of output error on each element due to the spatial and temporal discretizations, which we will call

$$\beta_{e,k}^{\text{space}} \quad \text{and} \quad \beta_{e,k}^{\text{time}}, \tag{3.29}$$

respectively, where

$$\beta_{e,k}^{\text{time}} = 1 - \beta_{e,k}^{\text{space}}. \tag{3.30}$$

These error fractions can then be multiplied by the output error indicator on each element to determine the individual spatial and temporal errors.

3.2.2.2 Adaptation Mechanics

Finally, once we have computed an approximation of $\mathbf{\Psi}_h$ and the spatial/temporal errors on each element, we are ready to adapt the mesh. The next question is: *how* should we adapt the mesh?

In unsteady problems, critical flow features such as vortices move throughout the domain as time progresses. In order to maintain resolution of these (and other) features, we would like our mesh adaptation algorithm to "track" them as they propagate, provided they are deemed important by the adjoint. Thus, we would like the spatial mesh resolution to change **dynamically** in time (i.e. to be different at each time step). Furthermore, in order to address temporal errors, we would like to selectively **refine** or **coarsen** time step sizes. By combining these two procedures, we can arrive at an algorithm that is capable of eliminating errors as they propagate in both space and time.

Note that when deciding which elements and time steps to adapt, we do not use the error estimates directly as our adaptive indicator. Since in the end we are interested in reducing the most error for the least computational cost, this cost must be factored into the adaptive indicator as well. Thus, as our final adaptive indicator, we compute a "**figure of merit**" that represents the amount of **output error on a given element or time slab** *divided by* the **cost of refinement** (i.e. the number of new degrees of freedom introduced). Since refining a single spatial element introduces a different number of degrees of freedom than refining an entire time step, this figure of merit will give a different (and ideally, more computationally efficient) result than refining based on error values alone. Further details on the figure of merit will be provided in Chapter V.

3.2.2.3 Summary: Unsteady Error Estimation and Mesh Adaptation

Figure 3.2 provides an overview of the unsteady error estimation and mesh adaptation process, the steps of which are listed below.

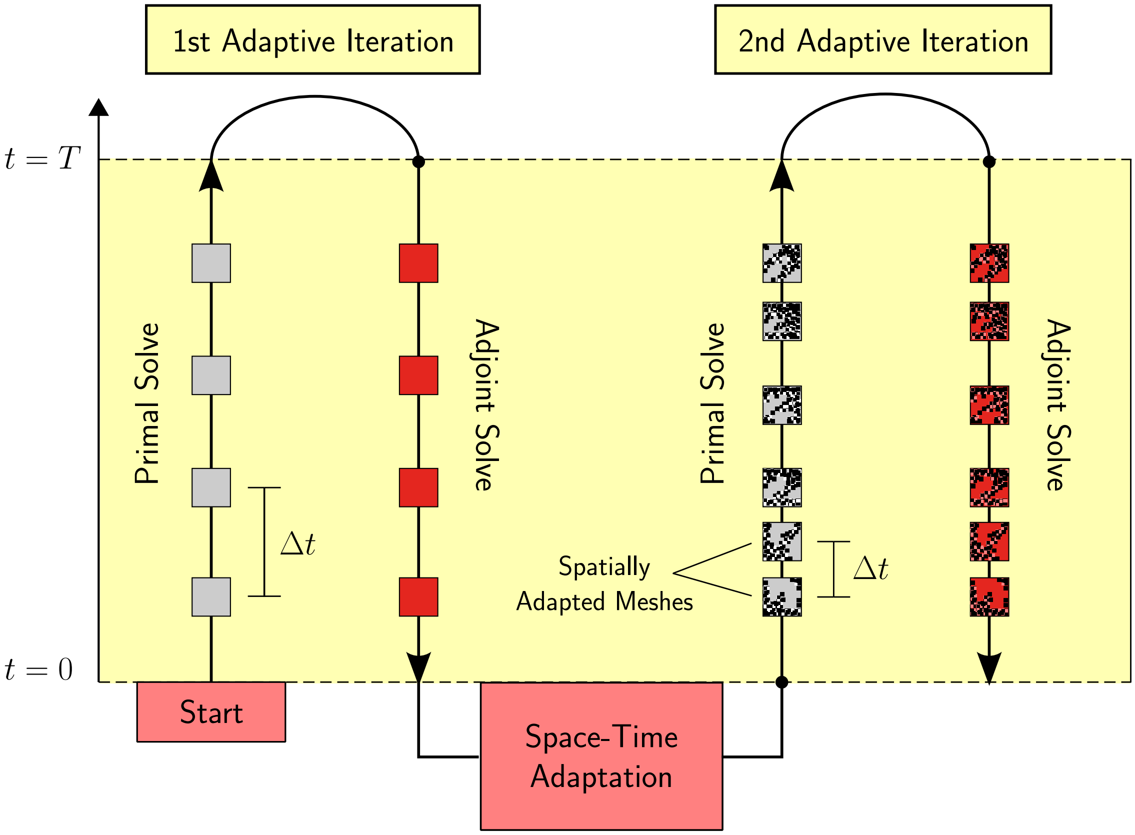


Figure 3.2: For each adaptive iteration, the primal problem is marched forward in time, and the adjoint problem is marched backward in time (on a uniformly refined space-time mesh). An output error estimate is then computed and mesh adaptation in space and time is performed. The spatial mesh is adapted differently at each time-step (as indicated by the variable shading in the 2nd iteration above), and time steps are selectively refined or coarsened.

Procedure:

1. Solve $\overline{\mathbf{R}}_H(\mathbf{U}_H) = 0$ on the coarse space, $\mathcal{V}_H$, to obtain $\mathbf{U}_H$.

2. Evaluate the output of interest, $J(\mathbf{U}_H)$.

3. Inject $\mathbf{U}_H$ to an order-incremented or uniformly refined space-time mesh, $\mathcal{V}_h$. (Compute $\mathbf{U}_h^H$.)

4. Evaluate the space-time residuals on $\mathcal{V}_h$ with the injected solution. (Compute $\overline{\mathbf{R}}_h(\mathbf{U}_h^H)$.)

5. Solve (or approximate) the fine-space adjoint equation

$$\left.\frac{\partial \overline{\mathbf{R}}_h}{\partial \mathbf{U}_h}^T\right|_{\mathbf{U}_h^H} \mathbf{\Psi}_h = \left.\frac{\partial J_h}{\partial \mathbf{U}_h}^T\right|_{\mathbf{U}_h^H} \tag{3.31}$$

   for $\mathbf{\Psi}_h$ by marching backward in time.

6. Compute the error estimate $\delta J_{\text{est}} \approx -\mathbf{\Psi}_h^T \overline{\mathbf{R}}_h(\mathbf{U}_h^H)$.

7. Correct the original $J(\mathbf{U}_H)$ with this error estimate. (Compute $J_{\text{corrected}} = J(\mathbf{U}_H) + \delta J_{\text{est}}$.) This corrected output is more accurate than $J(\mathbf{U}_H)$.

8. Localize δJ_{est} to individual space-time elements in the mesh.

9. Compute **space-time anisotropy** fractions $\beta_{e,k}^{\text{space}}$ and $\beta_{e,k}^{\text{time}}$, which indicate how much of δJ_{est} on each space-time element is due to the spatial vs. temporal discretization.

10. Compute a **figure of merit** representing the amount of output error eliminated by refining a given elment/step divided by the additional degrees of freedom introduced by the refinement.

11. Select a certain percentage of elements or time steps with the highest figure of merit and refine them. Coarsen a certain percentage of elements/steps with the lowest figure of merit.

12. Solve the primal problem on the new mesh and repeat steps 2-12 until the output error is driven below a desired tolerance.

CHAPTER IV

Governing Equations and Discretization

In this chapter, we give a brief overview of the governing equations and numerical discretizations used in this work. We begin with a review of unsteady conservation laws (such as the Navier-Stokes equations), which will be the focus of our output-based error estimation and mesh adaptation. We then describe the numerical discretizations used to approximate these equations: namely, finite element methods of a discontinuous Galerkin (DG) type.

4.1 Unsteady Conservation Laws

A general unsteady conservation law can be written as

$$\frac{\partial \mathbf{u}}{\partial t} + \nabla \cdot \vec{\mathbf{F}} = \mathbf{0}, \tag{4.1}$$

where the flux $\vec{\mathbf{F}}$ is given by

$$\vec{\mathbf{F}} = \vec{\mathbf{F}}^i(\mathbf{u}) - \vec{\mathbf{F}}^v(\mathbf{u}, \nabla\mathbf{u}). \tag{4.2}$$

Here, $\mathbf{u}(\vec{x}, t) \in \mathbb{R}^s$ is the state vector, $\vec{x} \in \mathbb{R}^d$ is the spatial coordinate, $t \in \mathbb{R}$ is time, and $\vec{\mathbf{F}}^i$ and $\vec{\mathbf{F}}^v \in \mathbb{R}^s$ are inviscid and viscous fluxes, respectively.

For example, the two-dimensional Navier-Stokes equations – which will be the focus of the subsequent chapter – have the following state vector and fluxes:

$$\mathbf{u} = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ \rho E \end{bmatrix}, \quad \vec{\mathbf{F}}^i = \begin{bmatrix} \rho u \\ \rho u^2 + p \\ \rho u v \\ \rho u H \end{bmatrix} \hat{i} + \begin{bmatrix} \rho v \\ \rho v u \\ \rho v^2 + p \\ \rho v H \end{bmatrix} \hat{j}\,, \tag{4.3}$$

and

$$\vec{\mathbf{F}}^v = \begin{bmatrix} 0 \\ \tau_{xx} \\ \tau_{xy} \\ \tau_{xx}u + \tau_{xy}v + \kappa_T \partial_x T \end{bmatrix} \hat{i} + \begin{bmatrix} 0 \\ \tau_{xy} \\ \tau_{yy} \\ \tau_{xy}u + \tau_{yy}v + \kappa_T \partial_y T \end{bmatrix} \hat{j} \, . \tag{4.4}$$

Here, τ_{ij} is the viscous stress tensor given by

$$\tau_{ij} = \mu(\partial_i v_j + \partial_j v_i) + \lambda \delta_{ij} \partial_k v_k \, , \tag{4.5}$$

where μ and λ are the dynamic and bulk viscosities, respectively, δ_{ij} is the Kronecker delta function, and summation is implied on k. The variable ρ is the fluid density, u and v are the x- and y-velocities (respectively), E is the total energy, p is the pressure, T is the temperature, H is the total enthalpy, and κ_T is the thermal conductivity. In this work, we treat the fluid as compressible and assume a calorically/thermally perfect gas with specific heat ratio $\gamma = 1.4$ and Prandtl number $Pr = .71$. In addition, we assume μ obeys Sutherland's law and λ obeys Stoke's hypothesis (i.e. $\lambda = -\frac{2}{3}\mu$). For additional details, see [33].

In order to solve equations of this form, we must discretize them in both space and time. In this work, we employ high-order finite element methods of a discontinuous Galerkin type for both spatial and temporal discretizations. The following sections provide a brief overview of these methods.

4.2 Discontinuous Galerkin (DG) Method

In this section, we describe the solution approximation and discretization procedures associated with a standard DG method.

4.2.1 Solution Approximation

To approximate the equation

$$\frac{\partial \mathbf{u}}{\partial t} + \nabla \cdot \vec{\mathbf{F}} = \mathbf{0} \tag{4.6}$$

on a given domain Ω (subject to appropriate boundary conditions), we first partition the domain into a tessellation T_H which consists of N_e elements – each with its own subdomain Ω_e – such that $T_H = \{\Omega_e\}$. (See Fig. 4.1a.)

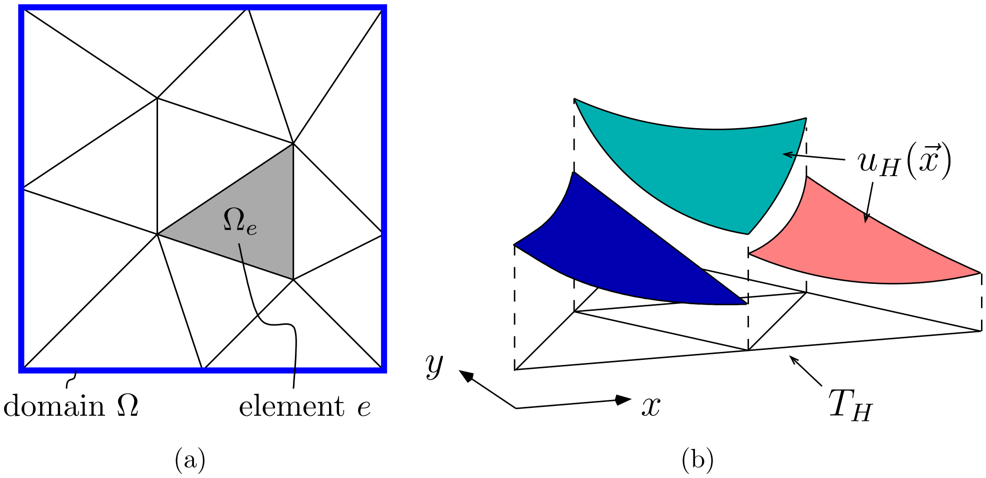


Figure 4.1: (a) Unstructured tessellation T_H (i.e. mesh) of a domain Ω, which can be used to compute a DG approximation to the solution of a PDE. (b) Solution approximation for a DG method, which is discontinuous between elements. (Figures reproduced from [32].)

As shown in Fig. 4.1b, a DG method then seeks a numerical approximation $\mathbf{u}_H$ to the solution, where $\mathbf{u}_H$ is a smooth function within each element, but is allowed to be discontinuous between elements.

On each element e, $\mathbf{u}_H$ is represented using a set of basis functions $\{\phi_{e,j}\}$ with local support, where j ranges from 1 to the total number of basis functions on the element. These $\phi_{e,j}$ are usually chosen to span a polynomial space of order p_e, which has an associated number of degrees-of-freedom (i.e. basis functions) N_{p_e}. (For example, for a full-order basis typically used on two-dimensional triangles, $N_{p_e} = (p_e+1)(p_e+2)/2$, while for a tensor product basis typically used on quadrilaterals, $N_{p_e} = (p_e + 1)^2$.)

The state $\mathbf{u}_H$ can then be written in terms of these basis functions as

$$\mathbf{u}_H(\vec{x}) = \sum_{e=1}^{N_e} \sum_{j=1}^{N_{pe}} \mathbf{U}_{e,j}(t)\phi_{e,j}(\vec{x}) \,, \tag{4.7}$$

where the $\mathbf{U}_{e,j}$ are the solution coefficients (i.e. the unknowns) weighting each basis function on element e. For unsteady problems, these coefficients are a function of time and can be updated according to a chosen time scheme.

The above description can be summarized more formally by stating that $\mathbf{u}_H \in \boldsymbol{\mathcal{V}}_H = [\mathcal{V}_H]^s$, where

$$\mathcal{V}_H = \{u \in L^2(\Omega) \, : \, u|_{\Omega_e} \in \mathcal{P}^{p_e}(\Omega_e) \quad \forall \Omega_e \in T_H\} \,. \tag{4.8}$$

Finally, note that in general, the polynomial order p_e can vary from element to element in the mesh, which is particularly useful if we wish to perform order adaptation – as employed later in this work.

4.2.2 Weak Form

With the solution approximation defined, a DG method then seeks to approximate the *weak form* of Eqn. 4.6. It therefore takes test functions $\mathbf{v}_H \in \boldsymbol{\mathcal{V}}_H$, enforces orthogonality of these test functions with respect to the PDE on each element, and integrates by parts, giving

$$\int\limits_{\Omega_e} \mathbf{v}_H^T \left[\frac{\partial \mathbf{u}}{\partial t} + \nabla \cdot \vec{\mathbf{F}} \right] d\Omega = \mathbf{0} \qquad \forall \mathbf{v}_H \in \boldsymbol{\mathcal{V}}_H \, , \tag{4.9}$$

$$\int\limits_{\Omega_e} \mathbf{v}_H^T \frac{\partial \mathbf{u}}{\partial t} \, d\Omega - \int\limits_{\Omega_e} (\nabla \mathbf{v}_H)^T \cdot \vec{\mathbf{F}} \, d\Omega + \int\limits_{\partial\Omega_e} \mathbf{v}_H^T \vec{\mathbf{F}} \cdot \vec{n} \, ds = \mathbf{0} \qquad \forall \mathbf{v}_H \in \boldsymbol{\mathcal{V}}_H \, , \tag{4.10}$$

where $\vec{n}$ is the normal vector pointing out of element e.

Next, inserting the state approximation $\mathbf{u}_H$ and replacing $\vec{\mathbf{F}} \cdot \vec{n}$ with a numerical flux $\hat{\mathbf{F}}$ on the element boundaries gives:

$$\int\limits_{\Omega_e} \mathbf{v}_H^T \frac{\partial \mathbf{u}_H}{\partial t} \, d\Omega - \int\limits_{\Omega_e} (\nabla \mathbf{v}_H)^T \cdot \vec{\mathbf{F}}(\mathbf{u}_H) \, d\Omega$$

$$+ \int\limits_{\partial\Omega_e} \mathbf{v}_H^T \hat{\mathbf{F}} \left(\mathbf{u}_H^+, \mathbf{u}_H^-, \nabla \mathbf{u}_H^+, \nabla \mathbf{u}_H^-, \vec{n} \right) ds \, + \, \text{D.C. Term} = \mathbf{0} \qquad \forall \mathbf{v}_H \in \boldsymbol{\mathcal{V}}_H \, . \tag{4.11}$$

The $()^+$ and $()^-$ notation above refers to quantities taken from the element interior and the neighbor element, respectively. On interior faces, a Roe approximate Riemann solver [82] is used to compute the convective portion of the flux $\hat{\mathbf{F}}$, while for the diffusive component, the second form of Bassi and Rebay (BR2) flux is employed [9]. Note that on boundary faces $\hat{\mathbf{F}}$ is replaced by a separate boundary flux, which incorporates a boundary state $\mathbf{u}_{H,B} = \mathbf{u}_{H,B}(\mathbf{u}_H^+, \text{BC Data})$, since in that case no neighbor-element state exists.

Finally, note that there is an additional ***dual consistency*** term (represented by the D.C. Term above) introduced by the BR2 discretization. If the viscous component of the flux $\vec{\mathbf{F}}$ is written in index notation as

$$F_{ik}^{\text{visc}} = A_{ijkl} \partial_j u_l \, , \tag{4.12}$$

then this dual consistency term has the form

$$\text{D.C. Term} = -\int\limits_{\partial\Omega_e} \partial_i v_k^+ A_{ijkl}^+ (u_l^+ - \hat{u}_l) n_j \, ds \, . \tag{4.13}$$

Here, $\hat{u}_l$ is defined to be the average of left and right states on an interior interface and $\mathbf{u}_{H,B}$ on a boundary face. The inclusion of this term, which vanishes upon mesh refinement and acts to symmetrize the diffusion discretization, leads to a dual consistent (i.e. adjoint consistent) method. More details on both the BR2 discretization and the boundary condition treatment can be found in [33, 9].

4.2.3 Discrete Form

To obtain a set of discrete residuals, we note that since $\mathcal{V}_H = \text{span}\{\phi_{e,j}\}$, we can choose $\mathbf{v}_H = \phi_{e,j}\mathbf{e}_r$ as the general form of a test function in Eqn. 4.11. (Here, $\mathbf{e}_r \in \mathbb{R}^s, r = 1...s$, is the Cartesian vector with a 1 in the rth entry and zeros in the others, which we use to denote the fact that each state component of the governing equations is weighted by an independent test function). If we then let the index e range over all N_e elements, j range over all N_{p_e} basis functions, and r range over all s state components, we obtain a set of $N = N_e \times N_{p_e} \times s$ equations (i.e. residuals). Each residual entry corresponds to a single choice of $\mathbf{v}_H$ in Eqn. 4.11.

The steady component of these residuals can then be lumped into a discrete ***spatial residual*** vector

$$\mathbf{R}(\mathbf{U}) \in \mathbb{R}^N \, , \tag{4.14}$$

while the unsteady term (i.e. the first term in Eqn. 4.11) can be written in terms of a discrete ***mass matrix***

$$\mathbf{M} \in \mathbb{R}^{N \times N} \, , \tag{4.15}$$

where

$$\mathbf{M}_{ij} = \mathbf{I}_s \int\limits_{\Omega} \phi_i \phi_j d\,\Omega \, . \tag{4.16}$$

Here, $\mathbf{I}_s \in \mathbb{R}^{s \times s}$ is an identity matrix and $1 \leq i, j \leq N$ range over all global degrees of freedom. Note that since each ϕ_i has support over only a single element, $\mathbf{M}$ is element-wise block diagonal.

The resulting equations can then be written in semi-discrete form as:

$$\mathbf{M}\frac{d\mathbf{U}}{dt} + \mathbf{R}(\mathbf{U}) = \mathbf{0}\,. \tag{4.17}$$

The final step is to choose the temporal discretization – i.e. the method used to approximate $d\mathbf{U}/dt$. In general, a standard multi-step scheme or Runge Kutta method could be employed. However, for error estimation purposes later on, it is more rigorous to remain within a finite element framework in both space and time. For this reason, we will use a so-called "DG-in-time" method to perform the temporal discretizaton. This method is discussed below.

4.3 DG-in-Time Method

In this section, we describe the solution approximation and discretization procedures associated with a DG-in-time scheme.

4.3.1 Solution Approximation

To integrate the equation

$$\mathbf{M}\frac{d\mathbf{U}}{dt} + \mathbf{R}\left(\mathbf{U}\right) = 0 \tag{4.18}$$

in time, a DG-in-time method first partitions the temporal domain into a set of time "**slabs**," or steps. Next, within a given time slab k (denoted by $T_k = [t_{k-1}, t_k]$), the state vector – denoted by $\mathbf{U}^k(t)$ – is represented as a smooth polynomial of order r. It can therefore be expressed as

$$\mathbf{U}^k(t) = \sum_{n=1}^{r+1} \mathbf{U}^{k,n}\varphi^n(t), \qquad \text{for } t \in T_k\,, \tag{4.19}$$

where the $\varphi^n(t)$ are temporal basis functions spanning a one-dimensional order-r space.

As with a spatial DG scheme, these $\varphi^n(t)$ have local support over each slab k, allowing for discontinuities between neighboring slabs. This situation is illustrated in Fig. 4.2. (Note that in this work, we assume a nodal Lagrange basis in time, so that $\varphi^1(t)$ and $\varphi^{r+1}(t)$ are equal to 1 at the beginning and end of a given time slab, respectively.)

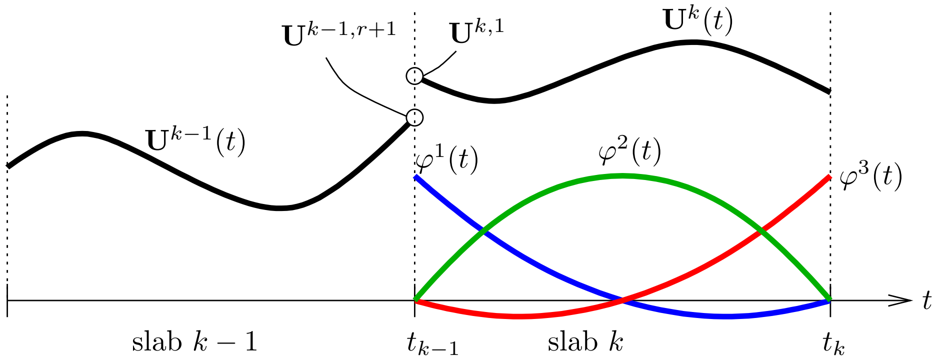


Figure 4.2: The DG-in-time method treats the numerical solution as a set of discontinuous polynomials on each time "slab." (Figure reproduced from [31].)

4.3.2 Weak Form

To integrate Eqn. 4.18 in time, the DG-in-time method approximates its weak form in a similar manner as spatial DG. Thus, we multiply Eqn. 4.18 by test functions $\varphi^m(t)$ and integrate by parts over each time slab k, resulting in

$$-\int_{T_k} \left(\frac{d\varphi^m(t)}{dt} \mathbf{M}\mathbf{U}^k(t) \right) dt \; + \; \varphi^m(t)\mathbf{M}\mathbf{U}(t) \Big|_{t_{k-1}}^{t_k} + \int_{T_k} \varphi^m(t) \mathbf{R}\left(\mathbf{U}^k(t)\right) dt = 0\,. \tag{4.20}$$

Next, since the state is double-valued at the ends of each slab, we must choose whether to use the "upwind" or "downwind" values of the state when evaluating the above boundary term. To obey the direction of physical information propagation, we use an upwind flux to evaluate this term. Thus, we replace $\mathbf{U}(t_{k-1})$ in the above equation with the right-most state on the previous time slab, $\mathbf{U}^{k-1,r+1}$. Making this substitution and replacing $\mathbf{U}^k(t)$ in the first term with its expansion in (4.19), we then obtain

$$-\int_{T_k} \left(\frac{d\varphi^m(t)}{dt} \mathbf{M} \sum_n \mathbf{U}^{k,n} \varphi^n(t) \right) dt + \varphi^m(t_k)\mathbf{M}\mathbf{U}^{k,r+1} - \varphi^m(t_{k-1})\mathbf{M}\mathbf{U}^{k-1,r+1} + \int_{T_k} \varphi^m(t)\mathbf{R}\left(\mathbf{U}^k(t)\right) dt = 0\,. \tag{4.21}$$

In the first term, only $\varphi^m(t)$ and $\varphi^n(t)$ depend on time, so $\mathbf{M}$ and $\mathbf{U}^{k,n}$ can be brought outside of the integral, giving

$$-\left(\int\limits_{T_k} \varphi^n(t)\frac{d\varphi^m(t)}{dt}dt\right)\mathbf{M}\mathbf{U}^{k,n} + \varphi^m(t_k)\mathbf{M}\mathbf{U}^{k,r+1} - \varphi^m(t_{k-1})\mathbf{M}\mathbf{U}^{k-1,r+1} + \int\limits_{T_k} \varphi^m(t)\mathbf{R}\left(\mathbf{U}^k(t)\right)dt\; = 0, \tag{4.22}$$

with implied summation over the n index. To further simplify, we can combine the first and second terms and write:

$$\underbrace{a^{m,n}\,\mathbf{M}\mathbf{U}^{k,n} - \varphi^m(t_{k-1})\mathbf{M}\mathbf{U}^{k-1,r+1} + \int\limits_{T_k} \varphi^m(t)\mathbf{R}\left(\mathbf{U}^k(t)\right)dt}_{\overline{\mathbf{R}}^{k,m}}\; = 0, \tag{4.23}$$

where $\overline{\mathbf{R}}^{k,m}$ is now defined to be the ***unsteady residual*** vector and the $a^{m,n}$ are constant coefficients given by

$$a^{m,n} = \varphi^n(t_k)\,\varphi^m(t_k) - \int\limits_{t_{k-1}}^{t_k} \varphi^n(t)\frac{d\varphi^m(t)}{dt}\,dt. \tag{4.24}$$

By the properties of Lagrange bases, the first term in this expression for $a^{m,n}$ is nonzero only when $n = m = r+1$, and we recover the second term in Equation 4.22, as desired.

Finally, for simulations in which the spatial order p_e changes with time, Equation 4.23 must be altered slightly, since the spatial basis functions (and hence $\mathbf{M}$) change in time. In these cases, we can write the final form of the unsteady residual as follows:

$$\underbrace{a^{m,n}\,\mathbf{M}^{k,k}\mathbf{U}^{k,n} - \varphi^m(t_{k-1})\,\mathbf{M}^{k,k-1}\mathbf{U}^{k-1,r+1} + \int\limits_{T_k} \varphi^m(t)\,\mathbf{R}\left(\mathbf{U}^k(t)\right)\,dt}_{\overline{\mathbf{R}}^{k,m}} = 0, \tag{4.25}$$

where $\mathbf{M}^{k,l}$ denotes the mass matrix formed from the spatial basis functions on slabs k and l, which may in general differ. This $\mathbf{M}^{k,l}$ can be written explicitly on each element as

$$\mathbf{M}^{k,l}\Big|_e = \left[\int\limits_{\Omega_e} \phi^k_{e,i}(\vec{x})\, \phi^l_{e,j}(\vec{x})\, d\Omega \right] \otimes \mathbf{I}_s \,, \tag{4.26}$$

where $\mathbf{I}_s$ is the $s \times s$ identity matrix. If an element transitions from a low p to a higher p at a given time, then the action of this matrix is the same as if a lossless injection of the low-p state into the high-p space were performed. On the other hand, if an element transitions from a high p to a low p, the result is the same as if a least-squares projection of the high-p state into the low-p space were performed.

4.3.2.1 Newton Solve

Since the spatial residual $\mathbf{R}\left(\mathbf{U}^k(t)\right)$ in Eqn. 4.25 is evaluated with the current state on time slab k, the unsteady residual $\overline{\mathbf{R}}^k$ represents an ***implicit*** set of equations over each time slab. (Here, we have dropped the nodal index m, so that $\overline{\mathbf{R}}^k \in \mathbb{R}^{N(r+1)}$ is a column vector of the residuals associated with all temporal nodes in slab k.)

To solve this equation, we can therefore perform a Newton iteration on each time slab, where the update vector $\Delta\mathbf{U}^k \in \mathbb{R}^{N(r+1)}$ is computed by solving the following linear system:

$$\frac{\partial \overline{\mathbf{R}}^k}{\partial \mathbf{U}^k}\bigg|_{\mathbf{U}^k} \Delta\mathbf{U}^k = -\overline{\mathbf{R}}^k\left(\mathbf{U}^k\right) \,. \tag{4.27}$$

The residual Jacobian in the above equation has dimension $N(r+1) \times N(r+1)$, which for large problems may be infeasible to invert or even store. For these problems, rather than computing $\frac{\partial \overline{\mathbf{R}}^k}{\partial \mathbf{U}^k}$ exactly, we instead perform an approximate linearization. This approximation consists of evaluating the linearization of the spatial residual (i.e. the rightmost term in Eqn. 4.25) at just a single point within the time slab (e.g. the midpoint of the slab), and then employing an approximate solution scheme introduced by Richter [81, 37] to solve Eqn. 4.27. In the end, while this linearization procedure is inexact, so long as the Newton iteration converges, the unsteady residual (Eqn. 4.25) is satisfied exactly.

CHAPTER V

Output-based Mesh Adaptation for Navier-Stokes Simulations on Deformable Domains

With the DG discretization and the theory of output-based error estimation discussed, we now attempt to apply these methods to practical aerospace problems. Specifically, we present an output-based mesh adaptation strategy for high-order Navier-Stokes simulations on deforming domains. These simulations have far-reaching applications, from bio-inspired flight to aircraft maneuver and flutter analysis. After presenting the method, we evaluate this output-based strategy on a series of low Reynolds number flapping-wing flight problems in both two and three dimensions. This work can be found in published form in [61, 58].

5.1 Introduction

As we have discussed, when performing a practical CFD simulation, the primary interest of the user is often to compute a certain output – such as lift or drag – accurately. If this is the case, then resolving all regions of the flow with equal precision is both unnecessary and inefficient. Instead, a better strategy is to (1) compute the output and estimate its corresponding error, (2) determine where this output error originates from, and (3) drive this error down by targeting the regions of the mesh responsible for it. These steps can be accomplished by applying the adjoint-based error estimation and mesh adaptation techniques discussed earlier.

The concept of adjoint-based (or output-based) error estimation is not new, and has received considerable attention for steady problems [79, 10, 49, 95, 88, 74, 36, 18, 104, 19, 24, 101]. However, attention has only recently shifted toward use of these methods for unsteady problems. One issue is that unsteady problems can – particularly at high Reynolds numbers – become chaotic, making them extremely sensitive to perturbations. This causes the adjoint, which reflects this sensitivity, to grow unbounded as it is marched backward in time. This is an issue that deserves attention, and the proper way to define the adjoint for chaotic problems is the subject of ongoing research [98, 99]. That said, there are many unsteady problems whose nature is not chaotic, and for these cases, the unsteady adjoint can provide vital sensitivity data. This is the case for the low-Reynolds-number problems considered here.

Another issue is that computing an unsteady adjoint is expensive, due in part to the significant storage requirements for nonlinear problems, in which the entire space-time state must be saved to disk for use in the adjoint problem. Techniques for reducing these costs – such as checkpointing methods – are also the subject of ongoing research [45, 83, 63, 73, 102].

Fortunately, however, the large costs of an unsteady adjoint can come with an even larger payoff. These payoffs have been observed in optimization problems [72, 97, 92] where it is vital to obtain accurate sensitivities with respect to a large number of inputs. This same reasoning extends to unsteady CFD simulations, where small errors made in remote regions of the space-time domain can coalesce into large errors in the output of interest. In many cases, identifying these errors with an unsteady adjoint and eliminating them through mesh adaptation can lead to significant computational savings.

Some work in this area has already been done. In a finite element context, output error estimation for scalar parabolic problems was studied in [70] and [86], with a high-order reconstructed adjoint used to drive dynamic space-time mesh adaptation. Recently, spatial-only [7] and combined space-time [12] adaptation have been performed for two-dimensional Navier-Stokes simulations on static domains. Within a finite volume framework, temporal-only adaptation has been shown for the Euler equations on deforming domains [67, 68], while on static domains a spatial-only [11] and preliminary space-time adaptation [38] have been demonstrated. Finally, in recent work [37, 66, 35, 103], combined space-time adaptation strategies for Euler and Navier-Stokes simulations on static domains have been presented. In each of the above works, improvements in output convergence were obtained through the use of unsteady output-based adaptation.

In this work, we extend these output-based techniques to problems that are receiving increasing attention: those involving low-Reynolds-number flapping-wing flight. These problems have applications in the biological fields as well as in the field of Micro Aerial Vehicles (MAVs), the latter of which has seen rapid growth over the last decade. Obtaining accuracy in quantities such as the time-averaged lift-to-drag ratio and the total energy expenditure is particularly important for the design of these vehicles.

To simulate the type of motion inherent in these problems, the mesh must deform in some way. Thus, extending output-based techniques to these flows means extending them to problems with *deforming domains*. In particular, our goal in this work is to extend these techniques to high-order Navier-Stokes simulations on deforming domains in both two and three dimensions.

To perform the simulations, we first implement a discontinuous Galerkin Arbitrary Lagrangian-Eulerian (ALE) method introduced by Persson *et al* [78], which can handle arbitrary deformations of the mesh (which are analytically prescribed). We then implement an output-based space-time adaptation procedure, with **dynamic** ***p*****-refinement** (as well as **time slab refinement** and **coarsening**) used to resolve the flow features deemed critical by the adjoint.

Furthermore, on deformable domains, satisfaction of a so-called **Geometric Conservation Law (GCL)** requires the adjoint system to change. In this chapter, we present the required modifications to the adjoint system and derive a discrete adjoint for the GCL equation itself. We then incorporate this adjoint into the error estimation and adaptation strategy and evaluate its performance on a series of 2D and 3D test cases. These cases verify the validity of the output-based strategy and show the computational savings that can be achieved.

5.2 Arbitrary Lagrangian-Eulerian Mapping

The Navier-Stokes equations can be written in conservation form as

$$\frac{\partial \mathbf{u}}{\partial t} + \nabla \cdot \vec{\mathbf{F}}(\mathbf{u}, \nabla \mathbf{u}) = \mathbf{0}, \qquad \vec{\mathbf{F}} = \vec{\mathbf{F}}^i(\mathbf{u}) - \vec{\mathbf{F}}^v(\mathbf{u}, \nabla \mathbf{u}), \tag{5.1}$$

where $\mathbf{u}(\vec{x}, t) \in \mathbb{R}^s$ is the state vector, $\vec{x} \in \mathbb{R}^d$ is the spatial coordinate, $t \in \mathbb{R}$ is time, and $\vec{\mathbf{F}}^i$ and $\vec{\mathbf{F}}^v$ are the inviscid and viscous fluxes, respectively [9]. For the cases considered in this work, the physical domain in which Eqn. 5.1 holds is deforming in time, and a direct solution would be difficult to obtain. Instead, we can map the problem to a fixed reference domain and solve using an arbitrary Lagrangian-Eulerian (ALE) approach. Since the reference domain remains fixed for all time, standard numerical methods for static problems can then be employed to obtain the solution. A simple and effective ALE method for DG was recently introduced by Persson *et* *al.*[78], and we follow their approach here. In this method, the physical equations are transformed to equivalent reference-domain equations, and the deformation of the mesh is encapsulated within the mapping between these domains.

5.2.1 ALE mapping

Following Persson *et al.*, we begin by assuming we have some physical domain $v(t)$, which is deforming in time. We would like to map this problem to a fixed "reference" domain V, which remains static for all time. This V is the domain on which we will generate the mesh and compute the solution. Thus, in the end, we will be able to solve the deforming domain problem on a fixed mesh, which will allow us to employ a standard DG method in both space and time.

We assume that each point $\vec{X}$ in the reference domain can be mapped to a deformed location $\vec{x}$ in the physical domain via some analytical mapping

$$\vec{x} = \vec{x}(\vec{X}, t)\,. \tag{5.2}$$

For example, a mapping we will use later in two dimensions is:

$$\begin{aligned} x_1 &= X_1 + 2.0\sin\left(\frac{2\pi X_1}{20}\right)\sin\left(\frac{\pi X_2}{7.5}\right)\sin\left(\frac{2\pi t}{3}\right), \\ x_2 &= X_2 + 1.5\sin\left(\frac{2\pi X_1}{20}\right)\sin\left(\frac{\pi X_2}{7.5}\right)\sin\left(\frac{4\pi t}{3}\right). \end{aligned} \tag{5.3}$$

This mapping causes points within the physical domain $v(t)$ to "wave" back and forth sinusoidally in time. Other useful mappings include rotation, translation, and shear transformations, which can be found in many texts.

The general transformation between reference and physical domains is summarized graphically in Fig. 5.1, and definitions of relevant variables are given in Table 5.1.

Table 5.1: Definitions of variables used in the ALE mapping. Bold indicates a state vector and an arrow indicates a spatial vector.

$\vec{X}$	=	reference-domain coordinates	$\vec{x}$	=	physical-domain coordinates
$\mathbf{u}_X$	=	state on reference domain	$\mathbf{u}$	=	physical state
$\vec{\mathbf{F}}_X$	=	flux vector on reference domain	$\vec{\mathbf{F}}$	=	flux vector on physical domain
dA	=	differential area on reference domain	da	=	differential area on physical domain
$\vec{N}$	=	unit normal on reference domain	$\vec{n}$	=	unit normal on physical domain
$\mathcal{G}$	=	mapping Jacobian matrix	$\vec{v}_G$	=	grid velocity
g	=	determinant of mapping Jacobian			

Points in the physical domain may contract or expand relative to those in the

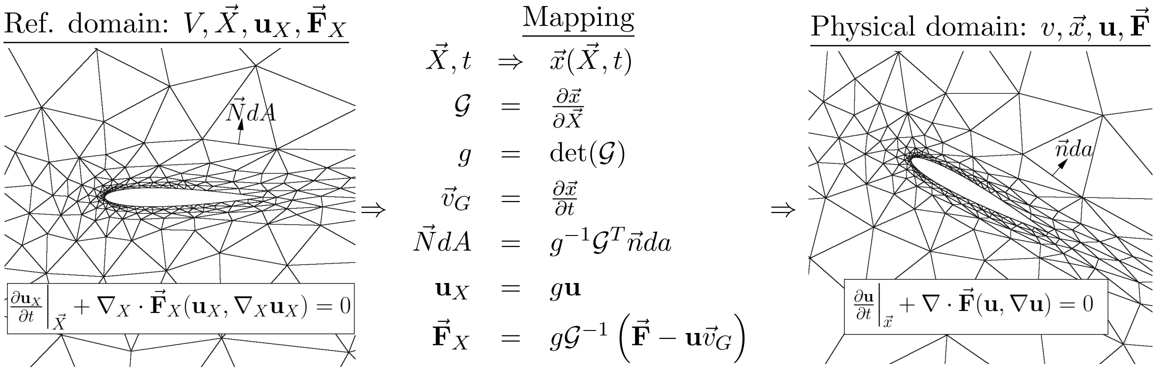

Ref. domain: $V, \vec{X}, \mathbf{u}_X, \vec{\mathbf{F}}_X$
$\vec{N}dA$
$\left.\frac{\partial \mathbf{u}_X}{\partial t}\right|_{\vec{X}} + \nabla_X \cdot \vec{\mathbf{F}}_X(\mathbf{u}_X, \nabla_X \mathbf{u}_X) = 0$
$\Rightarrow$
Mapping
$\vec{X}, t \Rightarrow \vec{x}(\vec{X}, t)$
$\mathcal{G} = \frac{\partial \vec{x}}{\partial \vec{X}}$
$g = \det(\mathcal{G})$
$\vec{v}_G = \frac{\partial \vec{x}}{\partial t}$
$\vec{N}dA = g^{-1}\mathcal{G}^T\vec{n}da$
$\mathbf{u}_X = g\mathbf{u}$
$\vec{\mathbf{F}}_X = g\mathcal{G}^{-1}\left(\vec{\mathbf{F}} - \mathbf{u}\vec{v}_G\right)$
$\Rightarrow$
Physical domain: $v, \vec{x}, \mathbf{u}, \vec{\mathbf{F}}$
$\vec{n}da$
$\left.\frac{\partial \mathbf{u}}{\partial t}\right|_{\vec{x}} + \nabla \cdot \vec{\mathbf{F}}(\mathbf{u}, \nabla \mathbf{u}) = 0$

Figure 5.1: Summary of the mesh motion mapping. The physical domain deforms according to a user-defined analytical mapping, $x(\vec{X}, t)$. The equations are then mapped to and solved on the reference domain, which remains fixed for all time. Note: when denoting reference domain quantities, we use a subscript X rather than $\vec{X}$ for visual clarity.

reference domain, so that a differential volume dv in the physical domain is related to that in the reference domain via

$$dv = gdV, \tag{5.4}$$

where g is the determinant of the mapping Jacobian defined in Fig. 5.1. Thus, if g is less than unity, a region in the physical domain is contracted relative to the corresponding region in the reference domain, and vice versa. Furthermore, infinitesimal vectors transform via the relation

$$d\vec{l} = \mathcal{G}d\vec{L}, \tag{5.5}$$

where $\mathcal{G}$ is the mapping Jacobian.

By combining these differential volume and vector relationships and noting that $dV = d\vec{L} \cdot \vec{N}\, dA$ and $dv = d\vec{l} \cdot \vec{n}\, da$, it is straightforward to derive the following relationships between normal vectors and areas[1] on the two domains:

$$\vec{n}da = g\mathcal{G}^{-T}\vec{N}dA \qquad \text{and} \qquad \vec{N}dA = g^{-1}\mathcal{G}^T\vec{n}da\,. \tag{5.6}$$

Here, the transpose appears as a result of expressing the dot products in the definitions of dv and dV using their equivalent transpose notation.

[1] Or lengths, in two dimensions.

5.2.2 Conservation Law on Reference Domain

With these transformations in mind, our goal now is to express the Navier-Stokes equations (i.e. Eqn. 5.1) on the reference domain. We begin by taking the conservation law

$$\frac{\partial \mathbf{u}}{\partial t} + \nabla \cdot \vec{\mathbf{F}}(\mathbf{u}, \nabla \mathbf{u}) = \mathbf{0}, \tag{5.7}$$

and applying the divergence theorem over the physical domain $v(t)$:

$$\int\limits_{v(t)} \frac{\partial \mathbf{u}}{\partial t} dv + \int\limits_{\partial v} \vec{\mathbf{F}} \cdot \vec{n}\, da = \mathbf{0}. \tag{5.8}$$

Here, $\vec{n}$ is the outward-pointing normal on $v(t)$, while da is a differential area (or length, in 2D) on the boundary of $v(t)$.

Next, we transform the flux integral (i.e. the second term in Eqn. 5.8) to the reference domain by using the relations in Eqn. 5.6:

$$\int\limits_{\partial v} \vec{\mathbf{F}} \cdot \vec{n}\, da \;=\; \int\limits_{\partial V} \vec{\mathbf{F}} \cdot \underbrace{(g \mathcal{G}^{-T} \vec{N})\, dA}_{\vec{n}\, da} \;=\; \int\limits_{\partial V} (g \mathcal{G}^{-1} \vec{\mathbf{F}}) \cdot \vec{N}\, dA. \tag{5.9}$$

Note that the integrals are now over the reference domain boundary, ∂V. Similarly, using the relations in Eqn. 5.6 along with Liebniz's rule[2], we can transform the first term in Eqn. 5.8 (i.e. the time derivative term) as follows:

$$\begin{aligned}
\int\limits_{v(t)} \frac{\partial \mathbf{u}}{\partial t}\, dv \;&=\; \frac{d}{dt} \int\limits_{v(t)} \mathbf{u}\, dv \;-\; \int\limits_{\partial v} (\mathbf{u} \vec{v}_G) \cdot \vec{n}\, da \\
&=\; \frac{d}{dt} \int\limits_{V} \mathbf{u}\, \underbrace{g\, dV}_{dv} \;-\; \int\limits_{\partial V} (\mathbf{u} \vec{v}_G) \cdot \underbrace{(g \mathcal{G}^{-T} \vec{N})\, dA}_{\vec{n}\, da} \\
\int\limits_{v(t)} \frac{\partial \mathbf{u}}{\partial t}\, dv \;&=\; \int\limits_{V} \frac{\partial (g \mathbf{u})}{\partial t}\, dV \;-\; \int\limits_{\partial V} (g \mathbf{u} \mathcal{G}^{-1} \vec{v}_G) \cdot \vec{N}\, dA.
\end{aligned} \tag{5.10}$$

In the last step, the time derivative is brought back inside the integral since the domain V does not change in time.

Substituting Eqns. 5.10 and 5.9 back into Eqn. 5.8 then gives the integral form of

[2] Note that Liebniz's rule is just Reynolds Transport Theorem in this case.

the conservation law on the reference domain:

$$\int\limits_V \frac{\partial(g\mathbf{u})}{\partial t}\,dV + \int\limits_{\partial V} \underbrace{(g\mathcal{G}^{-1}\vec{\mathbf{F}} - g\mathbf{u}\mathcal{G}^{-1}\vec{v}_G)}_{\vec{\mathbf{F}}_X} \cdot \vec{N}\,dA. \tag{5.11}$$

If we define the reference-domain state to be the quantity inside the time derivative,

$$\mathbf{u}_X \equiv g\mathbf{u}, \tag{5.12}$$

and the reference-domain flux to be the quantity inside the boundary integral,

$$\vec{\mathbf{F}}_X \equiv g\mathcal{G}^{-1}\vec{\mathbf{F}} - g\mathbf{u}\mathcal{G}^{-1}\vec{v}_G, \tag{5.13}$$

then Eqn. 5.11 becomes

$$\int\limits_V \frac{\partial \mathbf{u}_X}{\partial t} dV + \int\limits_{\partial V} \vec{\mathbf{F}}_X \cdot \vec{N}\,dA = \mathbf{0}. \tag{5.14}$$

This is the standard form of an integral conservation law on V. Applying the divergence theorem one more time then gives the differential form of the conservation law on V:

$$\left.\frac{\partial \mathbf{u}_X}{\partial t}\right|_X + \nabla_X \cdot \vec{\mathbf{F}}_X(\mathbf{u}_X, \nabla_X \mathbf{u}_X) = \mathbf{0}. \tag{5.15}$$

We see then that by redefining the state variable and flux to be

$$\mathbf{u}_X = g\mathbf{u} \qquad \text{and} \tag{5.16}$$

$$\vec{\mathbf{F}}_X = g\mathcal{G}^{-1}\vec{\mathbf{F}} - g\mathbf{u}\mathcal{G}^{-1}\vec{v}_G \tag{5.17}$$

rather than the standard $\mathbf{u}$ and $\vec{\mathbf{F}}$, we can solve a deforming domain problem on a fixed mesh in a similar manner as any other conservation law.

Looking back at the definition of the physical flux, $\vec{\mathbf{F}}$, we see from Eqn. 5.1 that it can be split into separate inviscid and viscous components. Inserting these components into Eqn. 5.17 then allows us to write the reference domain flux in terms of inviscid and viscous components as well. If we lump the grid velocity term in

Eqn. 5.17 into the inviscid flux, we can define:

$$\vec{\mathbf{F}}_X = \vec{\mathbf{F}}^i_X - \vec{\mathbf{F}}^v_X, \tag{5.18}$$

$$\vec{\mathbf{F}}^i_X = g\mathcal{G}^{-1}\vec{\mathbf{F}}^i - \mathbf{u}_X\mathcal{G}^{-1}\vec{v}_G, \tag{5.19}$$

$$\vec{\mathbf{F}}^v_X = g\mathcal{G}^{-1}\vec{\mathbf{F}}^v, \tag{5.20}$$

where $\vec{\mathbf{F}}^i_X$ and $\vec{\mathbf{F}}^v_X$ are the inviscid and viscous components of the reference-domain flux, respectively.

5.2.3 Implementation

Here, we mention a few points related to the implementation of the above equations. Additional details can be found in Appendix B.

5.2.3.1 Inviscid Flux

As derived above, the inviscid flux on the reference domain has the form

$$\vec{\mathbf{F}}^i_X = g\mathcal{G}^{-1}\left(\vec{\mathbf{F}}^i - \mathbf{u}\vec{v}_G\right) . \tag{5.21}$$

The $\mathbf{u}\vec{v}_G$ term represents a Galilean transformation due to the change of reference frames (and is nonzero even in the case of an undeformed – e.g. a purely translating – grid), while the multiplication by $g\mathcal{G}^{-1}$ takes into account the local grid contraction/dilation effects. This multiplication by $g\mathcal{G}^{-1}$ can be implemented as a postprocessing step during the flux computation. The subtraction of $\mathbf{u}\vec{v}_G$ is slightly more code-intrusive since the Roe flux on interior faces must be modified to act on the quantity $\left(\vec{\mathbf{F}}^i - \mathbf{u}\vec{v}_G\right) \cdot \vec{n}$ rather than just $\vec{\mathbf{F}}^i \cdot \vec{n}$. Furthermore, the boundary conditions, which are specified in the physical domain, must incorporate any motion of the grid boundary itself. For example, near a wall boundary, flow tangency must be satisfied, which means the fluid velocity near the wall must have a normal component equal to the normal component of the grid velocity $\vec{v}_G$ there.

5.2.3.2 Viscous Flux

The viscous flux on the reference domain is given by Eqn. 5.20 as

$$\vec{\mathbf{F}}^v_X = g\mathcal{G}^{-1}\vec{\mathbf{F}}^v. \tag{5.22}$$

While this flux is not influenced by a translation of the grid, the incorporation of mesh contraction/dilation (i.e. the multiplication by $g\mathcal{G}^{-1}$) can again be performed as a post-processing step in the flux computation. Furthermore, since the computation of the physical flux $\vec{\mathbf{F}}^v$ involves the application of a diffusion matrix to the physical gradient $\nabla\mathbf{u}$, additional terms arise when this $\nabla\mathbf{u}$ is computed from the reference-domain state. These terms arise from applying the chain and product rules, since

$$\begin{aligned}\nabla\mathbf{u} = \frac{\partial\mathbf{u}}{\partial x_j} = \frac{\partial(g^{-1}\mathbf{u}_X)}{\partial X_d}\frac{\partial X_d}{\partial x_j} &= \left(g^{-1}\frac{\partial\mathbf{u}_X}{\partial X_d} - g^{-2}\frac{\partial g}{\partial X_d}\mathbf{u}_X\right)\frac{\partial X_d}{\partial x_j} \\ &= g^{-1}\left(\frac{\partial\mathbf{u}_X}{\partial X_d} - g^{-1}\frac{\partial g}{\partial X_d}\mathbf{u}_X\right)\mathcal{G}_{dj}^{-1}\,. \end{aligned} \tag{5.23}$$

Here, we see the two separate terms that emerge from the computation of $\nabla\mathbf{u}$, both of which must be included in the computation of $\vec{\mathbf{F}}^v$ on the reference domain. For more details, see Appendix B.

5.3 The Geometric Conservation Law

The ability to preserve a free stream is a desirable property of numerical schemes. However, for a DG method employing a finite-dimensional basis (e.g. a set of polynomials in space-time), a constant state $\bar{\mathbf{u}}$ in the physical domain will generally not be a solution to the discrete form of Eqn. 5.15 in the reference domain. This means that for an arbitrary motion of the mesh, an initially free-stream state will not be preserved.

This lack of free-stream preservation can be explained by noting that, for general mappings, the Jacobian g will be non-polynomial in both space and time. Hence, the reference state $\mathbf{u}_X = g\bar{\mathbf{u}}$ will likewise be non-polynomial, which means it cannot be represented exactly with the reference-domain bases. This inexact representation will introduce both spatial and temporal errors into an initially free-stream state, which manifest as conservation errors that accumulate in time.

If we wish to eliminate these conservation errors, a separate Geometric Conservation Law (GCL) can be enforced alongside the governing equations. The idea of the GCL is twofold: (1) to address the representation issues mentioned above, replace the analytical g with a new variable, $\bar{g}$, which is a polynomial approximation to g in space-time; and (2) to ensure that this $\bar{g}$ actually allows for free-stream preservation, compute it from the following equation (as discussed in [78]):

$$\frac{\partial \bar{g}}{\partial t} - \nabla_X \cdot (g\mathcal{G}^{-1}\vec{v}_G) = 0. \tag{5.24}$$

This equation ensures that the change in element area (i.e. the change in $\bar{g}$) is directly linked to what the grid velocities on element boundaries "claim" it should be. Hence, there is no disagreement between grid velocities and Jacobians on what the current mesh geometry is, and in that sense we have "geometric conservation."

The strategy then is to use this $\bar{g}$ to define a new reference-domain state $\mathbf{u}_{\bar{X}} = \bar{g}g^{-1}\mathbf{u}_X = \bar{g}\mathbf{u}$, which is used instead of the original state $\mathbf{u}_X = g\mathbf{u}$. If $\bar{g}$ is discretized using the same spatial basis as the state and is marched in time using the same unsteady solver, a free-stream state $\bar{\mathbf{u}}$ will be preserved. In the end, what we have done is replaced the original analytical g with a "best fit" space-time polynomial $\bar{g}$, which then makes the free-stream reference state $\mathbf{u}_{\bar{X}} = \bar{g}\bar{\mathbf{u}}$ exactly representable in the discrete space.

Once $\bar{g}$ is obtained on each element, it is used instead of g to convert the stored reference state to the physical state. The final form of the reference domain equation is then

$$\left.\frac{\partial \mathbf{u}_{\bar{X}}}{\partial t}\right|_X + \nabla_X \cdot \vec{\mathbf{F}}_{\bar{X}}(\mathbf{u}_{\bar{X}}, \nabla_X \mathbf{u}_{\bar{X}}, \bar{g}) \quad = \quad \mathbf{0}, \tag{5.25}$$

where $\vec{\mathbf{F}}_{\bar{X}}$ is just $\vec{\mathbf{F}}_X$ but with $\mathbf{u}_{\bar{X}}/\bar{g}$ replacing $\mathbf{u}_X/g$ in the calculation of the physical state.

Fig. 5.2 shows the effect of the GCL on a free-stream preservation test. In this case, we solve the Navier-Stokes equations on a rectangular domain using the analytical sinusoidal mapping defined in Eqn. 5.3. The temporal and spatial discretization are both discontinuous Galerkin, with order $r = 1$ and p, respectively. Without the GCL, the free-stream solution is not preserved, though the L^2 error converges with both h and p refinement of the spatial mesh. With the GCL, the free-stream is maintained to residual tolerance, which was approximately ten orders of magnitude for these runs. To achieve this level of accuracy, high-order quadrature rules are required, with rules of order $6p$ used in this case to demonstrate the GCL's full effect. In practical cases we use more modest rules, namely $2p + 5$ for the Navier-Stokes equations, since discretization errors tend to dominate and make high quadrature rules unnecessary.

Finally, note that Eqn. 5.24 has introduced an additional numerical quantity, $\bar{g}$, which is subject to discretization errors just like the state quantities. This will be

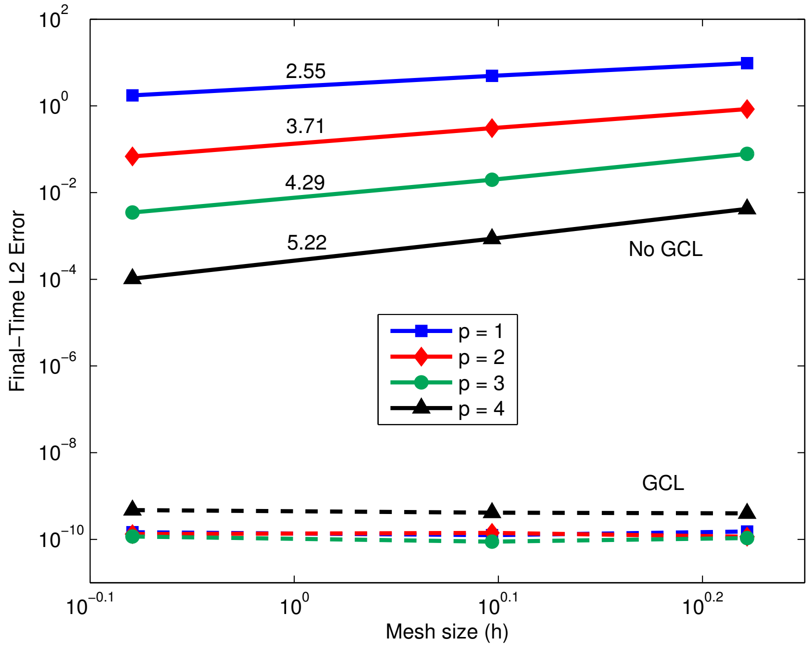


Figure 5.2: Free-stream errors with and without the GCL. A large number of time steps is used so that the spatial error dominates the temporal error in each case.

relevant for error estimation later on.

5.3.1 Blended Mesh Motion

With or without the GCL, the ALE method described above requires an analytically defined mapping between reference and physical domains. Therefore, the user must explicitly prescribe the motion throughout the domain. However, for many applications, we need only a certain object – such as an airfoil – to move. In these cases, defining a mapping throughout the entire domain would be unnecessary and (oftentimes) difficult. Instead, if only a portion of the domain needs to move, we can prescribe a mapping centered about this region, then smoothly blend the mapping to zero as we move radially outward. This allows the farfield boundaries and any other objects in the domain to remain uninfluenced by the local region of mesh motion.

A typical scenario is to have an inner disk in 2D (or a sphere in 3D) undergo a prescribed rigid-body motion, and to then blend this motion into the static mesh via a so-called blending function. Fig. 5.3 shows a diagram of the rigid-body and blending regions around a theoretical object, while Fig. 5.4 shows an example of this blending for 2D airfoil and 3D wing meshes. As Persson *et al.* present in [78], a

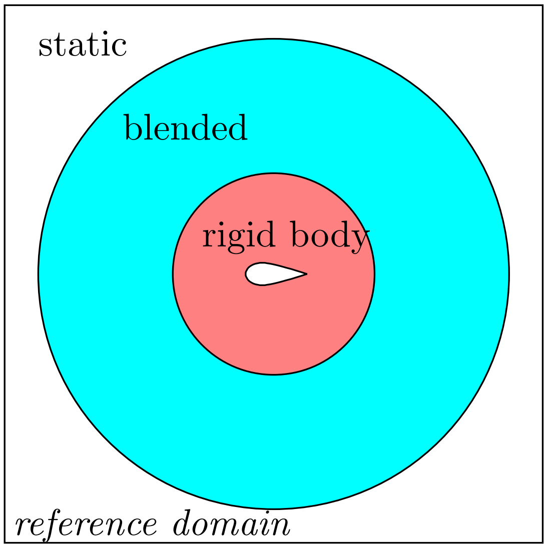


Figure 5.3: If an object such as an airfoil needs to move, a rigid-body motion around that object can be blended into the static mesh further out. This keeps the boundaries of the domain fixed while the object moves.

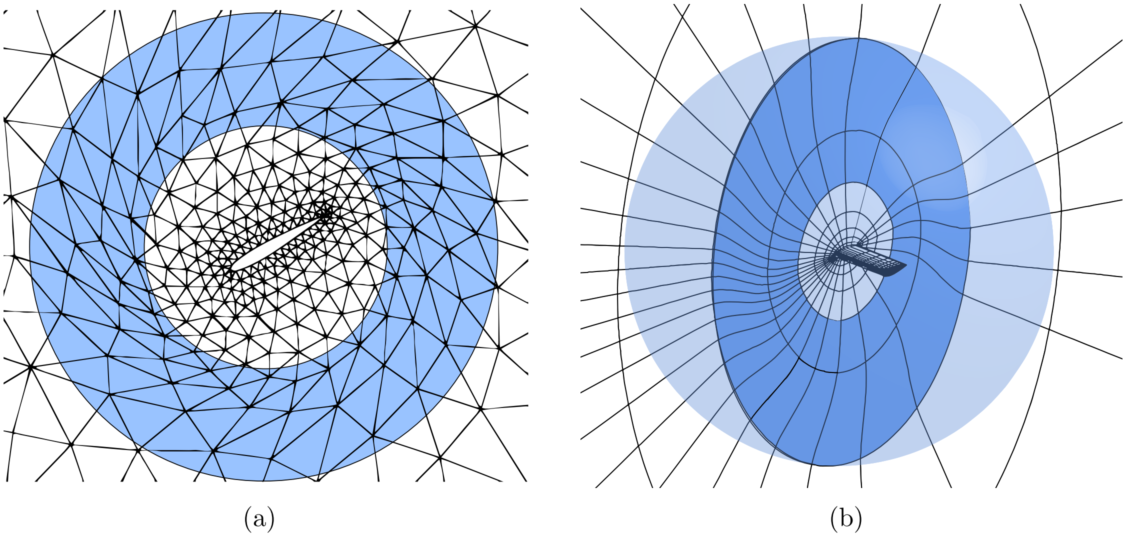


Figure 5.4: Airfoil (a) and wing (b) undergoing analytical motions. The blue regions are those in which the prescribed inner motion is blended into the static outer mesh. The boundaries of these blending regions are circular in 2D and spherical in 3D.

polynomial blending function is a simple way to transition between the deforming and static regions. The strategy is then as follows.

First, we assume that an object centered about some point $\vec{X}_C$ in the reference domain undergoes a transformation $\mathcal{T}$. This transformation could represent e.g. a rigid-body rotation or translation. We then introduce the following parameters:

$$
\begin{aligned}
\vec{X}_C &= \text{center of rigid-body motion region (i.e. region where } \mathcal{T} \text{ acts)} \\
C &= \text{circle centered about } \vec{X}_C \text{ (or sphere in 3D)} \\
R_C &= \text{radius of } C\text{, within which } \mathcal{T} \text{ applies} \\
D &= \text{distance from } \vec{X}_C \text{ beyond which } \mathcal{T} \text{ has no effect (i.e. static region)} \\
d(\vec{X}) &= ||\vec{X} - \vec{X}_C|| - R_C = \text{distance a point } \vec{X} \text{ lies outside of } C \\
r_n(\xi) &= \text{1D } n\text{th-order blending polynomial} \\
b(d) &= \text{blending function}
\end{aligned}
\tag{5.26}
$$

Here, $r_n(\xi)$ is an (odd) nth-order polynomial such that $r_n(0) = 0$, $r_n(1) = 1$ and its first $(n-1)/2$ derivatives vanish at $\xi = 0$ and $\xi = 1$. For example, for $n = 5$, it is given by $r_5(\xi) = 10\xi^3 - 15\xi^4 + 6\xi^5$. The blending function $b(d(\vec{X}))$ is then defined in the following manner:

$$
b(d) = \begin{cases} 0 & \text{if } d < 0 & \text{(rigid body region)} \\ 1 & \text{if } d > D & \text{(static region)} \\ r_n(d/D) & \text{otherwise} & \text{(blended region)} \end{cases}
\tag{5.27}
$$

Here, D is chosen by the user, and represents the furthest distance from $\vec{X}_C$ that any motion due to $\mathcal{T}$ is felt. Thus, beyond D, the domain remains static. Note that D should typically be chosen as large as the problem allows, in order to create a smoothly blended mapping. If D is too small, then elements in the blending region could become significantly warped, generating large errors.

With these definitions, the blended mapping over the entire domain is given by:

$$
\vec{x} = b(d)\vec{X} + (1 - b(d))\mathcal{T}(\vec{X})\,.
\tag{5.28}
$$

From this expression, we see that if we are in a region where $b(d)$ is zero, the mapping reduces to $\mathcal{T}$ – i.e. the rigid-body transformation. On the other hand, if $b(d)$ is unity, $\vec{x} = \vec{X}$, meaning the region is static. Finally, if $0 \leq d \leq D$, we are between these regions and the mapping is blended smoothly by the polynomial r_n.

5.4 Primal Discretization

With the mapping and blending functions defined, we move on to the solution procedure. To discretize the state and GCL equations (Eqns. 5.25 and 5.24), we use a DG finite element method in both space and time. As discussed in Chapter IV, each element has a spatial approximation order p_e, which can differ between elements and vary dynamically in time. Temporally, the discretization consists of time slabs on which the solution variation is approximated with polynomials of order r. At a given time, the width of a time slab (i.e. Δt) is the same for all spatial elements in the mesh. However, the widths of slabs at different times will vary once adaptation is performed.

5.4.1 Approximation

Each space-time element is identified by two indices, (e, k), where e is the spatial element index and k is the time slab index. On each element (e, k), the state and GCL variable are approximated as

$$\mathbf{u}_{\bar{X},H}(\vec{X},t)\Big|_{e,k} = \underbrace{\mathbf{U}^{k,n}_{\bar{X},H,e,j}}_{\in \mathbb{R}^s} \underbrace{\phi^k_{H,e,j}(\vec{X})}_{\text{order } p_{e,k}} \underbrace{\varphi^n_H(t)}_{\text{order } r}, \tag{5.29}$$

$$\bar{g}_H(\vec{X},t)\Big|_{e,k} = \underbrace{\bar{G}^{k,n}_{H,e,j}}_{\in \mathbb{R}^1} \underbrace{\phi^k_{H,e,j}(\vec{X})}_{\text{order } p_{e,k}} \underbrace{\varphi^n_H(t)}_{\text{order } r}, \tag{5.30}$$

where s is the number of governing equations, $1 \leq j \leq \text{dof}(p_{e,k})$ is the spatial degree-of-freedom index, and $1 \leq n \leq r+1$ is the temporal degree-of-freedom index. Note that for a full-order approximation on triangles (which we use for our two-dimensional simulations), $\text{dof}(p_{e,k}) = (p_{e,k}+1)(p_{e,k}+2)/2$. Furthermore, in this work, we use Lagrange bases in both space and time, with the spatial basis functions $\phi^k_{H,e,j}(\vec{X})$ specific to a given element and time slab, and the temporal basis functions $\varphi^n_H(t)$ the same for each time slab. Finally, the subscript H on the above terms just indicates that these quantities are defined on our current ("coarse") space-time mesh. When discussing error estimation later on, we will be dealing with a fine mesh as well, which will be denoted with a lowercase h.

For compactness of notation, we lump all spatial degrees of freedom associated with time node n on slab k into one vector, for both the state and the GCL variable,

$$\mathbf{U}^{k,n}_H = \left\{\mathbf{U}^{k,n}_{\bar{X},H,e,j}\right\}_{\forall e,j} \in \mathbb{R}^{sN^k_H} \quad \text{and} \quad G^{k,n}_H = \left\{\bar{G}^{k,n}_{H,e,j}\right\}_{\forall e,j} \in \mathbb{R}^{N^k_H}, \tag{5.31}$$

where $N_H^k = \sum_e \mathrm{dof}(p_{e,k})$ is the total number of spatial degrees of freedom on time slab k. As a shorthand, we will denote by $\mathbf{U}_H^k$ and G_H^k the sets of unknowns over a whole time slab, k. Finally, the set of states over the entire space-time domain will be referred to simply as $\mathbf{U}_H$ and G_H.

5.4.2 Residuals

A nonlinear system of equations on each time slab is obtained by substituting the approximations from Eqns. 5.29 and 5.30 into Eqns. 5.25 and 5.24, multiplying by test functions in the same space as the approximation functions, and integrating by parts to incorporate discontinuities at time slab and spatial element interfaces. Thus, we take the equations

$$\begin{aligned}
&\int_{T_k}\int_{\Omega_{e,k}} \varphi_H^n(t)\,\phi_{H,j}(\vec{X}) \underbrace{\left[\left.\frac{\partial \mathbf{u}_{\bar{X}}}{\partial t}\right|_X + \nabla_X \cdot \vec{\mathbf{F}}_{\bar{X}}\left(\mathbf{u}_{\bar{X}}, \nabla_X \mathbf{u}_{\bar{X}}, \bar{g}\right)\right]}_{\text{Navier-Stokes Eqns.}} d\Omega\, dt \;=\; \mathbf{0} \\
&\int_{T_k}\int_{\Omega_{e,k}} \varphi_H^n(t)\,\phi_{H,j}(\vec{X}) \underbrace{\left[\frac{\partial \bar{g}}{\partial t} - \nabla_X \cdot \left(g \mathcal{G}^{-1} \vec{v}_G\right)\right]}_{\text{GCL Eqn.}} d\Omega\, dt \;=\; 0
\end{aligned} \tag{5.32}$$

(where $1 \leq j \leq \mathrm{dof}(p_{e,k})$, $1 \leq n \leq r+1$, and $\Omega_{e,k}$ and T_k represent a given spatial element and time slab, respectively) and express them in terms of the following discrete $r+1$ residual vectors on each time slab k:

$$\begin{aligned}
\text{State Residual:} \quad \overline{\mathbf{R}}_{\mathbf{U},H}^{k,m} \;\equiv\; & a^{m,n}\,\mathbf{M}_H^{k,k}\mathbf{U}_H^{k,n} - \varphi_H^m(t_{k-1})\,\mathbf{M}_H^{k,k-1}\mathbf{U}_H^{k-1,r+1} \\
& + \int_{t_{k-1}}^{t_k} \varphi_H^m(t)\,\mathbf{R}_{\mathbf{U},H}\left(\mathbf{U}_H^k(t), G_H^k(t)\right)\, dt = \mathbf{0},
\end{aligned} \tag{5.33}$$

$$\begin{aligned}
\text{GCL Residual:} \quad \overline{R}_{G,H}^{k,m} \;\equiv\; & a^{m,n}\,\mathbf{M}_H^{k,k}G_H^{k,n} - \varphi_H^m(t_{k-1})\,\mathbf{M}_H^{k,k-1}G_H^{k-1,r+1} \\
& + \int_{t_{k-1}}^{t_k} \varphi_H^m(t)\,R_{G,H}(t)\, dt = \mathbf{0},
\end{aligned} \tag{5.34}$$

The derivation of these residuals follows from Chapter IV. The temporal approximations of the state and GCL variable on time slab k are given by $\mathbf{U}_H^k(t) = \sum_n \mathbf{U}_H^{k,n}\varphi_H^n(t)$ and $G_H^k(t) = \sum_n G_H^{k,n}\varphi_H^n(t)$, respectively, and the spatial state and

GCL residuals lie in $\mathbf{R}_{\mathbf{U},H} \in \mathbb{R}^{sN_H^k}$ and $R_{G,H} \in \mathbb{R}^{N_H^k}$. Note that the GCL residual is independent of the state, while conversely, the state residual depends on both the state and GCL variable. This coupling will be reflected in the adjoint system derived later.

As discussed in Chapter IV, the flux functions used in the spatial discretization are Roe's inviscid flux [82] and the second form of Bassi and Rebay (BR2) for the viscous flux [9]. Finally, the $\mathbf{M}_H^{k,l}$ terms in Eqn. 5.33 refer to the mass matrices formed from the spatial basis functions on neighboring time slabs k and l, which will in general differ due to dynamic order refinement. Note that we slightly abuse matrix-vector product notation with the mass matrix in Eqn. 5.34, since $G_H^{k,n}$ has a state rank of 1 rather than s. However, there is no rank mismatch in the actual implementation, since the mass matrix acts independently on the state components of a vector.

5.4.3 Implementation

Since the state residual depends on the GCL variable, the GCL must be advanced in time before advancing the state. This is a small cost due to the simplicity of the GCL residual and the fact that $\bar{g}_H$ is a scalar field. In addition, because of the nonlinear nature of the ALE mapping, we increase the default numerical quadrature order used to evaluate the spatial integrals in $\mathbf{R}_{\mathbf{U},H}$ and $R_{G,H}$. For the results in this paper, we use a quadrature order of $2p + 5$ rather than the standard order of $2p + 1$. Finally, as discussed in Chapter IV, we note that the unsteady solver used in this work is an iterative technique based on an inexact linearization of the unsteady residuals, so that the residual Jacobian storage costs do not exceed those of a steady solve [81, 37].

5.5 Output Error Estimation and Adjoint Formulation

Above, we have described the technique for simulating problems on deforming domains. Our main purpose is then to answer: for an output like lift or drag at a given time, (1) what is the error in this output computed with our current space-time mesh, and (2) how can we adapt the mesh to efficiently reduce this error?

Fundamentally, errors in an output arise due to discretization errors, which are a consequence of solving the governing equations in a finite-dimensional space. Typically, outputs of interest such as lift or drag are functions of the physical state, $\mathbf{u}$, which means errors in these outputs are directly related to errors in the physical state. However, from the transformations in Sec. 5.2, when the GCL is used, the physical state is a function of both the reference state $\mathbf{u}_{\bar{X}}$ and the GCL variable $\bar{g}$. Therefore, for simulations satisfying the GCL, the error in a given output will be related to errors in *both* $\mathbf{u}_{\bar{X}}$ and $\bar{g}$.

To estimate the effect of these errors on an output J_H, we will again require use of a fine-space adjoint and a fine-space residual evaluation, as discussed in Chapters II and III. However, the presence of the GCL makes the adjoint and error estimation procedure more complicated. In this section, we will describe how to modify this procedure while employing the GCL. The derivation will follow a similar line of reasoning as in Chapters II and III.

5.5.1 Estimating Errors with the GCL

To estimate the error in an output J_H computed on our current mesh (denoted by subscript H), we consider the value of the output on a finer space-time mesh, denoted by subscript h. This fine-space output J_h, were it known, could be expanded about the coarse-space solution with a truncated Taylor series as follows:

$$J_h\left(\mathbf{U}_h, G_h\right) \;\approx\; \underbrace{J_h\left(\mathbf{U}_h^H, G_h^H\right)}_{\approx J_H} + \left.\frac{\partial J_h}{\partial \mathbf{U}_h}\right|_{\left(\mathbf{U}_h^H, G_h^H\right)} \delta\mathbf{U} \;+\; \left.\frac{\partial J_h}{\partial G_h}\right|_{\left(\mathbf{U}_h^H, G_h^H\right)} \delta G\,, \tag{5.35}$$

$$\Rightarrow \quad J_h - J_H \approx \left.\frac{\partial J_h}{\partial \mathbf{U}_h}\right|_{\left(\mathbf{U}_h^H, G_h^H\right)} \delta\mathbf{U} \;+\; \left.\frac{\partial J_h}{\partial G_h}\right|_{\left(\mathbf{U}_h^H, G_h^H\right)} \delta G\,. \tag{5.36}$$

Eqn. 5.36 gives an estimate of the output error $J_h - J_H$ between fine and coarse spaces. Here, $\mathbf{U}_h^H$ and G_h^H are just injections of the coarse solution ($\mathbf{U}_H$, G_H) into the fine space, while the perturbations $\delta\mathbf{U} = \mathbf{U}_h - \mathbf{U}_h^H$ and $\delta G = G_h - G_h^H$ are the differences between fine and coarse solutions, which are at this point unknown.

Next, if we assume the fine space equations are satisfied, our set of state and GCL residuals over the whole space-time domain, $\overline{\mathbf{R}}_{\mathbf{U},h}$ and $\overline{R}_{G,h}$, must be zero. Just as we did with J, we can expand these residuals about the coarse-space solution to get

$$\overline{\mathbf{R}}_{\mathbf{U},h}\left(\mathbf{U}_h, G_h\right) = 0 \approx \overline{\mathbf{R}}_{\mathbf{U},h}\left(\mathbf{U}_h^H, G_h^H\right) + \left.\frac{\partial \overline{\mathbf{R}}_{\mathbf{U},h}}{\partial \mathbf{U}_h}\right|_{\left(\mathbf{U}_h^H, G_h^H\right)} \delta\mathbf{U} + \left.\frac{\partial \overline{\mathbf{R}}_{\mathbf{U},h}}{\partial G_h}\right|_{\left(\mathbf{U}_h^H, G_h^H\right)} \delta G \tag{5.37}$$

$$\overline{R}_{G,h}\left(G_h\right) = 0 \approx \overline{R}_{G,h}\left(G_h^H\right) \;+\; \left.\frac{\partial \overline{R}_{G,h}}{\partial G_h}\right|_{G_h^H} \delta G\,. \tag{5.38}$$

Due to discretization errors, the solution on the coarse mesh will generally not satisfy the fine-space equations, and hence $\overline{\mathbf{R}}_{\mathbf{U},\,h}\,(\mathbf{U}_h^H, G_h^H)$ and $\overline{R}_{G,\,h}\,(G_h^H)$ will be nonzero.

From Eqn. 5.38 we can obtain an expression for δG, which we can then use in Eqn. 5.37 to obtain $\delta\mathbf{U}$. Doing so gives

$$\delta G \approx -\left[\frac{\partial \overline{R}_{G,\,h}}{\partial G_h}\right]^{-1} \overline{R}_{G,\,h}\,(G_h^H)\,, \tag{5.39}$$

$$\delta\mathbf{U} \approx -\left[\frac{\partial \overline{\mathbf{R}}_{\mathbf{U},\,h}}{\partial \mathbf{U}_h}\right]^{-1} \left(\overline{\mathbf{R}}_{\mathbf{U},\,h}\,(\mathbf{U}_h^H, G_h^H) - \frac{\partial \overline{\mathbf{R}}_{\mathbf{U},\,h}}{\partial G_h}\left[\frac{\partial \overline{R}_{G,\,h}}{\partial G_h}\right]^{-1} \overline{R}_{G,\,h}\,(G_h^H)\right), \tag{5.40}$$

where all Jacobian matrices and their inverses are evaluated using the $(\mathbf{U}_h^H,\, G_h^H)$ states injected from the coarse mesh. Since Eqns. 5.39 and 5.40 are computable from the coarse solution alone, we could in theory calculate $\delta\mathbf{U}$ and δG directly, then insert them into Eqn. 5.36 to obtain the output error estimate. However, while this would provide us with a total output error estimate, it would not tell us *where* in the mesh that output error originated from. Since we are interested in adapting the mesh to reduce the error, this is a problem, since it would effectively leave us blind.

The solution to this problem can be found by inserting Eqns. 5.39 and 5.40 into Eqn. 5.36 and grouping all terms multiplying the residual perturbations:

$$\begin{aligned} J_h - J_H \;\approx\; & -\underbrace{\frac{\partial J_h}{\partial \mathbf{U}_h}\left[\frac{\partial \overline{\mathbf{R}}_{\mathbf{U},\,h}}{\partial \mathbf{U}_h}\right]^{-1}}_{\mathbf{\Psi}_{\mathbf{U},\,h}^T} \overline{\mathbf{R}}_{\mathbf{U},\,h}\,(\mathbf{U}_h^H, G_h^H) \\ & - \underbrace{\left\{\frac{\partial J_h}{\partial G_h} - \frac{\partial J_h}{\partial \mathbf{U}_h}\left[\frac{\partial \overline{\mathbf{R}}_{\mathbf{U},\,h}}{\partial \mathbf{U}_h}\right]^{-1} \frac{\partial \overline{\mathbf{R}}_{\mathbf{U},\,h}}{\partial G_h}\right\}\left[\frac{\partial \overline{R}_{G,\,h}}{\partial G_h}\right]^{-1}}_{\Psi_{G,\,h}^T} \overline{R}_{G,\,h}\,(G_h^H)\,. \end{aligned} \tag{5.41}$$

We can now define the quantity multiplying the state residual pertubation as the state adjoint $\mathbf{\Psi}_{\mathbf{U},h}^T$, and the quantity multiplying the GCL residual pertubation as the GCL adjoint $\Psi_{G,h}^T$. These adjoint vectors represent the sensitivity of the output J to perturbations in the state and GCL residuals, respectively. By rearranging the terms in the definition of the adjoints, we see that the following equations must hold:

$$\text{State Adjoint Eqn.:} \quad \left(\frac{\partial \overline{\mathbf{R}}_{\mathbf{U},h}}{\partial \mathbf{U}_h}\right)^T \boldsymbol{\Psi}_{\mathbf{U},h} = \left(\frac{\partial J_h}{\partial \mathbf{U}_h}\right)^T, \tag{5.42}$$

$$\text{GCL Adjoint Eqn.:} \quad \left(\frac{\partial \overline{R}_{G,h}}{\partial G_h}\right)^T \Psi_{G,h} = \left(\frac{\partial J_h}{\partial G_h}\right)^T - \left(\frac{\partial \overline{\mathbf{R}}_{\mathbf{U},h}}{\partial G_h}\right)^T \boldsymbol{\Psi}_{\mathbf{U},h}\,. \tag{5.43}$$

Notice that we have two linear systems for $\boldsymbol{\Psi}_{\mathbf{U},h}$ and $\Psi_{G,h}$, which can be solved via the same iterative method used for the primal problem. Furthermore, once obtained, $\boldsymbol{\Psi}_{\mathbf{U},h}$ and $\Psi_{G,h}$ provide the sensitivity of J to residual perturbations at specific locations in the mesh, which is essential for driving adaptation. Finally, the output error estimate, defined as $\delta J_{\text{est}} = J_h - J_H$, is readily computable from the adjoints as

$$\delta J_{\text{est}} \approx -\boldsymbol{\Psi}^T_{\mathbf{U},h}\,\overline{\mathbf{R}}_{\mathbf{U},h}\,(\mathbf{U}^H_h, G^H_h) \;-\; \Psi^T_{G,h}\,\overline{R}_{G,h}\,(G^H_h)\,. \tag{5.44}$$

Thus, by solving for the state and GCL adjoints on a finer mesh, we can obtain both an adaptive indicator and an estimate for the output error using only our original solution on the coarse mesh.

5.5.2 Some notes on the GCL adjoint

For static simulations (or those not employing a GCL), the state adjoint would provide all relevant sensitivity information, and the GCL adjoint would not exist. For deforming domain problems satisfying the GCL, the GCL adjoint relates errors made in the motion itself to errors in the output of interest. For example, the analytical $g(t)$ that describes the desired motion could have a sinusoidal variation in space and time, but $\bar{g}(t)$ will only approximate this variation with a polynomial. Therefore, on a given mesh, we may be sufficiently resolving the states $\mathbf{U}$, but could be getting an accurate answer for the wrong motion! Injecting the coarse G_H into the fine-space $\overline{R}_{G,h}$ and weighting by $\Psi_{G,h}$ tells us where the mesh should be refined to more accurately represent the true motion.

The form of the GCL adjoint equation (5.43) is also worth noting. We see that the state adjoint $\boldsymbol{\Psi}_{\mathbf{U},h}$ appears as a source term on the right hand side, while conversely, the state adjoint is independent of $\Psi_{G,h}$ in Eqn. 5.42. This is due to the fact that the state residual depends on the GCL variable, but the GCL residual does not depend on the state. From an implementation standpoint, it means that on a given time slab the state adjoint must be computed before the GCL adjoint can be obtained.

5.5.3 Unsteady Adjoint Equations

Equations 5.42 and 5.43 represent the adjoint equations at a high level, with all space-time residuals and adjoints lumped into the $\overline{\mathbf{R}}_{\mathbf{U}}/\overline{R}_G$ and $\mathbf{\Psi}_{\mathbf{U}}/\Psi_G$ terms. To actually solve them, we need to consider the details of the space-time Jacobian. The sparsity pattern of the full space-time Jacobian for a DG-in-time discretization is shown in Fig. 5.5. This pattern reveals the temporal dependencies of the residuals on the states, which enables us to write the following form of the adjoint equations:

Jacobian matrix

	$\mathbf{U}^1$	G^1	$\mathbf{U}^2$	G^2	$\mathbf{U}^3$	G^3
$\overline{\mathbf{R}}^1_{\mathbf{U}}$	●	●				
$\overline{R}^1_G$		●				
$\overline{\mathbf{R}}^2_{\mathbf{U}}$	●		●	●		
$\overline{R}^2_G$		●		●		
$\overline{\mathbf{R}}^3_{\mathbf{U}}$			●		●	●
$\overline{R}^3_G$				●		●

Jacobian matrix transpose

	$\overline{\mathbf{R}}^1_{\mathbf{U}}$	$\overline{R}^1_G$	$\overline{\mathbf{R}}^2_{\mathbf{U}}$	$\overline{R}^2_G$	$\overline{\mathbf{R}}^3_{\mathbf{U}}$	$\overline{R}^3_G$
$\mathbf{U}^1$	●		●			
G^1	●	●		●		
$\mathbf{U}^2$			●		●	
G^2			●	●		●
$\mathbf{U}^3$					●	
G^3					●	●

Figure 5.5: Sparsity patterns for the coupled state/GCL system in a DG-in-time discretization, shown for the first three time slabs. Each entry has the dimension of a Jacobian matrix (with respect to either $\mathbf{U}_h$ or G_h) over the temporal nodes and spatial domain.

$$\left(\frac{\partial \overline{\mathbf{R}}^{k,m}_{\mathbf{U},h}}{\partial \mathbf{U}^{k,n}_h}\right)^T \mathbf{\Psi}^{k,m}_{\mathbf{U},h} + \left(\frac{\partial \overline{\mathbf{R}}^{k+1,m}_{\mathbf{U},h}}{\partial \mathbf{U}^{k,n}_h}\right)^T \mathbf{\Psi}^{k+1,m}_{\mathbf{U},h} = \left(\frac{\partial J_h}{\partial \mathbf{U}^{k,n}_h}\right)^T \tag{5.45}$$

$$\left(\frac{\partial \overline{\mathbf{R}}^{k,m}_{\mathbf{U},h}}{\partial G^{k,n}_h}\right)^T \mathbf{\Psi}^{k,m}_{\mathbf{U},h} + \left(\frac{\partial \overline{R}^{k,m}_{G,h}}{\partial G^{k,n}_h}\right)^T \Psi^{k,m}_{G,h} + \left(\frac{\partial \overline{R}^{k+1,m}_{G,h}}{\partial G^{k,n}_h}\right)^T \Psi^{k+1,m}_{G,h} = \left(\frac{\partial J_h}{\partial G^{k,n}_h}\right)^T \tag{5.46}$$

where k represents the time slab index, and m and n index the temporal nodes within each slab. These equations are just a more explicit form of Eqns. 5.42 and 5.43, and are marched backward in time to obtain $\mathbf{\Psi}_{\mathbf{U},h}$ and $\Psi_{G,h}$.

5.5.4 Adjoint Implementation

The adjoint equations above require several derivative terms, including residual Jacobians and output linearizations. In the current work, we perform all differentiation analytically, with the exception of $\partial \overline{\mathbf{R}}^{k,m}_{\mathbf{U},h}/\partial G^{k,n}_h$, which we evaluate using finite differences for ease of implementation.

In solving the GCL adjoint equation (Eqn. 5.46), the derivatives of $\overline{R}_{G,h}$ with respect to G are straightforward to obtain, as the result is just a mass matrix. Obtaining the derivative of the output J with respect to $G_h^{k,n}$ is done by simply applying the chain rule on derivatives with respect to the physical state, which are already used in the state adjoint equation. In addition, when solving the adjoint equations, we use the entire time history of the primal state and GCL, which we store to disk during the primal solve. While for the present work this storage has not been prohibitive, for larger problems, solution checkpointing [45] or local-in-time adjoint solvers [102] may be worth considering.

5.5.5 Error Estimate Implementation

The error estimate in Eqn. 5.44 requires an evaluation of the fine-space unsteady residuals associated with the coarse solution, as well as the fine-space adjoints $\mathbf{\Psi}_{\mathbf{U},h}$ and $\Psi_{G,h}$. In this work, when computationally feasible, we solve the fine-space adjoint equations to machine precision to minimize additional sources of error in our estimates. However, for more complex problems (or when interested in reducing CPU time), we smooth or reconstruct the coarse-space adjoints in order to minimize computational cost [34].

When Galerkin orthogonality holds, coarse-space approximations of the adjoints can be subtracted from the fine space adjoints appearing in Eqn. 5.44. (See e.g. Chapter II, Eqn. 2.104.) Theoretically this has no effect on δJ_{est}, but in practice it minimizes errors due to converging residuals only to a finite tolerance. We note however that care must be taken when using the BR2 viscous discretization, which does not exhibit Galerkin orthogonality for coarse-space solutions injected into an order-enriched fine space. This is due to an order-dependence of the BR2 stabilization terms. We employ a simple remedy [103], which consists of using the coarse-space orders to approximate the stabilization terms when evaluating the fine-space residuals. In addition, as quadrature effects tend to be more pronounced for simulations on deformable domains, we use the coarse-space quadrature rules in these fine-space residuals.

5.5.6 Error Localization

Since our aim is to adapt the mesh to reduce the output error, a global error estimate is not enough – we need to localize the error contributions to individual space-time elements in the mesh. As suggested in Chapter III, this can done by noting that the output error estimate in Eqn. 5.44 can be written as a sum over all space-time elements,

$$\delta J_{\mathrm{est}} = \sum_k \sum_e \varepsilon_{e,k}, \tag{5.47}$$

where the error contribution of a given space-time element (e, k) is

$$\varepsilon_{e,k} = \left(-\boldsymbol{\Psi}^m_{\mathbf{U},h}\right)^T \overline{\mathbf{R}}^m_{\mathbf{U},h}\left(\mathbf{U}^H_h, G^H_h\right)\bigg|_{e,k} + \left(-\Psi^m_{G,h}\right)^T \overline{R}^m_{G,h}(G^H_h)\bigg|_{e,k}. \tag{5.48}$$

This is just the adjoint-residual product restricted to the element (e, k), with a sum taken over the intra-slab temporal degrees of freedom m (implied by the repeated index above). The error indicator is then taken as the absolute value of this elemental contribution to the output error,

$$\text{error indicator} = \varepsilon_{e,k} = \left|\varepsilon_{e,k}\right|.$$

This indicator identifies the space-time elements that contribute most to the output error.

5.5.7 Space-Time Anisotropy

For adaptation purposes, however, we require still more information – specifically, is the error on a given space-time element due primarily to the *spatial* or *temporal* discretization?

This information is obtained from a space-time anisotropy measure, which we calculate in the same manner as presented in [34, 35]. Specifically, we calculate the error anisotropy using separate projections of the fine-space adjoints onto semi-coarsened spatial and temporal spaces.

The idea is that the amount of spatial error on a given element can be revealed by keeping the temporal grid fixed and refining the mesh solely in space. Likewise, the temporal error can be uncovered by keeping the spatial mesh fixed and refining only in time. The issue, however, is that we have already computed the adjoints on a mesh refined in both space *and* time, which leads to these errors becoming "mixed."

In order to "unmix" the spatial and temporal errors, we take our fine-space adjoint $\boldsymbol{\Psi}_{\mathbf{U},h}$ (likewise for $\Psi_{G,h}$) and perform least-squares projections onto semi-coarsened meshes:

$$\boldsymbol{\Psi}_{\mathbf{U},Hh} = \Pi_H^{\text{space}} \, \boldsymbol{\Psi}_{\mathbf{U},h} \tag{5.49}$$

$$\boldsymbol{\Psi}_{\mathbf{U},hH} = \Pi_H^{\text{time}} \, \boldsymbol{\Psi}_{\mathbf{U},h} \, . \tag{5.50}$$

Here, Π_H^{space} and Π_H^{time} represent least-squares projections to spatial order p_e and temporal order r, respectively. $\boldsymbol{\Psi}_{\mathbf{U},Hh}$ then contains only fine *temporal* information, while $\boldsymbol{\Psi}_{\mathbf{U},hH}$ contains only fine *spatial* information. We then project these semi-coarsened adjoints back to the fine-space mesh to make them dimensionally consistent with the fine-space residuals, resulting in $\boldsymbol{\Psi}_{\mathbf{U},h}^{Hh} \in \mathbb{R}^{N_h}$ and $\boldsymbol{\Psi}_{\mathbf{U},h}^{hH} \in \mathbb{R}^{N_h}$.

Estimates for the amount of spatial and temporal error on an element (e, k) are then given by

$$\varepsilon_{e,k}^{\text{space}} = \left(-\boldsymbol{\Psi}_{\mathbf{U},h}^{hH,m}\right)^T \overline{\mathbf{R}}_{\mathbf{U},h}^m \left(\mathbf{U}_h^H, G_h^H\right)\bigg|_{e,k} + \left(-\Psi_{G,h}^{hH,m}\right)^T \overline{R}_{G,h}^m \left(G_h^H\right)\bigg|_{e,k} \tag{5.51}$$

$$\varepsilon_{e,k}^{\text{time}} = \left(-\boldsymbol{\Psi}_{\mathbf{U},h}^{Hh,m}\right)^T \overline{\mathbf{R}}_{\mathbf{U},h}^m \left(\mathbf{U}_h^H, G_h^H\right)\bigg|_{e,k} + \left(-\Psi_{G,h}^{Hh,m}\right)^T \overline{R}_{G,h}^m \left(G_h^H\right)\bigg|_{e,k} \tag{5.52}$$

Finally, we use the ratio of these values to estimate the fractions (β) of spatial and temporal error on element (e, k) as

$$\beta_{e,k}^{\text{space}} = \frac{|\varepsilon_{e,k}^{\text{space}}|}{|\varepsilon_{e,k}^{\text{space}}| + |\varepsilon_{e,k}^{\text{time}}|}, \qquad \beta_{e,k}^{\text{time}} = 1 - \beta_{e,k}^{\text{space}}. \tag{5.53}$$

5.6 Spatial and Temporal Adaptation

With the above estimates of spatial and temporal error, we have the fundamental information needed for adaptation. The next step is to decide *what* to adapt. In this work, to address spatial errors, we will adapt the spatial order p on every element dynamically in time (see Fig. 5.6 for a schematic). To address temporal errors, we will adapt the temporal grid by selectively reducing or increasing the time slab widths (while potentially adding or removing time slabs as necessary).

The "objects" to be adapted then are both (1) individual space-time elements and (2) time slabs. Adaptive indicators identifying the spatial error on each space-time

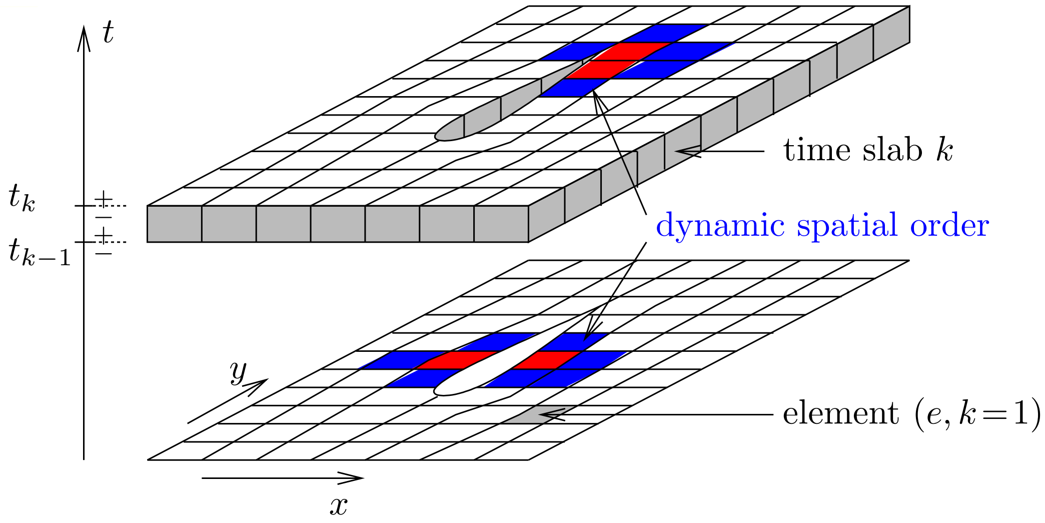


Figure 5.6: An illustration of dynamic-p refinement, in which spatial interpolation orders change in time to track relevant flow features.

element and the temporal error on each time slab are given by

$$\begin{aligned} \text{spatial indicator on space-time element } e, k = \quad & \varepsilon_{e,k}^{\text{space}} && = \varepsilon_{e,k}\beta_{e,k}^{\text{space}}, && (5.54) \\ \text{temporal indicator on time slab } k = \quad & \varepsilon_{k}^{\text{time}} && = \sum_{e} \varepsilon_{e,k}\beta_{e,k}^{\text{time}}, && (5.55) \end{aligned}$$

where the sum indexed by e is taken over all spatial elements on a given time slab. With these indicators, we can then conceivably lump all time slabs and space-time elements into the same "bin," rank them according to highest and lowest error indicators, and determine which to adapt based on their relative positions in that ranking.

However, from a computational perspective, this is not quite what we want to do. The issue is that, if (for example) a time slab and a space-time element had the same error, they would be equal candidates for refinement. But refining an entire time slab generally results in a much larger increase in degrees of freedom than refining a single element. Thus, rather than adapting directly on errors, we adapt on a slightly different **figure of merit** – the amount of error on a given element or slab (Eqns. 5.54 and 5.55) ***divided by*** the additional degrees of freedom associated with adapting that element or slab. This figure of merit then ensures that we eliminate the most output error for the least additional cost.

The above figure of merit is used in a fixed-growth adaptive strategy in which some combination of time slabs and space-time elements is marked for coarsening or refinement. The user specifies the fraction of degrees of freedom to be coarsened (f^{coarsen}), as well as the growth factor f^{growth}, which dictates the total number of degrees of freedom in the next mesh as $D^{\text{next}} = f^{\text{growth}}\, D^{\text{current}}$ (where D^{current} is the number of current degrees of freedom). The coarsening and refinement budgets are then

$$\begin{aligned} B^{\text{coarsen}} &= f^{\text{coarsen}}\, D^{\text{current}}, \\ B^{\text{refine}} &= (f^{\text{growth}} - 1)\, D^{\text{current}} + B^{\text{coarsen}}. \end{aligned}$$

In practice, we typically use a coarsening fraction of $\sim 5\%$ ($f^{\text{coarsen}} = 0.05$) and a growth factor of 1.30-1.35. With these budgets defined, the following algorithm is then used to decide which space-time elements or time slabs to adapt:

1. **Sort**

   Sort all space-time elements and time slabs based on the figure of merit. As mentioned, the figure of merit is the amount of output error addressed, Eqns. 5.54 and 5.55, divided by the degrees of freedom added if the element/slab were to be refined. For temporal refinement, the latter is approximated as the degrees of freedom in the targeted slab k, $\text{dof}_k \equiv \sum_e \text{dof}(p_{e,k})$, and for spatial refinement as the number of additional degrees of freedom, $\text{dof}(p_{e,k} + 1) - \text{dof}(p_{e,k})$, associated with an order increase of element e, k.

2. **Coarsen**

   (a) Set coarsening degree-of-freedom tally to zero.

   (b) Choose an unmarked space-time element or time slab with the lowest figure of merit.

   (c) If a time slab was chosen, mark it for a factor of 2 coarsening and add $0.5\, \text{dof}_k$ to the coarsening tally.

   (d) If a space-time element was chosen, mark it for an order decrement and add $\text{dof}(p_{e,k}) - \text{dof}(p_{e,k} - 1)$ to the coarsening tally.

   (e) If the tally meets or exceeds the coarsening budget, B^{coarsen}, stop. Otherwise return to step 2b.

3. **Refine**

   (a) Set refinement degree-of-freedom tally to zero.

(b) Choose an unmarked space-time element or time slab with the highest figure of merit.

(c) If a time slab was chosen, mark it for a factor of 0.5 refinement and add dof_k to the refinement tally.

(d) If a space-time element was chosen, mark it for an order increment and add $\mathrm{dof}(p_{e,k}+1) - \mathrm{dof}(p_{e,k})$ to the refinement tally.

(e) If the refinement budget, B^{refine}, is met or exceeded, stop. Else, return to step 3b.

When we say "factor of 2 coarsening" (in 2c) and "factor of 0.5 refinement" (in 3c), this essentially means that the width of a slab marked for coarsening will be doubled, while the width of a slab marked for refinement will be halved. In fact, if *only* refinement occurs, this is exactly the case – each marked slab will be perfectly bisected. However, when coarsening also occurs, the boundaries of a coarsened slab will typically encroach on those of the neighboring slabs, and the entire temporal grid must be shuffled to make room for the coarsened slab. To perform this "shuffling," time slabs are redistributed using one-dimensional metric-based meshing, the details of which are given below (with an additional schematic provided in Fig. 5.7).

1. For each time slab k, define $\widetilde{\Delta t}_k^{\text{desired}} = c\Delta t_k^{\text{current}}$ where c is 0.5 for refinement, 2 for coarsening, and 1 if the time slab is not marked. $\widetilde{\Delta t}_k^{\text{desired}}$ represents the new time step size that we desire on the current time slab k. The given choices of c are consistent with the choices made for the degree-of-freedom counts above.

2. Define $N^{\text{desired}} = \lceil \sum_k (\Delta t_k^{\text{current}} / \widetilde{\Delta t}_k^{\text{desired}}) \rceil$, where $\lceil\ \rceil$ is the greatest integer (ceiling) function. This will be the total number of time slabs on the new temporal mesh. Note that the ceiling function ensures that this number is an integer.

3. To ensure that the desired time step size is consistent with this total number of time steps, define the new desired time step size as

$$\Delta t_k^{\text{desired}} = \widetilde{\Delta t}_k^{\text{desired}} \frac{\sum_k (\Delta t_k^{\text{current}} / \widetilde{\Delta t}_k^{\text{desired}})}{N^{\text{desired}}}.$$

4. Finally, define a function $n(t)$ that is piecewise-constant over the current time slabs, and that takes on the value $1/\Delta t_k^{\text{desired}}$ on each current slab k. Define the new time slab breakpoints as times t_l where $\int_0^{t_l} n(t)dt$ is an integer.

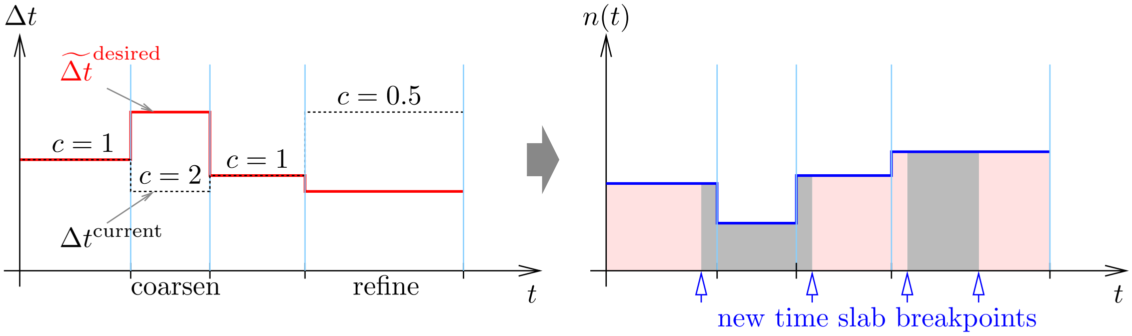

Δt
$\widetilde{\Delta t}^{\text{desired}}$
$c = 1$
$c = 2$
$c = 1$
$c = 0.5$
$\Delta t^{\text{current}}$
coarsen
refine
t
$n(t)$
t
new time slab breakpoints

Figure 5.7: Demonstration of the one-dimensional remeshing algorithm used to define new time slab breakpoints.

5.6.1 Summary of Adaptation Procedure

The above procedure describes how output-based adaptation of the current space-time mesh is performed. For a given number of degrees of freedom, this adaptation represents our best effort at eliminating errors in the output of interest. During the overall adaptive process, several of these adaptations are performed, and the unsteady problem is solved on successively refined space-time meshes. After each primal solve, the adjoint equations are marched backward in time, new error indicators are obtained, and a new adapted mesh is generated. This process is repeated until the output error drops below a specified tolerance.

Fig. 3.2, repeated below, provides a visual summary of this process. A summary of the steps in the algorithm, which parallels Sec. 3.2.2.3, is also provided below.

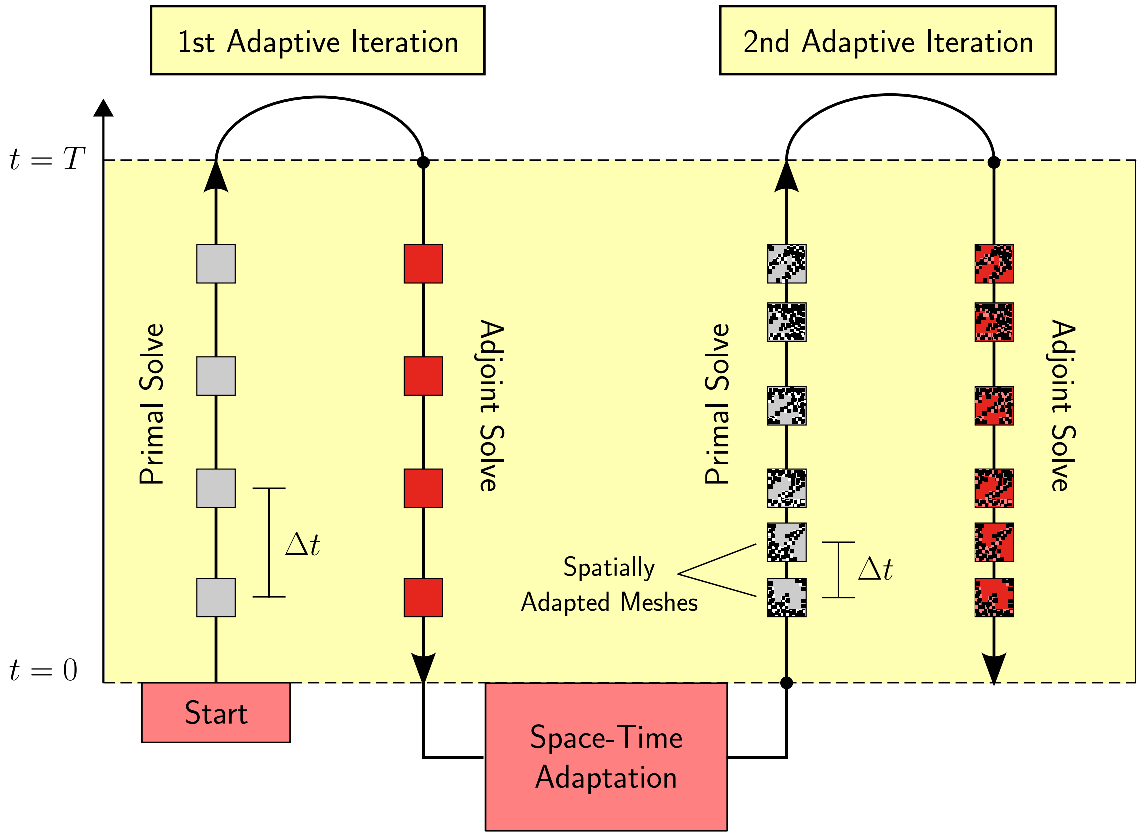


Figure 5.8: For each adaptive iteration, the primal problem is marched forward in time, and the adjoint problem is marched backward in time (on a uniformly refined space-time mesh). An output error estimate is then computed and mesh adaptation in space and time is performed. The spatial mesh is adapted differently at each time-step (as indicated by the variable shading in the 2nd iteration above), and time steps are selectively refined or coarsened.

Procedure:

1. Solve $\overline{R}_{G,H}(G_H) = 0$ and $\overline{\mathbf{R}}_{\mathbf{U},H}(\mathbf{U}_H, G_H) = 0$ on the coarse space to obtain G_H and $\mathbf{U}_H$. On a given time slab, G_H must be computed first.

2. Evaluate the output of interest, $J(\mathbf{U}_H, G_H)$.

3. Inject $\mathbf{U}_H$ and G_H to a uniformly order-incremented space-time mesh, $\mathcal{V}_h$. (Compute $\mathbf{U}_h^H$ and G_h^H.)

4. Evaluate the space-time residuals on $\mathcal{V}_h$ with the injected states. (Compute $\overline{\mathbf{R}}_{\mathbf{U},h}(\mathbf{U}_h^H, G_h^H)$ and $\overline{R}_{G,h}(G_h^H)$.)

5. Solve (or approximate) the fine-space adjoint equations

$$\left.\frac{\partial \overline{\mathbf{R}}_h}{\partial \mathbf{U}_h}^T\right|_{\mathbf{U}_h^H, G_h^H} \boldsymbol{\Psi}_{\mathbf{U},h} = \left.\frac{\partial J_h}{\partial \mathbf{U}_h}^T\right|_{\mathbf{U}_h^H, G_h^H}$$

$$\left.\frac{\partial \overline{R}_{G,h}}{\partial G_h}^T\right|_{\mathbf{U}_h^H, G_h^H} \Psi_{G,h} = \left.\frac{\partial J_h}{\partial G_h}^T\right|_{\mathbf{U}_h^H, G_h^H} - \left.\frac{\partial \overline{\mathbf{R}}_{\mathbf{U},h}}{\partial G_h}^T\right|_{\mathbf{U}_h^H, G_h^H} \boldsymbol{\Psi}_{\mathbf{U},h} \qquad (5.56)$$

   for $\boldsymbol{\Psi}_{\mathbf{U},h}$ and $\Psi_{G,h}$ by marching backward in time. On a given time slab, $\boldsymbol{\Psi}_{\mathbf{U},h}$ must be computed first.

6. Compute the error estimate $\delta J_{\text{est}} \approx -\boldsymbol{\Psi}_{\mathbf{U},h}^T \overline{\mathbf{R}}_{\mathbf{U},h}(\mathbf{U}_h^H, G_h^H) - \Psi_{G,h}^T \overline{R}_{G,h}(G_h^H)$.

7. Correct the original $J(\mathbf{U}_H, G_H)$ with this error estimate. (Compute $J_{\text{corrected}} = J + \delta J_{\text{est}}$.) This corrected output is more accurate than the original J.

8. Localize δJ_{est} to individual space-time elements in the mesh.

9. Compute space-time anisotropy fractions $\beta_{e,k}^{\text{space}}$ and $\beta_{e,k}^{\text{time}}$, which indicate how much of δJ_{est} on each space-time element is due to the spatial vs. temporal discretization.

10. Compute the "figure of merit" representing the amount of spatial or temporal error divided by the cost of refining the element or time slab.

11. Select a certain percentage of elements or time steps with the highest figure of merit and refine (and/or coarsen) them.

12. Solve the primal problem on the new mesh and repeat steps 2-12 until the output error is driven below a desired tolerance.

5.6.2 Alternative Adaptive Methods

In the results section below, we compare our output-based adaptation procedure to three alternative strategies:

1. **Uniform** h-refinement in space combined with uniform slab bisection in time.
2. **Uniform** p-refinement in space combined with uniform slab bisection in time.
3. **Residual-based** dynamic-p adaptation in space combined with slab coarsening/refinement.

The residual indicator is given by a form similar to the output error (Eqn. 5.48), but without the adjoint and with absolute values on the individual residual components:

$$\varepsilon^{\text{res}}_{e,k} = \sum_m \left|\bar{\mathbf{R}}^m_{\mathbf{U},h}(\mathbf{U}^H_h)\right| \Bigg|_{(e,\,k)} .$$

This indicator targets areas of the space-time domain where the governing equations[3] are not well-satisfied (i.e. where truncation errors are large), and is commonly used in practice as an adaptive metric. A space-time anisotropy measure based on the size of the solution discontinuities between elements [37] is used to determine $\beta^{\text{space}}_{e,k}$ and $\beta^{\text{time}}_{e,k}$ for this metric. Otherwise, the same dynamic-p and temporal adaptation algorithms are used as for the output-based method, providing a fair comparison of the strategies.

5.7 Results

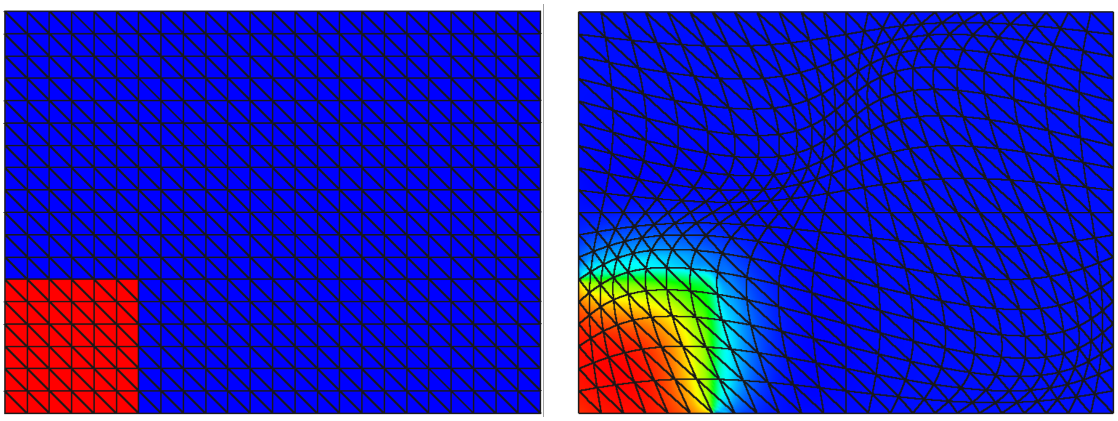
In this section, we present the results for several test cases. The first is a verification of the adjoint and error estimation procedures described above. The following cases then employ this error indicator to drive unsteady mesh adaptation for problems of engineering interest. These problems consist of airfoils and wings pitching and plunging at low Reynolds number, and the convergence of the lift on these bodies is compared for the adaptive methods described above.

5.7.1 Error Estimate Verification

Here, we verify the proposed error estimation strategy with a simple Navier-Stokes problem. The problem consists of a density perturbation in a stagnant fluid, situated in the corner of a rectangular basin bounded by no-slip walls (Fig. 5.9). The density perturbation is allowed to diffuse for two time units, and the final vertical force on the left wall is taken as our output of interest. The force is kept dimensional while using the following convenient units: density = 1, density perturbation = 0.25, total energy

[3] Note that we have included only the state residual rather than the GCL residual in this indicator. We do not expect this to have a large influence on the results. Since the dimensions of the equations in the residual are different anyway, there is always an *ad-hoc* choice involved in deciding how to sum its components. Furthermore, the poor performance seen later with this indicator is not due to the absence of the GCL residual, since the GCL errors are too small to be generating the observed effects.

= 2.675, gas constant = 1, laminar viscosity = 1. One time unit then corresponds to $t = 1$ in the physical evolution of the equations.



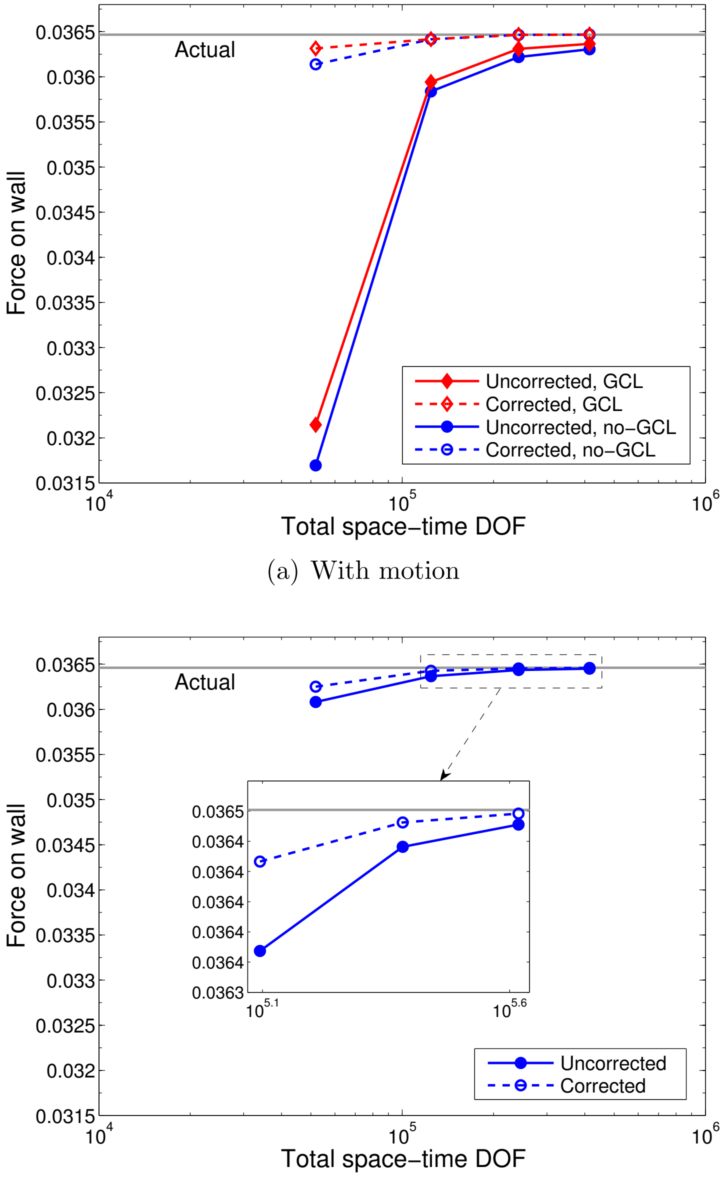
(a) Density, $t = 0$

(b) Density, $t = 2$

Figure 5.9: Initial and final meshes and densities. The initial density perturbation is 25% above the nominal value.

Since no boundaries are moving here, mesh motion is not required and the problem could be solved with a fixed grid. However, to test the error estimation with mesh motion, we instead choose to wave the mesh in the background, using the formula in Eqn. 5.3 to map the domain from reference to physical space. Since this mapping is relatively violent, with large variations in g in both space and time, it should introduce motion-related errors and provide a good test of our error estimation procedure. Note that the movement of the mesh should in theory have no impact on the physics of the problem, and the solution should converge upon refinement to that with no mesh motion.

To verify the error estimation, three separate cases were run: (i) a no-motion case; (ii) a case with motion but no GCL; and (iii) a case with both motion and GCL. These cases were run at interpolation orders ranging from $p = 1$ to 4, with a DG1 time scheme and 10, 12, 14, and 16 time steps, respectively. For each run, the output J_H on the current mesh is first recorded. Next, error estimation based on a $(p + 1, r + 1)$ fine space is performed, and the predicted change in the output relative to the fine space (δJ_{est}) is computed. Finally, the primal problem is solved directly on the $(p + 1, r + 1)$ space, and the output J_h is determined. The difference $\delta J_{\text{act}} = J_h - J_H$ between coarse and fine outputs can then be obtained and compared to δJ_{est}. If the error estimation is working properly, these values should correspond closely. Of course, since the problem is nonlinear, the correspondence will generally not be exact.



(a) With motion

(b) Without motion

Figure 5.10: Output convergence for both motion and no-motion cases, shown with the same scale. Despite larger-magnitude errors, the corrected output converges rapidly to the true value for motion cases.

Fig. 5.10 shows both the output and corrected output ($J + \delta J_{\mathrm{est}}$) convergence for all cases. Though the errors for the motion cases are significantly larger than those without motion, the error estimate brings the corrected output very close to the true value. Table 5.2 gives a more quantitative comparison between the actual and predicted errors. From the table, we see that the error estimates with mesh motion display 93-99% accuracy relative to the actual output errors. The no-motion error estimates, despite being relatively less accurate initially, also achieve greater than 95% accuracy as the mesh is refined. The reason for the initial inaccuracy is not completely known, but the smaller magnitude of the no-motion errors means low-level noise due to (e.g.) numerical quadrature has a larger effect on the percent accuracy.

For the GCL runs specifically, the above results verify that the GCL adjoint and error estimation procedures were implemented correctly. The breakdown of errors due to the state and GCL individually are given in Table 5.3 (which includes an additional $p = 0$ run), and we see that the GCL-related errors constitute from 8-42% of the total error estimate.

	GCL			**No GCL**			**No Motion**		
p	δJ_{est}	δJ_{act}	% Error	δJ_{est}	δJ_{act}	% Error	δJ_{est}	δJ_{act}	% Error
1	4.17e-3	4.02e-3	**3.7**	4.44e-3	4.49e-3	**1.0**	1.69e-4	2.88e-4	**41.4**
2	4.75e-4	5.08e-4	**6.6**	5.78e-4	6.12e-4	**5.5**	5.92e-5	7.03e-5	**15.7**
3	1.55e-4	1.54e-4	**0.5**	2.45e-4	2.44e-4	**0.4**	1.61e-5	1.55e-5	**3.5**
4	1.02e-4	1.01e-4	**1.4**	1.63e-4	1.61e-4	**1.1**	7.31e-6	7.02e-6	**4.2**

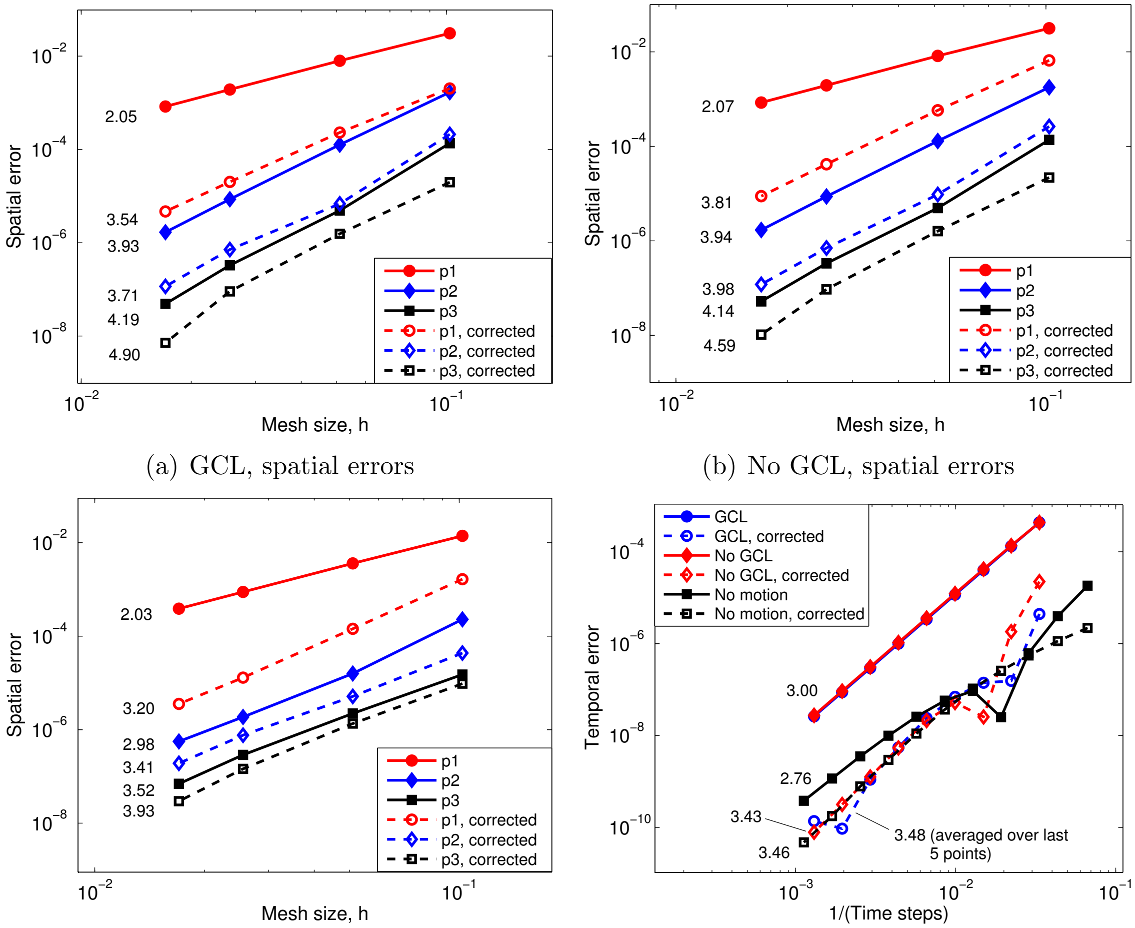
Table 5.2: Relative accuracy of error estimates for motion and no-motion cases at different orders p. "% Error" denotes the error in δJ_{est} relative to δJ_{act}.

p	0	1	2	3	$p = 4$
State Errors	-3.59e-2	4.61e-3	5.93e-4	2.53e-4	1.68e-4
GCL Errors	-2.63e-2	-4.38e-4	-1.18e-4	-9.74e-5	-6.53e-5
GCL % of Total	**42.3**	**8.9**	**16.6**	**27.8**	**28.0**

Table 5.3: Contribution of GCL and state errors to total error estimate.

While these results demonstrate the qualitative and quantitative accuracy of the error estimates, it is also worthwhile to look at formal convergence rates. To ensure that the singularities in the corners of the basin do not inhibit these rates, we change the problem slightly. The basin walls are replaced by periodic boundaries, and the perturbed region (which now consists of both density and velocity variations) is shifted so that it no longer touches the boundary. Since the walls have vanished, the output is taken to be the final-time integral of pressure over the domain.

We are interested in how the uncorrected and corrected outputs ($J + \delta J_{\text{est}}$) converge in both space and time. We first fix the number of time steps and perform error estimation in space only (relative to a $p + 1$ fine space), for various values of p and several $h-$refinements of the spatial mesh. From these refinements, we can plot the



(a) GCL, spatial errors

(b) No GCL, spatial errors

(c) No motion, spatial errors

(d) Temporal errors

Figure 5.11: Spatial and temporal convergence histories for the corrected and uncorrected final-time output. Average convergence rates are shown next to each curve, and expected rates are achieved for all runs. Note that different reference values are used for the GCL, no-GCL, and no-motion cases, so a direct comparison of error magnitudes should not be made.

spatial convergence rates for GCL, no-GCL, and no-motion runs (shown in Fig. 5.11). For nearly all runs, we see $p + 1$ convergence rates for the uncorrected outputs and the expected $p + 2$ rates for the corrected outputs. A rate of $p + 2$ for the corrected output is anticipated because, ideally, the error estimate should correct the coarse-space output to the fine-space value, and the fine space value itself converges at a rate of $p + 2$ in this case. We note that the $p = 2$ motion runs are somewhat anomalous, since they appear to converge at fourth, rather than third, order.[4]

[4] It is possible that the true convergence rates here are actually $2p$, and that the discontinuity in the initial condition is limiting the rates for the $p > 2$ runs.

Next, we fix the spatial grid and interpolation order (at $p = 2$), and perform both error estimation and refinement in time only, using a DG time scheme with order $r = 1$. The results of this temporal convergence study are shown in the final plot of Fig. 5.11. Since the output is computed at the final time, it lies on a downwind time node and is expected to super-converge [1] at a rate of $2r + 1 = 3$. This rate was obtained, and is evident in the plot. Error estimation for these runs is performed relative to an $r + 1$ space, and a convergence rate of $2(r + 1) = 4$ is expected for the corrected outputs. (Note that while the fine-space output itself would converge at 5th order, the error estimate is limited to 4th-order accuracy due to the linearization error, which is of order $\delta\mathbf{u}^2 = 2(r + 1)$. See Eqn. 3.27.) The actual corrected outputs achieve a rate of approximately 3.5 before numerical precision causes a bottoming out.

5.7.2 Dynamic Mesh Adaptation for Pitching Airfoil

With the performance of the error estimates confirmed, we next use them to drive mesh adaptation for a case of engineering interest – an airfoil pitching sinusoidally at low Reynolds number. The airfoil starts from an impulsive free-stream condition and undergoes three periods of pitching motion (with amplitude 30° and period $T = 2.5$) at a Strouhal number of 1.0, a Reynolds number of 400, and a free-stream Mach number of 0.2. The airfoil is a NACA 0012 situated in the center of a 60 x 60 chord-length domain, and the mesh consists of 1,454 triangular elements.

Entropy contours at several phases of the motion are shown in Fig. 5.15. A series of vortices forms behind the airfoil as it completes the motion cycle, and these vortices combine with the free-stream flow and inertial effects to generate forces on the airfoil. Our output of interest is the lift component of these forces integrated over the final 2.5% of the simulation time.

To compute this output, the three adaptive methods described in Sec. 5.6.2 (output-based, residual, and uniform refinement) are considered. Several adaptations are performed starting from an initial $p = 1$, 70 time step solution. The spatial order is constrained to lie within the range $0 \leq p \leq 5$, and an $r = 1$ DG time scheme is used for all runs. For the output error and residual methods, 5% of space-time elements are coarsened during each adaptive iteration, while the overall size of the space-time mesh is increased by a growth factor of 30%. All simulations are performed in parallel using 72 processors on the *Diamond* supercomputing cluster at the U.S. Army Engineering Research and Development Center (ERDC).

5.7.2.1 Results

The output convergence for each adaptive method as a function of total space-time degrees of freedom (DOF) is shown in Fig. 5.12. For the output-based method, both the output J and the corrected output $J + \delta J_{\text{est}}$ are given. As seen, the output-based adaptation converges much faster than uniform refinement, requiring approximately two orders of magnitude fewer degrees of freedom to converge to the true output value. It also significantly outperforms residual adaptation, which becomes distracted by acoustic waves emanating from the airfoil and fails to converge even after 14 adaptive iterations.

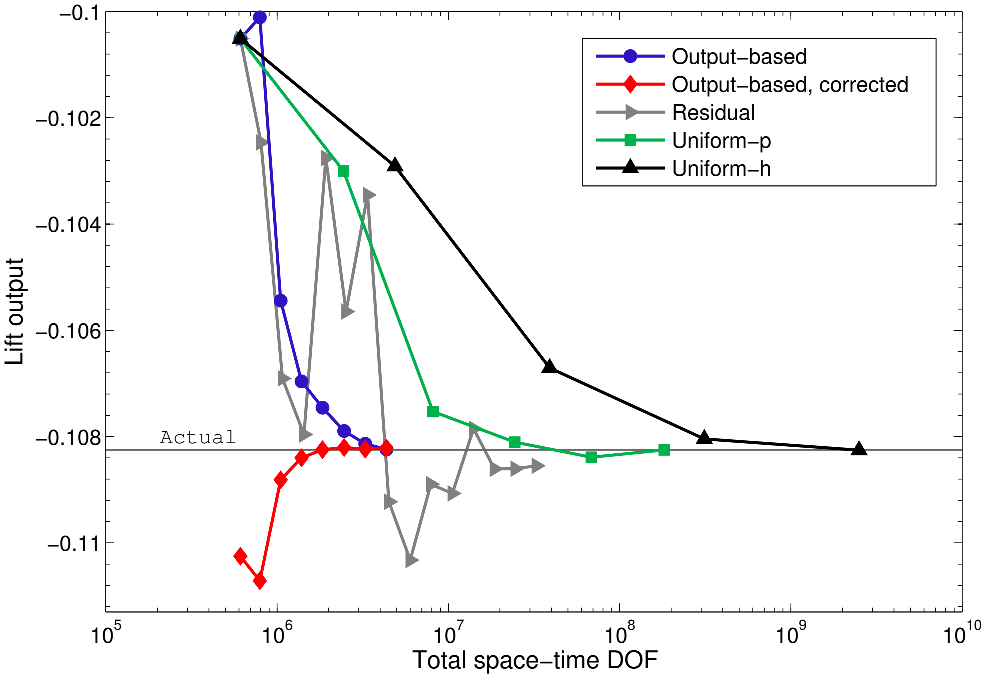


Figure 5.12: Single airfoil: Output convergence for the various adaptive methods. The output-based method outperforms both residual-based and uniform refinement by orders of magnitude. Note that the red and blue curves come from the same runs, but the red curve is the “corrected” output, $J + \delta J_{\text{est}}$, while the blue curve shows J.

The temporal and spatial grids from the final output-adapted run are shown in Fig.s 5.14 and 5.16, respectively. In Fig. 5.16, we see that the regions where vortex shedding occurs are heavily targeted, while a circular region surrounding the airfoil is refined to a lesser extent. In the far field, most elements have been coarsened to $p = 0$, as expected. Temporally, the periods of strongest vortex shedding are targeted, with a preference given to those occurring in the first half of the simulation. This preference for earlier times makes sense, since the initial vortices influence the vortex shedding during later parts of the simulation, and must therefore be resolved accurately. The residual indicator also targets this vortex shedding, but does so blindly, giving roughly equal weight to each vortex shed throughout the simulation (as evidenced by the equally-spaced refinement in Fig. 5.14).

To highlight one of the factors driving the adaptation, contours of the GCL adjoint are shown alongside the entropy contours in Fig. 5.15. The first contour shows a band of inward-moving sensitivity waves converging upon the trailing edge near time $t = T/3 = 0.833$. Similar behavior is seen in the other adjoint components, and this is reflected in the adaptation, which heavily targets the times following $t \approx 0.8$. The final adjoint contour in Fig. 5.15 shows the sensitivity field collapsing onto the airfoil near the end of the simulation. This collapse occurs because, as the simulation comes to an end, any errors made in regions farther from the airfoil no longer have time to propagate to the airfoil and influence the output. Thus, the adjoint goes to zero everywhere except in regions near the airfoil.

Finally, since the GCL adjoint and error estimates are of particular interest in this paper, a breakdown of the output error into its state and GCL components is provided in Table 5.4. From the table, we see that the GCL errors initially make up only a small percentage of the total error, but as the adaptations proceed, their contribution increases to over half of the total error estimate. While this alone does not prove that the GCL is essential to obtaining accurate outputs, it does imply that *if* the GCL is used, a corresponding GCL adjoint is necessary to obtain accurate error estimates.

5.7.3 CPU Time Comparison

Above, we solved the fine-space adjoint to machine precision to ensure accurate error estimates and efficient allocation of mesh degrees of freedom. In practice, CPU time is another important factor, and solving the adjoint to machine precision is generally not the most efficient strategy. Previous work [34] has shown that a few smoothing iterations on the fine-space can provide acceptable error estimates at reduced computational cost. In Fig. 5.21, we show a wall time comparison for the various adaptive strategies, with the adjoint smoothed to a residual tolerance of 1×10^{-4}. This tolerance is not necessarily optimal, and we note that the code itself has not been optimized for CPU time. However, our aim is simply to give an idea of the relative timings. While the performance gains for the output-based method are not as substantial in this context, it does converge faster than the uniform refinement strategies. It also significantly outperforms the residual-based adaptation, which fails

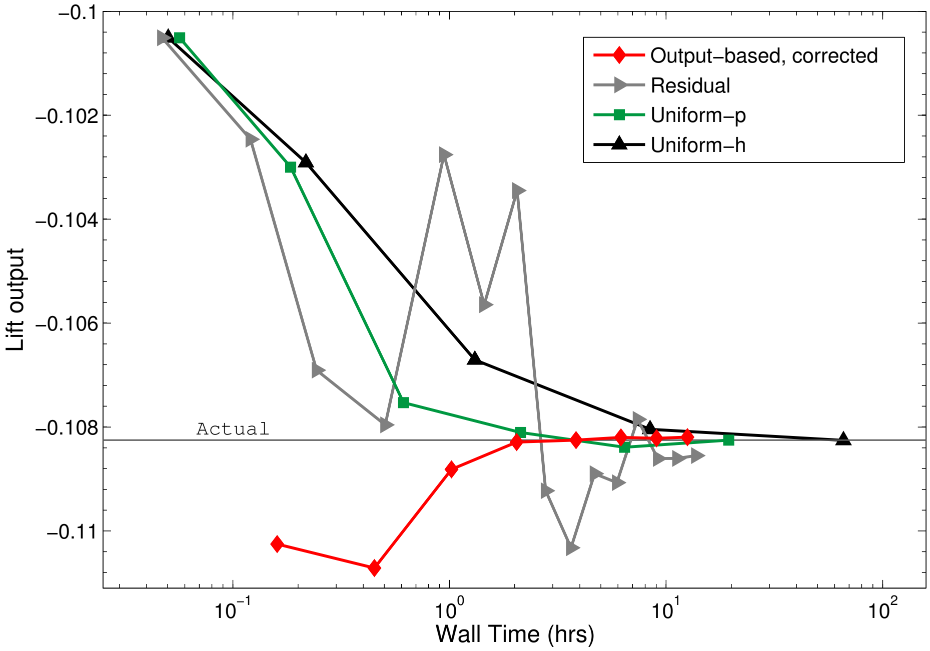


Figure 5.13: Single airfoil: Wall time comparison for the various adaptive methods. The output-based method outperforms the other strategies.

to converge by the time of the final adaptation.

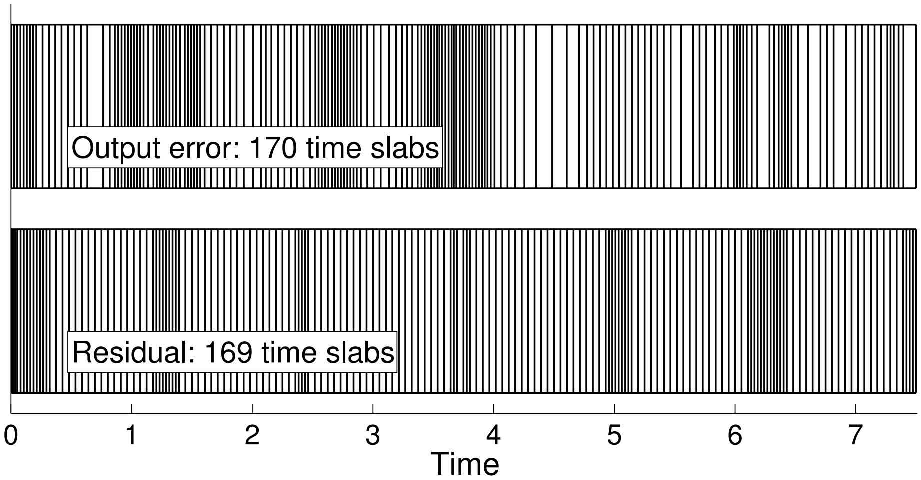
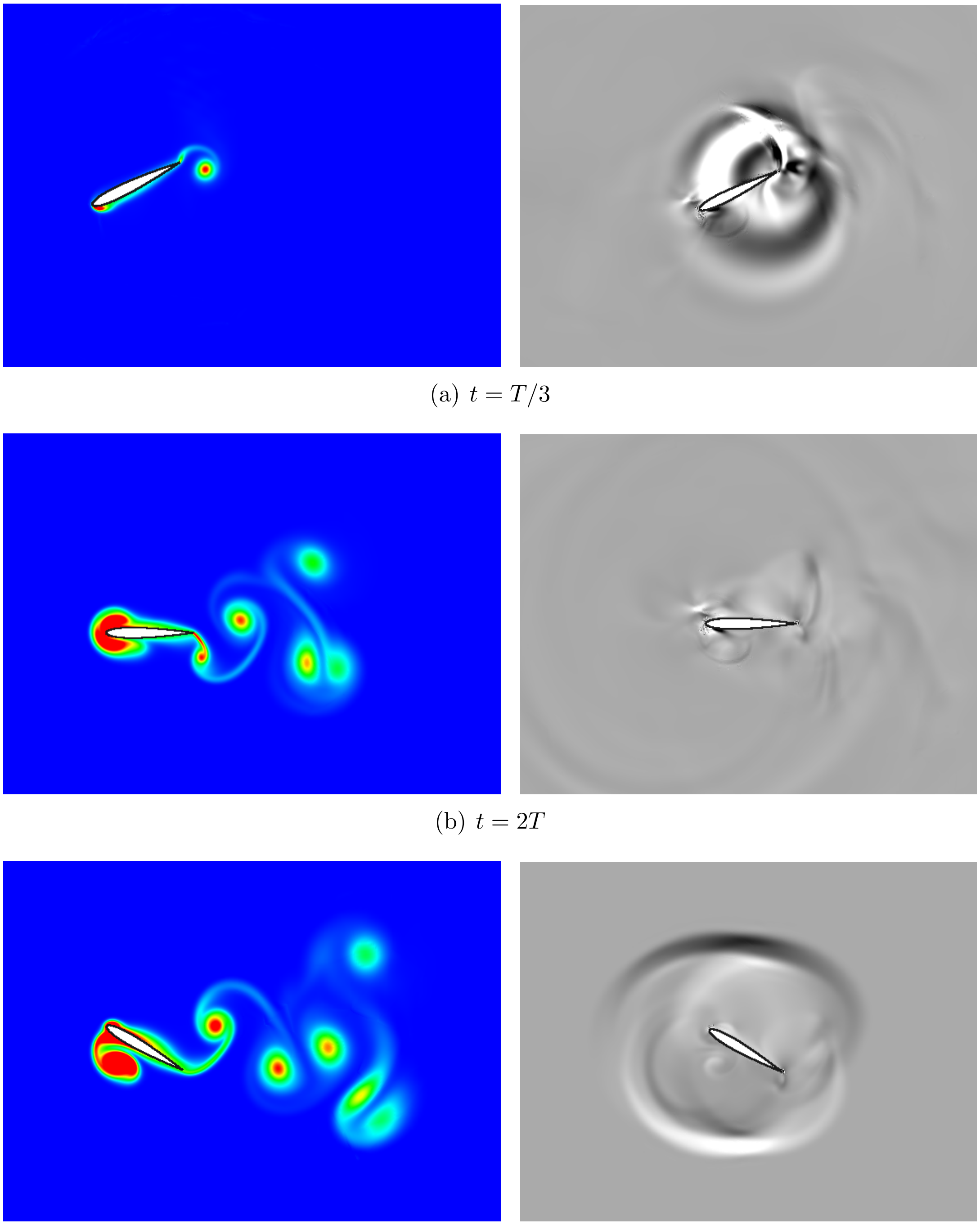


Figure 5.14: Single airfoil: Temporal grids from the seventh adaptation of both adjoint and residual runs.

Adapt Iter.	1	2	3	4	5	6	7	8
State Errors	-9.9e-3	-1.1e-2	-3.8e-3	-1.9e-3	-6.6e-4	-4.0e-4	-4.5e-4	-3.1e-4
GCL Errors	1.3e-4	2.7e-4	4.5e-4	4.9e-4	-1.3e-4	8.9e-5	3.5e-4	3.4e-4
GCL % of Total	**1.3**	**2.5**	**10.6**	**20.2**	**16.1**	**18.2**	**43.7**	**51.8**

Table 5.4: Single airfoil: Contribution of GCL and state errors to the total error estimate for all iterations of output-based adaptation.

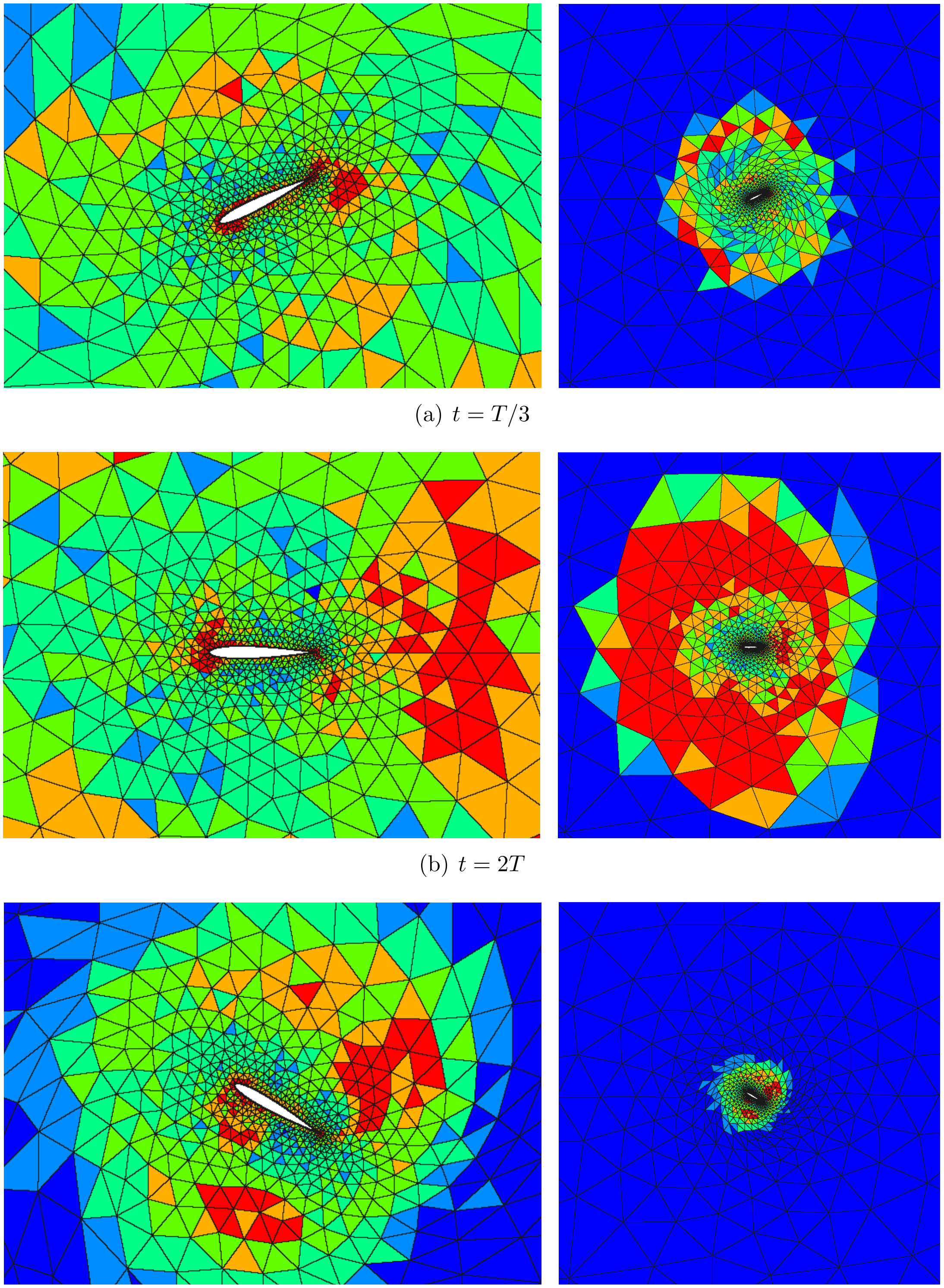


(a) $t = T/3$

(b) $t = 2T$

(c) $t = 11T/4$

Figure 5.15: Single airfoil: Entropy (left) and GCL adjoint (right) contours at various stages of the pitch motion on a fine mesh. Note that the GCL adjoint contours have been re-scaled to more clearly show the features. (Black is 1.5, white is -0.75.)



(a) $t = T/3$

(b) $t = 2T$

(c) $t = 11T/4$

Figure 5.16: Single airfoil: Output-adapted meshes at various stages of the pitch motion. Blue is $p = 0$, red is $p = 5$.

5.7.4 Dynamic Mesh Adaptation for Pitching and Plunging Airfoils

Next, we try a more complicated case – two airfoils pitching and plunging in series. The airfoils start from an impulsive free-stream condition and undergo three periods of motion. The plunge amplitude is 0.25 chords, the pitch amplitude is 30°, and the period of both motions is $T = 2.5$. The Strouhal, Mach, and Reynolds numbers are 2/3, 0.3, and 1200, respectively. The airfoils are offset 4.5 chords horizontally and 1 chord vertically, and are situated in a 60 x 60 chord-length mesh with $3,534$ triangular elements.

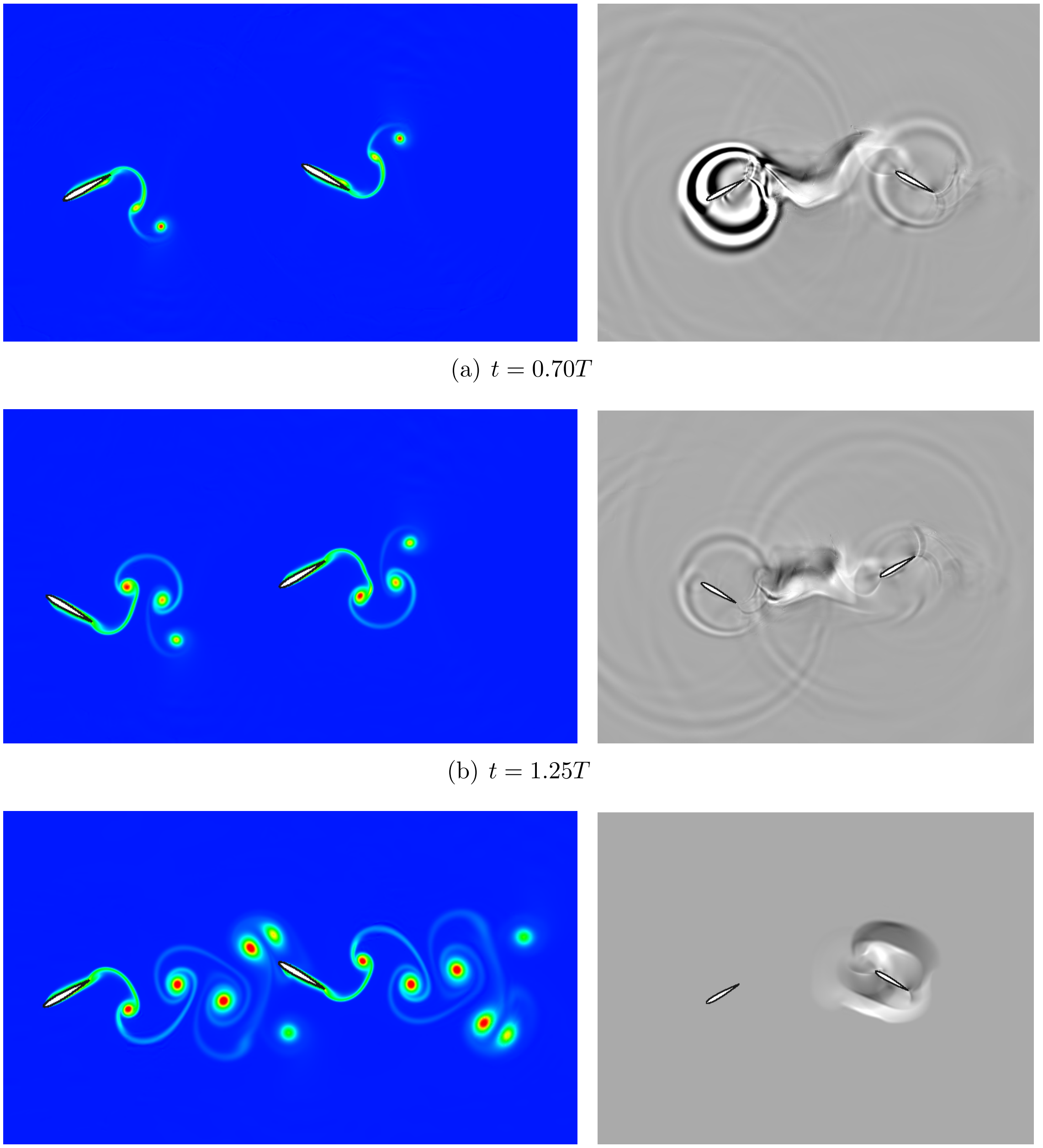
Entropy contours at various phases of the motion are shown in Fig. 5.17. A reverse Kármán vortex street develops behind each airfoil, and the second airfoil interacts with the wake from the first airfoil near the end of the simulation. Our output of interest is the lift on the second airfoil integrated from time $t = 7.25$ to $t = 7.5$ (the final time).

To compute this output, the adaptive methods described in Sec. 5.6 (output-based, residual, and uniform h- and p-refinement) are again considered. Adaptations are performed starting from an initial $p = 1$, 90 time step solution, with a 35% growth factor and 5% coarsening factor used for the output- and residual-based methods. The spatial order p is constrained to lie between 0 and 5, and an $r = 1$ DG scheme is used in time. The simulations are again performed in parallel using 72 processors on the *Diamond* supercomputing cluster at the U.S. Army ERDC.

5.7.4.1 Results

The output convergence for each adaptive method as a function of total space-time degrees of freedom is shown in Fig. 5.18. We see that the output-based adaptation converges significantly faster than uniform refinement, requiring roughly two orders of magnitude fewer degrees of freedom. These gains relative to uniform refinement are impressive, though not necessarily unexpected. Equally interesting is the difference between the output-based and residual adaptation.

The residual indicator targets regions of the domain where the governing equations are not well-satisfied, and hence usually performs well for static problems. However, in this case, its performance is erratic and no better than uniform refinement. As in the single-airfoil case, this erratic behavior is a consequence of the acoustic waves that emanate from the airfoils as they pitch back and forth. The residual indicator becomes distracted by these waves and exhausts degrees of freedom trying to resolve them as they propagate throughout the domain. The output-based method, on the



(a) $t = 0.70T$

(b) $t = 1.25T$

(c) $t = 2.75T$

Figure 5.17: Two airfoil case: Entropy (left) and GCL adjoint (right) contours at various stages of the motion on a fine mesh. The GCL adjoint contours have been re-scaled to more clearly show the features (black is 2, white is -1). Both acoustic and convective modes of error propagation can be seen in the first two contours, while at the final time, the adjoint field collapses on the second airfoil.

other hand, deems the majority of these waves irrelevant to the output and does not expend resources on them.

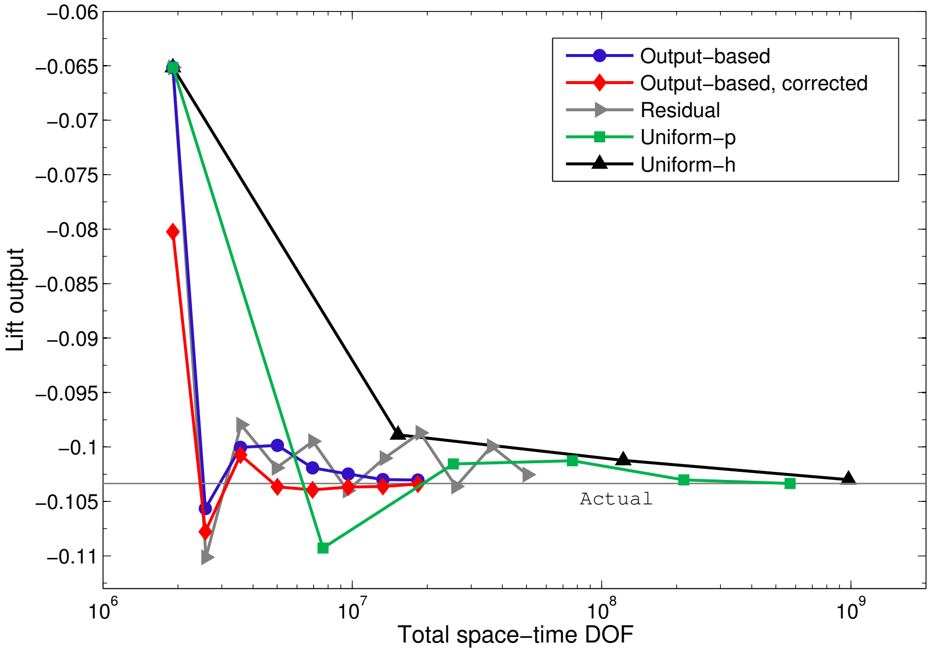
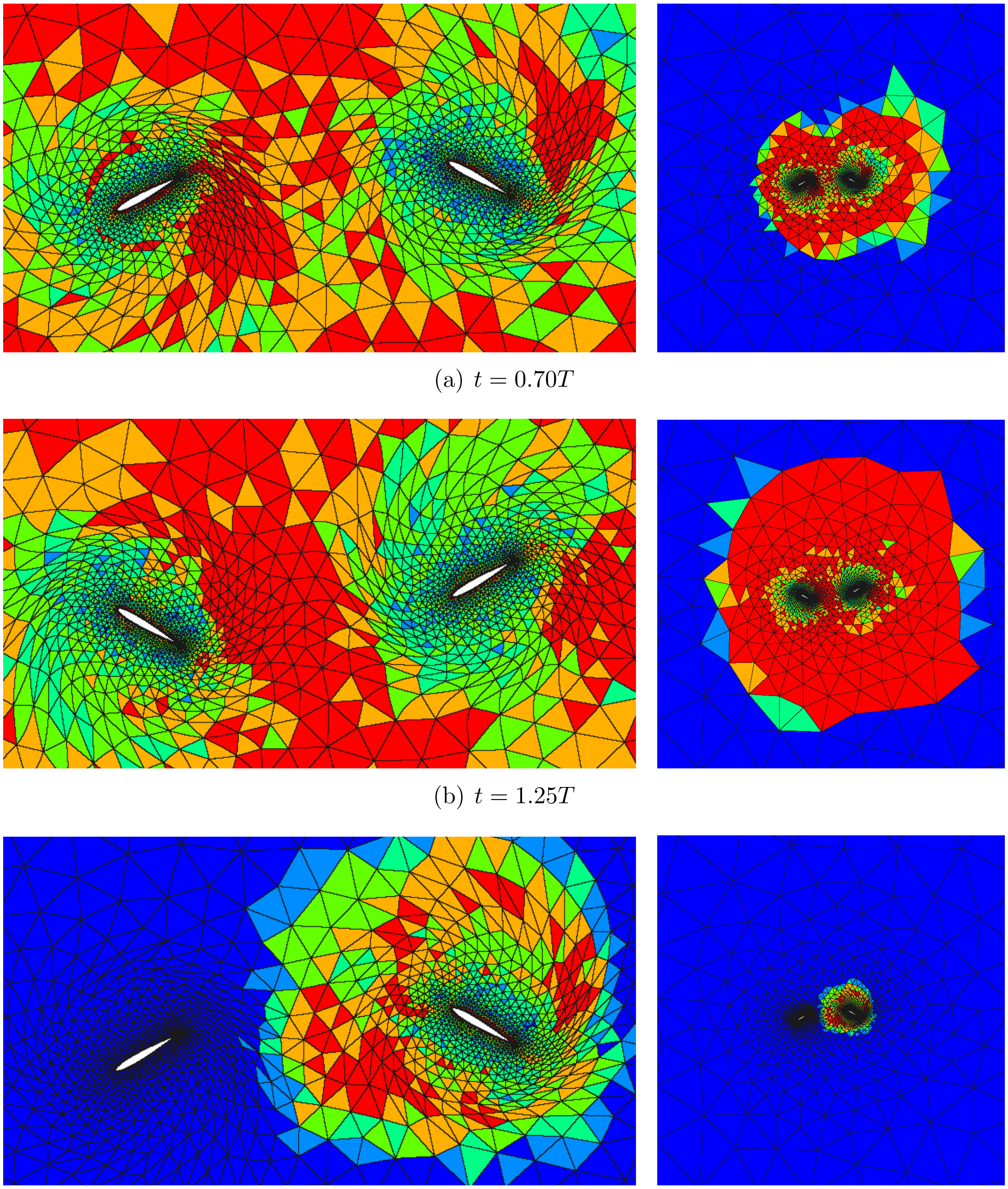


Figure 5.18: Two-airfoil case: Output convergence for various adaptive methods. The output-based method performs the best.

The spatial and temporal meshes from the final output-based adaptation are shown in Fig.s 5.19 and 5.20, respectively. We see that the near-airfoil and vortex shedding regions are targeted for adaptation, as well as the group of large elements surrounding the mesh motion regions. While somewhat difficult to observe in the still-frames, the initial vortex shed from the first airfoil is heavily targeted throughout the simulation, since this vortex later collides with the second airfoil near the final time.

Contours of the GCL adjoint (which are similar in nature to the state adjoint) are shown alongside the entropy contours in Fig. 5.17. The time $t = 0.70T$ is the instant before the initial vortex is shed, and the large sensitivity of the output to this event can be seen in the adjoint contours. As the simulation proceeds, the output sensitivity gradually shifts from the first airfoil to the second, before collapsing upon the second airfoil at the final time.

Some other aspects of the GCL adjoint are worth pointing out. In the first two contours, the near-circular rings represent inward-moving (adjoint) acoustic waves, which converge upon a particular region as the simulation proceeds. The existence of a ring implies that an important event in space-time is about to occur, and any errors made within the circumference of the ring have the ability to influence this event. In this simulation, the important events tend to be instances of vortex shedding, and the



(a) $t = 0.70T$

(b) $t = 1.25T$

(c) $t = 2.75T$

Figure 5.19: Two-airfoil case: Output-adapted meshes at various stages of the motion. Blue is $p = 0$, red is $p = 5$.

rings converge on the trailing edge regions. Lastly, between the two airfoils, a path can be seen tethering them together. This path appears because any errors within it ultimately reach the second airfoil via convection, and can therefore directly affect

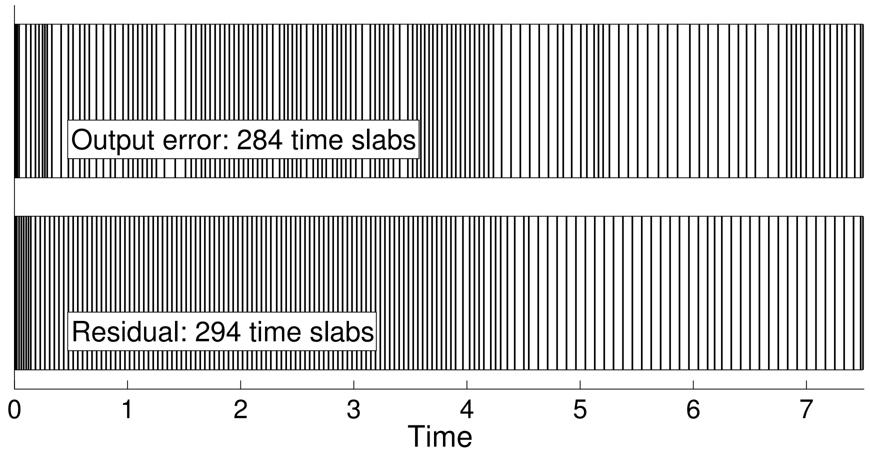


Figure 5.20: Two-airfoil case: Temporal grids from the seventh adaptation of both output-based and residual runs. For clarity, only every other time slab is plotted.

the output.

Finally, a breakdown of the output error into its state and GCL components is provided in Table 5.5. From the table, we see that the GCL errors initially make up only a small percentage of the total error, but as the adaptations proceed, their contribution accounts for a significant fraction of the total – again underscoring the importance of incorporating the GCL adjoint when the GCL is employed.

Adapt Iter.	1	2	3	4	5	6	7	8
State Errors	-1.5e-2	-2.2e-3	-7.4e-4	-3.8e-3	-2.4e-3	-1.0e-3	-3.6e-4	5.8e-4
GCL Errors	1.5e-4	9.3e-5	3.8e-5	3.9e-5	3.5e-4	-1.7e-4	-2.8e-4	-9.5e-4
GCL % of Total	**1.0**	**4.0**	**4.8**	**1.0**	**12.8**	**14.0**	**43.4**	**62.2**

Table 5.5: Two-airfoil case: Contribution of GCL and state errors to the total error estimate for all iterations of output-based adaptation.

5.7.4.2 CPU Time Comparison

For the above results, we solved the fine-space adjoint to machine precision. However, if we are interested in reducing CPU time, we can again smooth the adjoint to some finite residual tolerance. In Fig. 5.21, we show a wall time comparison for the various adaptive strategies, with the adjoint smoothed to a residual tolerance of 1×10^{-3}. We see that the output-based method converges roughly 5-10 times faster than the uniform refinement strategies. It also significantly outperforms the residual-based adaptation, which again fails to converge in any amount of time.

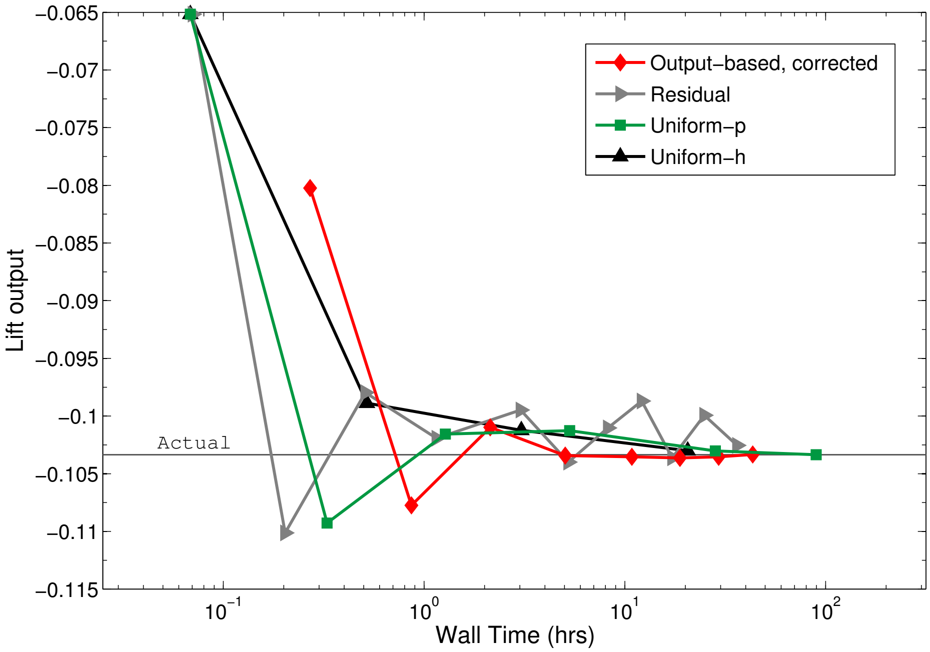


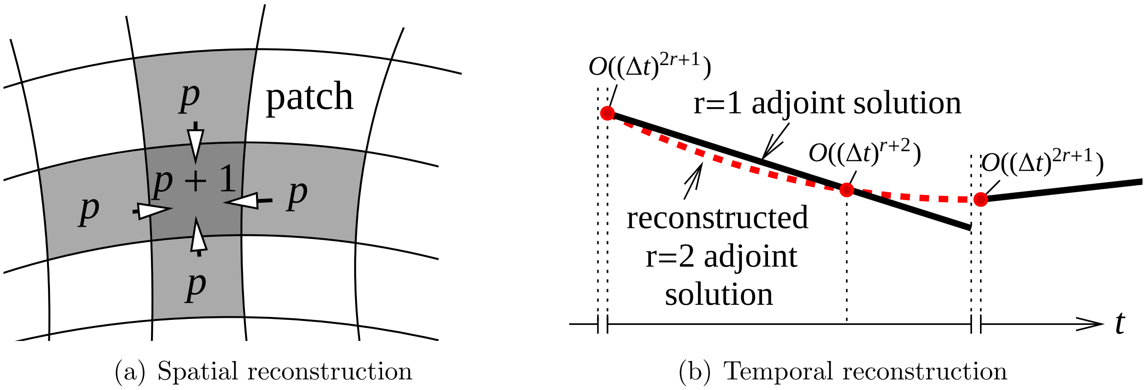
Figure 5.21: Two-airfoil case: Wall time comparison for the various adaptive methods. The output-based method converges the fastest.

5.7.5 Mesh Adaptation for a Three-Dimensional Flapping Wing

The above cases show the effectiveness of our output-based strategy in two dimensions. Next, we attempt to apply a similar strategy in three dimensions. In 3D, the underlying ideas for mesh motion and adaptation remain the same, but the increased algorithmic complexity and new dimensional scaling do not allow for a simple extrapolation of the 2D results. In particular, the extra dimension makes the scaling for the adjoint problem less favorable. For a $(p=1,\ r=1)$ primal solve in 2D, the dimension of the fine-space adjoint is approximately 3 times that of the primal, while in 3D this factor increases to over 5. When combined with additional CPU costs, computing (or even smoothing) the adjoint on the fine space becomes an expensive proposition.

To address this issue, we give up the idea of computing the fine-space adjoint directly. Instead, we compute the adjoint in the same space as the primal problem, and then reconstruct it in both space and time to obtain an approximation to the fine-space adjoint. This reconstruction is performed locally, with patches of neighboring elements used to generate a least-squares spatial reconstruction, and pairs of neighboring time slabs used to obtain a high-order temporal interpolant [34]. See Fig. 5.22 for a depiction of these procedures.

The lower dimension of the coarse-space adjoint combined with the local nature



(a) Spatial reconstruction

(b) Temporal reconstruction

Figure 5.22: Illustration of spatial and temporal adjoint reconstructions. (a) shows a patch of nearest-neighbor elements used for spatial high-order reconstruction via least-squares interpolation. (b) shows reconstruction of an $r = 1$ adjoint to $r = 2$ using the left node from the adjacent future time slab and the super-convergent nodes on the current time slab (which correspond to the roots of the left Radau polynomial for $r = 1$).

of the reconstruction makes it cheaper to compute than the true fine-space adjoint. However, this savings comes at the expense of accuracy, since the reconstructed adjoint will only be a good approximation to the fine-space adjoint when the problem is nearing asymptotic convergence. By employing this strategy then, we accept that our global output error esimate may be compromised, but assume that the reconstructed adjoint will still identify the correct regions of the mesh to be adapted.

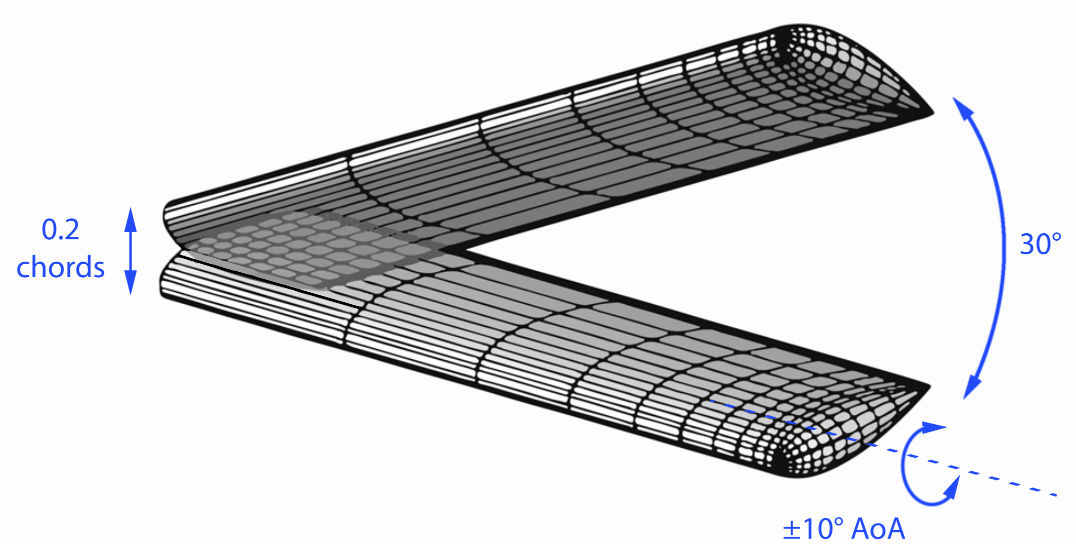


Figure 5.23: 3D wing: Schematic of the flapping motion. The flow regime is approximately that of a small house-fly.

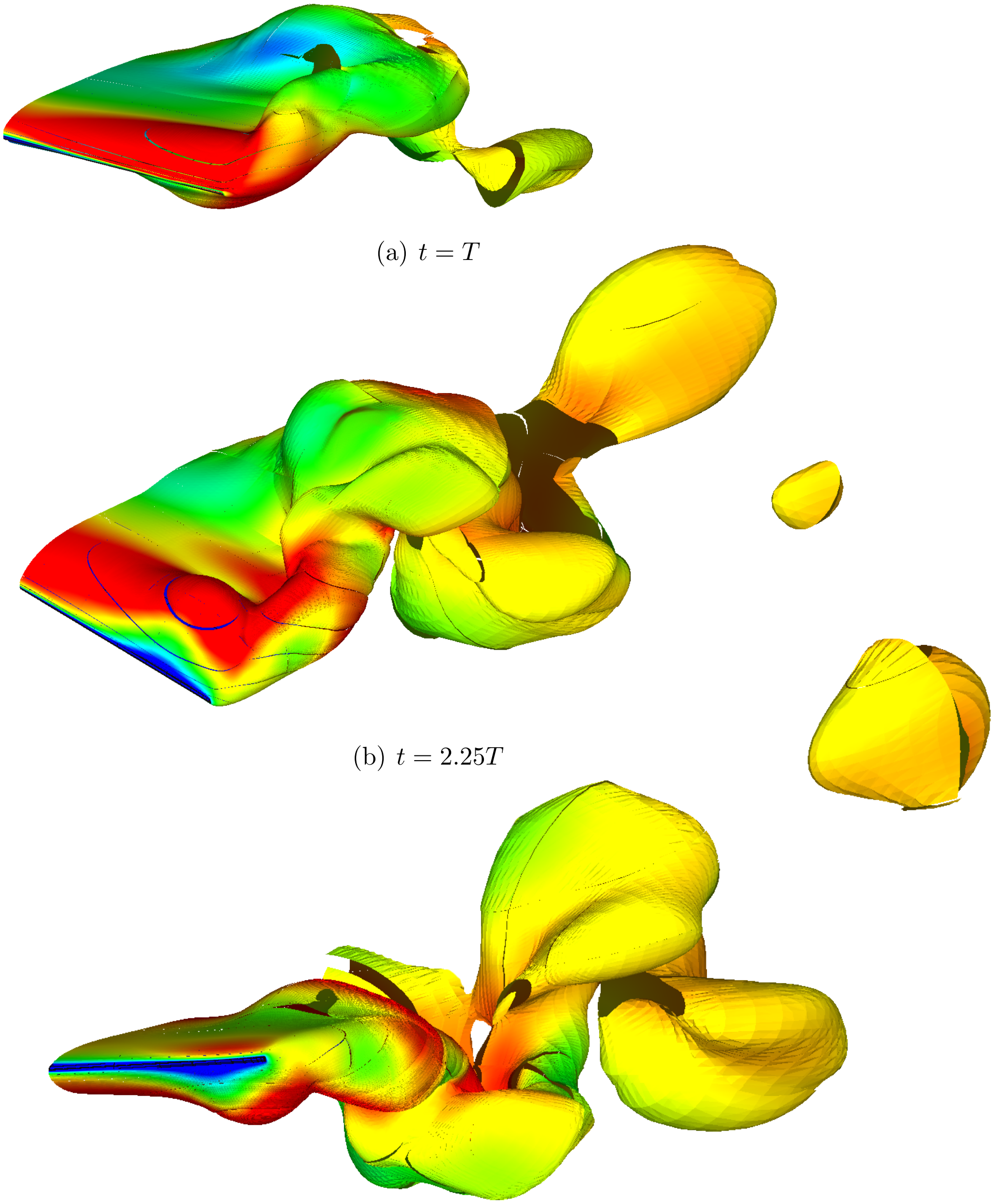
To test this strategy, we proceed in a similar manner as in 2D, though now simulating a full wing rather than just an airfoil. The wing is shown in Fig. 5.23, along with a schematic of the flapping motion. The wing moves in all dimensions, with a 30° stroke angle, a slight vertical plunge simulating movement of the "shoulder joint," and an angle of attack variation of ±10°. The Mach number is 0.3, while the Reynolds number of 500, Strouhal number of 0.4, and aspect ratio of 2 place the wing in the flight regime of a small house-fly.

Three periods of motion are simulated, and the solution at various times is shown in Fig. 5.24. Strong vortex cores develop near the leading edge and wingtip regions before detaching and shedding into the wake. For adaptation purposes, the output of interest is taken to be the lift integrated over the final 5% of the simulation time (from $t = 7.1$ to $t = 7.5$). To solve the problem, output-based, residual, and uniform-p refinement strategies are employed. Adaptations are performed starting from an initial $p = 0$, 75 time step solution, with a 30% growth factor and 5% coarsening factor used for the output- and residual-based methods. As before, the spatial order p is constrained to lie between 0 and 5, and an $r = 1$ DG scheme is used in time. The simulations are performed on the *Diamond* supercomputing cluster at the U.S. Army ERDC. The number of processors used for the output-based, residual, and uniform-p strategies is 304, 48, and 168, respectively. When comparing wall time later on, the run times are scaled appropriately according to both the number of processors and the results of a parallel efficiency study performed on the cluster.

5.7.5.1 Results

The convergence for each adaptive method is shown in Fig. 5.25, and the results are encouraging. We see that although reconstructed adjoints were used, the output-based adaptation still performs well, and converges with about two orders of magnitude fewer degrees of freedom than uniform refinement. Note that the accuracy of the error estimate itself is not ideal, as expected, but it is enough to guide the adaptation in the correct direction. Convergence as a function of wall time is shown in Fig. 5.25, and we see that the method also performs well in this context.

Of additional interest is the performance of the residual-based adaptation, which is so poor that the curve is easy to miss in the plots. In the 2D cases, the residual convergence was oscillatory, but at least hovered near the true output value. Here, the residual adaptation fails outright, and the output value does not converge at all from its initial value. The reason for this behavior is evident in Fig. 5.26, which shows the orders midway through the final residual adaptation. The residual indicator flags



(a) $t = T$

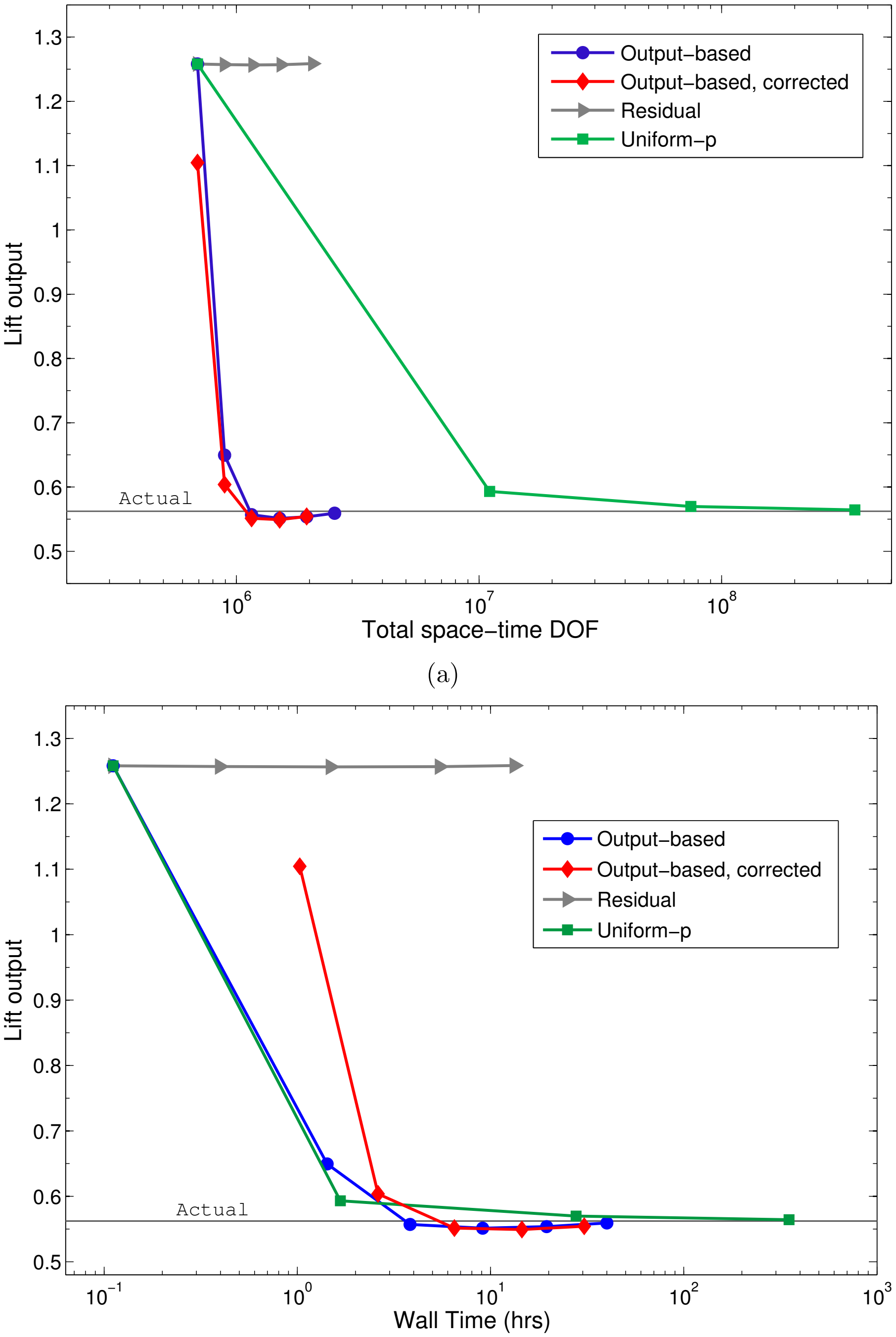
(b) $t = 2.25T$

(c) $t = 2.8T$

Figure 5.24: 3D Wing: Mach contours projected onto entropy isosurfaces, shown for several stages of the flapping motion. The maximum Mach number (in red) is approximately 0.5. The images are taken from the final ($p\!=\!3$) uniform refinement.

only large elements in the blending region for refinement, and leaves all elements near the wing at their original low order.

The output-based adaptation, on the other hand, refines elements near the wing as needed. The adapted spatial meshes from the output-based runs are shown in



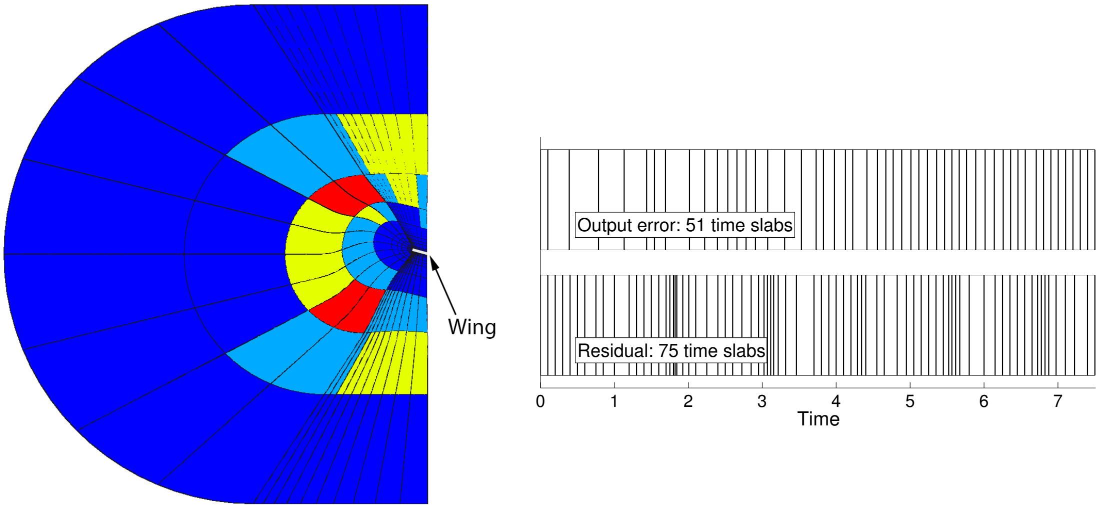
(a)

(b)

Figure 5.25: 3D wing: Output convergence as a function of (a) degrees of freedom and (b) total wall time.

Fig. 5.27, while adapted temporal grids are shown in Fig. 5.26. We see that, at early times, the output-based indicator leaves the mesh relatively coarse, but progressively increases the resolution in both space and time as the simulation progresses.

Finally, the state vs. GCL error breakdown for the output-based runs is shown in Table 5.6. As in 2D, the GCL errors make up a relatively large percentage of the total error estimate, again indicating the importance of the GCL adjoint. Thus, overall, we find that both the performance of the adaptive algorithm and the conclusions drawn about the GCL extend from two to three dimensions.



(a) Residual-adapted orders

(b) Temporal grids

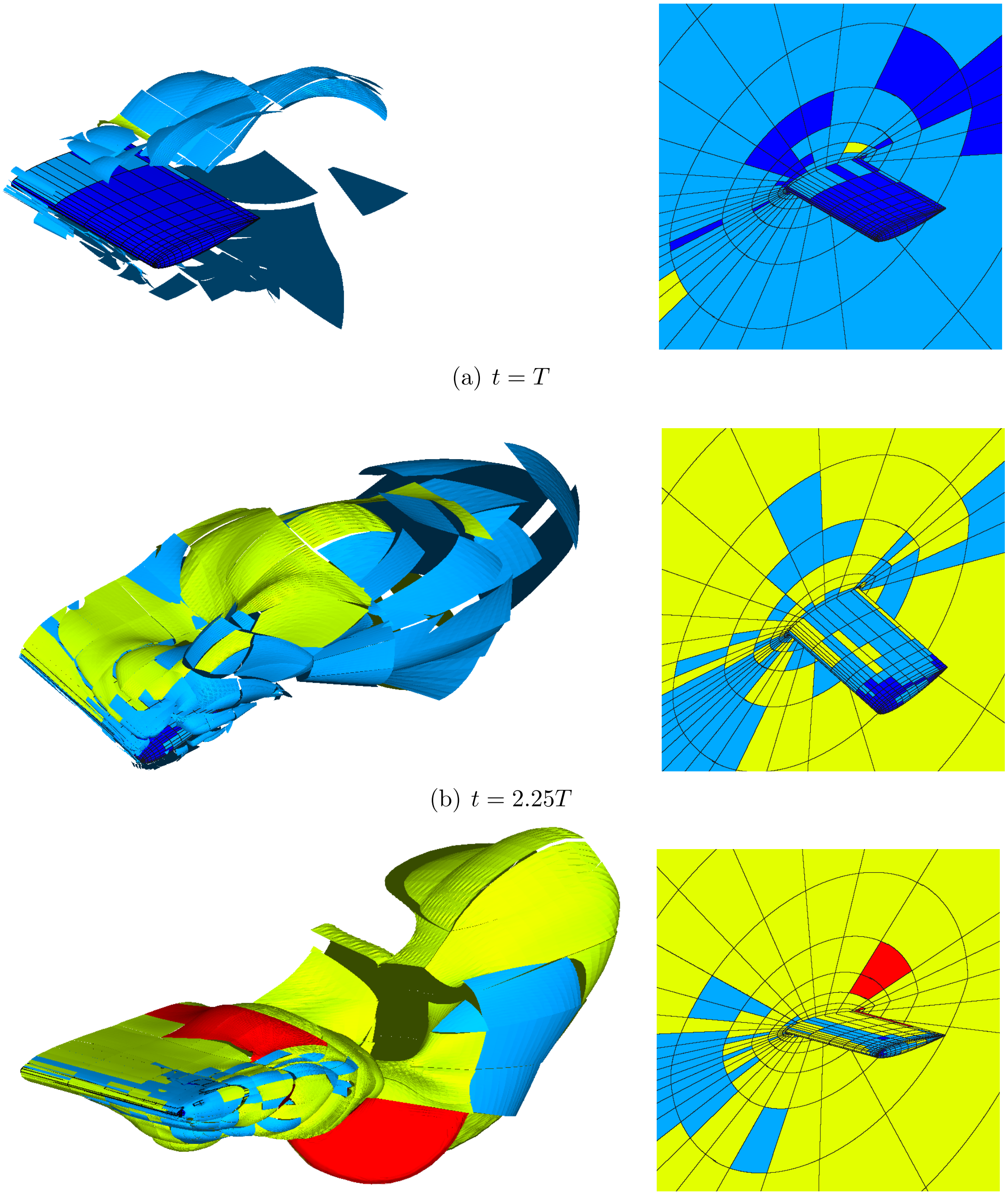
Figure 5.26: 3D wing: (a) Spatial orders from the residual adaptation midway through the simulation (blue is $p = 0$, red is $p = 3$). The adaptation targets only elements far from the wing, leading to poor output convergence. (b) Temporal grids from the final output-based and residual adaptations. The residual adaptation refines periodically with the motion, while the output-based adaptation increases the resolution near the final times.

Adapt Iter.	1	2	3	4	5
State Errors	-1.7e-1	-5.6e-2	-2.0e-2	-1.8e-2	-1.8e-2
GCL Errors	1.4e-2	9.9e-3	1.4e-2	1.6e-2	1.9e-2
GCL % of Total	**7.8**	**15.1**	**41.8**	**47.0**	**50.7**

Table 5.6: 3D wing: Contribution of state and GCL errors to the total error estimate for all iterations of output-based adaptation.

5.8 Relevance of the GCL

In the cases presented, we have shown that output-based error estimation and mesh adaptation lead to efficient output convergence in terms of both mesh size and



(a) $t = T$

(b) $t = 2.25T$

(c) $t = 2.8T$

Figure 5.27: 3D Wing: Spatial orders from the final output-based adaptation, shown at several stages of the flapping motion. Dark blue is $p = 0$, red is $p = 3$. The images on the left show the interpolation orders projected onto entropy isosurfaces.

computational time. For these cases, the GCL was used to ensure conservation, and we saw that accurate error estimates were obtained only if a corresponding GCL adjoint was used. However, an important question is: what if the GCL were ignored from the start? If we accept the possibility of conservation errors, we can neglect the GCL equation and simply solve the original form of the governing equations (Eqn. 5.15). If this is done, no separate GCL adjoint is required, and the state adjoints provide the information necessary for error estimation and adaptation.

Ignoring the GCL for the airfoil cases presented above gives the output convergence histories shown in Fig. 5.28. We see that use of the GCL for these cases does provide a slight improvement in output convergence; however, if error estimation is performed, both GCL and no-GCL runs show nearly identical corrected outputs. This suggests that satisfying the GCL may not be critical to obtaining accurate outputs, especially if those outputs are corrected by an *a posteriori* error estimate.

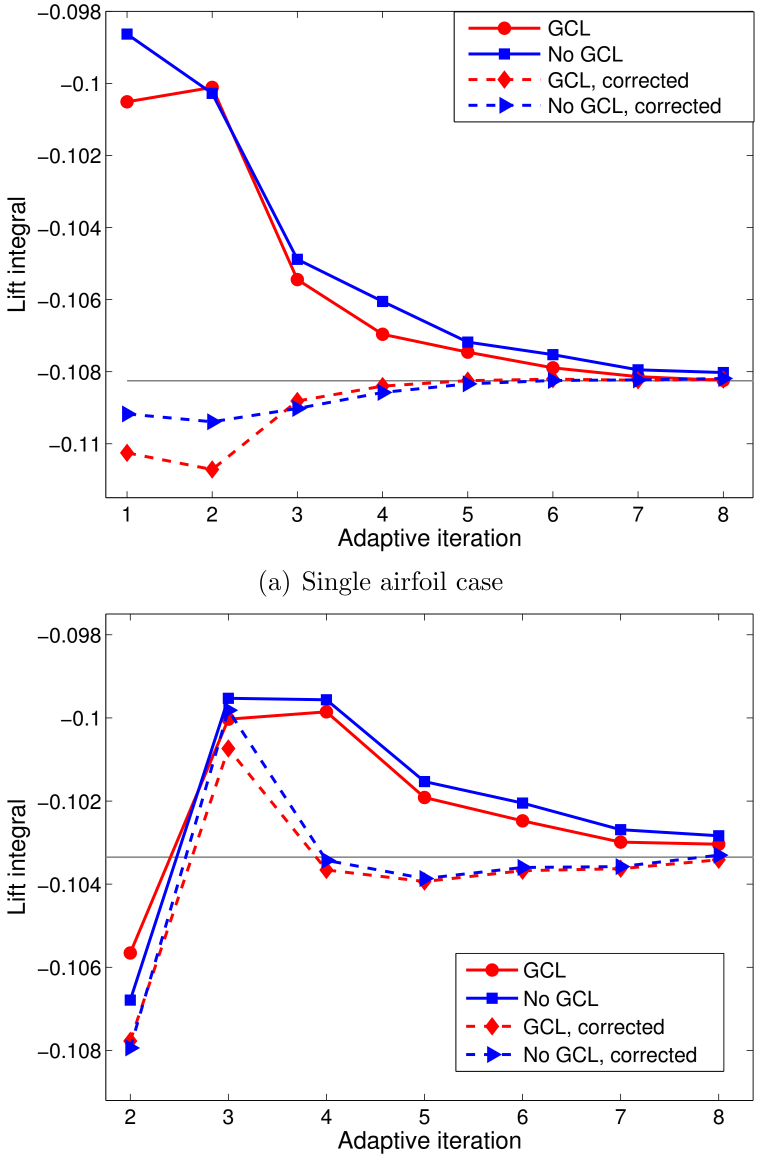
Granted, these cases have a relatively low Mach number, and we might expect the difference between GCL and no-GCL runs to increase as compressibility becomes more important. In particular, for cases with shocks, it is known that incorrect shock speeds can be obtained if a numerical method is not strictly conservative [52, 64]. However, we have found that even for cases with relatively strong shocks, correct solutions are obtained with no-GCL runs provided the order and/or mesh resolution is high enough (see Appendix C).

Overall, for cases where the GCL can be safely ignored, the primary savings would be in implementation rather than computation time. The GCL represents a small ($\sim$3%) addition to the CPU time of the primal problem, while the GCL adjoint adds roughly 6% to the adjoint solve time. (This adjoint time could be reduced further as well, since it includes a relatively inefficient finite differencing of the residuals with respect to the GCL variable.) This means that the total reduction in CPU time obtained by neglecting the GCL would be about 5%.

5.9 Conclusions

In this work, we derived unsteady adjoints for both the governing equations and the geometric conservation law, and showed that using these adjoints to drive adaptation provides orders-of-magnitude savings in the mesh size required for output convergence. A significant reduction in computational time was also achieved when the adjoints were reconstructed or smoothed to an appropriate tolerance. More common adaptation methods, such as uniform or residual-based refinement, were shown to be either inefficient or erratic for cases with mesh motion. These results were demonstrated in both two and three dimensions.

The overall impact of the GCL on error estimation was also assessed. While



(a) Single airfoil case

(b) Two-airfoil case

Figure 5.28: Output convergence for the single and two-airfoil runs with and without the GCL.

the contribution of the GCL to the output error is generally smaller than the state contribution, it can become significant during later stages of adaptation. Hence, if the GCL is enforced, a corresponding GCL adjoint is required to ensure accurate error estimates. However, results indicate that for certain cases, accurate output values may be obtained while ignoring the GCL altogether, particularly if the output is corrected by an *a posteriori* error estimate.

Overall, the proposed output-based adaptation represents an unquestionable improvement over more heuristic strategies when considering total mesh size. While reductions in CPU time were also obtained, further gains in this area are possible. For parallel computations, an algorithm for dynamically repartitioning the mesh could prove beneficial, while changes to the adaptive strategy itself may provide additional improvement. These issues are the subject of ongoing research.

Below, we list some additional topics for future work.

- Since DG-in-time schemes are relatively expensive, it would be useful to extend the above output-based techniques to more common temporal discretizations, such as Runge-Kutta schemes or multi-step methods. Runge Kutta schemes are adjoint-consistent for variable time-step sizes [84], so dynamic space-time adaptation is a viable option. However, since the temporal solution is known only at discrete nodes/stages with these methods, one question is how to define an injection into the "fine-space" when performing error estimation. While there is not a unique choice for this procedure, reconstructing a polynomial by interpolating a combination of the discrete state values and temporal derivatives should prove sufficient for error estimation.

- The adaptation performed in this work consists of dynamic-p adaptation in space and slab bisection/coarsening in time. For flows with sharp gradients (such as higher-Reynolds-number flows or flows with shocks), it would be useful to incorporate both h and p adaptation in space. Alternatively, r-adaptation, in which the nodes defining the mesh elements are adjusted to minimize the error, could also be incorporated. The adaptation in time could also be improved, by (for example), refining time step sizes uniquely for each spatial element in the mesh. While all of the above options are feasible, the challenge is to implement them in an efficient and robust manner.

- The unsteady output-based techniques presented here should be evaluated on higher-Reynolds-number problems and extended to chaotic flows. In chaotic flows, outputs are extremely sensitive to perturbations in the residuals, causing the magnitude of the adjoint to grow unbounded as it is marched backward in time. However, from a physical perspective, it should still be possible to define useful sensitivities for time-averaged outputs in these cases. The most effective way to define and compute the adjoint for these problems is the subject of ongoing research [98, 99], and a successful resolution of this issue will be critical to many engineering applications.

CHAPTER VI

Optimal Test Functions for Boundary Accuracy in Discontinuous Finite Element Methods

As discussed in the preceding chapters, the typical strategy for achieving output accuracy is to use a standard (e.g. discontinuous Galerkin) method in combination with output-based mesh adaptation. In this case, the numerical method itself is a "general-purpose" scheme, while the mesh bears the burden of providing accuracy in outputs of interest. By successively refining the mesh in the regions indicated by an adjoint vector, we are – in essence – "optimizing" the mesh while leaving the numerical method alone.

However, we might wonder: is optimizing the mesh the only way to reduce output errors? Or could the numerical method *itself* be optimized to achieve output accuracy?

In this chapter, we take a step back and investigate this alternative – and somewhat overlooked – idea.

Specifically, we investigate whether the **test space** of a finite element method can be optimized to obtain accuracy in certain outputs, and – in particular – in those outputs defined along the domain **boundaries**. These boundary outputs, which include quantities such as lift, drag, and moment, are often the most important from a practical standpoint.

As we will show, it turns out that adjoints play a critical role in the optimization of the test space, providing a further link between the present chapter and our previous work. From the previous chapters, we know that an adjoint-weighted residual represents the error in a certain output. But, by definition, in finite element methods it is the test functions that weight the residual (in the weak form). Since this test-function-weighted residual is set to zero when solving the problem, it follows that if we choose the test functions themselves to be adjoint solutions, we will be directly setting the error in certain outputs to zero. By defining these outputs appropriately, we can therefore choose which quantities the finite element method achieves accuracy in.

In the end, to compute the optimal test functions in a practical manner, we will not be solving *global* adjoint problems, but rather local ones defined on each element. As we will show, we will define these local adjoints such that, when used as test functions, they minimize the amount of error in the fluxes leaving each element. This accuracy in the local fluxes then propagates globally, leading to accuracy in outputs along the domain boundaries.

We are not the first to consider optimizing a finite element test space. There have been several schemes employing the idea of optimal test functions in the past (some fairly successfully), though for the most part their emphasis has been on obtaining "stability" rather than accuracy in certain quantities. In the following section, we discuss the idea of optimal test functions in a more historical context and highlight the place of our present work (which can be found in published form in [60, 59]) within this context.

6.1 Introduction

Over the years, finite element methods have become the primary computational tool in many branches of engineering. When initially applied within structural mechanics, the principal components of these methods – the test and trial functions – were chosen to be identical, resulting in a so-called Galerkin formulation. Based on this choice, the continuous Galerkin (CG) method has since seen wide application, owing both to its simplicity and provable optimality for many problems of interest.

When CG was applied to fluid dynamics problems, however, the results were poor. For convection-diffusion equations, spurious oscillations arose in the presence of gradients and boundary layers, corrupting the numerical solution. Over time, it became apparent that this lack of "stability" could be blamed on a suboptimal test space, and many so-called Petrov-Galerkin (or "stabilized") schemes have since arisen to address this issue [15, 39, 54, 13]. These schemes, the most popular of which is the Streamline Upwind Petrov-Galerkin (SUPG) method [15], improve the stability of CG by modifying its test space in an upwind-biased manner.

With the emergence of discontinuous finite element methods, however, the impetus for optimizing the test space largely disappeared, since the use of a Riemann flux provides an inherent measure of stability [14]. Thus, methods of a discontinuous

Galerkin (DG) type, which take the simplest possible test space (setting it equal to the trial space), have come to dominate the literature [8, 22, 3, 50, 56, 96, 41, 87]. But the question naturally arises: is this the best option? Or, for discontinuous methods, is there a better – optimal – choice of test space?

This is the question we investigate here. Before doing so, however, we must say what we mean by "optimal." For discontinuous methods, this requires a slight shift in perspective. Rather than viewing the test space as a means to "fix" stability issues, we can instead view it as a means to obtain a specific, goal-oriented approximation of the true solution – one that provides accuracy in the regions we care about. This is, in the end, the role of the test space in *any* finite element method: to define its goal. The appearance of oscillations or "instabilities" is just evidence that, in some sense, that goal has not been well defined.

Typically, the goal of a simulation is to achieve accuracy in a certain norm of interest. The optimal test functions can then be defined as those that render the numerical solution the best approximation to the true solution in the desired norm. This idea has been pursued by several authors in a continuous context [5, 27, 44, 17, 51, 4], dating back to the work of Barrett and Morton in 1984 [5]. In addition, the test functions of stabilized schemes such as SUPG (which achieves H^1 optimality for certain problems [55]) can be viewed as optimal in a similar sense. More recently, Demkowicz and Gopalakrishnan have introduced discontinuous Petrov-Galerkin (DPG) methods, which employ optimal test functions within a more general, discontinuous framework [25, 26, 105, 20, 16]. These methods, developed initially within an "ultraweak" [26] context and adapted to hybrid methods in [71], have interior L^2 optimality as their primary goal.

In the present work, we pursue a different goal. We note that while domain-interior accuracy is important, from an engineering standpoint the regions of greatest interest are often the domain *boundaries*. Indeed, obtaining the forces, fluxes, and distributions of quantities along the boundaries is often the principal goal of a simulation. We therefore make this our aim in the current work. In addition, for purposes of both familiarity and computational efficiency, we pursue this aim within the context of standard DG and hybrid DG (HDG) methods.

To that end, we present a simple framework for deriving and computing optimal test functions. These test functions render the solution optimal in a desired error norm, which in this case we choose to emphasize boundary accuracy. While in general the optimal test functions would satisfy global differential equations, when boundary accuracy is desired they can be computed in a purely element-local manner. When used within a standard DG or HDG framework, they result in a scheme we call the boundary discontinuous Petrov-Galerkin (BDPG) method.

Here, we list some relevant properties of the optimal test functions and the corresponding BDPG method:

1. The optimal test functions are elementwise adjoint solutions (i.e. generalized Greens functions [30]), similar to the fine-scale Greens functions used in several multiscale methods [54, 53, 55].
2. As adjoints, they ensure that information is properly "upwinded" within each element, and they attempt to minimize the amount of error that (after being generated on the element interior) ultimately reaches the element boundaries.
3. They therefore lead to accuracy in the element-interface fluxes, which propagates globally to the domain boundaries.
4. They have close ties to *a posteriori* error estimation [10, 36], which explains their ability to minimize the errors in the fluxes.
5. For one-dimensional linear problems, if the test functions are well-represented, exact boundary fluxes are obtained.
6. For multi-dimensional linear problems, if the test functions *and* interface fluxes are well-represented, exact boundary fluxes are obtained.
7. For nonlinear problems, the optimal test functions lead to boundary flux rates of order $2p+2$ (an improvement over the $2p+1$ rates associated with a standard DG method).

While the theory applies to general PDEs, here we focus on steady-state fluid applications, showing results for advection-diffusion, Euler, Navier-Stokes, and other equations. We begin with a simple approximation problem and a one-dimensional example that emphasizes the main ideas of the method. We then move on to multiple dimensions and systems of equations, as well as to nonlinear problems. The benefits and limitations of optimal test functions are discussed, and remaining challenges are identified.

Finally, we note that we have tried to keep the discussion in this chapter at a relatively high level, without much of the formalism often associated with finite element methods. While we are by no means experts in functional analysis, this is done primarily in the hope that the main concepts can be seen clearly.

6.2 An Approximation Problem

In this section, we begin by discussing a type of "curve-fitting" or function approximation problem, which on the surface has little to do with finite element methods. However, as we will see, these ideas will be relevant to the derivation of optimal test functions later on.

6.2.1 One-Dimensional Function Approximation

Imagine we have some one-dimensional function, $u(x)$, the shape of which is known to us. (See Fig. 6.1.) Now assume that we would like to approximate this $u(x)$ with a polynomial.[1] If we call our polynomial approximation $u_H(x)$, then we can expand this $u_H(x)$ as

$$u_H(x) = \sum_{i=1}^{N} U_i \, \phi_i(x) \, , \tag{6.1}$$

where the ϕ_i are basis functions for our chosen polynomial space, and the U_i are discrete coefficients. If our polynomial space is of order p, then the dimension N is $N = p + 1$.

Now, we would like to find the coefficients U_i that make u_H the "best" approximation to the original curve u. However, before we can do this, we must specify which norm we would like the best approximation *in*. For example, if we desire a least-squares approximation of u, then we need u_H to minimize the following norm:

$$||e||^2 = \int\limits_{\Omega} (u_H - u)^2 \, dx \, . \tag{6.2}$$

Here, $e \equiv u_H - u$ is the approximation error, while $\Omega = [x_L, x_R]$ is the extent of the domain over which u is defined.

While the above norm would ensure that u_H provides an accurate approximation on the domain interior, we may also wish to obtain accuracy on the domain *bound-*

[1] In other words, we would like to perform a type of continuous "curve fit."

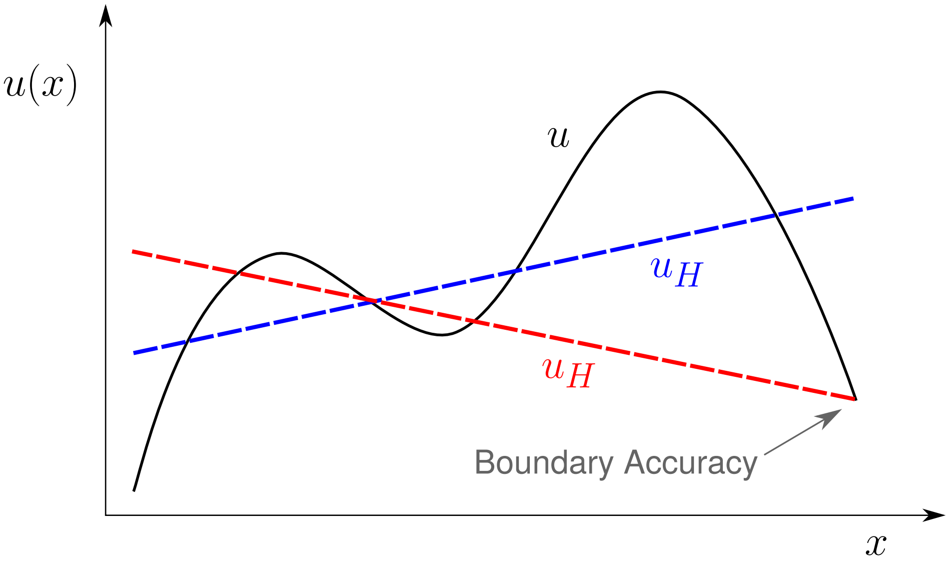


Figure 6.1: Two $p = 1$ approximations of a one-dimensional function $u(x)$. The blue curve provides interior accuracy, while the red curve provides right-boundary accuracy. The amount of boundary accuracy depends solely on the type of error norm minimized – not on the type of polynomial itself.

aries. If we are interested in accuracy near the right boundary, for example, then we could modify our error norm to read:

$$||e||^2 = \int\limits_{\Omega} (u_H - u)^2 \, dx \, + \, w_R \, (u_H - u)^2 \Big|_{x_R} . \tag{6.3}$$

Here, w_R is a weight that determines how much emphasis is placed on the boundary; if it is taken large, more accuracy is obtained there.

Assume that Eqn. 6.3 is our error norm of interest. Then in order to find the coefficients U_i that minimize this norm, we take the partial derivative of $||e||^2$ with respect to each U_i, and set this equal to zero. By basic calculus, the derivative of a function is zero at a critical point, and in the case of $||e||^2$ this critical point is in fact a minimum.

Since $u_H = \sum_{i=1}^{N} U_i \, \phi_i$, differentiating Eqn. 6.3 with respect to U_i gives:

$$\frac{\partial ||e||^2}{\partial U_i} = 0 = \int\limits_{\Omega} 2 \, (u_H - u) \, \phi_i \, dx \, + \, 2 \, w_R \, (u_H - u) \, \phi_i \Big|_{x_R} . \tag{6.4}$$

Thus, if u_H is to provide the minimum error in $||e||^2$, it must satisfy the following N equations:

$$\int_{\Omega} \phi_i \left(u_H - u\right) dx \; + \; w_R \, \phi_i \left(u_H - u\right)\bigg|_{x_R} = 0 \qquad i = 1..N \quad . \tag{6.5}$$

Fig. 1 shows two potential u_H curves that satisfy the above equations – one that emphasizes boundary accuracy (corresponding to large w_R) and another that emphasizes interior accuracy (corresponding to small w_R). From the figure, a few relevant conclusions can be drawn.

First, we see that the amount of boundary accuracy obtained depends solely on the choice of error norm minimized. Thus, if the goal is to achieve boundary accuracy, the type of polynomial used in the approximation (i.e. the "trial" space) is not relevant. While this is strictly true only in one dimension, a similar idea holds in general: the lower the dimension of a given region of interest (in this case a zero-dimensional boundary), the more important the choice of error norm becomes and the less important the choice of trial space becomes.[2]

As we will show, in the context of a finite element method, the function u plays the role of the exact solution to a PDE, u_H is analogous to the numerical solution, and the error norm can be controlled by modifying the test space. It is clear then that if boundary accuracy is desired in this context, it is a test-space optimization that should play the central role. This idea, along with Eqn. 6.5, will be critical in deriving an optimal finite element scheme.

6.3 Optimal Test Functions (One-Dimensional Example)

Let us shift focus now and discuss our actual goal: solving PDEs. For the sake of clarity, consider a differential equation of the following form:

$$\begin{aligned} \underbrace{a\frac{\partial u}{\partial x} + cu}_{Lu} = f & \qquad x \in \Omega \\ u = u_L & \qquad x = x_L \end{aligned} \tag{6.6}$$

This is a linear advection-reaction problem with a source term $f(x)$, where we assume $a > 0$ so the Dirichlet condition u_L is well-posed. The differential operator L is given by $L \equiv a\frac{\partial ()}{\partial x} + c()$, and the residual is defined as $r(u) \equiv Lu - f$. The domain

[2] Though as we will see, in multiple dimensions the trial space does remain relevant.

$\Omega = [x_L, x_R]$ is the same as in Sec. 6.2 and will be assumed here to consist of a single element.[3]

To obtain the weak form of the above problem, we multiply the residual by a test function and integrate, giving

$$\int\limits_{\Omega} v\,(Lu - f)\,dx = 0 \qquad \forall v \in \mathcal{V}\,, \tag{6.7}$$

where v is any test function in some continuous space $\mathcal{V}$. For sufficiently smooth[4] u and v, integrating the vLu term by parts then gives

$$\int\limits_{\Omega} \underbrace{\left[-a\frac{\partial v}{\partial x} + cv\right]}_{L^* v} u\,dx \,+\, vau\Big|_{x_L}^{x_R} - \int\limits_{\Omega} vf\,dx \,=\, 0 \qquad \forall v \in \mathcal{V}\,. \tag{6.8}$$

Here, we have defined the operator that emerges after integration by parts as $L^* \equiv -a\frac{\partial ()}{\partial x} + c()$. Finally, after inserting the boundary condition and moving the "known" terms to the right-hand side, we obtain

$$\underbrace{\int\limits_{\Omega} L^* v\, u\,dx \,+\, vau\Big|_{x_R}}_{b(u,v)} = \underbrace{\int\limits_{\Omega} vf\,dx \,+\, vau_L\Big|_{x_L}}_{l(v)} \qquad \forall v \in \mathcal{V}\,. \tag{6.9}$$

This equation, which relates the bilinear form $b(u, v)$ to the load $l(v)$, is satisfied by the exact solution u for any smooth test function v.

Now, to compute an approximate solution to the PDE in Eqn. 6.6, a standard (upwind) DG method attempts to mimic Eqn. 6.9. In other words, it seeks an approximate solution $u_H \in \mathcal{U}_H$ that satisfies

$$\underbrace{\int\limits_{\Omega} L^* v_h\, u_H\,dx \,+\, v_h a u_H\Big|_{x_R}}_{b(u_H,v_h)} = \underbrace{\int\limits_{\Omega} v_h f\,dx \,+\, v_h a u_L\Big|_{x_L}}_{l(v_h)} \qquad \forall v_h \in \mathcal{V}_h \tag{6.10}$$

where v_h is any test function in some discrete space $\mathcal{V}_h$. Once a basis $\{\phi_i\}$ is chosen for the approximation space $\mathcal{U}_H$ (typically assumed to be a polynomial space of order

[3] The single-element setting allows us to assume a smooth numerical solution. While in the end we are interested in discontinuous finite element methods, the ideas in this section will ultimately be applied only *within* each element, where the smoothness assumption is justified.

[4] We assume that it is valid to evaluate u and v on the domain boundaries.

p), the discrete u_H can be represented as

$$u_H = \sum_{i=1}^{N} U_i \, \phi_i(x) \, , \tag{6.11}$$

where the U_i are the unknown solution coefficients. Then the remaining question is: what should our test space $\mathcal{V}_h$ be? A standard Galerkin method would choose $\mathcal{V}_h = \mathcal{U}_H$, so that the test space is identical to the trial space. But is this the best choice?

We have arrived at the critical point: we have used the word "best." *Best in what way?* Recall that in Sec. 6.2 we encountered the same issue. To get a "best" approximation, we had to first define the norm we wanted the best approximation in. In the same way, when solving a PDE, we must say *how* we would like the numerical solution u_H to approximate the exact u.

Typically, we would like u_H to obtain some amount of interior and (particularly in this work) boundary accuracy. For the current problem, the relevant boundary to request accuracy in is the right boundary, since the solution on the left is already known from the Dirichlet condition. Thus, the norm we desire the best approximation in may look something like:

$$||e||^2 = \int_{\Omega} (u_H - u)^2 \, dx \; + \; w_R \, (u_H - u)^2 \bigg|_{x_R} \, .$$

This is the same norm used in Sec. 6.2. Recall from that section that if u_H is to minimize this norm, it must satisfy the zero-derivative condition given by Eqn. 6.5, i.e.

$$\int_{\Omega} \phi_i \, (u_H - u) \, dx \; + \; w_R \, \phi_i \, (u_H - u) \Bigg|_{x_R} = 0 \qquad i = 1..N.$$

Now, we claim that with a certain ("optimal") choice of test functions, we can ensure that Eqn. 6.5 is satisfied by our finite element solution. First, note that since $\mathcal{V}_h \subset \mathcal{V}$, we can choose a common test function $v_h \in \mathcal{V}_h \subset \mathcal{V}$ for Eqns. 6.9 and 6.10.

Doing so and subtracting the two equations then results in:

$$\underbrace{\int\limits_{\Omega} L^{*}v_h\,(u_H - u)\,dx \;+\; v_h a\,(u_H - u)\Big|_{x_R}}_{b(e,v_h)} \;=\; 0 \qquad \forall v_h \in \mathcal{V}_h\,. \tag{6.12}$$

This equation is just the bilinear form evaluated with the solution error, and is satisfied regardless of how the test space is chosen. However, to see how an optimal scheme can be created, note that the above expression involves some quantity that is equal to zero. Next, note that the error minimization statement (Eqn. 6.5) also involves a quantity that is equal to zero. Then the idea is this: if we can make the above $b(e, v_h)$ *look* like the error minimization statement, our finite element solution u_H will necessarily minimize that error norm.

To make this clearer, we first make a simple notational change. Regardless of how the test space is chosen, it must have the same dimension (N) as the trial space, in order for the number of equations to equal the number of unknowns. Therefore, we can replace the general test function v_h above with a specific test function v_i, where i ranges from 1 to N. Doing so gives:

$$\int\limits_{\Omega} L^{*}v_i\,(u_H - u)\,dx \;+\; a\,v_i\,(u_H - u)\Big|_{x_R} \;=\; 0 \qquad i = 1..N. \tag{6.13}$$

Now, our goal is to make this equation look like the error minimization statement in Eqn. 6.5, which is:

$$\int\limits_{\Omega} \phi_i\,(u_H - u)\,dx \;+\; w_R\,\phi_i\,(u_H - u)\Big|_{x_R} \;=\; 0 \qquad i = 1..N.$$

By simply comparing these equations, we see that the way to make them identical is to set

$$\left.\begin{array}{ll} L^{*}v_i = \phi_i & x \in \Omega \\[2ex] a\,v_i = w_R\,\phi_i & x = x_R \end{array}\right\} \qquad i = 1..N \quad . \tag{6.14}$$

Thus, if the discrete solution u_H is to minimize the error given by Eqn. 6.3, the test functions v_i must satisfy the above differential equation(s) and boundary condition(s). The test functions that satisfy these equations are the ***optimal test functions***, in the sense that they make u_H the best approximation to u in the desired error norm.

At this point, it is worth making a few remarks.

Remark 6. While for simple problems the optimal test functions can be found analytically, in general we must compute them numerically. This can be done by (e.g.) approximating them in a high-order polynomial space. A similar strategy has been pursued in much of the DPG literature [26].

Remark 7. The idea of making $b(e, v_h)$ reduce to the derivative of a desired error norm has arisen previously in the context of continuous finite element methods [5, 4, 44, 27]. More recently, the DPG methods of Demkowicz, Gopalakrishnan, *et al.* have employed similar ideas, primarily within the context of a discontinuous "ultra-weak" formulation [25, 26, 105, 20]. In contrast, here we consider standard DG and HDG formulations[5], with the specific goal of achieving boundary accuracy.

Remark 8. Recall that to achieve boundary accuracy we simply choose w_R to be large. This emphasizes the boundary term in the error norm (Eqn. 6.3), which in the limit of large w_R ensures that u_H provides the best approximation to u on the boundary. When the corresponding optimal test functions are used within a DG or HDG framework, we refer to the resulting scheme as a "boundary discontinuous Petrov-Galerkin" (BDPG) method.

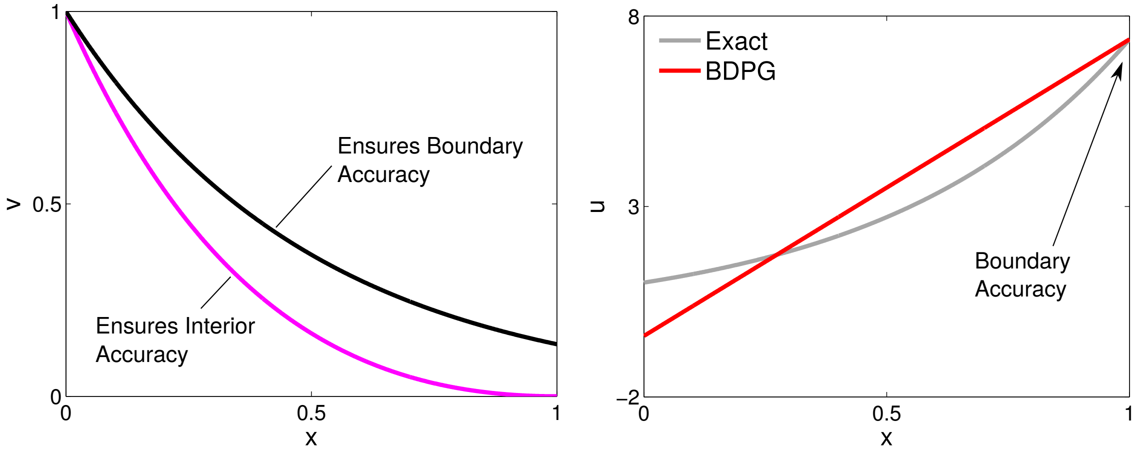
In practice, the precise choice of w_R is up to the user, and its magnitude is correlated with the amount of accuracy desired on the boundaries. For example, for one-dimensional linear problems, choosing $w_R = 10^3$ will provide $\mathcal{O}(10^{-3})$ accuracy in the right-boundary state, while choosing $w_R = 10^{10}$ will provide $\mathcal{O}(10^{-10})$ accuracy. The optimal test functions can also be reformulated with w_R taken to infinity, but since this would make the error norm defined in Eqn. 6.3 no longer strictly a norm, we will take w_R large but finite in this work. Furthermore, in practice, errors in the computation of the test functions often make the use of an infinite w_R unnecessary.

Remark 9. Note that since information propagates to the right in the current advection example, choosing w_R large – i.e. requesting accuracy in the right-boundary state $u_H|_{x_R}$ – may also be viewed as requesting accuracy in the *outgoing flux.* This is the view we adopt when generalizing to multidimensional systems.

[5] Note that for certain problems (such as pure advection), the ultra-weak DPG formulation may be preferable to DG or HDG formulations, since its independent treatment of interior and flux unknowns enables an independent optimization of each [25]. However, for problems such as advection-diffusion, the HDG formulation is preferred since its number of globally coupled unknowns is just half that of the corresponding ultra-weak formulation.

Remark 10. We have posed the above derivations in a single-element setting, which means the equations satisfied by the optimal test functions are in fact *global* differential equations. In practice, solving these would be prohibitive, so we will need to localize them in some way.

Remark 11. Finally, use of the optimal test functions will not, in general, result in a conservative scheme. This is because the solutions to the test function equations (Eqn. 6.14) will not always contain the constant mode. However, from a purely geometric standpoint, when boundary accuracy is desired we do not necessarily want the scheme to be conservative. For example, imagine that we are in a situation like the one depicted in Fig. 6.1 and wish to obtain accuracy on both left *and* right boundaries. Then requesting that u_H interpolate u on both boundaries would preclude it from satsifying a "conservation" relation such as $\int_\Omega u_H \, dx = \int_\Omega u \, dx$, since the linear interpolant of the boundary data would necessarily have a smaller integral value than u itself. However, if the user does desire conservation, this could theoretically be added as a constraint during the computation of the optimal test functions.



(a) Optimal test functions

(b) Single-element BDPG solution

Figure 6.2: One-dimensional advection-reaction: (a) Normalized optimal test functions corresponding to a $p = 1$ trial space and large w_R. The black test function (v_2 in Eqn. 6.15) provides right-boundary accuracy, while the remaining test function (v_1 in Eqn. 6.15) provides interior L^2 accuracy. Note the upwind/leftward bias of both functions. (b) The solution obtained using the optimal test functions. Right-boundary accuracy is achieved.

We will elaborate on these remarks later on. For now, let us return to our one-dimensional advection-reaction problem. For this problem, we can solve Eqn. 6.14

analytically to find the optimal test functions. If we assume a $p = 1$ trial space with Lagrange basis functions $\phi_1 = 1 - x$ and $\phi_2 = x$, we can compute two corresponding test functions. For a parameter choice of $a = 1$, $c = -2$, and $f(x) = 0$ in Eqn. 6.6, these test functions are

$$v_1 = \frac{1}{4}e^{2(1-x)} + \frac{x}{2} - \frac{3}{4} \qquad \text{and} \qquad v_2 = \left(w_R - \frac{1}{4}\right)e^{2(1-x)} - \frac{x}{2} + \frac{1}{4}\,. \tag{6.15}$$

These functions are plotted in Fig. 6.2, where w_R has been taken large and the maximum values have been normalized to unity. The solution obtained by using these test functions in Eqn. 6.10 is also shown. Note that the solution achieves accuracy on the right boundary, as desired.

From the figure, we see that in contrast to the symmetric nature of the standard Galerkin test functions, the optimal test functions display a clear upwind bias. Thus, from a physical perspective, we can think of these test functions as performing a proper upwinding of information within the domain. This idea is related to another important property of the optimal test functions – namely, that they are *adjoint* solutions.

6.4 Optimal Test Functions as Adjoints

To see that the optimal test functions satisfy adjoint equations, recall from Chapter II (Eqn. 2.28) that for a given differential operator L, the definition of its adjoint operator L^* is:

$$(Lu, v) \;=\; (u, L^* v) \qquad \forall u \in \mathcal{U},\;\; \forall v \in \mathcal{V}, \tag{6.16}$$

where $\mathcal{U}$ and $\mathcal{V}$ are function spaces over which the above inner product is defined. In practical cases, L^* is simply the operator that emerges after integrating by parts, and hence is exactly the operator in the test function equations (Eqn. 6.14).

As derived in Chapter II, an adjoint equation provides the sensitivity of a certain output to perturbations in the residuals of the governing equation. Therefore, if the optimal test functions themselves satisfy an adjoint equation, this begs the question: *for what output*?

By examining Eqn. 6.14, we see that it is identical in form to the continuous adjoint equation (Eqn. 2.31) derived in Chapter II for steady advection. Thus, just as the right-hand side of Eqn. 2.31 represents an output linearization, the right-hand side of Eqn. 6.14 must also represent an output linearization. By inspection, we see that the output whose linearization (with respect to u) matches Eqn. 6.14 is:

$$J_i = \int\limits_{\Omega} \phi_i u \, dx \; + \; w_R \phi_i u \bigg|_{x_R} . \tag{6.17}$$

This output represents a *projection* of the exact solution u against the i-th trial basis function.

Now, from *a posteriori* error estimation (e.g. Eqn. 2.82), we know that the adjoint-weighted residual represents the error in a given output. Thus, since the optimal test functions v_i are adjoints for the outputs J_i, when we ultimately use them in the finite element weighted residual,

$$\int\limits_{\Omega} v_i \underbrace{(Lu_H - f)}_{r(u_H)} \, dx \;\; = \;\; 0 \;\; = \;\; -\delta J_i \qquad i = 1..N, \tag{6.18}$$

we are directly enforcing that the error in each projection output J_i, i.e. $\delta J_i \equiv J_i(u) - J_i(u_H)$, is zero. This implies that the discrete solution u_H is the direct projection of the exact solution u into the trial space, with respect to the desired error norm. This is the same conclusion arrived at in the previous section, but viewed from a different perspective.

Finally, to further clarify the relationship between the outputs J_i and the minimization of the error $||e||^2$, consider the following. We have said that using the optimal v_i gives zero error in the J_i. But from the above definition of J_i, zero error in J_i implies

$$-\delta J_i = J_i(u_H) - J_i(u) = \int\limits_{\Omega} \phi_i (u_H - u) \, dx \; + \; w_R \phi_i (u_H - u) \bigg|_{x_R} \; = \; 0. \tag{6.19}$$

This is identical to the statement that $\frac{\partial ||e||^2}{\partial U_i} = 0$, i.e. it is identical to Eqn. 6.5, and thus implies that $||e||^2$ is minimized.

Remark 12. Note that if the test functions are approximated numerically (e.g. at some order p_{test}), the error in the J_i will not be identically zero. However, from Eqn. 6.18, the convergence rate of this error will at a minimum correspond to the sum of the test function and residual convergence rates (i.e. $p_{\text{test}} + p + 1$), and at a maximum to a value of $2p_{\text{test}} + 1$ (via a Galerkin orthogonality argument). Thus, taking $p_{\text{test}} > p$ will yield higher output convergence rates than the standard $2p + 1$ rates of DG.

Remark 13. Since adjoint solutions are a type of "generalized" Green's function [30], the optimal test functions are closely related to the fine-scale Green's functions used in many multiscale methods, such as the Variational Multiscale Method [53]. Considered in this context, approximating the optimal test functions with $p_{\text{test}} > p$ can be thought of as bringing in "fine-scale" information that ultimately leads to improved solution accuracy.

6.5 Localization of Test Functions

With the nature of the optimal test functions discussed, we now turn to the issue of localization. In the above sections, we derived the optimal test functions while assuming a single-element mesh. This means that the adjoint equations satisfied by the test functions are in fact *global* differential equations. Since in practice these would be prohibitive to solve, we need to find a way to localize the test functions without giving up their accuracy.

Fortunately, for norms $||e||^2$ emphasizing boundary accuracy, localization is straightforward. We simply loop over each element in the mesh and apply the theory from Sec. 6.3 *inside* each element. We thus solve purely local adjoint problems (with support over a single element) to compute the test functions, with the local outputs defined by

$$J_i = \int\limits_{\Omega_e} \phi_i u \, dx \, + \, w_R \phi_i u \bigg|_{x_{e,R}} . \tag{6.20}$$

Here, Ω_e represents the domain of a given element, while $x_{e,R}$ represents its right (downwind) boundary. Note that this output has the same form as in Eqn. 6.17, but is defined over element Ω_e rather than the entire domain.

In this localized context, the optimal test functions then minimize the desired error norm within each element. Thus, taking w_R large in Eqn. 6.20 will provide accuracy in the outgoing flux on each *element* boundary, rather than on the domain boundary. However, the critical idea is that, if flux accuracy is obtained on each element boundary, then this accuracy will propagate downstream, ultimately yielding accuracy on the domain boundary. A similar idea holds for more general problems, including those with diffusion terms. Since the fluxes represent the only means by which elements in the mesh communicate, if these local fluxes can be made accurate, global accuracy follows naturally.

This idea is supported analytically as well. Appendix D shows that for the current problem, as w_R is taken large, the test space formed by the local adjoints actually contains the global adjoints corresponding to the domain-boundary fluxes. This implies that (if well-represented) the local test functions are in fact globally optimal.

6.6 Implementation (One-Dimensional Example)

With the localization defined, we are ready to use the optimal test functions in a practical setting. First, we note that the above ideas can be applied to several discontinuous formulations. For the current problem, we will apply them to a standard DG formulation, while later on we will use a hybrid (HDG) method. Regardless of the method used, there are two main steps to be performed: (1) the computation of optimal test functions on each element; and (2) the use of these test functions in the bilinear form.

6.6.1 Computation of Test Functions

For most problems, it will not be possible to solve for the optimal test functions analytically. Instead, we must approximate them numerically. If the mesh is uniform or the element-wise adjoint problems are self-similar, the localized test functions can be computed just once on a reference element and can then be "copied" onto each element in turn. Otherwise, independent test functions must be computed on each element. The simplest way to compute the test functions is to solve the local adjoint problems using a DG method. For the current advection-reaction problem, we thus solve the following equation for each v_i on a given element[6]:

Find $v_i \in \mathcal{U}_{\text{test}}$ such that:

$$\underbrace{\int_{\Omega_e} L^* v_i \, \delta u \, dx \; + \; a v_i \, \delta u \Big|_{x_{e,R}}}_{b_e(\delta u, v_i)} = \underbrace{\int_{\Omega_e} \phi_i \, \delta u \, dx \; + \; w_R \phi_i \delta u \Big|_{x_{e,R}}}_{J_i(\delta u)} \qquad \forall \delta u \in \mathcal{U}_{\text{test}} \, . \tag{6.21}$$

Note that this is just the weak form of the adjoint equations given in Eqn. 6.14 (which is analogous to the adjoint equation derived in Eqn. 2.50). These equations are solved in an enriched space $\mathcal{U}_{\text{test}}$ with corresponding order p_{test}, which must be *higher* than the original order p of the space $\mathcal{U}_H$. The higher the p_{test}, the more accurate the

[6] Of course, for this problem, we know the analytical form of the test functions from the previous section, but we compute them numerically here to demonstrate the procedure used for more general problems.

test functions, and the more accurate the final solution on the boundaries. Indeed, in one dimension, if the local test functions are represented exactly (and w_R is taken large), machine-precision element and boundary fluxes are be obtained.

For DG codes that construct a primal Jacobian matrix, the above adjoint equations do not need to be explicitly discretized. Instead, a discrete adjoint approach (defined by Eqn. 2.15) can be taken in which the equations

$$\frac{\partial \mathbf{R}}{\partial \mathbf{U}}^T \mathbf{V}_i = \frac{\partial J_i}{\partial \mathbf{U}}^T \qquad i = 1..N\,, \tag{6.22}$$

are solved for the test function coefficients $\mathbf{V}_i$ on each element, where $\partial\mathbf{R}/\partial\mathbf{U}$ and $\partial J_i/\partial\mathbf{U}$ are the elementwise order-p_{test} Jacobian and output linearization, respectively.

6.6.1.1 Choice of Boundary Weight

As mentioned earlier, the choice of w_R in Eqn. 6.21 is up to the user, and the larger this value is taken, the greater the amount of boundary accuracy achieved. (For a demonstration of this effect, the reader may look ahead to Fig. 6.9b.) While it may seem that we should take w_R to infinity, in practice we choose w_R to be large but finite. There are several reasons for this:

1. Keeping w_R finite allows us to remain within the setting of minimizing an error norm.

2. Numerical errors in the test function computation itself will typically render an infinite w_R unnecessary. Particularly on coarse meshes or for low values of p_{test}, these errors will dominate over the errors associated with a finite w_R. Theoretically, as the mesh is refined or the order p_{test} is increased, the value of w_R will play a larger role and should be increased proportionally. However, taking w_R to be a large value (e.g. $\mathcal{O}(10^{10})$) from the beginning also suffices.

3. As we will see, for multi-dimensional and nonlinear problems, additional sources of error due to the trial space representation and the linearization will again render an infinite w_R unnecessary.

For the problems in this work, we will typically choose w_R to be in the range $10^8 - 10^{15}$, depending on the problem type and the accuracy desired. Finally, since taking w_R large leads to large test function magnitudes, we orthonormalize the optimal test functions after computation with respect to a discrete or continuous L^2 norm. This ensures that the primal system (discussed below) remains well-conditioned.

6.6.2 Construction of Primal System

Once computed, the optimal test functions can be used in place of the standard DG test functions, and the primal problem can be solved as usual. Upon doing so, our first inclination would be to use high-order quadrature rules to perform all integrations, due to the high-order nature of the test functions. However, it turns out that high-order quadrature is, for the most part, unnecessary.

For example, for our current problem, the primal equation on a given element Ω_e is (from Eqn. 6.10):

$$\underbrace{\int\limits_{\Omega_e} L^* v_i \, u_H \, dx \; + \; v_i a u_H \Big|_{x_{e,R}}}_{b_e(u_H, v_i)} = \int\limits_{\Omega_e} v_i f \, dx \; + \; v_i a u_{e-1} \Big|_{x_{e,L}} \qquad \forall v_i \in \mathcal{V}_{\text{test}} \tag{6.23}$$

where we assume the v_i have been computed from Eqn. 6.21, and u_{e-1} is the neighbor-element state. But note that, since u_H is contained in the space $\mathcal{U}_{\text{test}}$, Eqn. 6.21 implies that the left-hand side of Eqn. 6.23 can be rewritten as:

$$\underbrace{\int\limits_{\Omega_e} \phi_i \, u_H \, dx \; + \; w_R \phi_i u_H \Big|_{x_{e,R}}}_{J_i(u_H)} = \int\limits_{\Omega_e} v_i f \, dx \; + \; v_i a u_{e-1} \Big|_{x_{e,L}} \qquad \forall v_i \in \mathcal{V}_{\text{test}} \tag{6.24}$$

Thus, for implementation purposes, we can use the above formulation of the primal problem rather than Eqn. 6.23.[7,8] Note that all terms involving the optimal test functions have vanished from the left-hand side, and have been replaced with low-order integrals. Furthermore, in practical cases the source term f on the right-hand side is often zero, so that *all* high-order interior integrals disappear. If this is the case, the optimal test functions appear only in the right-most term of Eqn. 6.24 – i.e. on the upwind boundary. An interesting conclusion is that, for most problems, the accuracy of BDPG rests solely on obtaining accurate test function values on the element boundaries.

[7] A similar idea was proposed in the context of continuous finite elements by Givoli in 1988 [44]

[8] Note that the left-hand side of this equation (i.e. the element self-block of the Jacobian) is symmetric and positive definite. However, the coupling to u_{e-1} via the upwind flux means that (unlike other DPG methods [26]) the global Jacobian is nonsymmetric.

6.6.3 Summary of Method

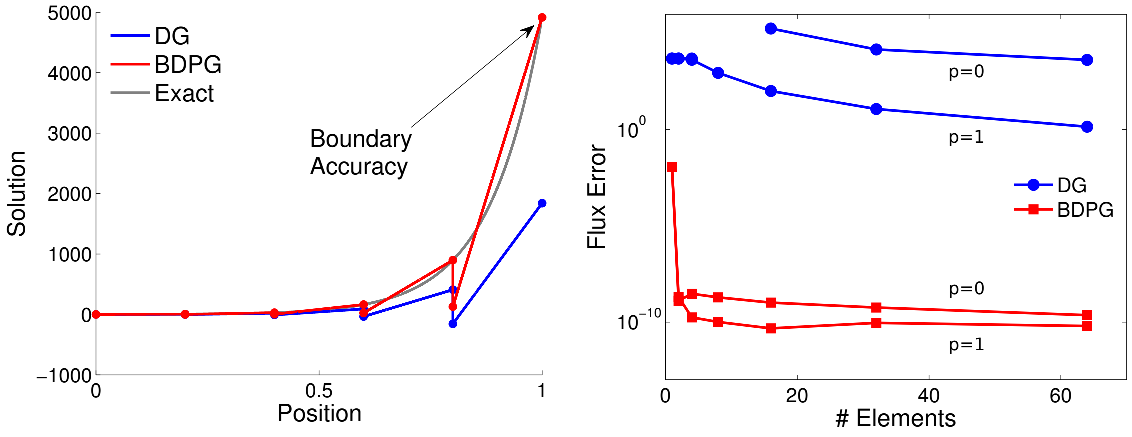
The BDPG method can be summarized as follows:

1. Loop over each element.
2. Compute the local optimal test functions (adjoints) v_i by solving Eqn. 6.22 at order p_{test}, where $p_{\text{test}} \geq p$.
3. Normalize the v_i with respect to (e.g.) a discrete or continuous L^2 norm.
4. Use the v_i in place of the standard DG test functions. To avoid high-order quadrature, the alternate form of the primal problem (Eqn. 6.24) can be used.
5. Obtain accelerated convergence of the boundary fluxes. In 1D, a minimum rate of $p_{\text{test}} + p + 1$ will be obtained, with rates of up to $2p_{\text{test}} + 1$ possible. By choosing $p_{\text{test}} > p$, these rates are necessarily higher than the maximum DG rate of $2p + 1$.

6.7 Results (One-Dimensional Example)

With the above procedures established, we are ready to solve a multi-element problem. As an example, we solve the advection-reaction equation with $a = 1$, $c = -8.5$, and $f = 0$. To see if BDPG obtains boundary accuracy, we choose the boundary weight in the error norm to be high – in this case, taking it to be $w_R = 10^{12}$ – and choose the test space order to be $p_{\text{test}} = 10$. (Note again that we orthonormalize the test functions after computation, so that the large w_R value does not lead to poor conditioning of the primal problem.)

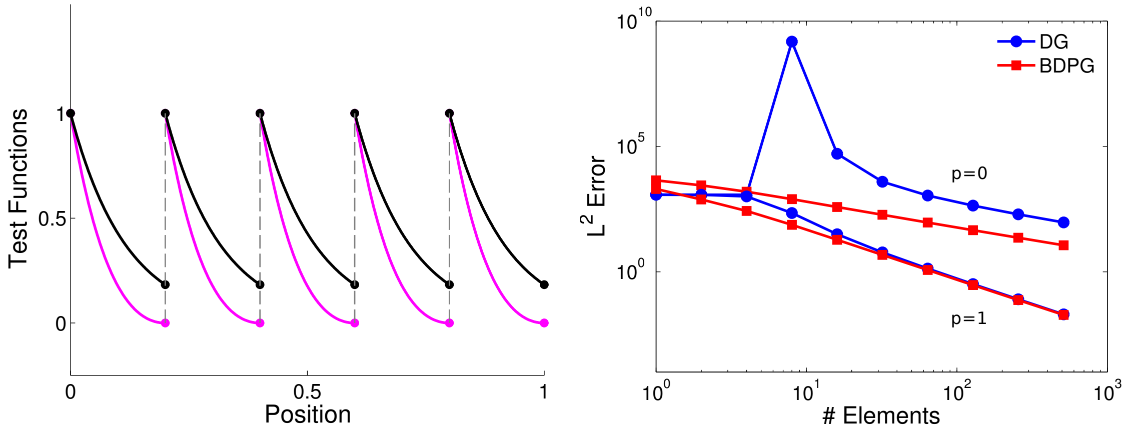
From Fig. 6.3, we see that BDPG achieves boundary errors approximately 10 orders of magnitude lower than standard DG. This is encouraging, and confirms that our purely local test space (shown in Fig. 6.4) is capable of achieving global optimality. Note that the initial convergence rate of the BDPG fluxes (which is observed to be $2p_{\text{test}} + 1$) is due solely to the inexact representation of the test functions, and that if analytical test functions were used, exact boundary fluxes would be obtained. Finally, while the primary goal of BDPG is to achieve boundary accuracy, from Fig. 6.4 we see that it also performs well in an L^2 sense, exhibiting none of the preasymptotic behavior seen with standard DG.



(a) Numerical solutions

(b) Boundary flux error convergence

Figure 6.3: One-dimensional advection-reaction: (a) $p = 1$ solutions for DG and BDPG on a 5-element mesh. (b) The error in the right-boundary flux for $p = 0$ and $p = 1$ runs. The BDPG fluxes converge at a rate of $2p_{\text{test}} + 1$ and quickly attain machine precision accuracy.



(a) Localized optimal test functions

(b) L^2 error convergence

Figure 6.4: One-dimensional advection-reaction: (a) Localized optimal test functions for the 5-element BDPG solution shown in Fig. 6.3a. The two test functions on each element correspond to the two $p = 1$ trial bases. (b) L^2 error convergence for $p = 0$ and $p = 1$ DG and BDPG solutions. The L^2 performance of the methods is similar, with BDPG showing greater stability on coarse meshes.

6.8 General Theory for Multi-Dimensional Systems

So far, we have derived the optimal test functions in the setting of a one-dimensional advection-reaction problem. However, the relevant concepts extend naturally to systems of equations, as well as to multiple dimensions. We will describe those extensions here. To simplify the presentation, we will again assume that the domain Ω consists of a single element.

A general steady-state conservation law in multiple dimensions can be written as

$$\nabla \cdot \vec{\mathbf{F}}(\mathbf{u}, \vec{\mathbf{q}}) = \mathbf{0}, \tag{6.25}$$

$$\vec{\mathbf{q}} - \nabla \mathbf{u} = \vec{\mathbf{0}}\,, \tag{6.26}$$

where $\mathbf{u}$ is the state vector and $\vec{\mathbf{q}}$ represents the gradient of the state. $\vec{\mathbf{F}}$ is a flux vector, which may contain both advective and diffusive components, and consists of r state components in dim dimensions. (Note that boldface indicates a state vector, while an arrow indicates a spatial vector.) In general, a source term $\mathbf{S}(\mathbf{u})$ could be added to Eqn. 6.25, though we omit it for brevity here. Furthermore, we assume that $\vec{\mathbf{q}}$ is an independent unknown, since this is the case for the hybridized method presented later. However, this is not critical to the theory.

To obtain the weak form of the above problem, we weight Eqns. 6.25 and 6.26 by test functions $\mathbf{v}$ and $\vec{\boldsymbol{\tau}}$, respectively, giving a total weighted residual (upon summation) of

$$\mathcal{R} \equiv \int_{\Omega} \vec{\boldsymbol{\tau}}^T \cdot (\vec{\mathbf{q}} - \nabla \mathbf{u})\; d\Omega + \int_{\Omega} \mathbf{v}^T (\nabla \cdot \vec{\mathbf{F}})\, d\Omega \;=\; 0\,. \tag{6.27}$$

Note that this residual is just a scalar value. We next integrate both terms in Eqn. 6.27 by parts, giving

$$\mathcal{R} \;=\; \int_{\Omega} \vec{\boldsymbol{\tau}}^T \cdot \vec{\mathbf{q}}\, d\Omega \;+ \int_{\Omega} \nabla \cdot \vec{\boldsymbol{\tau}}^T\, \mathbf{u}\, d\Omega \;- \int_{\Omega} (\nabla \mathbf{v})^T \cdot \vec{\mathbf{F}}\, d\Omega \;+ \int_{\partial\Omega} \mathbf{v}^T (\vec{\mathbf{F}} \cdot \vec{n})\, ds$$

$$- \int_{\partial\Omega} (\vec{\boldsymbol{\tau}} \cdot \vec{n})^T\, \mathbf{u}\, ds \;= 0. \tag{6.28}$$

If we now assume Dirichlet boundary conditions (denoted by $\mathbf{u}_B$), the right-most term above becomes a "known" value and can be moved to the right-hand side. After making this change and for convenience defining $\hat{\mathbf{F}} = \vec{\mathbf{F}} \cdot \vec{n}$, we obtain:

$$\underbrace{\int\limits_{\Omega} \vec{\boldsymbol{\tau}}^T \cdot \vec{\mathbf{q}}\, d\Omega + \int\limits_{\Omega} \nabla \cdot \vec{\boldsymbol{\tau}}^T \mathbf{u}\, d\Omega - \int\limits_{\Omega} (\nabla \mathbf{v})^T \cdot \vec{\mathbf{F}}\, d\Omega + \int\limits_{\partial\Omega} \mathbf{v}^T \hat{\mathbf{F}}\; ds}_{b\,(\mathbf{u}, \vec{\mathbf{q}}, \mathbf{v}, \vec{\boldsymbol{\tau}})} = \underbrace{\int\limits_{\partial\Omega} (\vec{\boldsymbol{\tau}} \cdot \vec{n})^T \mathbf{u}_B\, ds}_{l\,(\mathbf{v}, \vec{\boldsymbol{\tau}})} . \tag{6.29}$$

From this equation, we are able to define the bilinear form $b\,(\mathbf{u}, \vec{\mathbf{q}}, \mathbf{v}, \vec{\boldsymbol{\tau}})$.

Next, as in one dimension (i.e. Eqn. 6.9), we would like to write this bilinear form as a product of the state variables and the adjoint operator applied to the test functions. In order to do this, we must first write all domain integrals explicitly in terms of $\mathbf{u}$ and $\vec{\mathbf{q}}$. To start, we rewrite the flux $\vec{\mathbf{F}}$, assumed linear, as

$$\vec{\mathbf{F}} = \frac{\partial \vec{\mathbf{F}}}{\partial \mathbf{u}} \mathbf{u} + \frac{\partial \vec{\mathbf{F}}}{\partial \mathbf{q}_j} \mathbf{q}_j\, , \tag{6.30}$$

where summation over the spatial dimension j is implied.[9] We now substitute this expression (Eqn. 6.30) into Eqn. 6.29 and transpose the first three terms, giving

$$\begin{aligned} b \;=\; & \int\limits_{\Omega} \vec{\mathbf{q}}^T \cdot \vec{\boldsymbol{\tau}}\, d\Omega + \int\limits_{\Omega} \mathbf{u}^T \nabla \cdot \vec{\boldsymbol{\tau}}\, d\Omega \\ & - \int\limits_{\Omega} \left(\mathbf{u}^T \left[\frac{\partial \vec{\mathbf{F}}}{\partial \mathbf{u}} \right]^T + \mathbf{q}_j^T \left[\frac{\partial \vec{\mathbf{F}}}{\partial \mathbf{q}_j} \right]^T \right) \cdot (\nabla \mathbf{v})\; d\Omega + \int\limits_{\partial\Omega} \mathbf{v}^T \hat{\mathbf{F}}\; ds\, . \end{aligned} \tag{6.31}$$

Grouping the $\mathbf{u}$ and $\vec{\mathbf{q}}$ terms then results in

$$\begin{aligned} b \;=\; & \int\limits_{\Omega} \mathbf{q}_j^T \underbrace{\left(\boldsymbol{\tau}_j - \left[\frac{\partial \vec{\mathbf{F}}}{\partial \mathbf{q}_j} \right]^T \cdot \nabla \mathbf{v} \right)}_{L^*_{q,j}(\vec{\boldsymbol{\tau}}, \mathbf{v})} d\Omega \\ & + \int\limits_{\Omega} \mathbf{u}^T \underbrace{\left(\nabla \cdot \vec{\boldsymbol{\tau}} - \left[\frac{\partial \vec{\mathbf{F}}}{\partial \mathbf{u}} \right]^T \cdot \nabla \mathbf{v} \right)}_{L^*_{u}(\vec{\boldsymbol{\tau}}, \mathbf{v})} d\Omega + \int\limits_{\partial\Omega} \mathbf{v}^T \hat{\mathbf{F}}\; ds\, . \end{aligned} \tag{6.32}$$

[9] Note that for nonlinear problems, a similar expression would hold for the Fréchet linearization of the flux.

Next, if we define group variables (denoted by a tilde) for the states and test functions as

$$\tilde{\mathbf{u}} \equiv \begin{bmatrix} \mathbf{q}_j \\ \mathbf{u} \end{bmatrix} \qquad \text{and} \qquad \tilde{\mathbf{v}} \equiv \begin{bmatrix} \boldsymbol{\tau}_j \\ \mathbf{v} \end{bmatrix}, \tag{6.33}$$

and we define the ***adjoint*** operator L^* (based on the operators $L^*_{q,j}$ and L^*_u in Eqn. 6.32) as

$$L^* \equiv \begin{bmatrix} L^*_{q,j} \\ \\ L^*_u \end{bmatrix} = \begin{bmatrix} I(\,) & -\left[\dfrac{\partial \vec{\mathbf{F}}}{\partial \mathbf{q}_j}\right]^T \cdot \nabla(\,) \\ \\ \partial_j(\,) & -\left[\dfrac{\partial \vec{\mathbf{F}}}{\partial \mathbf{u}}\right]^T \cdot \nabla(\,) \end{bmatrix}, \tag{6.34}$$

then Eqn. 6.32 can be rewritten in the following form:

$$b(\tilde{\mathbf{u}}, \tilde{\mathbf{v}}) \;=\; \int\limits_{\Omega} \tilde{\mathbf{u}}^T \,(\, L^* \tilde{\mathbf{v}}\,)\; d\Omega + \int\limits_{\partial\Omega} \mathbf{v}^T\, \hat{\mathbf{F}}(\tilde{\mathbf{u}})\; ds \;=\; l\,(\tilde{\mathbf{v}}) \qquad \forall \tilde{\mathbf{v}} \in \tilde{\mathcal{V}}, \tag{6.35}$$

where the operator L^* acts as a matrix on the group variable $\tilde{\mathbf{v}}$. Note that the above $b(\tilde{\mathbf{u}}, \tilde{\mathbf{v}})$ is essentially the same as the one-dimensional bilinear form (i.e. Eqn. 6.9), except that the states and test functions are now vectors, and we have a general flux $\hat{\mathbf{F}}$ on the boundary rather than the one-dimensional flux au. Thus, from this point on, the development will parallel the one-dimensional theory.

To approximate the above equation, a DG method chooses a set of discrete states and test functions, $\tilde{\mathbf{u}}_H$ and $\tilde{\mathbf{v}}_h$, as well as a numerical flux $\hat{\mathbf{F}}(\tilde{\mathbf{u}}_H)$, resulting in the discrete bilinear form:

$$b(\tilde{\mathbf{u}}_H, \tilde{\mathbf{v}}_h) \;=\; \int\limits_{\Omega} \tilde{\mathbf{u}}_H^T \,(\, L^* \tilde{\mathbf{v}}_h\,)\; d\Omega + \int\limits_{\partial\Omega} \mathbf{v}_h^T\, \hat{\mathbf{F}}(\tilde{\mathbf{u}}_H)\; ds \;=\; l\,(\tilde{\mathbf{v}}_h) \qquad \forall \tilde{\mathbf{v}}_h \in \tilde{\mathcal{V}}_h\,. \tag{6.36}$$

Once a basis is chosen for the trial space representations of $\mathbf{u}_H$ and $\vec{\mathbf{q}}_H$, these states can be expanded as

$$u_{H,s} = \sum_{m=1}^{n_U} U_{s,m}\, \phi_{s,m}(\vec{x}) \quad \text{and} \quad q_{H,s,d} = \sum_{m=1}^{n_Q} Q_{s,d,m}\, \phi_{s,d,m}(\vec{x})\,. \tag{6.37}$$

Here, s indexes the state component (ranging from 1 to the state rank, r), m indexes the basis function (ranging from 1 to the number of nodes, n_U or n_Q), and d indexes the dimension (ranging from 1 to dim). Finally, $U_{s,m}$ and $Q_{s,d,m}$ represent the unknown solution coefficients, the total number of which is given by $N \equiv N_U + N_Q = r\, n_U \, + \, r\, n_Q \cdot dim$.

The remaining task is to define the test space. In order to derive the optimal test space, we follow a similar strategy as before: we first define an error norm we wish to minimize, then choose the test functions such that the bilinear form reduces to the *derivative* of that norm. To help determine an appropriate error norm to minimize, we first choose a common test function $\tilde{\mathbf{v}}_h$ for Eqns. 6.35 and 6.36. This allows us to equate (and subtract) the exact and discrete bilinear forms, resulting in the equation

$$b(\tilde{\mathbf{e}}, \tilde{\mathbf{v}}_h) = 0\,, \tag{6.38}$$

where $\tilde{\mathbf{e}}$ is a group variable representing the errors in the states. If we now set $\tilde{\mathbf{v}}_h = \tilde{\mathbf{v}}_i$, where i ranges from 1 to N, then writing out Eqn. 6.38 in a form similar to Eqn. 6.32 gives:

$$\begin{aligned} b(\tilde{\mathbf{e}}, \tilde{\mathbf{v}}_i) \; = \; 0 \; = \int\limits_{\Omega} (\mathbf{q}_{H,j} - \mathbf{q}_j)^T \; L^*_{q,j}\,(\tilde{\mathbf{v}}_i)\, d\Omega + \int\limits_{\Omega} (\mathbf{u}_H - \mathbf{u})^T \; L^*_u\,(\tilde{\mathbf{v}}_i)\, d\Omega \\ + \int\limits_{\partial\Omega} \mathbf{v}_i^T \, \left[\hat{\mathbf{F}}(\mathbf{u}_H, \vec{\mathbf{q}}_H) - \hat{\mathbf{F}}(\mathbf{u}, \vec{\mathbf{q}})\right] \; ds\,. \end{aligned} \tag{6.39}$$

This equation is satisfied regardless of how the test space is chosen. However, when using optimal test functions, we would like this expression to represent the minimization of a certain error norm. It is important to note that, due to the form of $b(\tilde{\mathbf{e}}, \tilde{\mathbf{v}}_i)$, only certain types of error norm are valid to minimize. In other words, we must be careful to select an error norm whose derivative it is *possible* to represent by $b(\tilde{\mathbf{e}}, \tilde{\mathbf{v}}_i)$. To that end, we propose minimizing the following norm:

$$\boxed{\begin{aligned} ||\,\tilde{\mathbf{e}}\,||^2 \;=\; & \underbrace{\sum_{s=1}^{r}\sum_{d=1}^{dim}\int\limits_{\Omega}(q_{H,s,d}-q_{s,d})^2\;d\Omega}_{\text{interior } q \text{ accuracy}} \;+\; \underbrace{\sum_{s=1}^{r}\int\limits_{\Omega}(u_{H,s}-u_s)^2\;d\Omega}_{\text{interior } u \text{ accuracy}} \\ & \qquad\qquad + \underbrace{\sum_{s=1}^{r} w_s \int\limits_{\partial\Omega}\left[\hat{F}_s(\mathbf{u}_H,\vec{\mathbf{q}}_H)-\hat{F}_s(\mathbf{u},\vec{\mathbf{q}})\right]^2 ds}_{\text{boundary flux accuracy}} \end{aligned}} \tag{6.40}$$

This norm contains errors in the state, its gradients, and the boundary flux – all of which are present in the above $b(\tilde{\mathbf{e}}, \tilde{\mathbf{v}}_i)$. Furthermore, with this norm, we see that choosing the weights w_s to be large emphasizes accuracy in the boundary fluxes, which, as in one dimension, is our ultimate goal.

Finally, note that the boundary fluxes in the above equation can correspond to *any* type of physical boundary – e.g. an outflow boundary, a wall boundary, etc. Regardless of the type of boundary, the method will seek to minimize the error in the particular information flowing *from* the domain interior *out* of that boundary. For a true outflow boundary, this information may be the fluid itself, while for a wall boundary it may be pressure or heat flux information.

If we are to minimize the above norm, we need its derivatives with respect to both the $U_{k,m}$ and $Q_{k,d,m}$ coefficients to be zero. Thus, we need

$$\begin{aligned} \frac{1}{2}\frac{\partial||\,\tilde{\mathbf{e}}\,||^2}{\partial U_{k,m}} \;=\; 0 \;=\; & \int\limits_{\Omega}(u_{H,k}-u_k)\,\phi_{k,m}\;d\Omega \\ & + \sum_{s=1}^{r} w_s \int\limits_{\partial\Omega}\left[\hat{F}_s(\mathbf{u}_H,\vec{\mathbf{q}}_H)-\hat{F}_s(\mathbf{u},\vec{\mathbf{q}})\right]\frac{\partial\hat{F}_s}{\partial u_{H,k}}\,\phi_{k,m}\,ds \end{aligned} \tag{6.41}$$

and

$$\begin{aligned} \frac{1}{2}\frac{\partial||\,\tilde{\mathbf{e}}\,||^2}{\partial Q_{k,d,m}} \;=\; 0 \;=\; & \int\limits_{\Omega}(q_{H,k,d}-q_{k,d})\,\phi_{k,d,m}\;d\Omega \\ & + \sum_{s=1}^{r} w_s \int\limits_{\partial\Omega}\left[\hat{F}_s(\mathbf{u}_H,\vec{\mathbf{q}}_H)-\hat{F}_s(\mathbf{u},\vec{\mathbf{q}})\right]\frac{\partial\hat{F}_s}{\partial q_{H,k,d}}\,\phi_{k,d,m}\,ds\,. \end{aligned} \tag{6.42}$$

For these equations to be satisfied by our finite element method, we must choose the test functions $\tilde{\mathbf{v}}_i$ such that the bilinear form $b(\tilde{\mathbf{e}}, \tilde{\mathbf{v}}_i)$ reduces to them. A given $\tilde{\mathbf{v}}_i$ will then ensure that *one* of the above equations is satisfied. Since Eqns. 6.41 and 6.42 represent N derivative equations altogether, with N test functions (i.e. a square system) we can ensure that each of them is satisfied in turn.

By comparing $b(\tilde{\mathbf{e}}, \tilde{\mathbf{v}}_i)$ (Eqn. 6.39) to Eqn. 6.41, we see that to make these expressions identical the test functions must satisfy:

$$\boxed{ i = 1 \dots N_U \quad \begin{cases} L^*_{q,j}(\tilde{\mathbf{v}}_i) \ = \ \mathbf{0} & j = 1 \dots dim \quad x \in \Omega \\[2ex] L^*_{u,s}(\tilde{\mathbf{v}}_i) \ = \ \phi_{k,m}\, \delta_{s,k} & s = 1 \dots r \quad\;\; x \in \Omega \\[2ex] v_{i,s} \ = \ w_s \, \dfrac{\partial \hat{F}_s}{\partial u_{H,k}} \, \phi_{k,m} & s = 1 \dots r \quad\;\; x \in \partial\Omega \end{cases} } \tag{6.43}$$

Here, $\delta_{s,k}$ denotes the Kronecker delta function, $L^*_{u,s}$ denotes the sth component (i.e. equation) associated with the operator L^*_u, and repeated indices do not imply summation. As before, we see that the optimal test functions satisfy adjoint equations in which the trial bases appear as source terms on the right-hand side. The above equations are solved for each u basis function $\phi_{k,m}$, with the test function index i enumerating all combinations of (k, m). Since $1 \leq k \leq r$ and $1 \leq m \leq n_U$, there are a total of $N_U = r\, n_U$ basis functions altogether, which provides, in the end, a corresponding N_U test functions.

Next, to make $b(\tilde{\mathbf{e}}, \tilde{\mathbf{v}}_i)$ reduce to Eqn. 6.42, we see that the remaining test functions should satisfy:

$$\boxed{ i = 1 \dots N_Q \quad \begin{cases} L^*_{q,j,s}(\tilde{\mathbf{v}}_i) \ = \ \phi_{k,d,m}\, \delta_{j,d}\, \delta_{s,k} & j, s = 1 \dots dim, r \quad x \in \Omega \\[2ex] L^*_{u}(\tilde{\mathbf{v}}_i) \ = \ \mathbf{0} & \qquad\qquad\qquad\quad x \in \Omega \\[2ex] v_{i,s} \ = \ w_s \, \dfrac{\partial \hat{F}_s}{\partial q_{H,k,d}} \, \phi_{k,d,m} & s = 1 \dots r \qquad\quad\; x \in \partial\Omega \end{cases} } \tag{6.44}$$

This set of equations is solved for each q trial basis $\phi_{k,d,m}$, with the test function index i enumerating all combinations of (k, d, m), where $1 \leq k \leq r$, $1 \leq d \leq dim$, $1 \leq m \leq n_Q$. The result is an additional $N_Q = r\, n_Q \cdot dim$ test functions, for a total of $N_U + N_Q = N$. When used in place of the standard Galerkin test functions, these optimal test functions ensure that Eqns. 6.41 and 6.42 are satisfied, and hence that the error in Eqn. 6.40 is minimized.

Finally, as in one dimension, the optimal test functions can be interpreted as adjoint solutions for certain "projection" outputs. These outputs are closely related to the error norm derivatives. By inspection of Eqns. 6.41 and 6.42, we can write the effective outputs as

$$J^u_{k,m} \;=\; \int\limits_{\Omega} u_k\, \phi_{k,m}\, d\Omega \;+\; \sum_{s=1}^{r} w_s \int\limits_{\partial\Omega} \hat{F}_s(\mathbf{u}, \vec{\mathbf{q}})\, \frac{\partial \hat{F}_s}{\partial u_{H,k}}\, \phi_{k,m}\, ds \tag{6.45}$$

and

$$J^q_{k,d,m} \;=\; \int\limits_{\Omega} q_{k,d}\, \phi_{k,d,m}\, d\Omega \;+\; \sum_{s=1}^{r} w_s \int\limits_{\partial\Omega} \hat{F}_s(\mathbf{u}, \vec{\mathbf{q}})\, \frac{\partial \hat{F}_s}{\partial q_{H,k,d}}\, \phi_{k,d,m}\, ds\,. \tag{6.46}$$

It is easy to verify that enforcing zero error in these outputs is equivalent to enforcing zero derivative of $||\, \tilde{\mathbf{e}}\, ||^2$ – which of course is the actual goal.

6.8.1 Localization and Adjoint Consistency

As in 1D, the above test functions satisfy global adjoint equations. However, if we choose the flux weights w_s to be large, we can again localize the adjoint problems to individual elements, since accuracy in the local fluxes will propagate globally. Thus, for multi-element meshes, the outputs J^u and J^q are defined over each element in turn, and the $\hat{F}_s$ terms are chosen to reflect the fluxes on a given element's boundaries.

Specifically, to define the adjoint problems for a domain-interior element, the neighbor-element states are treated as local Dirichlet conditions, and the fluxes $\hat{F}_s$ are defined as if a single-element DG or HDG problem were being solved. Since there is no physical boundary condition on interior faces, the convective part of $\hat{F}_s$ is just taken to be a Roe flux [82] between the element and the neighbor states. This Roe flux ensures that information is properly upwinded and that the local adjoint problems remain well-posed.[10]

[10] For these local problems, use of a non-upwinding flux such as local Lax-Friedrichs would result in an adjoint inconsistency [65, 48], and would lead to oscillations in (and suboptimal performance of) the test functions.

Likewise, for an element with a domain-boundary face, the flux $\hat{F}_s$ is defined as usual for a DG or HDG method – for example, the convective flux is just the analytical flux function evaluated with a corresponding boundary state. Furthermore, if the boundary condition specifies a certain flux component directly (e.g. if it is a wall boundary, where zero mass flux is specified), then there is no need to request accuracy in this flux component, and it should be removed from the outputs J^u and J^q. This removal occurs automatically when using a discrete adjoint approach, since the discrete $\partial \hat{F}_s / \partial u_H$ and $\partial \hat{F}_s / \partial q_H$ will be zero and will hence vanish from both the output definitions and the residual Jacobian.

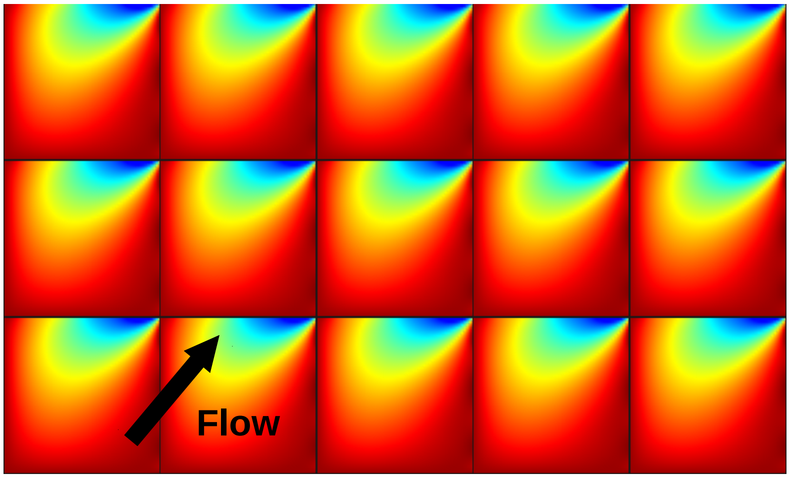


Figure 6.5: Optimal test functions corresponding to the q_y (gradient in the vertical direction) trial basis in the upper-right corner of each element. Blue corresponds to a large value, while red is near zero. These test functions ensure that accuracy in the top flux of each element is obtained. Similar test functions ensure accuracy in the remaining fluxes. Note the upwinding nature of the test functions.

Fig. 6.5 shows a set of localized optimal test functions corresponding to a low Reynolds number advection-diffusion problem. The "upwinding" nature of the optimal test functions is apparent, and we see that their role is to ensure that the fluxes on a given element face have the proper domain of dependence *within* that element.

6.8.2 Boundary Enrichment of Trial Space

Localization of the test functions has obvious advantages: it makes their computation relatively inexpensive and their application within an existing method straightforward. However, in multiple dimensions, localization is – in some ways – a double-edged sword. By using localized test functions, we are giving up certain global properties of the test space, and focusing all of our attention on achieving accuracy in the interface fluxes. And while it is true that *if* these local fluxes can be made accurate, then this accuracy will propagate globally, it is also true that if these fluxes *cannot* be made accurate, then this *inaccuracy* will propagate globally.

In one dimension, the fluxes are just scalar values, since the boundaries of a given element are zero-dimensional. Thus, if the optimal test functions are well-represented, the flux errors can be driven to zero regardless of the trial space. However, in two dimensions, the fluxes themselves are one-dimensional profiles along the element boundaries. In this case, the pointwise errors in the fluxes cannot in general be driven to zero: their magnitude depends inevitably on the *trial* space resolution near element boundaries. For example, if the exact fluxes happen to be quadratic along a given face, but the trial space is linear, then $O(h^2)$ errors in the fluxes will necessarily remain. These flux errors will then propagate globally, and, in many cases, would render methods using localized optimal test functions no better than standard DG methods.

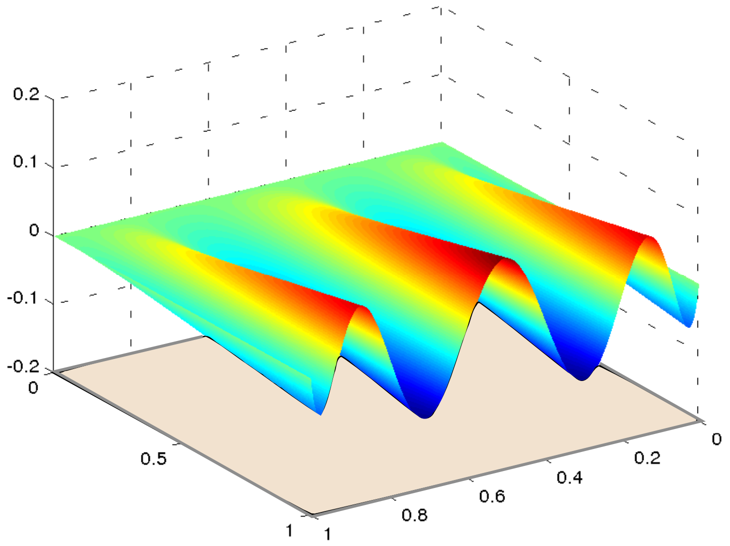


Figure 6.6: An eighth-order Lobatto function defined along an edge of a quadrilateral reference element. These functions are added to the trial space to improve flux resolution.

Thus, if optimal test functions are to provide a benefit in multiple dimensions, not only must the test functions *request* accuracy in the fluxes, but the trial space must be capable of *providing* that accuracy. To ensure that the latter requirement is satisfied, in multiple dimensions the trial space can be enriched near element boundaries. In the current work, this is done by adding high-order one-dimensional Lobatto functions [91] along the element faces, which are then blended linearly into the element interior. Fig. 6.6 shows an example of a blended eighth-order Lobatto function defined on a single edge of a reference quadrilateral.

In the results section to follow, this is the enrichment strategy used. Note that we keep the interior interpolation order, p_I, at a (low) value of 1, and then enrich this $p_I = 1$ space with Lobatto functions of some higher order, p_B. The combination of linear interior basis with order-p_B Lobatto functions ensures that the trial space on element boundaries spans a full order-p_B space. In the end, this is equivalent to using a standard order-p_B hierarchical basis, but with all interior modes removed.

6.8.3 Benefit in Multiple Dimensions

Before moving on, it is worth pausing to reflect on the potential benefit of optimal test functions in multiple dimensions. The primary advantage is that, by requiring trial space resolution only near element boundaries, a BDPG scheme requires fewer degrees of freedom than a DG method. For example, on a two-dimensional quadrilateral mesh, BDPG requires $4p_B$ degrees of freedom on each element, whereas for a similar level of accuracy DG would require $(p_B + 1)^2$. Thus, the number of degrees of freedom scales as p^{dim} for DG methods, but as p^{dim-1} for BDPG. In a sense then, BDPG may be viewed as a form of "hybridization," since hybridization of a standard DG scheme results in a similar reduction in the number of globally coupled unknowns.

However, even within an existing hybrid framework (such as HDG), optimal test functions can still provide a benefit. Since BDPG requires trial space resolution only near element boundaries, it opens up the possibility of performing a targeted trial space optimization in those regions. For example, if the trial space were tuned to include the primary "modes" of the true interface fluxes, then hybridized BDPG schemes could significantly outperform standard HDG schemes. As a step in this direction – and to show that optimal test functions can be used within a hybrid framework – in the following sections we present a hybridized BDPG method.

6.9 A Hybridized BDPG Method

In this section, we first give a brief overview of hybridized discontinuous Galerkin (HDG) methods [77, 75, 31]. We then describe how these methods can be modified to incorporate optimal test functions, resulting in a hybridized BDPG scheme. The primary advantage of hybridized methods is that, by introducing new unknowns (which we call $\hat{\mathbf{u}}$) on element interfaces, they decouple elements during the linear solve and result in a smaller global system than DG (for sufficiently high order, p [77]). An illustration of the primary differences between DG and HDG is provided in Fig. 6.7.

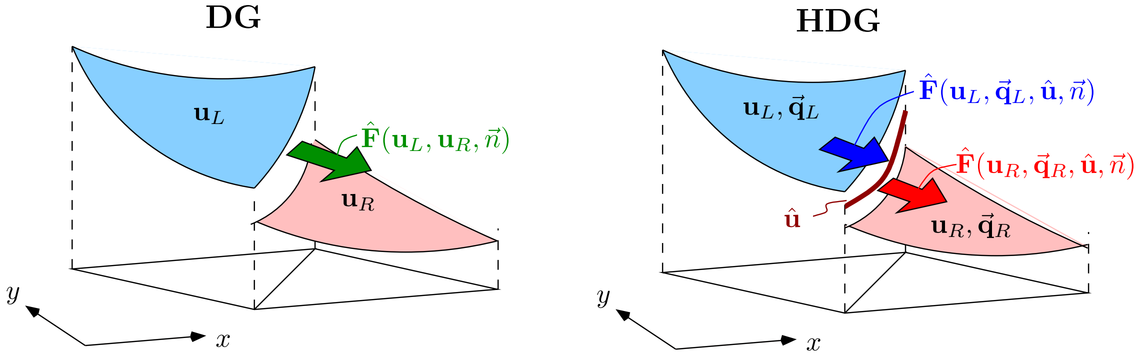

DG
$\mathbf{u}_L$
$\hat{\mathbf{F}}(\mathbf{u}_L, \mathbf{u}_R, \vec{n})$
$\mathbf{u}_R$
y
x
HDG
$\mathbf{u}_L, \vec{\mathbf{q}}_L$
$\hat{\mathbf{F}}(\mathbf{u}_L, \vec{\mathbf{q}}_L, \hat{\mathbf{u}}, \vec{n})$
$\hat{\mathbf{F}}(\mathbf{u}_R, \vec{\mathbf{q}}_R, \hat{\mathbf{u}}, \vec{n})$
$\hat{\mathbf{u}}$
$\mathbf{u}_R, \vec{\mathbf{q}}_R$
y
x

Figure 6.7: In the HDG method, introducing additional $\hat{\mathbf{u}}$ unknowns on element interfaces allows for elimination of the element-interior unknowns during the global solve. This results in a global system in which the number of unknowns scales as p^{dim-1} instead of p^{dim} (as for DG).

6.9.1 HDG Discretization

While the HDG method can be applied to general nonlinear systems, here we consider a linear steady-state problem written as a first-order system:

$$\vec{\mathbf{q}} - \nabla \mathbf{u} = \vec{\mathbf{0}}, \tag{6.47}$$

$$\nabla \cdot \vec{\mathbf{F}}(\mathbf{u}, \vec{\mathbf{q}}) = \mathbf{0}, \tag{6.48}$$

where $\mathbf{u}$ is the state, $\vec{\mathbf{q}}$ is the state gradient, and $\vec{\mathbf{F}}$ is the conservative flux. We assume that a second-order (diffusive) term is present in the governing equations. If not, then the first equation is not required, but the derivation is otherwise identical. Weighting the above equations with test functions $\vec{\boldsymbol{\tau}}_H$ and $\mathbf{v}_H$ (respectively), discretizing, and integrating by parts over an element Ω_e then gives:

$$\mathcal{R}_H^Q \equiv \int\limits_{\Omega_e} \vec{\boldsymbol{\tau}}_H^T \cdot \vec{\mathbf{q}}_H \, d\Omega + \int\limits_{\Omega_e} \nabla \cdot \vec{\boldsymbol{\tau}}_H^T \mathbf{u}_H \, d\Omega - \int\limits_{\partial\Omega_e} \left(\vec{\boldsymbol{\tau}}_H^T \cdot \vec{n}\right) \; \hat{\mathbf{u}}_H \, ds = 0 \quad \forall \vec{\boldsymbol{\tau}}_H \in [\boldsymbol{\mathcal{V}}_H]^{dim} \tag{6.49}$$

$$\mathcal{R}_H^U \equiv -\int\limits_{\Omega_e} \nabla \mathbf{v}_H^T \cdot \vec{\mathbf{F}} \, d\Omega + \int\limits_{\partial\Omega_e} \mathbf{v}_H^T \, \hat{\mathbf{F}}_L \, ds = 0 \qquad \forall \mathbf{v}_H \in \boldsymbol{\mathcal{V}}_H \, . \tag{6.50}$$

Here, $\boldsymbol{\mathcal{V}}_H \equiv [\mathcal{V}_H]^s$, where $\mathcal{V}_H$ is typically taken as

$$\mathcal{V}_H = \{u \in L^2(\Omega) \, : \, u|_{\Omega_e} \in \mathcal{P}^p \quad \forall \Omega_e \in T_H\} \, .$$

The unknowns $\mathbf{u}_H \in \boldsymbol{\mathcal{V}}_H$ and $\vec{\mathbf{q}}_H \in [\boldsymbol{\mathcal{V}}_H]^{\mathrm{dim}}$ are approximated as polynomials within each element and are allowed to be discontinuous between elements. In addition, note that we have introduced the interface unknown $\hat{\mathbf{u}}_H$ within $\mathcal{R}_H^Q$ (and, although not apparent, within $\mathcal{R}_H^U$ as well through $\hat{\mathbf{F}}_L$), which means that we will need an additional equation to close this system. This additional equation is defined as

$$\mathcal{R}_H^\Lambda \equiv \int\limits_f \boldsymbol{\mu}_H^T \left\{ \hat{\mathbf{F}}_L + \hat{\mathbf{F}}_R \right\} \, ds = 0 \qquad \forall \boldsymbol{\mu}_H \in \boldsymbol{\mathcal{M}}_H . \tag{6.51}$$

Here, f denotes an element face, while $\boldsymbol{\mathcal{M}}_H = [\mathcal{M}_H]^r$, where $\mathcal{M}_H$ is the space of polynomials on the set of interior faces $\mathcal{E}_H$, i.e.

$$\mathcal{M}_H = \{\mu \in L^2(\mathcal{E}_H) \, : \, \mu|_f \in \mathcal{P}^p(f) \quad \forall f \in \mathcal{E}_H\} \, . \tag{6.52}$$

The above $\mathcal{R}_H^\Lambda$ equation (Eqn. 6.51) is a weak flux continuity statement required to close the system (and in so doing, enforce conservation), since the fluxes on either side of an interface f need not match pointwise. These "one-sided" fluxes are defined as

$$\hat{\mathbf{F}}_L \;=\; \vec{\mathbf{F}} \left(\hat{\mathbf{u}}_H, \vec{\mathbf{q}}_{H,L}\right) \cdot \vec{n}_L + \underline{\underline{\mathbf{S}}} \left(\hat{\mathbf{u}}_H\right) \left(\mathbf{u}_{H,L} - \hat{\mathbf{u}}_H\right) \tag{6.53}$$

and likewise for $\hat{\mathbf{F}}_R$, where the L and R indicate the side of a given face with which the flux is associated. (Note that in Eqn. 6.50 we arbitrarily designate the "left" side as that lying within element Ω_e, and define $\vec{n}_L$ to be the outward-pointing normal to Ω_e.) From the definition of the flux in Eqn. 6.53, we see that $\hat{\mathbf{F}}_L$ communicates only with the element-interior state $\mathbf{u}_{H,L}$ and the interface state $\hat{\mathbf{u}}_H$. It does not directly communicate with the state on the neighboring element, $\mathbf{u}_{H,R}$, and is for this reason termed "one-sided."

Finally, the $\underline{\underline{\mathbf{S}}}$ term in the above expression is a stabilization tensor, which to obtain a Roe-like flux can be chosen as

$$\underline{\underline{\mathbf{S}}} = \underline{\underline{\mathbf{R}}}\,|\underline{\underline{\mathbf{\Lambda}}}|\,\underline{\underline{\mathbf{L}}} + \tau_{\text{visc}}\,\underline{\underline{\mathbf{I}}}\,. \tag{6.54}$$

Here, $\tau_{\text{visc}} = \nu/\ell_{\text{visc}}$ includes the viscosity ν and a user-specified viscous length scale ℓ_{visc}, while the matrices $\underline{\underline{\mathbf{R}}}$, $\underline{\underline{\mathbf{\Lambda}}}$, and $\underline{\underline{\mathbf{L}}}$ come from an eigen-decomposition of the convective flux Jacobian (dotted with the normal) evaluated about $\hat{\mathbf{u}}_H$.

The above one-sided fluxes and $\hat{\mathbf{u}}_H$ are defined on all interior faces. For faces on domain boundaries, no $\hat{\mathbf{u}}_H$ is employed, and the one-sided fluxes are instead replaced by a standard boundary flux. As with DG, this boundary flux (which we will call simply $\hat{\mathbf{F}}$) consists of the analytical flux function evaluated with an appropriate boundary state $\mathbf{u}_{H,B}$.[11] Finally, a stabilization tensor $\underline{\underline{\mathbf{S}}}_B = \tau_{\text{visc}}\,\underline{\underline{\mathbf{I}}}$ is also included, so that the total boundary flux is given by

$$\hat{\mathbf{F}} \;=\; \vec{\mathbf{F}}\left(\mathbf{u}_B, \vec{\mathbf{q}}_{H,L}\right) \cdot \vec{n}_L + \underline{\underline{\mathbf{S}}}_B\left(\mathbf{u}_{H,L} - \mathbf{u}_{H,B}\right)\,. \tag{6.55}$$

6.9.1.1 Solution of HDG System

After discretizing and inserting known boundary conditions, Eqns. 6.49, 6.50, and 6.51 can be written as:

$$\begin{bmatrix} \vec{\mathbf{R}}^Q \\ \mathbf{R}^U \\ \mathbf{R}^\Lambda \end{bmatrix} = \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix} \begin{bmatrix} \vec{\mathbf{Q}} \\ \mathbf{U} \\ \mathbf{\Lambda} \end{bmatrix} + \begin{bmatrix} \vec{\mathbf{L}}^Q \\ \mathbf{L}^U \\ \mathbf{L}^\Lambda \end{bmatrix} = \begin{bmatrix} \vec{\mathbf{0}} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix}. \tag{6.56}$$

Here, $\vec{\mathbf{Q}}$, $\mathbf{U}$, and $\mathbf{\Lambda}$ are the discrete unknowns in the approximation of $\vec{\mathbf{q}}_H$, $\mathbf{u}_H$, and $\hat{\mathbf{u}}_H$, respectively; $\vec{\mathbf{R}}^Q$, $\mathbf{R}^U$, and $\mathbf{R}^\Lambda$ are the discrete residual vectors; and $\vec{\mathbf{L}}^Q$, $\mathbf{L}^U$, and $\mathbf{L}^\Lambda$ represent any source terms and boundary conditions. The Jacobian matrix, which

[11] In certain cases, e.g. at farfield boundaries, a Roe flux is used rather than the analytical convective flux.

is shown partitioned into four blocks above, can be written explicitly as:

$$\left[\begin{array}{c|c} \mathbf{A} & \mathbf{B} \\ \hline \mathbf{C} & \mathbf{D} \end{array}\right] \equiv \left[\begin{array}{cc|c} \dfrac{\partial \vec{\mathbf{R}}^Q}{\partial \vec{\mathbf{Q}}} & \dfrac{\partial \vec{\mathbf{R}}^Q}{\partial \mathbf{U}} & \dfrac{\partial \vec{\mathbf{R}}^Q}{\partial \boldsymbol{\Lambda}} \\ \dfrac{\partial \mathbf{R}^U}{\partial \vec{\mathbf{Q}}} & \dfrac{\partial \mathbf{R}^U}{\partial \mathbf{U}} & \dfrac{\partial \mathbf{R}^U}{\partial \boldsymbol{\Lambda}} \\ \hline \dfrac{\partial \mathbf{R}^\Lambda}{\partial \vec{\mathbf{Q}}} & \dfrac{\partial \mathbf{R}^\Lambda}{\partial \mathbf{U}} & \dfrac{\partial \mathbf{R}^\Lambda}{\partial \boldsymbol{\Lambda}} \end{array}\right] . \qquad (6.57)$$

The important point is now that, since the HDG system couples elements only indirectly through the interface states $\boldsymbol{\Lambda}$, the matrix $\mathbf{A}$ can be inverted by purely element-local solves. That is, we can ***statically condense*** the element-interior degrees of freedom out of the system to obtain a smaller global system,

$$\underbrace{\left(\mathbf{D} - \mathbf{C}\mathbf{A}^{-1}\mathbf{B}^T\right)}_{\mathbf{K}} \boldsymbol{\Lambda} + \left(\mathbf{L}^\Lambda - \mathbf{C}\mathbf{A}^{-1}\left[\vec{\mathbf{L}}^Q, \mathbf{L}^U\right]^T\right) = \mathbf{0}\,, \qquad (6.58)$$

which can be solved for $\boldsymbol{\Lambda}$. (Note that static condensation amounts to solving for $\vec{\mathbf{Q}}$ and $\mathbf{U}$ in terms of $\boldsymbol{\Lambda}$ using the first row of Eqn. 6.56, then substituting the result into the second row of Eqn. 6.56.) Furthermore, note that for high-p simulations, the dimension of the above $\mathbf{K}$ matrix is smaller than the system for a DG discretization of the same order, so that HDG achieves a reduction in the number of globally coupled unknowns.

In fact, since only the element-face degrees of freedom are involved in the HDG global solve, and since faces represent a one-dimension-lower space than element areas, the number of global degrees of freedom scales like DOF $\sim p^{\mathrm{dim}}$ for DG but only DOF $\sim p^{\mathrm{dim}-1}$ for HDG. The approximate degree-of-freedom counts for DG and

HDG on triangular and quadrilateral elements (per mesh vertex) are shown below.

DOF on Triangles:

	$p=1$	$p=2$	$p=3$	$p=4$
DG	6	12	20	30
HDG	6	9	12	15

DOF on Quadrilaterals:

	$p=1$	$p=2$	$p=3$	$p=4$
DG	4	9	16	25
HDG	4	6	8	10

Finally, note that after solving Eqn. 6.58 for the interface states $\mathbf{\Lambda}$, the interior states $\vec{\mathbf{Q}}$ and $\mathbf{U}$ can be obtained by performing local solves on each element, which represent an "undoing" of the original static condensation.

6.9.2 Optimal Test Function (BDPG) Implementation

Next, we give a brief overview of using optimal test functions within the above HDG framework. First, we note that for single-element problems, Eqn. 6.51 vanishes and the sum of $\mathcal{R}_H^Q$ and $\mathcal{R}_H^U$ (Eqns. 6.49 and 6.50) reduces to the bilinear form in Eqn. 6.29. The test function theory derived in Sec. 6.8 therefore carries over directly, and the optimal test functions make Eqns. 6.49 and 6.50 reduce to the error norm derivatives in Eqns. 6.41 and 6.42, thus minimizing the desired error.

For multi-element problems, localized optimal test functions can be computed as described in Sec. 6.8.1. This amounts to solving the following $N_U + N_Q$ adjoint problems for the test functions $\tilde{\mathbf{v}}_i = [\boldsymbol{\tau}_j \ \mathbf{v}]^T$ on each element Ω_e:

$$b_e\left(\delta\tilde{\mathbf{u}}, \tilde{\mathbf{v}}_i\right) = J_i^u\left(\delta\tilde{\mathbf{u}}\right) \qquad \forall\, \delta\tilde{\mathbf{u}} \in \tilde{\mathcal{U}}_{\text{test}} \qquad i = 1..N_U \tag{6.59}$$

$$b_e\left(\delta\tilde{\mathbf{u}}, \tilde{\mathbf{v}}_i\right) = J_i^q\left(\delta\tilde{\mathbf{u}}\right) \qquad \forall\, \delta\tilde{\mathbf{u}} \in \tilde{\mathcal{U}}_{\text{test}} \qquad i = 1..N_Q \tag{6.60}$$

Here, $b_e(\cdot,\cdot)$ is the bilinear form given by Eqn. 6.36 (corresponding to a single-element problem on element Ω_e), J_i^u and J_i^q are the outputs defined in Eqns. 6.45 and 6.46, and $\tilde{\mathcal{U}}_{\text{test}}$ is an enriched space of order p_{test}. These equations can be written in discrete form as

$$\frac{\partial \tilde{\mathbf{R}}}{\partial \tilde{\mathbf{U}}}^T \tilde{\mathbf{V}}_i = \frac{\partial J_i^u}{\partial \tilde{\mathbf{U}}}^T \qquad i = 1..N_U \tag{6.61}$$

$$\frac{\partial \tilde{\mathbf{R}}}{\partial \tilde{\mathbf{U}}}^T \tilde{\mathbf{V}}_i = \frac{\partial J_i^q}{\partial \tilde{\mathbf{U}}}^T \qquad i = 1..N_Q \tag{6.62}$$

and solved to find the test function coefficients $\tilde{\mathbf{V}}_i$. Here, the single-element Jacobian matrix $\partial\tilde{\mathbf{R}}/\partial\tilde{\mathbf{U}}$ contains contributions from both the $\mathcal{R}_H^Q$ and $\mathcal{R}_H^U$ residuals, while $\tilde{\mathbf{V}}_i$ and $\tilde{\mathbf{U}}$ contain the coefficients associated with $\vec{\boldsymbol{\tau}}_i, \mathbf{v}_i$ and $\vec{\mathbf{q}}, \mathbf{u}$, respectively.

When used to weight the $\mathcal{R}_H^Q$ and $\mathcal{R}_H^U$ residuals, the optimal test functions result in the following set of equations on each interior element and its faces:

$$J_i^u(\mathbf{u}_H, \vec{\mathbf{q}}_H) + \int\limits_{\partial\Omega_e} \mathbf{v}_i^T \left(\hat{\mathbf{F}}_L - \hat{\mathbf{F}}\right) ds = \int\limits_{\partial\Omega_e} (\vec{\boldsymbol{\tau}}_i \cdot \vec{n})^T \, \hat{\mathbf{u}}_H \, ds \qquad i = 1..N_U \tag{6.63}$$

$$J_i^q(\mathbf{u}_H, \vec{\mathbf{q}}_H) + \int\limits_{\partial\Omega_e} \mathbf{v}_i^T \left(\hat{\mathbf{F}}_L - \hat{\mathbf{F}}\right) ds = \int\limits_{\partial\Omega_e} (\vec{\boldsymbol{\tau}}_i \cdot \vec{n})^T \, \hat{\mathbf{u}}_H \, ds \qquad i = 1..N_Q \tag{6.64}$$

$$\int\limits_{f} \boldsymbol{\mu}_H^T \left\{\hat{\mathbf{F}}_L + \hat{\mathbf{F}}_R\right\} ds = 0 \qquad \forall \boldsymbol{\mu}_H \in \mathcal{M}_H. \tag{6.65}$$

Here, the $\hat{\mathbf{F}}_L - \hat{\mathbf{F}}$ terms arise due to the fact that the flux $\hat{\mathbf{F}}$ used in the adjoint problems is not necessarily identical to the one-sided flux $\hat{\mathbf{F}}_L$. This is because, as discussed in Sec. 6.8.1, $\hat{\mathbf{F}}$ represents the flux of a single-element Dirichlet problem (and contains a full Roe flux), whereas $\hat{\mathbf{F}}_L$ depends solely on "one-sided" information and in most cases is only approximately equal to $\hat{\mathbf{F}}$.

Finally, note that we leave the face residual (Eqn. 6.65) the same as in standard HDG – i.e. we do not compute optimal test functions for the interface states $\hat{\mathbf{u}}_H$. Instead, we view the interface states as a passive "glue" that transmits the boundary accuracy on a given element to its neighbors across each face. Thus, if the interior trial space is of order p_B on element boundaries, we choose $\mathcal{M}_H$ to be a standard order-p_B polynomial space, and take $\hat{\mathbf{u}}_H \in \mathcal{M}_H$. Lastly, we note that in the final formulation (Eqns. 6.63-6.65) the optimal test functions appear only on element boundaries, so that no high-order interior integration is required.[12]

[12] Assuming no element-interior source terms are present.

6.9.3 Summary of Hybrid BDPG Method

The multi-dimensional BDPG method can be summarized as follows:

1. Use an order-p_B hierarchical basis for the trial space of $\mathbf{u}_H$ and $\vec{\mathbf{q}}_H$, but with all interior modes removed.
2. Use an order-p_B space for the interface states $\hat{\mathbf{u}}_H$.
3. Loop over each element.
4. Compute the local optimal test functions $\tilde{\mathbf{v}}_i$ by solving Eqns. 6.61 and 6.62 at order p_{test}, where $p_{\text{test}} \geq p_B$. Choosing $p_{\text{test}} = p_B$ often suffices.
5. Normalize the $\tilde{\mathbf{v}}_i$ with respect to a discrete or continuous L^2 norm.
6. Use the $\tilde{\mathbf{v}}_i$ within $\mathcal{R}_H^Q$ and $\mathcal{R}_H^U$, while using a standard order-p_B test space for $\mathcal{R}_H^\Lambda$. This amounts to solving Eqns. 6.63-6.65.

6.10 Results

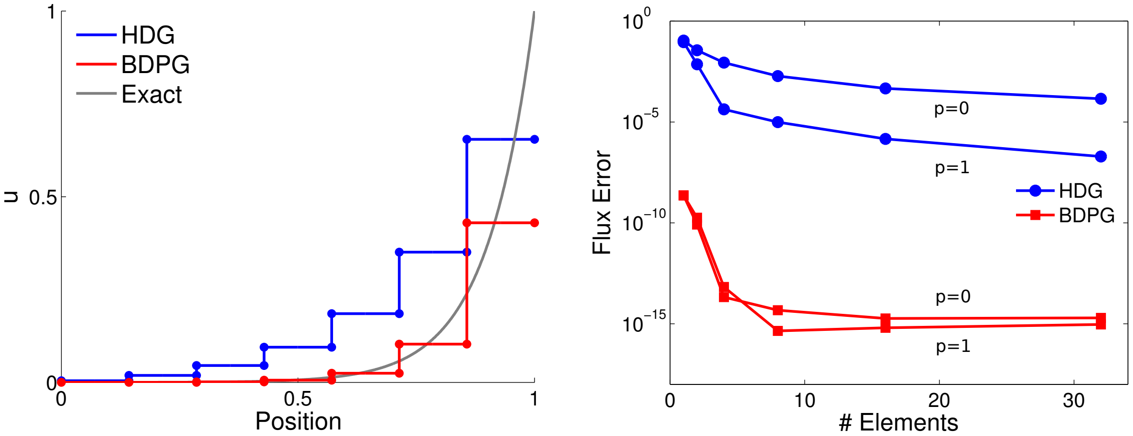
In this section, we present results for the hybrid BDPG method and compare its performance to standard HDG. We begin with a one-dimensional advection-diffusion problem before progessing to two-dimensional boundary layer and airfoil cases.

6.10.1 One-Dimensional Advection-Diffusion

We have shown that BDPG performs well for one-dimensional problems with advection. To demonstrate its performance for viscous problems, we solve the following advection-diffusion equation:

$$\begin{aligned} a\frac{\partial u}{\partial x} - \nu\frac{\partial^2 u}{\partial x^2} = 0 \qquad & x \in \Omega \\ u = 0 \qquad & x = x_L \\ u = 1 \qquad & x = x_R\,. \end{aligned} \tag{6.66}$$

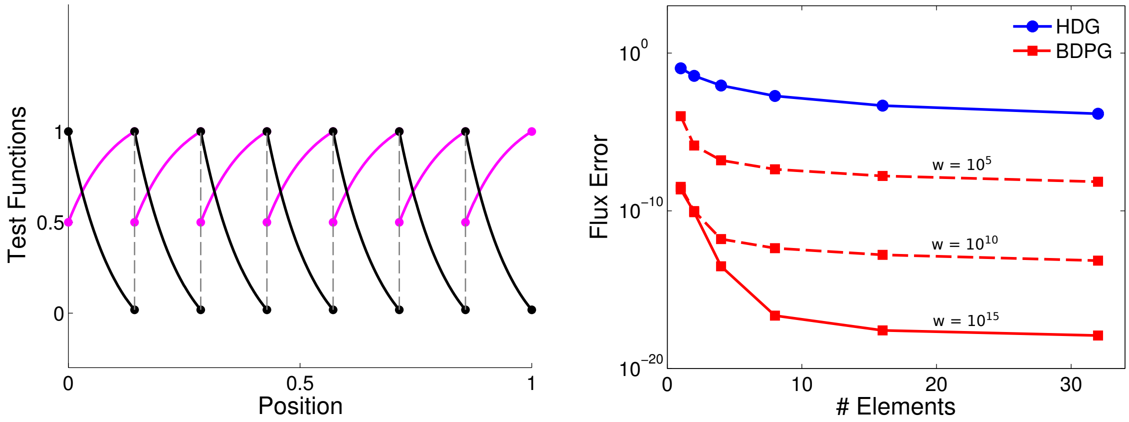
As an example, we choose the Reynolds number to be $aL/\nu = 10$ (where L is the domain width), the trial space orders to be $p = 0$ and $p = 1$, and the test space order



(a) Sample $p = 0$ solutions

(b) Right-flux error

Figure 6.8: One-dimensional advection-diffusion: (a) Sample $p = 0$ solutions for both HDG and BDPG. (b) Convergence of the right-boundary flux, where $w = 10^{15}$ was used for BDPG. BDPG provides interior accuracy while achieving significantly greater flux accuracy than HDG.



(a) Optimal test functions for $p = 0$

(b) Left-flux error

Figure 6.9: One-dimensional advection-diffusion: (a) Normalized v-component of the optimal test functions for the $p = 0$ solution in Fig. 6.8a. The black test functions are associated with the q trial bases, while the remaining test functions are associated with the u bases. (b) Convergence of the left-boundary flux for $p = 0$ and various choices of boundary weight, w. The higher the boundary weight, the more accurate the flux.

to be $p_{\text{test}} = 10$. The viscous length ℓ_{visc} is kept fixed at $O(1)$. Sample $p = 0$ solutions for HDG and BDPG are shown in Fig. 6.8a, while Fig. 6.8b gives the convergence of the right-boundary flux for $p = 0$ and $p = 1$ runs.

We see that, with a boundary weight of $w = 10^{15}$, BDPG provides nearly 10 orders of magnitude lower flux errors than HDG, while maintaining interior accuracy in u. Furthermore, Fig. 6.9b shows that (as expected) the choice of w determines the amount of flux accuracy obtained, with higher w leading to proportionally greater accuracy. Finally, the optimal test functions for the $p = 0$ case are shown in Fig. 6.9a. These test functions provide accuracy in both the left and right fluxes leaving each element, which leads ultimately to accuracy in the domain-boundary fluxes. As in the advection-reaction example, the initial convergence rate of the BDPG fluxes (which is observed here to be $p_{\text{test}} + p + 1$) is due solely to the inexact representation of the test functions. Thus, if analytical rather than numerical optimal test functions were used, machine-precision flux accuracy would be obtained on any mesh.

Remark 14. Note that for this problem, if we were to consider pure advection or pure diffusion (by setting $\nu = 0$ or $a = 0$, respectively), HDG would achieve exact fluxes without the need for optimal test functions. This is because, in general, the adjoints for the fluxes satisfy homogeneous equations of the form $L^*v = 0$. This equation reduces to $L^*v = -a\frac{\partial v}{\partial x} = 0$ for advection and $L^*v = -\nu\frac{\partial^2 v}{\partial x^2} = 0$ for diffusion. The solutions to these equations are just constant and linear functions, respectively. For $p \geq 1$, HDG already contains these adjoint solutions in its test space, so for these cases it is already optimal.

6.10.2 Two-Dimensional Advection-Diffusion: Manufactured Solution

Next, we move on to two dimensions. Before solving practical problems, we investigate two ideas related to the multidimensional test function theory: first, whether adequate trial space representation of the fluxes *is* actually important (as claimed in Sec. 8.2); and second, whether, given adequate flux representation, the localized optimal test functions *can* actually provide boundary accuracy.

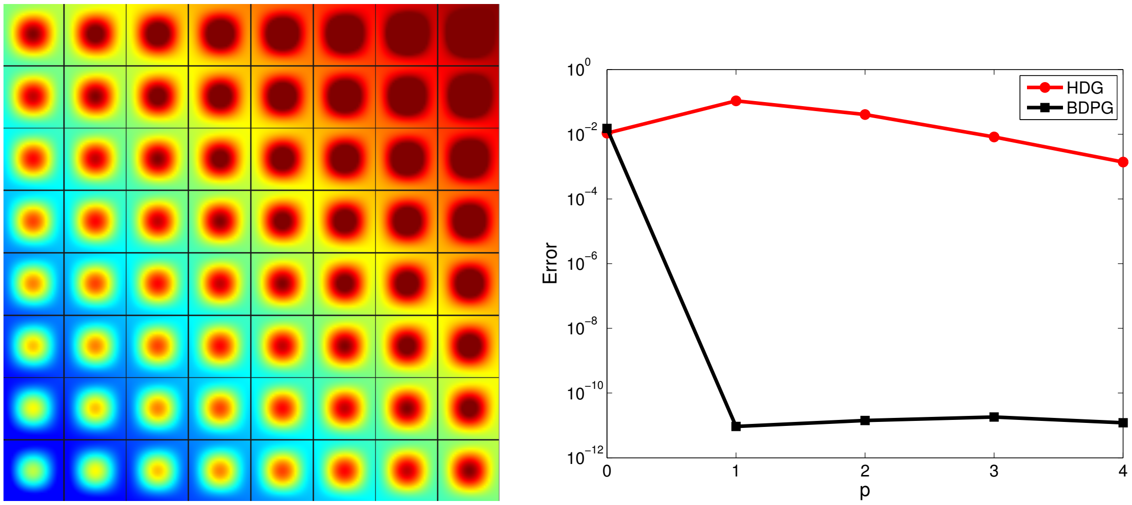
As mentioned, in two dimensions we expect BDPG to perform well only if the trial space is capable of adequately representing the true fluxes. In order to confirm this theory, we construct a manufactured solution whose true fluxes lie exactly in a $p = 1$ space. This solution is given by

$$u(x, y) = \sin{(8\pi x)}^2 \sin{(8\pi y)}^2 + x + y \,, \tag{6.67}$$

with contours shown in Fig. 6.10. Note that the sinusoidal terms vanish on all element boundaries, thus leaving a linear $(x + y)$ variation there. To define the problem, the advective velocity is chosen to be $\vec{a} = [0.4, 0.8]$, the viscosity is taken to be $\nu = 0.01$, and the domain has length $L = 1$.

We then solve the problem using BDPG with a standard trial space – i.e. with no Lobatto enrichment on element boundaries. When using a $p = 0$ trial space, we expect the performance of BDPG to suffer, since the true (linear) fluxes cannot be adequately represented. However, as soon as the trial space order is increased to $p = 1$, the true fluxes become representable, and we expect BDPG to be capable of delivering nearly exact boundary values.

To perform the test, we sweep through trial space orders from $p = 0$ to $p = 4$ and record the error in the boundary fluxes for both BDPG and HDG. For BDPG, we keep the test space order fixed at a high value of $p_{\text{test}} = 10$ to ensure that the test function representation has a minimal influence on the results.



(a) Manufactured solution

(b) Top flux error convergence

Figure 6.10: Two-dimensional advection-diffusion: (a) Manufactured solution with fluxes that are exactly representable in a $p = 1$ space. (b) Convergence of the top-boundary flux as a function of p. As the order is increased above $p = 0$, the fluxes become representable and BDPG attains machine-precision accuracy. This verifies the performance of BDPG and highlights the importance of flux resolution in multiple dimensions.

Fig. 6.10 shows the error in the top-boundary flux for both methods, which is representative of the fluxes on all boundaries. The results are as we expect: for $p = 0$, the BDPG errors are large since the flux is not representable, but as soon as $p = 1$ is used, the error drops to machine-precision levels. This highlights the importance of flux resolution for multidimensional problems, and justifies the idea of enriching the trial space near element boundaries. Furthermore, the performance of BDPG for $p \geq 1$ confirms that the optimal test space is functioning well, since if it were not, boundary accuracy – even *with* adequate flux resolution – would not be achieved. This point is demonstrated clearly by the performance of standard HDG, which due to its suboptimal test space has nearly 10 orders of magnitude larger errors than BDPG.

Finally, we note that in order to achieve the machine-precision accuracy shown in Fig. 6.10, the viscous length scale for BDPG had to be taken small; specifically, a value of $\ell_{\text{visc}} = 10^{-7}$ was used for the $p \geq 1$ runs. (A more standard value of $\ell_{\text{visc}} = 10^{-1}$ was used for all other runs.) This suggests that for multidimensional viscous problems, the test function localization becomes more effective as the elements become more tightly coupled, since the effect of a small viscous length is to penalize the inter-element jumps in u. While this issue warrants further analysis, we find that for practical problems (where the fluxes are not exactly representable) the performance of BDPG is relatively insensitive to the choice of viscous length, and more modest values of ℓ_{visc} can be used. Finally, note that this issue does not arise for inviscid problems, since in that case no ℓ_{visc} is defined. Indeed, when a similar manufactured solution is solved with the linearized Euler equations, BDPG attains machine-precision boundary fluxes with no "free" parameters involved.

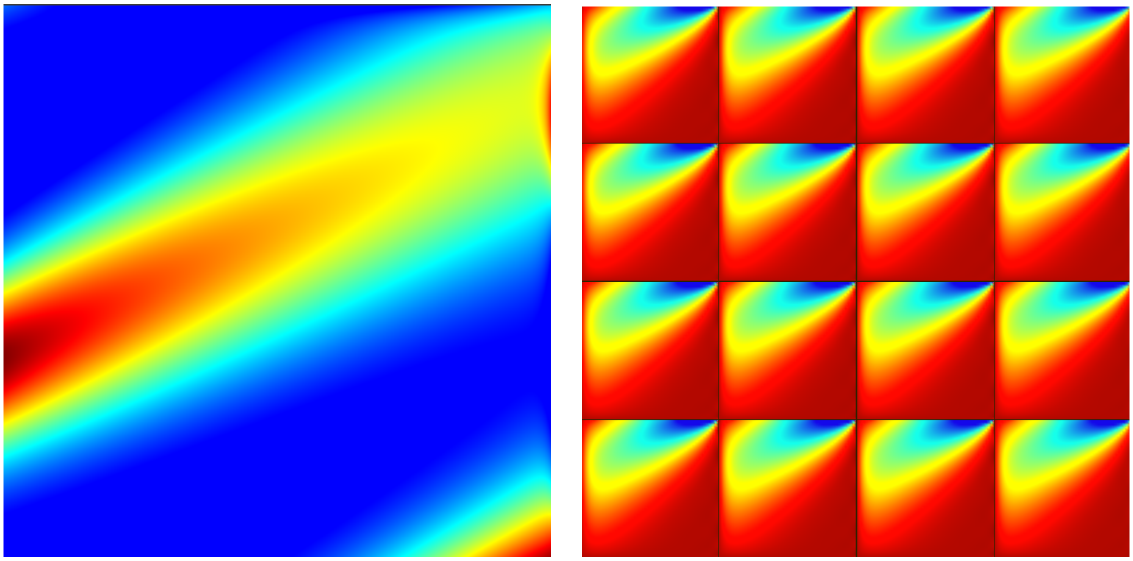
6.10.3 Two-Dimensional Advection-Diffusion: Boundary Layer

Next, we try a more practical advection-diffusion problem. With the same domain as above, we take $\vec{a} = [0.8, 0.4]$, $\nu = 0.01$ (so that $Re \approx 100$), and specify a Dirichlet boundary condition on all sides of the domain given by

$$u(x,y) = \exp\left[\frac{1}{2}\sin\left(-4x+6y\right) - \frac{4}{5}\cos\left(3x-8y\right)\right] \qquad \vec{x} \in \partial\Omega\,. \tag{6.68}$$

This condition generates boundary layers on the two outflow boundaries (the top and right), and provides a test as to whether BDPG can accurately predict these features. Fig. 6.11 shows contours of both the solution and the optimal test functions (which are again computed with $p_{\text{test}} = 10$) for a $p = 1$ trial space.

Since we verified above that the resolution of interface fluxes is critical for BDPG, we enrich the trial space with Lobatto functions near element boundaries. For the results shown in Fig. 6.12, we consider enrichment orders of $p_B = 6, 7, 8$, while keeping the interior basis at a low order of $p_I = 1$. We compare the BDPG results to a standard HDG method with the same interior trial space order of $p_I = 1$. Viscous lengths of $\ell_{\text{visc}} = 10^{-4}$ and $\ell_{\text{visc}} = 1$ are used for BDPG and HDG, respectively, with the results being relatively insensitive to these values.



(a) Solution, u

(b) Optimal test function component

Figure 6.11: Two-dimensional advection-diffusion: (a) The solution to a $Re = 100$ problem on a fine mesh. (b) The optimal test functions associated with the upper-right q_y trial basis on each element. Note the upwinding nature of the test functions.

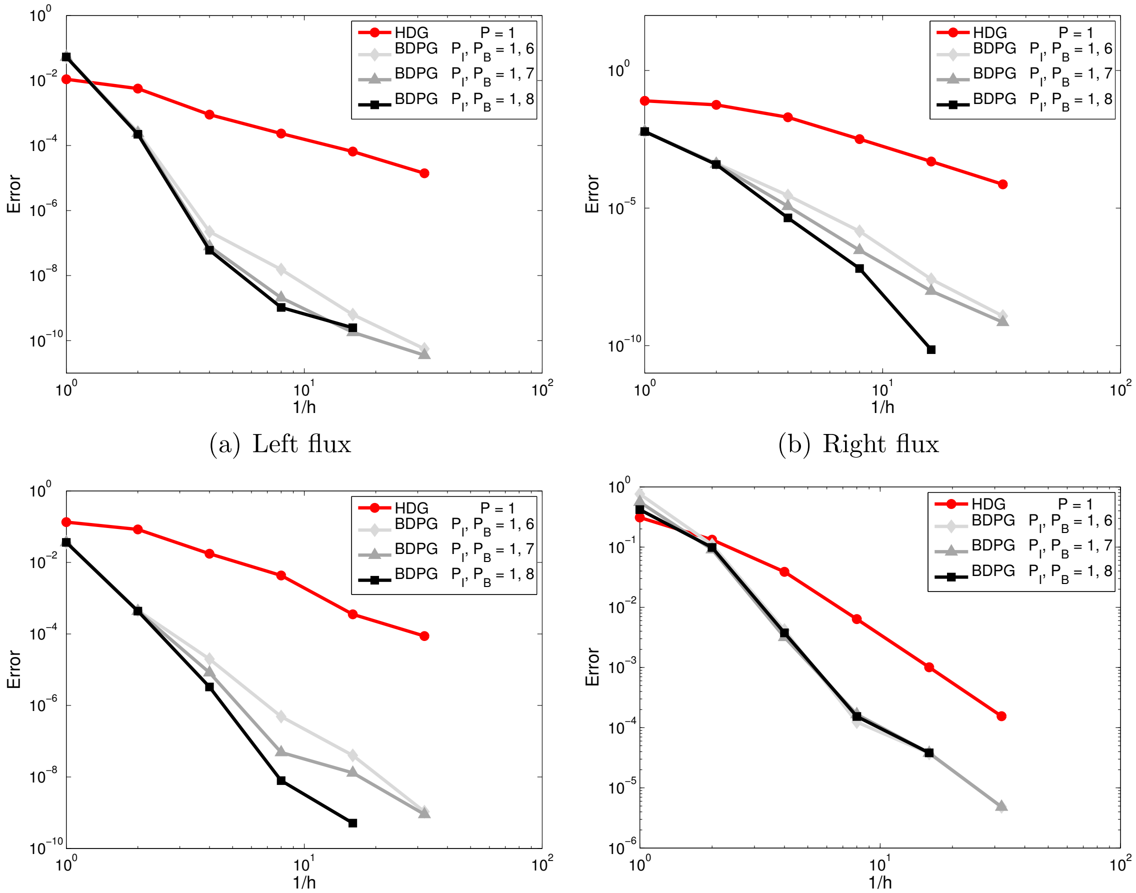
From Fig. 6.12, we see that BDPG with Lobatto enrichment can provide a nearly 6-orders-of-magnitude reduction in the flux errors, while maintaining accuracy in interior outputs. Furthermore, in addition to providing accuracy in the total flux through each boundary, BDPG also achieves accuracy in the solution *profiles* along the boundaries. The solution and gradient profiles along the right boundary of the domain are shown in Fig. 6.13, from which the enhanced accuracy of BDPG is apparent.

These results, of course, should be kept in perspective: the boundary enrichment of BDPG represents an additional expense (and additional degrees of freedom) compared to $p = 1$ HDG, so the comparison is in that sense unfair. However, the results demonstrate clearly that, with BDPG, the attainment of global boundary accuracy depends solely on the ability to resolve the interface fluxes – a fact that is not true of standard HDG,[13] and one that may be capitalized on in the future.

6.10.4 Two-Dimensional Linearized Euler: Manufactured Solution

With the performance of BDPG verified for scalar problems in one and two dimensions, we next move on to two-dimensional *systems*. In particular, we solve the

[13] Fig. 6.18b provides an explicit demonstration of the fact that, with standard HDG, flux resolution does not guarantee accuracy.



(a) Left flux

(b) Right flux

(c) Top flux

(d) Domain u^2 ouptut

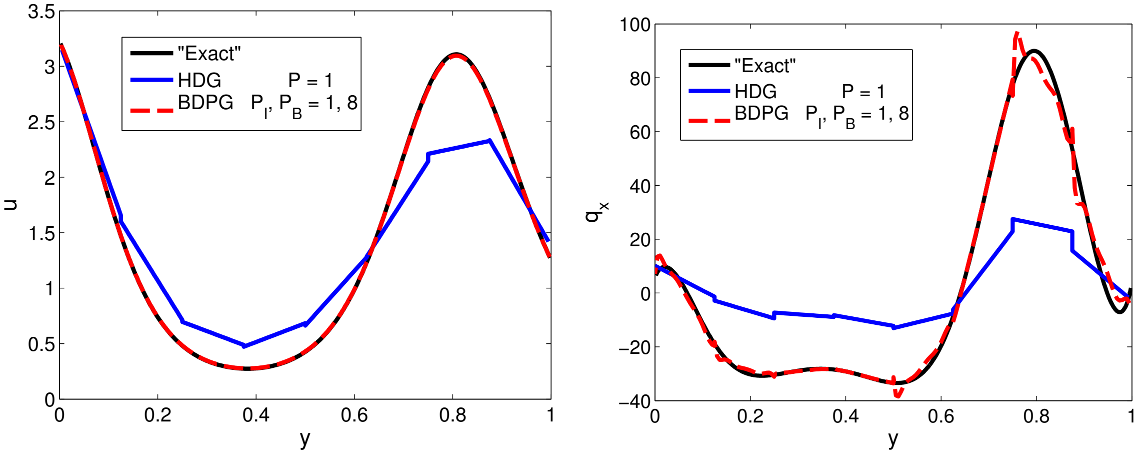
Figure 6.12: Two-dimensional advection-diffusion: Convergence rates for various outputs. Note that p_I and p_B denote the interior and boundary interpolation orders, respectively. Higher accuracy is obtained with BDPG as the amount of boundary enrichment increases. Note that BDPG also achieves accuracy in the interior u^2 output.

homentropic linearized Euler equations, with state variables and fluxes given by

$$\mathbf{u} = \begin{bmatrix} p \\ u_i \end{bmatrix}, \quad \mathbf{F}_j = \begin{bmatrix} u_{0_j} p + \rho_0 a_0^2 u_j \\ \frac{p}{\rho_0}\delta_{ij} + u_{0_j} u_i \end{bmatrix}, \tag{6.69}$$

where $1 < i, j < \text{dim}$. The state variables $\mathbf{u}$ represent velocity and pressure perturbations about the background state, which is described by the parameters a_0, u_{0_j}, and ρ_0 (speed of sound, velocity, and density, respectively).

As an initial test, we construct a manufactured solution on a square domain



(a) u right-boundary profile

(b) q_x right-boundary profile

Figure 6.13: Two-dimensional advection-diffusion: Solution profiles along the right boundary of the domain. BDPG with enrichment achieves greater accuracy than standard HDG.

$(L = 1)$ given by

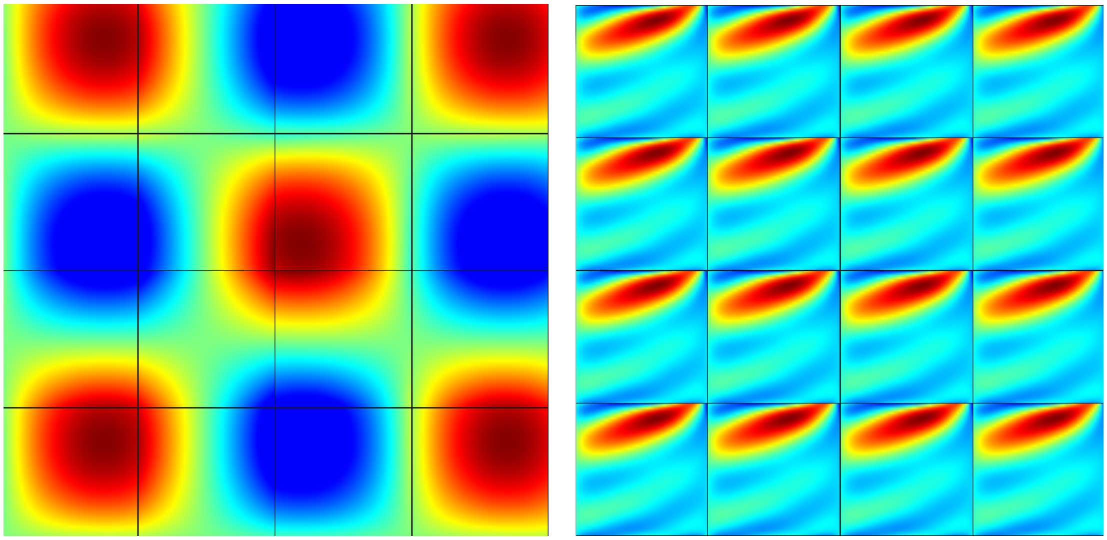
$$\begin{aligned} p(x, y) &= \sin(8.5x)\sin(8.5y) \\ u_i(x, y) &= 0\,. \end{aligned} \tag{6.70}$$

A plot of the pressure contours is provided in Fig. 6.14, along with a set of optimal test functions. The background state is chosen as $\rho_0 = 1$, $a_0 = 3$, $u_{0_1} = 0.8$, and $u_{0_2} = 0.2$, so that the Mach number is approximately 0.3. We again choose the test space order and boundary weights to be high (10 and 10^{10}, respectively), and consider the same boundary enrichment orders as in the previous section.

The results shown in Fig. 6.15 are encouraging, and mirror those obtained in the two-dimensional advection-diffusion case. We see that BDPG achieves output error reductions of over 10 orders of magnitude compared to HDG at the same interior trial space order. Furthermore, BDPG also obtains accurate boundary profiles, as shown in Fig. 6.16. These results verify the effectiveness of optimal test functions for systems of equations.

6.10.5 Two-Dimensional Linearized Euler: Cylinder and Airfoil

Lastly, we consider linearized Euler cases of engineering interest: subsonic flow over a cylinder and an airfoil. For these cases, the background state is chosen as



(a) Manufactured solution (pressure contours)

(b) Optimal test function component

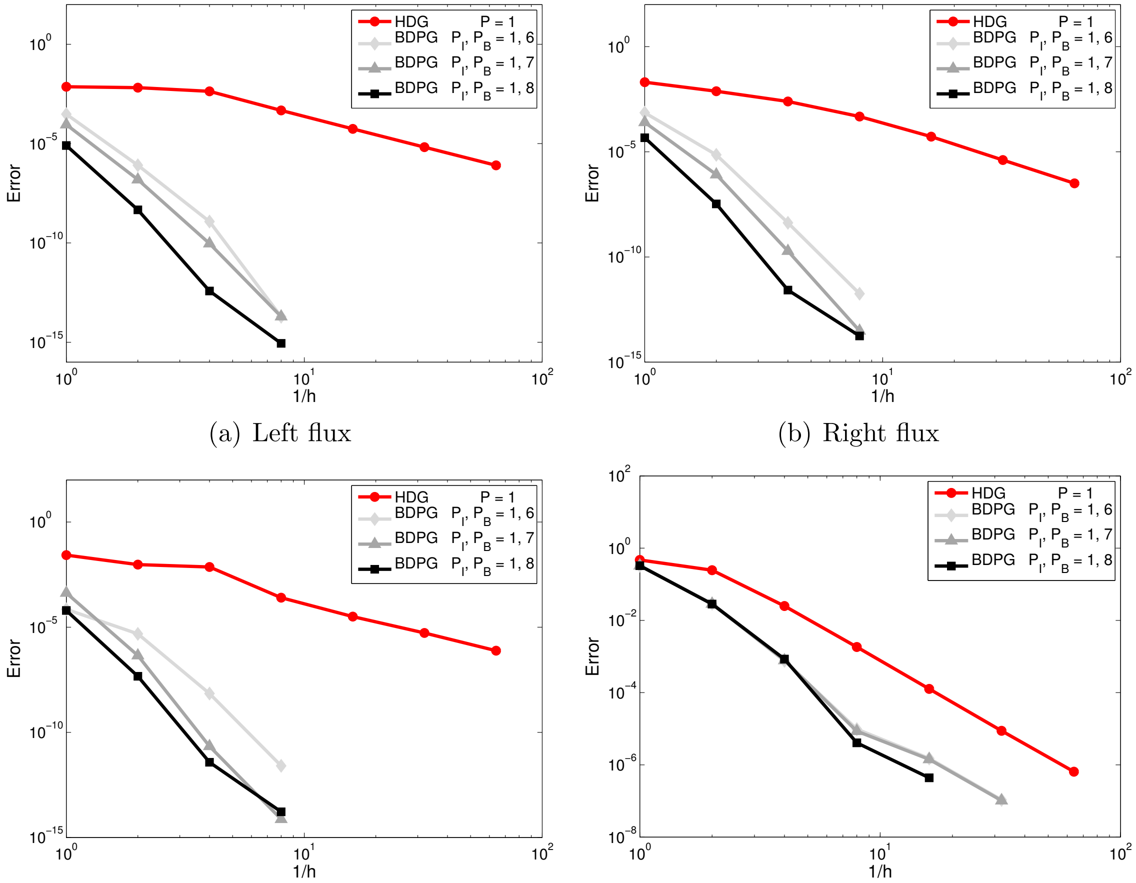
Figure 6.14: Two-dimensional linearized Euler: (a) Manufactured solution pressure contours. (b) Component of the optimal test functions corresponding to the trial basis in the upper-right corner of each element.

$\rho_0 = 1$, $a_0 = 3$, $u_{0_1} = 1$, $u_{0_2} = 0$, so that the flow is horizontal and the Mach number is approximately 0.3. In addition, the farfield boundary conditions on the state variables are $p = 1$, $u_1 = 1$, and $u_2 = 1$, so that a uniform perturbation travels upward and to the right. Finally, the mesh elements themselves are curved and are represented with $Q = 4$ polynomials, providing a first test of BDPG with curved geometry.

First, we simulate the flow around a cylinder of radius unity. Solution contours are shown in Fig. 6.17, while the convergence of the pressure flux along the cylinder wall is shown in Fig. 6.18a. If the trial space is enriched appropriately, we see that BDPG can achieve nearly 6 orders of magnitude lower flux errors than HDG.

To demonstrate that these accuracy gains are not *just* due to the trial space enrichment, we perform another HDG simulation in which the same boundary-enriched ($p_B = 8$) trial space is used as for BDPG. In this case, the only difference between BDPG and HDG is the test space. Fig. 6.18b shows the results of this comparison. We see that BDPG still achieves flux errors that are nearly 6 orders of magnitude lower than HDG. This again demonstrates that, in multiple dimensions, it is the combination of both optimal test functions *and* trial space resolution that is critical to achieving boundary accuracy.

Finally, to conclude our linear tests, we simulate the flow around a NACA 0012



(a) Left flux

(b) Right flux

(c) Top flux

(d) Domain p^2 ouptut

Figure 6.15: Two-dimensional linearized Euler: Output convergence for HDG and BDPG runs. The flux outputs represent the sum of all state components of the flux vector. (Note that this sum is taken so that the convergence behavior of all fluxes can be captured in a single plot – each of the individual flux components converges similarly to the results shown here.) Higher accuracy is obtained as the amount of BDPG boundary enrichment increases. BDPG also achieves accuracy in the interior p^2 output.

airfoil. The airfoil has a unit chord and the background state is the same as above. Pressure contours for this case are provided in Fig. 6.19, which also gives the convergence of the x-velocity flux through the airfoil. Although the trailing-edge singularity limits the uniform-refinement rates for this problem, we see that, once again, BDPG achieves superior boundary accuracy.

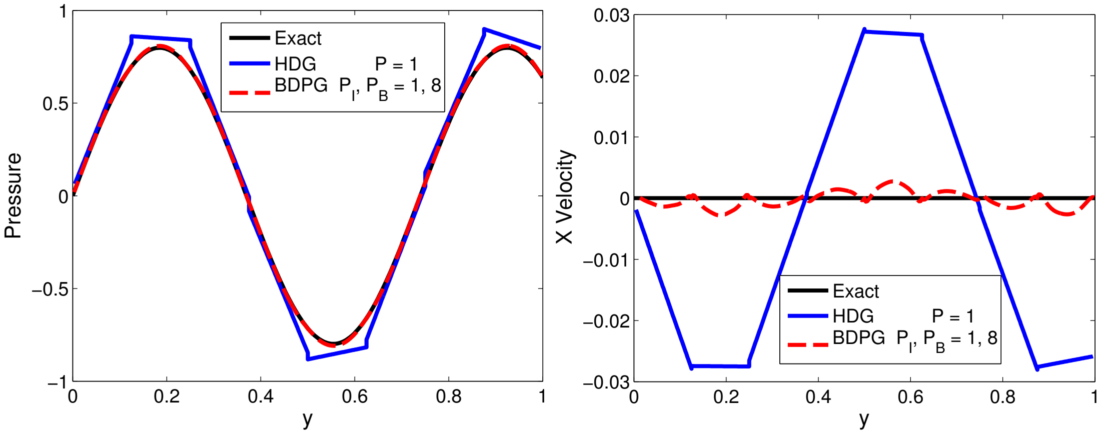
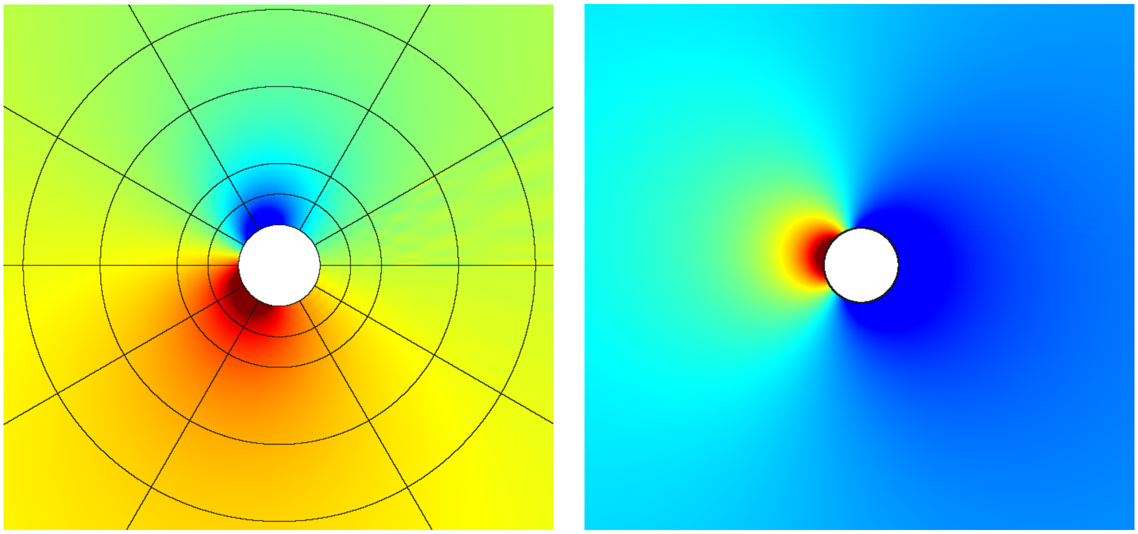


Figure 6.16: Two-dimensional linearized Euler: Solution profiles along the right boundary of the domain. BDPG with boundary enrichment is again more accurate than HDG with the same interior basis.



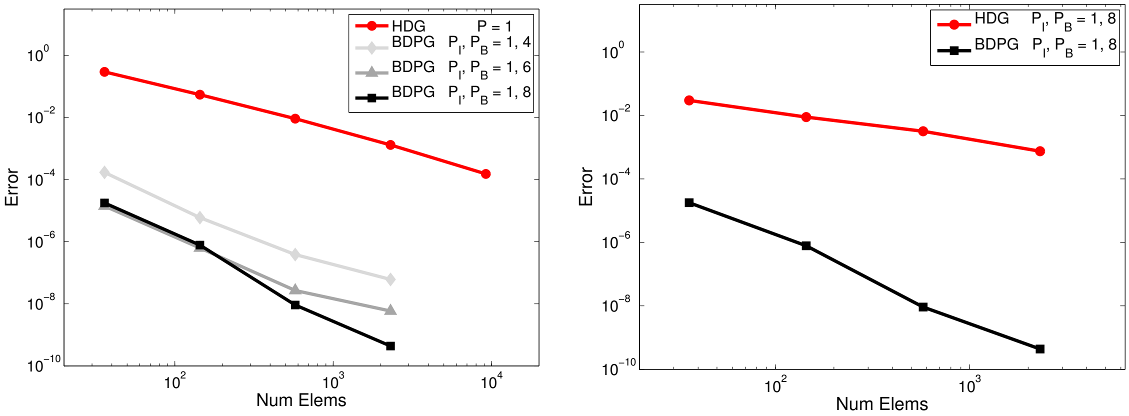
(a) Pressure contours

(b) y-velocity contours

Figure 6.17: Two-dimensional cylinder: (a) Pressure and (b) y-velocity contours from a high-order HDG solution.

6.11 Nonlinear Extension of BDPG

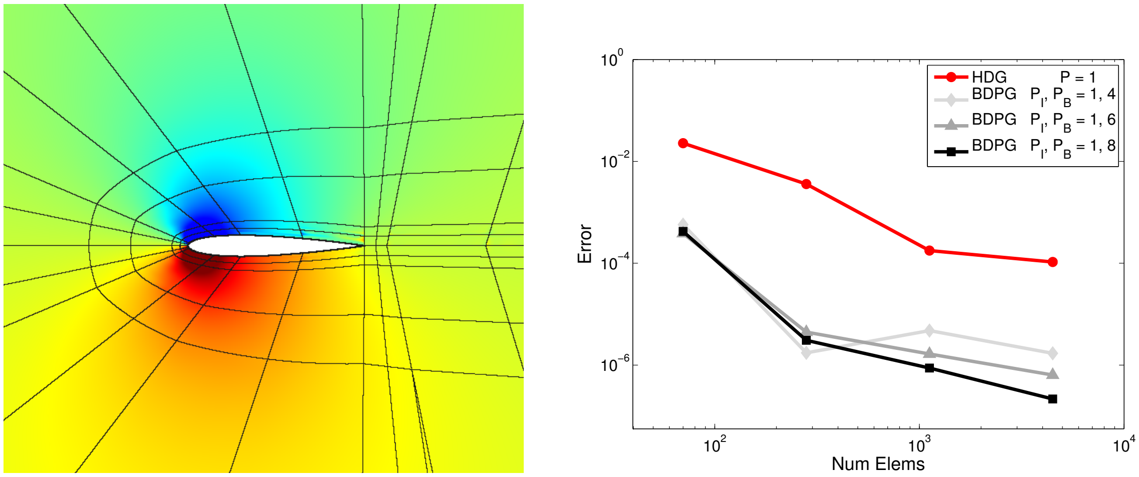
In this section, we discuss how the BDPG ideas can be extended to nonlinear problems. We then show preliminary results in one and two dimensions and identify remaining challenges and limitations.



(a) Pressure flux through cylinder, BDPG vs. $p = 1$ HDG

(b) Pressure flux through cylinder, BDPG vs. boundary-enriched HDG

Figure 6.18: Two-dimensional cylinder: (a) Convergence of the pressure flux through the cylinder wall for BDPG and $p = 1$ HDG. (b) Pressure flux convergence where the same $p_B = 8$ trial space is used for both BDPG and HDG. Since the only difference is the test space, the results show that the optimal test functions of BDPG are effective in reducing boundary errors.



(a) Pressure contours

(b) Airfoil x-velocity flux

Figure 6.19: Two-dimensional airfoil: (a) Pressure contours from a high-order BDPG solution. (b) x-velocity flux convergence for both BDPG and HDG. While the convergence rates are limited by the trailing-edge singularity, BDPG still provides a benefit over HDG.

6.11.1 Nonlinear Extension and Benefit

Our work so far has focused on linear problems. However, since the theory of optimal test functions is based on the solution of local adjoint problems, this theory can be extended to nonlinear problems in exactly the same way that adjoint problems can. In other words, we can keep the definition of the optimal test functions the same, but recognize that the adjoint problems used to define these test functions now represent a linearization of the full nonlinear governing equations.

There is, of course, a price to be paid for this linearization. Looking back at our derivation of the nonlinear error estimate in Eqn. 2.107, we see that there is an $\mathcal{O}(\delta \mathbf{u}^2)$ error associated with performing a linearization. Since $\delta \mathbf{u}$ itself converges at a rate of $p+1$, this error is of order $2p+2$. Thus, if we base our computation of the optimal test functions on a (linearized) adjoint problem, the highest rate of convergence we can hope for is $2p+2$. Note that a rate of $2p+2$ is one order higher than the rate of a standard DG/HDG method, which typically superconverges at a rate of $2p+1$. This means that the benefit of using optimal test functions for nonlinear problems is to increase the order of accuracy by *one*. Of course, this comes at the additional cost of computing the test functions, which is not negligible.

On the other hand, there is also an unintended "benefit" associated with the linearization error, which is that it gives us an indication of what p_{test} value should be used to compute the test functions. For nonlinear problems, there is a point beyond which increasing p_{test} will provide no further improvement in accuracy, since the $\mathcal{O}(h^{2p+2})$ linearization error will necessarily remain and eventually dominate. We can compute this "limiting" p_{test} value in a straightforward manner. Since BDPG outputs should converge at a rate of at least $p_{\text{test}}+p+1$ (which comes from summing the product of the test function and residual rates), choosing $p_{\text{test}}=p+1$ is enough to obtain an $\mathcal{O}(h^{2p+2})$ output convergence rate.

6.11.2 Nonlinear Test Function Computation

Since the test functions are solutions to elementwise adjoint problems, which for nonlinear problems depend on the value of the state, a question naturally arises: *which* state should the test functions be computed about?

Typically, to solve a nonlinear problem, an initial guess would be chosen and a Newton iteration would be employed to drive the initial guess to a converged solution. If the test functions were fixed (say, the standard polynomials used in DG/HDG), each step in the Newton iteration would just involve computing a state update. However, since the optimal test functions depend on the state, these test functions should also (in theory) be updated after each Newton step. In the results shown below, this is the strategy we adopt when using the BDPG method. This strategy leads to a back-and-forth in which we attempt to converge the residuals to zero while simultaneously changing their very definition (by virtue of the test functions changing).

In practice, this continual updating of the test functions is not actually required. An alternative strategy in which they are updated at every few Newton steps often suffices and is more efficient.

Another option – which is perhaps the most efficient choice – is to first solve a standard DG/HDG problem and to then compute the optimal test functions based on the linearization about this DG/HDG state. After this single update/computation of the test functions, they can be "frozen" for the remainder of the iterations, and the updated residuals can be smoothed back to machine-precision. Since the test function linearization is based on the $\mathcal{O}(h^{p+1})$ DG/HDG solution, this procedure allows for the optimal $2p+2$ rates to still be obtained. The only difference between this procedure (which requires only a single test function computation) and the "continual-update" procedure (which requires as many test function computations as there are Newton steps) is a potential shift in the error coefficient. However, with sufficiently refined meshes and/or $p > 1$, this difference is typically negligible.

Finally, for nonlinear problems, we note that it is less clear how the boundary weight w should be chosen in the error norm (Eqn. 6.40). For linear problems in which the fluxes are capable of being resolved, the accuracy obtained in these fluxes is proportional to $1/w$. However, for nonlinear problems, the final accuracy of the fluxes depends to some extent on the accuracy of the element-interior solution. Furthermore, due to the linearization error, there is a point beyond which increasing w leads to no further reduction in the flux errors. In the following examples, we find that choosing a relatively modest value of w (within the range of 1×10^6 - 1×10^8) typically works well. Ideally, a more rigorous means of choosing w can be found in the future.

6.11.3 Nonlinear One-Dimensional Examples

We now show results for one-dimensional nonlinear problems. As mentioned, the best flux convergence rate we can expect with BDPG is now $2p+2$, compared to the DG/HDG rate of $2p+1$.

6.11.3.1 1D Burgers

First, we solve the one-dimensional Burgers equation with the following source term and boundary condition:

$$u\frac{\partial u}{\partial x} + 4u^{3/2}\sin(kx) = 0 \qquad x \in \Omega \tag{6.71}$$

$$u = 1 \qquad x = x_L\,. \tag{6.72}$$

The source term is chosen to be a function of relatively high frequency, with $k = 7.73$ taken as an example. Information flows to the right in this problem, so that the boundary condition on the left side of the domain (defined as $\Omega = [-\frac{1}{2}, \frac{3}{2}]$) is well-posed.

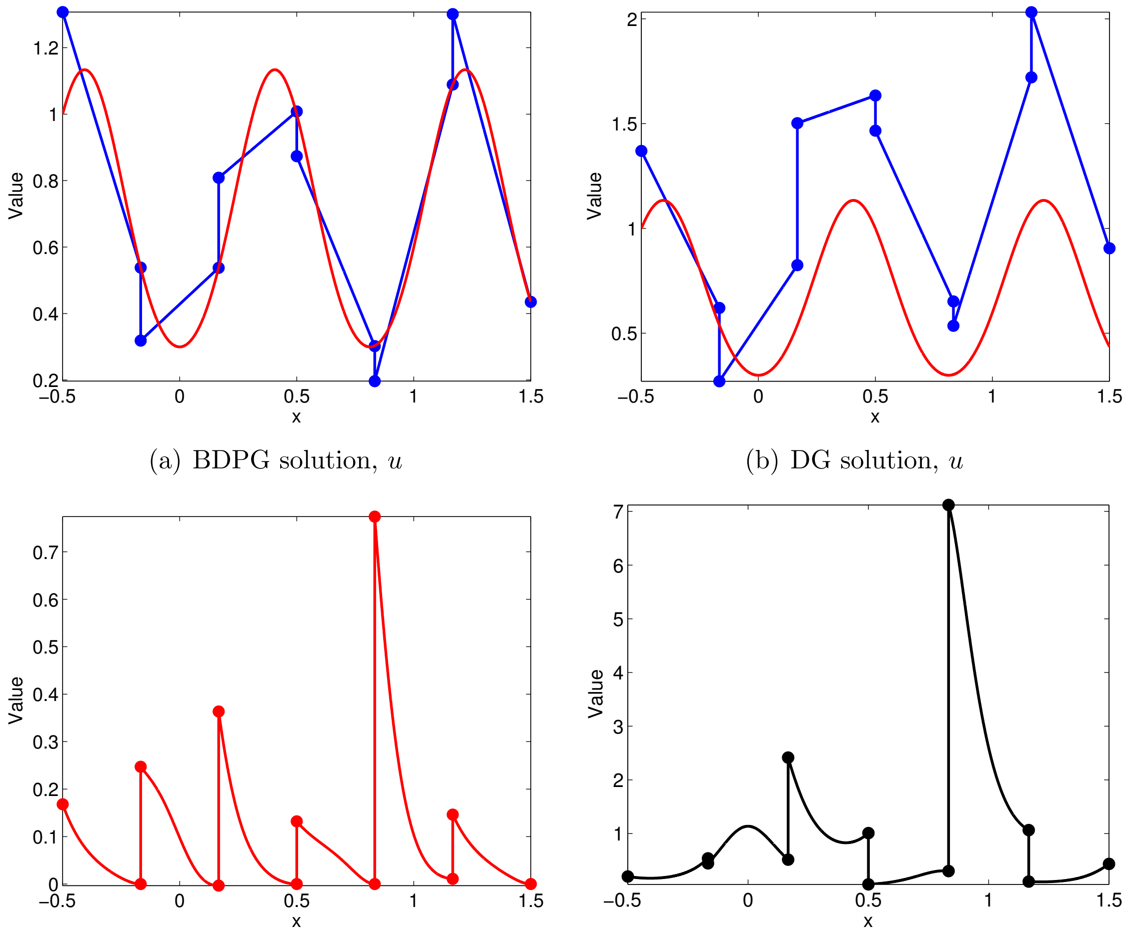
For the optimal test function problem, a large boundary weight of $w = 1 \times 10^8$ is chosen in order to achieve flux accuracy. Furthermore, although not required, we take $p_{\text{test}} = 12$ to ensure that the full benefits of BDPG are observed. As mentioned above (and as shown in the following section) an order of $p_{\text{test}} = p + 1$ is all that is needed to obtain optimal rates. Finally, the test functions are recomputed after each iteration as the problem is converged to a steady-state solution.

Fig. 6.20 shows a side-by-side comparison of $p = 1$ DG and BDPG solutions for this problem, with the exact solution shown in red. We see that the optimal test functions (which are also shown in Fig. 6.20) enable the BDPG solution to achieve accuracy in the downwind fluxes, which keeps the numerical solution “pinned” to the exact solution. With DG, on the other hand, inaccuracies in the fluxes compound as the flow moves downstream, so that the numerical solution eventually deviates from the exact solution.

The convergence of the flux errors on the domain outflow boundary are shown in Fig. 6.21 for both DG and BDPG, for trial space orders of $p = 0$ through $p = 3$. As expected, we see that the optimal test functions enable the BDPG fluxes to attain rates of $2p + 2$, whereas the DG fluxes achieve rates of only $2p + 1$. This confirms our earlier predictions about the performance of optimal test functions with nonlinear problems, and shows that they can achieve an order of accuracy that is one order higher than a standard DG method.

6.11.3.2 1D Euler

Next, we move from a scalar nonlinear problem to a nonlinear system of equations. In addition, we shift from the standard DG/BDPG methods used in the previous



(a) BDPG solution, u

(b) DG solution, u

(c) BDPG test functions, v_1

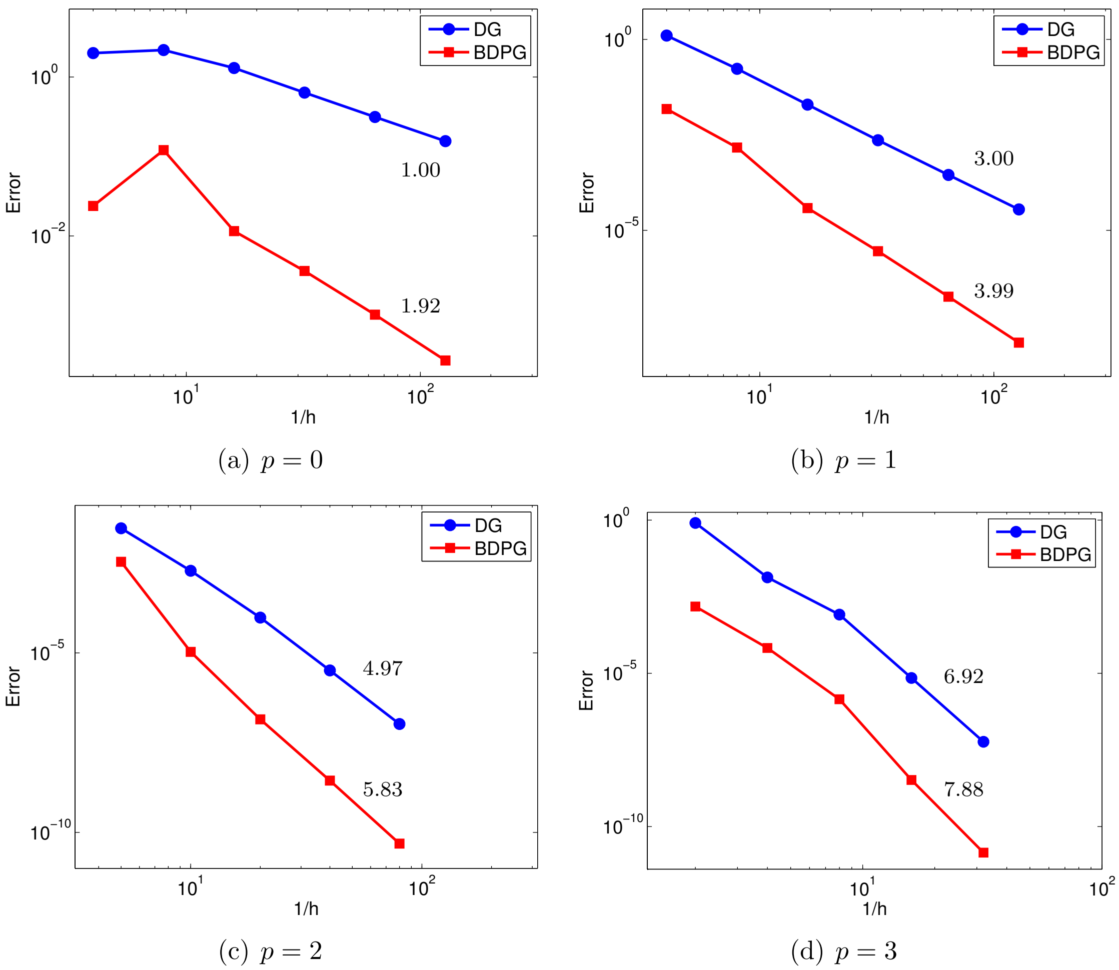
(d) BDPG test functions, v_2

Figure 6.20: 1D Burgers: 6-element, $p = 1$ (a) BDPG solution and (b) DG solution. Numerical solutions are shown in blue, while the exact solution is shown in red. Note the downwind accuracy of BDPG on each element. (c) First and (d) second BDPG test functions on each element, corresponding to the two Lagrange trial basis functions on each element. The test functions in (d) have been normalized by the flux weight, w, to keep their magnitudes $\mathcal{O}(1)$. Note that the vertical lines in all figures are drawn only for convenience – both the test functions and states on each element are independent and are not actually connected in any way.

section to hybrid DG/BDPG methods.

We first try a 1D subsonic Euler case, which consists of an inflow at the left boundary (with a total temperature of 7.5 and a total pressure of 3.389), an inviscid wall at the right boundary, and a source term

$$\mathbf{S}^T = \left[\, 0.4\,\rho^2 \;\; 0.7\,(\rho u)^2 \;\; 0.1\,(\rho E)^2 \,\right] \tag{6.73}$$



(a) $p = 0$

(b) $p = 1$

(c) $p = 2$

(d) $p = 3$

Figure 6.21: 1D Burgers: Right-boundary flux convergence for BDPG and DG methods. BDPG achieves rates of $2p + 2$, compared to the $2p + 1$ rates of DG.

added to the left-hand side of the equations. The flow enters from the left and collides with the wall on the right, while $\mathbf{S}$ acts as a sink that relieves the buildup of mass that would otherwise occur within the domain. Fig. 6.22a shows the steady-state values of the solution states (density, momentum, and energy) throughout the domain.

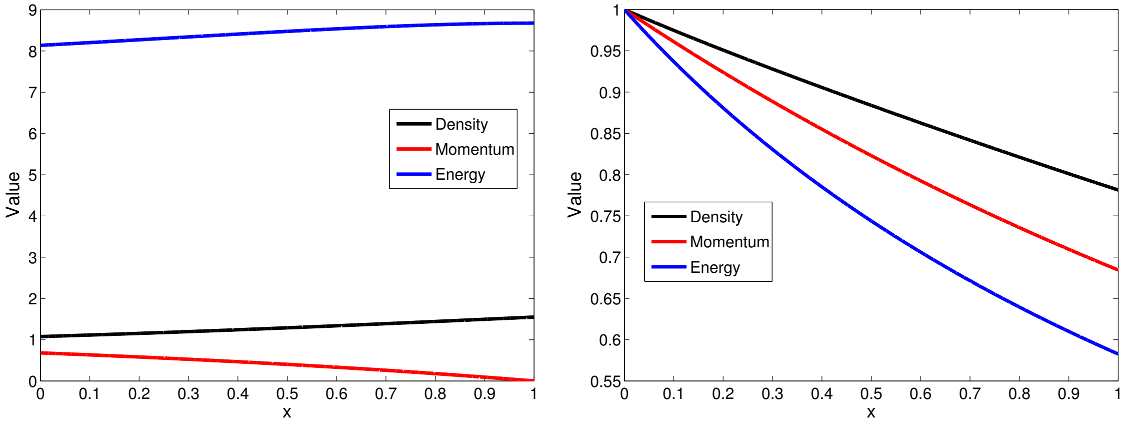
We solve this problem numerically with both HDG and (hybrid) BDPG for a trial space order of $p = 1$. For BDPG, we choose the test space order to be $p_{\text{test}} = p+1 = 2$ and the boundary weight to be large; specifically, $w = 10^8$. Note that since there are 3 state components associated with the 1D Euler equations, for a $p = 1$ trial basis, we must now compute a corresponding $n_U \cdot r = 2 \cdot 3 = 6$ test functions on each element.

Fig. 6.23a shows the convergence of the energy flux on the left boundary for both HDG and BDPG as the mesh is refined. For the BDPG runs, we observe a rate of 3.72, which is close to (though slightly below) the optimal rate of $2p + 2 = 4$. Similar rates are obtained for the remaining flux components on both left and right boundaries.

Next, we try a supersonic case, where the inflow Mach number is 1.89, corresponding to Dirichlet values of 1 in all state components on the left boundary. A source term

$$\mathbf{S}^T = \left[\, 0.4\,\rho^2 \;\; 0.7\,(\rho u)^2 \;\; 1.0\,(\rho E)^2 \,\right] \tag{6.74}$$

is added to the left-hand side of the equations, and steady-state solution profiles are shown in Fig. 6.22b. To solve the problem, we again use HDG and BDPG, with trial space orders ranging from $p = 0$ to $p = 2$. The test space order and boundary weight are again taken to be $p_{\text{test}} = p + 1$ and $w = 10^8$, respectively. Fig. 6.23 (parts *b*-*d*) shows convergence rates for various flux outputs and trial space orders p. We see that for this problem BDPG achieves the optimal rate of $2p + 2$ for all fluxes and trial space orders.



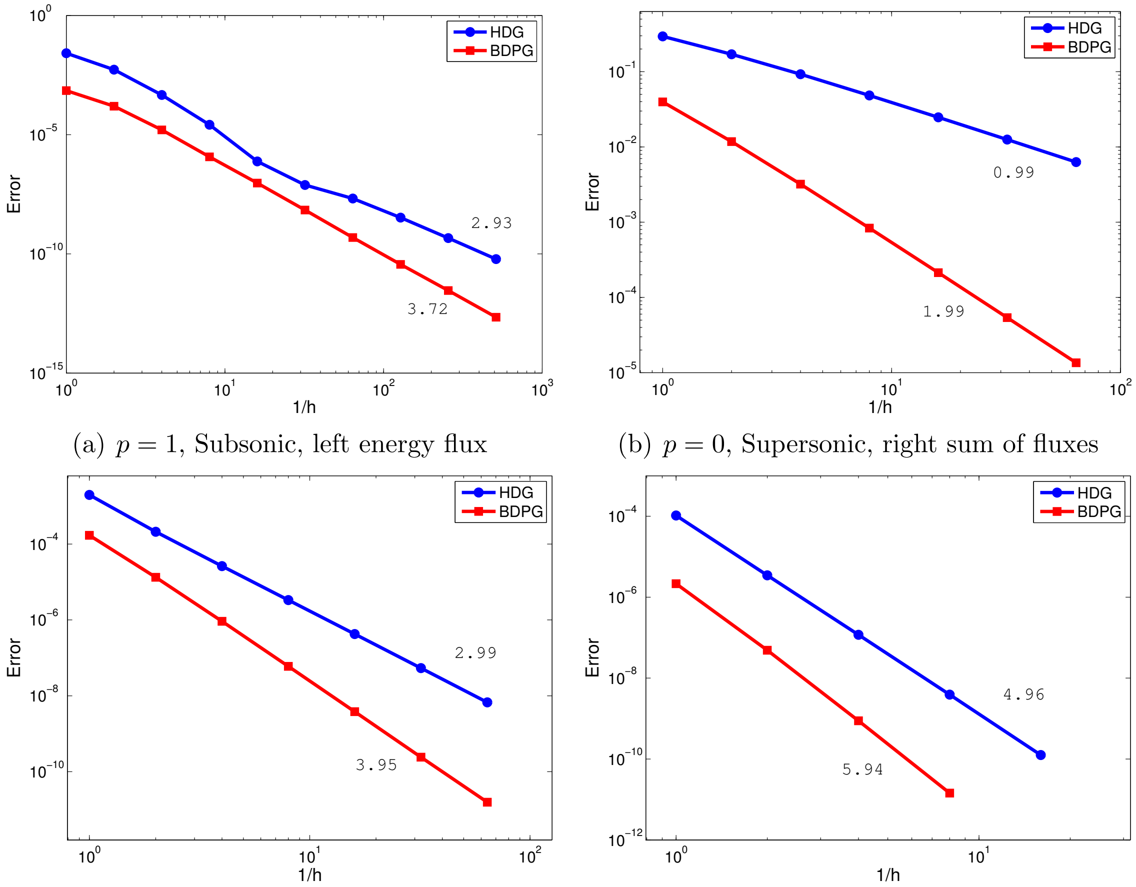
(a) Subsonic case

(b) Supersonic case

Figure 6.22: 1D Euler: (a) Subsonic case with inflow on the left and inviscid wall on the right side of the domain. (b) Supersonic case with Dirichlet conditions on left and outflow on right.

6.11.3.3 1D Navier-Stokes

Next, to confirm the performance of BDPG for nonlinear problems with viscosity, we try a Navier-Stokes case similar to the first Euler case above. The flow starts out supersonic at the inflow (with Mach number of 1.5, corresponding to a density of 1.0,



(a) $p = 1$, Subsonic, left energy flux

(b) $p = 0$, Supersonic, right sum of fluxes

(c) $p = 1$, Supersonic, right sum of fluxes

(d) $p = 2$, Supersonic, right sum of fluxes

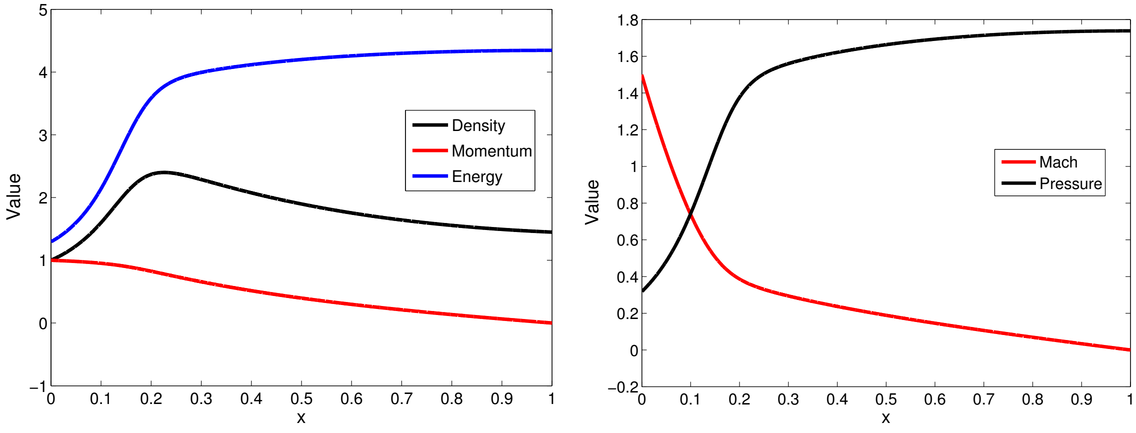
Figure 6.23: 1D Euler: Various flux outputs for subsonic and supersonic cases. (Note that "sum of fluxes" means all state components of the flux are summed to compute the output. The individual flux components converge similarly.) The supersonic BDPG runs achieve the optimal $2p + 2$ rate, while the subsonic BDPG run comes close to this rate. Standard DG obtains only $2p + 1$ rates for all cases.

momentum of 1.0, and energy of 1.2936) and becomes subsonic as it collides with an isothermal wall on the right boundary (with temperature of 3.0). The viscosity is chosen such that the inflow Reynolds number is 10, and the source term

$$\mathbf{S}^T = [\,0.3\,\rho^2 \;\; 0.1\,(\rho u)^2 \;\; 0.1\,(\rho E)^2\,], \tag{6.75}$$

is added to the left-hand side of the equations to enable a steady-state solution. Fig. 6.24 shows the corresponding solution profiles throughout the domain.

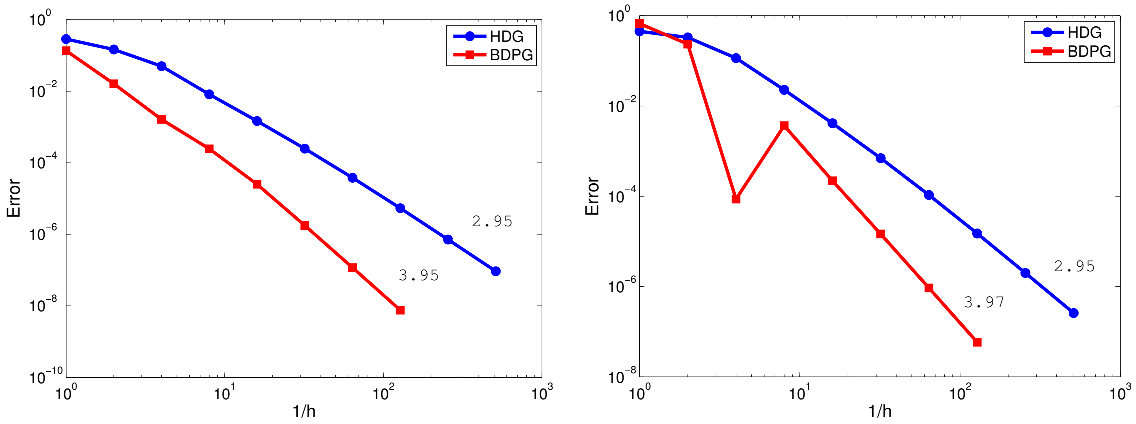
The boundary flux convergence for both HDG and hybrid BDPG is shown in Fig. 6.25, where the same test space properties are used for BDPG as in the Euler cases. We see that BDPG again outperforms HDG, and attains the optimal rate of $2p+2$ in the fluxes on both the left and right boundaries.



(a) Conservative states

(b) Pressure and Mach number

Figure 6.24: 1D Navier-Stokes: (a) State variables within the domain. (b) Mach number and pressure variation within the domain. Note that the flow transitions from supersonic to subsonic near the inflow.



(a) Right x-momentum flux

(b) Left energy flux

Figure 6.25: 1D Navier-Stokes: Convergence rates for a mixed supersonic/subsonic flow with $p=1$. BDPG obtains optimal $2p+2$ rates.

6.11.4 Nonlinear Two-Dimensional Problems

Next, we discuss nonlinear problems in two dimensions. When shifting from one dimension to multiple dimensions, we again encounter the issue that – if BDPG is to provide any benefit – the trial space on element boundaries must be able to represent the fluxes well. If it cannot, then BDPG will perform no better than HDG in terms of accuracy (and would be more expensive due to the test function computations). In general then, in order for BDPG to be worth pursuing in multiple dimensions, some type of basis optimization in which the trial space is tuned to include the primary "modes" of the true fluxes must be performed.

This optimization of the trial space remains a topic for future work. However, what we *can* show at this point is that, if the fluxes are well-represented – which for now we can ensure by adding additional "edge modes" to the trial basis – then BDPG can attain $2p_I + 2$ rates for nonlinear problems in multiple dimensions as well. Here, as before, p_I is the order spanned by the trial basis over the element interior. A standard HDG method, on the other hand, would have a maximum order of $2p_I + 1$, regardless of how many edge modes are added to the basis.

6.11.4.1 2D Euler

In order to show that BDPG can attain $2p_I + 2$ rates with well-represented fluxes, we solve a two-dimensional Euler problem. This problem consists of inviscid, compressible flow through a venturi at a Mach number of 0.5. (See Fig. 6.26 for pressure contours and a more detailed problem specification.)

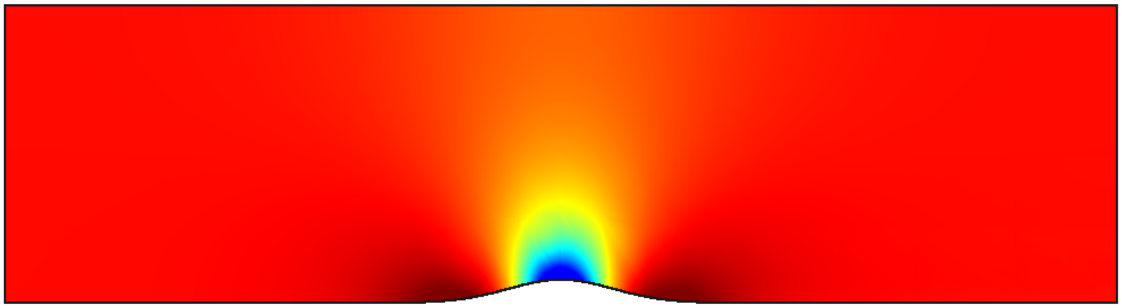


Figure 6.26: Pressure contours (blue=low, red=high) from a BDPG solution with $p_I = 2$, $p_B = 3$, and $p_{\text{test}} = 3$. The flow moves from left to right between inviscid walls at the top and bottom of the domain. A subsonic, horizontal inflow is specified at the left boundary with Mach number $M = 0.5$, total temperature $T_0 = 1.05$, and total pressure $p_0 = 1.1862$. A static-pressure outflow is specified at the right boundary with $p = 1.0$.

For both BDPG and HDG, we choose a (full-order) interior basis of $p_I = 2$. The

$p_I = 2$ trial basis functions on the unit reference triangle are given by

$$\begin{aligned}
\phi_1 &= 1 - \xi - \eta \\
\phi_2 &= \xi \\
\phi_3 &= \eta \\
\phi_4 &= -\sqrt{6}\,\xi\eta \\
\phi_5 &= \sqrt{6}\,(-1 + \xi + \eta)\,\eta \\
\phi_6 &= \sqrt{6}\,(-1 + \xi + \eta)\,\xi\,,
\end{aligned} \tag{6.76}$$

where ξ and η are coordinates in the two-dimensional reference space.

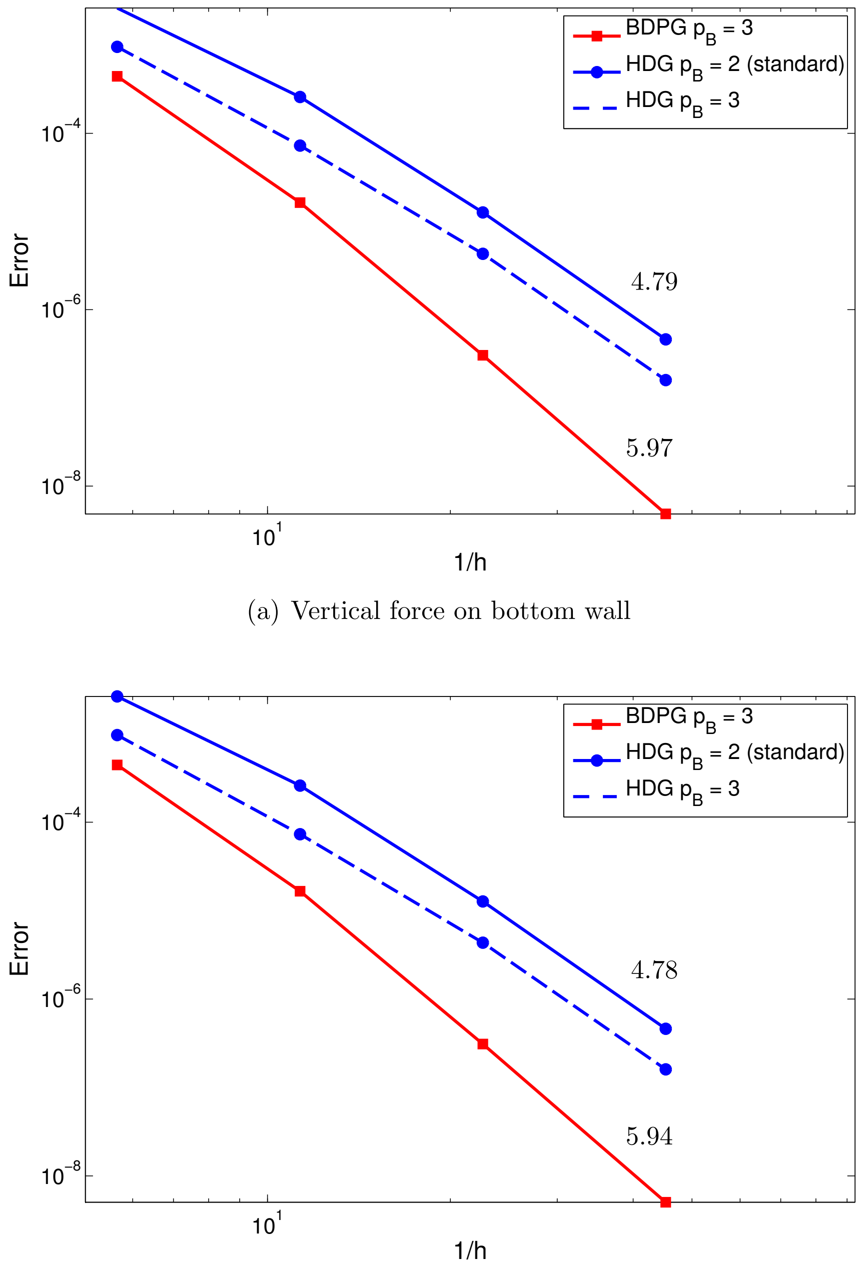
Next, we add $p_B = 3$ edge modes to this basis, which, after being linearly blended into the element interior, are given by

$$\begin{aligned}
\phi_7 &= -\sqrt{10}\,\xi\eta(\eta - \xi) \\
\phi_8 &= -\sqrt{10}\,(-1 + \xi + \eta)\,\eta(-1 + \xi + 2\eta) \\
\phi_9 &= \sqrt{10}\,(-1 + \xi + \eta)\,\xi(2\xi - 1 + \eta)\,.
\end{aligned} \tag{6.77}$$

Each of these basis functions corresponds to a different edge of the reference triangle and allows for a cubic polynomial to be represented along the edge.

For HDG, we then solve the problem as usual, employing this $p_I = 2, p_B = 3$ basis for both the trial and test spaces. For BDPG, we employ this $p_I = 2, p_B = 3$ basis for the trial space, but compute the optimal test functions from a full $p_{\text{test}} = 3$ space, which is chosen to match the order of p_B. (Note that it is not worth computing the test functions more accurately than this, since we are limited by the interpolation error on the edges.) Since we have 9 trial basis functions and 4 state components, we compute a total of 36 test functions on each element. While these problems are purely local, they represent a relatively significant computational and storage cost. Contours of the optimal test functions corresponding to a single trial basis function on each element are shown in Fig. 6.28.

The convergence rates of the vertical forces on both top and bottom walls are shown in Fig. 6.27. Looking at HDG first, we see that when $p_I = 2, p_B = 2$ (so that we are solving a standard $p = 2$ HDG problem), the force convergence rates approach $2p_I + 1$, as expected. When we then add the $p_B = 3$ edge modes to the basis, the error values decrease slightly, but the convergence rate remains $2p_I + 1$. This is because the test space of HDG does not take advantage of the extra resolution along the boundaries.



(a) Vertical force on bottom wall

(b) Vertical force on top wall

Figure 6.27: 2D Euler: Convergence rates for BDPG and HDG. All runs have interior order $p_I = 2$, along with the specified edge order p_B. For the BDPG runs, we take $p_{\text{test}} = p_B$. Note that BDPG achieves $2p_I + 2$ rates when 3rd-order (or higher) edge modes are added to the trial space. On the other hand, HDG remains at $2p_I + 1$ when these same 3rd-order edge modes are added.

On the other hand, we see a different result with BDPG. Here, when we add the $p_B = 3$ edge modes to the $p_I = 2$ trial basis, we obtain the anticipated $2p_I + 2$

rates. This is because the fluxes along the element boundaries are well-resolved by the $p_B = 3$ modes, and the BDPG test space then requests accuracy in those fluxes. The limiting factor is then the $\mathcal{O}(h^{2p_I+2})$ linearization error.

Overall then, we have confirmed that BDPG obtains $2p_I + 2$ rates for nonlinear problems if the fluxes along element boundaries are well-resolved. While this would certainly represent an improvement over a standard DG method, when compared to an HDG method, this fact alone does not make BDPG worth pursuing. This is because, since the p_B degrees of freedom are globally coupled, in terms of cost, we should really be comparing the $p_I = 2, p_B = 3$ BDPG scheme to a full $p = 3$ HDG scheme. But a $p = 3$ HDG scheme, converging at order $2p + 1 = 7$, would converge at an even greater rate than the $2p_I + 2 = 6$th order of BPDG.

Thus, ultimately, whether the cost of computing the optimal test functions is worth it for nonlinear problems in multiple dimensions will depend on whether a successful strategy for trial space optimization along element boundaries can be developed. This optimization would reduce the number of required edge degrees of freedom for BPDG, potentially allowing it to use fewer globally coupled degrees of freedom than HDG.

6.12 Optimal Test Function Summary

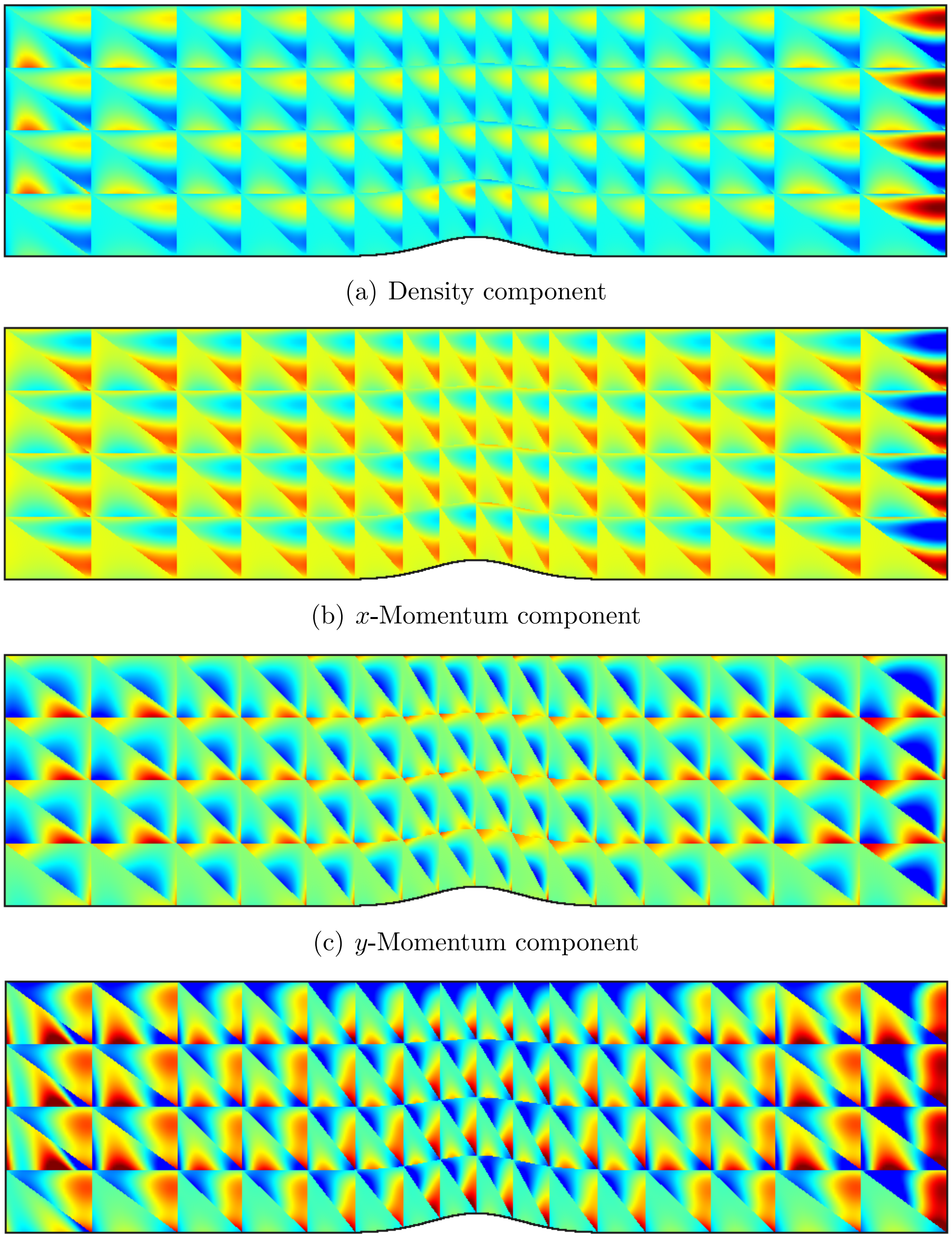
Here, we provide a brief summary of BDPG / optimal test function properties for general problems.

1. Optimal test functions can be computed in an element-local manner within existing DG/HDG formulations.

2. For 1D linear problems, if the optimal test functions are computed in an order-p_{test} space, the boundary flux accuracy will be at least order $p_{\text{test}}+p+1$. In cases where the local test space includes the global flux adjoints (such as advection), the boundary flux accuracy will be order $2p_{\text{test}} + 1$.

3. For 1D nonlinear problems, if the optimal test functions are computed in an order $p_{\text{test}} = p+1$ space, order $2p+2$ convergence rates can be achieved. (This is one order of accuracy higher than the order $2p+1$ rates of standard DG/HDG.)

4. For multi-dimensional linear problems, if the optimal test functions *and* interface fluxes are well-represented, exact boundary fluxes are obtained.



(a) Density component

(b) x-Momentum component

(c) y-Momentum component

(d) Energy component

Figure 6.28: 2D Euler: Optimal test function contours. The trial basis here has $p_I = 2$, $p_B = 3$, with $n_U = 9$. Accounting for the 4 state components then means that we have 36 trial basis functions on each element. We thus compute a corresponding 36 optimal test functions on each element. Plotted here are the state components of *one* of these optimal test functions. Specifically, we show the density, x-momentum, y-momentum, and energy components of an optimal test function associated with the energy component of the basis function ϕ_9 on each element.

   (a) To ensure that the exact multi-dimensional fluxes are representable, enrichment of the trial space basis with order-p_B "edge modes" can be performed.

   (b) Alternatively, a trial space optimization along element boundaries could be developed to incorporate the dominant "modes" of the true fluxes within the trial space basis.

5. For multidimensional nonlinear problems with an element-interior trial basis of order p_I and a boundary enrichment order of p_B, optimal test functions lead to $2p_I + 2$ flux rates if $p_B > p_I$ and $p_{\text{test}} = p_B$.

   (a) As in the linear case, a trial space optimization could be performed instead of boundary enrichment to achieve $2p_I + 2$ flux rates.

6.13 Cost of Optimal Test Functions

We next give a summary of boundary flux accuracy vs. global degrees of freedom (DOF) for DG, HDG, BDPG, and hybrid BDPG (which we will denote here by HBDPG) schemes. We assume inviscid problems for simplicity, so that DG fluxes converge at order $2p + 1$. We also assume smoothness of the optimal test functions.

1. For **1D linear** problems:

   (a) DG: Flux rates of $2p + 1$. Global DOF count $\sim p/h$.

   (b) BDPG: Flux rates of (at least) $p_{\text{test}} + p + 1$. Global DOF count $\sim p/h$.

   (c) HDG: Flux rates of $2p + 1$. Global DOF count $\sim 1/h$.

   (d) HBDPG: Flux rates of (at least) $p_{\text{test}} + p + 1$. Global DOF count $\sim 1/h$.

   Thus, for 1D linear problems, BDPG can achieve arbitrary boundary flux accuracy independent of p – and hence, independent of the global system size. This is a definite improvement over DG methods. On the other hand, HDG can also achieve flux accuracy independent of the global system size, so the difference between HDG and either BDPG or HBDPG would come down primarily to the cost of the local solves and ease of implementation.

2. For **1D nonlinear** problems:

   (a) DG: Flux rates of $2p + 1$. Global DOF count $\sim p/h$.

   (b) BDPG: Flux rates of $2p + 2$. Global DOF count $\sim p/h$.

(c) HDG: Flux rates of $2p+1$. Global DOF count $\sim 1/h$.

(d) HBDPG: Flux rates of $2p+2$. Global DOF count $\sim 1/h$.

For 1D nonlinear problems, BDPG gives improved flux convergence over DG for the same global system size. Likewise, HBDPG also gives improved flux convergence over HDG for the same global system size. Of course, since both HDG and HBDPG can achieve flux accuracy independent of the global DOF, the advantage of this is not always clear. An effective strategy may be to first solve the nonlinear problem using HDG, then "post-process" this HDG solution by computing the optimal test functions once and re-converging the residuals. This would yield an additional order of accuracy for minimal computational cost.

3. For **multi-dimensional linear** problems (where p_B is the edge enrichment order and p_I is the interior order):

   (a) DG: Flux rates of $2p+1$. Global DOF count $\sim p^{dim}/h^{dim}$.

   (b) BDPG: Flux rates of $2p_B+1$ (if $p_{\text{test}} = p_B$). Global DOF count $\sim (p_B^{dim-1} + p_I^{dim})/h^{dim}$.

   (c) HDG: Flux rates of $2p+1$. Global DOF count $\sim p^{dim-1}/h^{dim}$.

   (d) HBDPG: Flux rates of $2p_B+1$ (if $p_{\text{test}} = p_B$). Global DOF count $\sim p_B^{dim-1}/h$.

   For multidimensional linear problems, since the flux accuracy of BDPG depends only on the boundary order p_B, BDPG (at a minimum) provides a similar global DOF reduction as HDG when compared with standard DG. However, in general, a trial basis optimization along element boundaries would be required to make either BDPG or HBDPG more efficient than HDG itself.

4. For **multi-dimensional nonlinear** problems (where p_B is the edge enrichment order and p_I is the interior order):

   (a) DG: Flux rates of $2p+1$. Global DOF count $\sim p^{dim}/h^{dim}$.

   (b) BDPG: Flux rates of $2p_I+2$ (if $p_{\text{test}} = p_B = p_I+1$). Global DOF count $\sim (p_B^{dim-1} + p_I^{dim})/h^{dim}$.

   (c) HDG: Flux rates of $2p+1$. Global DOF count $\sim p^{dim-1}/h^{dim}$.

   (d) HBDPG: Flux rates of $2p_I+2$ (if $p_{\text{test}} = p_B = p_I+1$). Global DOF count $\sim p_B^{dim-1}/h$.

Here again, for multi-dimensional nonlinear problems, while an additional order of accuracy can be gained by BDPG, in general a trial space optimization would be required to give a potential benefit over HDG.

6.14 Remaining Challenges

Overall, the results shown in the above sections are encouraging, and verify the concept of using local optimal test functions to achieve global boundary accuracy. That said, before BDPG sees more widespread application, a few challenges remain.

The first of these, as mentioned, is the issue of trial space resolution near element boundaries. In the present work, we added order-p_B Lobatto functions to the trial space to ensure that the fluxes are well-represented. While this is an effective strategy for primal DG formulations, for an already-hybridized method it represents a relatively large computational expense, since these additional degrees-of-freedom are globally coupled. Thus, for most problems, to make hybrid BDPG more efficient than standard HDG a local optimization of the trial space near element boundaries is required. While a preliminary attempt at trial space optimization is shown in Appendix E, in general this remains an open problem.

An additional challenge is that, for nonlinear problems, recomputing the test functions during the Newton iteration can make this iteration less robust. One simple way to avoid this issue is to first solve a standard DG/HDG problem, then compute the optimal test functions just once – based on a linearization performed about the DG/HDG solution. While this can lead to slightly larger errors in the fluxes, the same $2p + 2$ rates can be achieved.

Finally, another important issue, which has not yet been emphasized, is the representation of the test functions themselves. For certain problems, the optimal test functions can exhibit nonsmooth behavior that makes their approximation difficult. Nonsmoothness of the test functions arises, for instance, for pure advection problems in two dimensions. In this case, the exact local adjoints (test functions) contain discontinuities within each element. Since polynomials cannot adequately resolve these discontinuities, the error between the discrete and exact optimal test functions can be large. This leads to errors in the elementwise fluxes, which propagate globally. A similar issue arises for high Reynolds number advection-diffusion cases, where – rather than discontinuities – steep boundary layers appear in the test functions.

These issues exist for many multiscale methods (including other DPG schemes, as discussed in e.g. [25, 21]), and there are various means of addressing them. One option is the use of a “subgrid” within each element to resolve the relevant fine-scale features. However, for BDPG methods, an alternative option presents itself. For most cases – as mentioned in Sec. 6.6.2 – it is only necessary to resolve the test functions on the *boundaries* of each element. Thus, if test function discontinuities or boundary layers exist *inside* a given element, these features do not actually need to be resolved. Therefore, when computing the optimal test functions, rather than using a standard DG or HDG method, we could instead atttempt to tailor the discretization to focus solely on obtaining boundary accuracy in the test functions. Indeed, since achieving boundary accuracy has been the primary goal of this work, it may be possible to apply some of the present ideas to the test function problem itself.

6.15 Conclusion

In this chapter, we presented a strategy for optimizing the test space of both primal and hybrid DG methods. The theory applies to linear PDEs and can be extended to nonlinear equations. We have shown that if the primary goal is to achieve boundary accuracy, the optimal test functions can be localized and computed independently on each element in the mesh. These test functions satisfy local adjoint equations and ensure that a proper upwinding of information occurs within each element. As shown, if the problem is linear and both test functions and fluxes are well-represented, exact boundary fluxes are obtained. The resolution of both test functions and fluxes are critical issues, and while we have addressed certain aspects of these issues, additional challenges remain.

CHAPTER VII

Conclusions and Future Work

7.1 Summary and Conclusions

In this section, we summarize the work performed and the conclusions drawn in this thesis, broken down into the two main topics addressed.

7.1.1 Unsteady Output-based Error Estimation and Mesh Adaptation

In this thesis, we have developed and tested an output-based mesh adaptation strategy for unsteady problems on deforming domains. This strategy relies on the solution of an unsteady adjoint problem, and reduces output errors by performing dynamic-p adaptation in space and slab refinement/coarsening in time. An Arbitrary Lagrangian-Eulerian (ALE) DG method was implemented to perform the deforming-domain simulations, along with a Geometric Conservation Law (GCL) to ensure conservation. The satisfaction of the GCL (and the introduction of a separate GCL variable) required the introduction of a corresponding GCL *adjoint*, which was solved alongside the state adjoint.

Results from low-Reynolds-number flapping-flight simulations in both two and three dimensions demonstrate that the output-based strategy outperforms more common strategies, including a residual-based method and uniform-h and -p refinements. For a given level of output accuracy, the output-based method required mesh sizes roughly 100 times smaller than either h- or p-refinements. Reductions in computational time were also observed – the largest being a factor of 5-10 reduction in wall time relative to the uniform refinement strategies. Furthermore, in every case simulated, the residual-based adaptation failed to converge to the true output value, indicating that this cheaper indicator is not adequate for these problems.

In addition, conclusions were drawn about the relevance of the GCL to output accuracy. For most problems, satisfaction of the GCL led to only a small improvement in output accuracy. However, if the GCL is employed, it is necessary to incorporate a GCL adjoint into the error estimation procedure, since errors associated with the GCL equation can make up more than half of the total output error during the final adaptive iterations.

7.1.2 Optimal Test Functions

In the latter half of the thesis, we returned to steady-state problems and investigated the idea of optimizing the test space of discontinuous finite element methods to achieve accuracy in quantities of interest. In particular, our goal was to achieve accuracy along domain *boundaries*. We showed that the test space of standard DG and HDG methods can be optimized to that end, and that this optimization can be performed in an element-local manner. We call the resulting method a boundary discontinuous Petrov-Galerkin (BDPG) method.

The optimal test functions satisfy local adjoint equations and ensure that a proper upwinding of information occurs within each element. For linear problems, if both test functions and fluxes are well-represented, exact boundary fluxes are obtained. In multiple dimensions, the use of optimal test functions within a DG method results in a similar degree-of-freedom reduction as HDG. For these cases, however, making BDPG more efficient than HDG would in general require a trial space optimization on element boundaries. This is a topic for future work.

For nonlinear problems, we showed that BDPG can achieve boundary-flux rates of $2p + 2$, which is one order higher than the $2p + 1$ rates of standard DG/HDG methods. However, once again, to improve upon HDG in multiple dimensions, some type of trial space optimization would be required. Additional topics for future work are mentioned in the following section.

7.2 Future Work

Some ideas for future work related to the topics in this thesis are given below.

1. **Unsteady Output-based Error Estimation and Mesh Adaptation**

   - **Extension to other temporal discretizations:** As mentioned in Sec. 5.9, the error estimation and adaptation techniques demonstrated here for DG-in-time can be extended to more standard time schemes such as Runge-Kutta or multi-step schemes. While some multi-step methods (such as the second-order backward difference method) are adjoint-inconsistent for variable time-step sizes [85], this is not an issue for Runge-Kutta methods [84]. Furthermore, while for Runge-Kutta methods there is some question over how to best represent the data for purposes of error estimation (due to the non-variational nature of the method), fitting a polynomial interpolant through some combination of the discrete nodal/stage/derivative values associated with these schemes should suffice. Some preliminary results to this effect look promising.

- **Unsteady *hp*- and *r*-adaptation:** As mentioned in Sec. 5.9, while the dynamic-p adaptation considered here is adequate for smooth problems, for flows with sharp gradients (such as high-Reynolds-number flows or flows with shocks), it would be useful to incorporate both h and p adaptation in space. Alternatively, node movement (r-adaptation) could be performed. The adaptation in time could also be improved by refining time step sizes uniquely for each spatial element, rather than refining entire time slabs. The challenge is to implement these options in a robust and efficient manner within an output-based framework.

- **Extension to chaotic problems:** As discussed in Sec. 5.9, another goal is to extend these output-based techniques to chaotic flows. In these flows, outputs are extremely sensitive to perturbations in the residuals, causing the magnitude of the adjoint to grow unbounded as it is marched backward in time. However, for suitably averaged outputs, it should be possible to obtain meaningful sensitivity data with respect to certain input parameters. Several ideas (such as the *Least Squares Shadowing Method*) have been put forward in the literature regarding the most effective way to define the adjoint for these problems [98, 99], and a successful resolution of this issue will be critical to many engineering applications.

- **Accounting for inter-element error cancellation:** Though not emphasized in this work, for both steady and unsteady problems, if the adjoint for an output has e.g. a discontinuity on a certain element, then the local output error estimate on that element will converge sub-optimally, at $\mathcal{O}(h^{p+1})$. (This makes sense mathematically, since the local adjoint and residual convergence rates will be order 1 and p, respectively, thus summing to $p + 1$.) However, oftentimes, the total output error will still superconverge at a rate of $2p+1$, indicating that the local $\mathcal{O}(h^{p+1})$ output errors must be cancelling as they propagate between elements. Some efficiency may be gained if the output-based adaptation were to take this inter-element cancellation into account, since it would not target these individual elements as aggressively.

2. **Optimal Test Functions**

   - **More rigorous proofs of BDPG properties:** We have presented the optimal test functions and corresponding BDPG method here as more of a "proof-of-concept" than a formal method backed by rigorous proofs of stability and convergence. While many of the ideas here can be mapped into a more formal setting (for example, the optimal test functions can be viewed as making the *inf-sup* and *continuity* constants equal to unity with respect to a desired norm – as in other DPG literature [25, 26]), more rigorous proofs of the optimality of the test-function localization and the choice of boundary weight should be performed. Furthermore, for viscous problems, a more formal means of choosing the stabilization parameters (such as the viscous length scale) should be determined.
   - **Cheaper computation of optimal test functions:** Though the test functions are computed in an element-local manner, for systems of equations and high-order trial spaces, their associated computation and storage costs can be large. Finding a means to reduce these costs would be advantageous. In certain cases (for example, for one-dimensional problems), only one or two optimal test functions on each element are relevant to achieving boundary accuracy, while the rest provide interior accuracy. Thus, to reduce costs in these cases, we could compute only those test functions relevant for boundary accuracy. Furthermore, in general, since the number of test functions that must be computed is identical to the number of trial functions, if a trial-space optimization is performed (particularly in multiple dimensions), this could significantly reduce both the number of trial functions *and* the number of test functions required on each element.
   - **Resolution of non-smooth test functions:** As mentioned in Sec. 6.14, for certain problems (such as two-dimensional advection), the optimal test functions contain singularites. A better method for resolving these singularities – such as the use of a "subgrid" within each element – could be

developed. We note however that if the primal problem has no element-interior source terms, accuracy in the test functions is required only along element *boundaries*. While a singularity such as a discontinuity could still exist on the element boundaries, this fact alleviates many of the representation issues.

- **Optimization of the trial space:** As discussed, to make BDPG / optimal test functions more efficient than HDG in multiple dimensions, some type of optimization of the trial space near element boundaries is required. While we have demonstrated preliminary trial space optimization results for two-dimensional advection in Appendix E, much work remains to be done. Trial-space optimization is an open-ended problem even for standard DG/HDG methods, and many strategies can be imagined.
- **Improvement in nonlinear BDPG rates:** For nonlinear problems, the current BDPG convergence rates are limited to $2p+2$ due to the linearization error associated with the local adjoint (i.e. test function) problems. While this is one order of accuracy higher than a standard DG method, it may be possible to improve on this rate. Specifically, from the theory of error estimation, it is known that by using a combined primal-dual form of the error estimate, the second-order ($2p+2$) linearization error can be improved to third-order ($3p+3$). (See Appendix A or [10].) Thus, it may be possible to improve the nonlinear BDPG convergence rates to $3p+3$ if some type of local primal-dual problem is solved.
- **Extension to unsteady problems:** It is possible to extend BDPG ideas to unsteady problems as well. If a space-time finite element method were used, the ideas would carry over directly, since the theory would be the same as for a multi-dimensional steady-state problem. It may also be possible to make an effective time-marching scheme based on the idea of optimal test functions. Since the temporal direction is one-dimensional, computing optimal test functions in time could give similar results as for (e.g.) a one-dimensional, steady advection problem.
- **Targeting of outputs/regions besides boundaries:** While we have focused on achieving accuracy in boundary outputs/distributions here, the idea of optimizing a numerical method (whether a finite element method or otherwise) to achieve accuracy in specific outputs could be pursued in a more general context. Pursuing boundary accuracy here turned out to be

the most practical goal, not only because it is often of engineering interest, but also because it enabled a straightforward localization of the optimal test functions.

APPENDICES

APPENDIX A

Output Error Estimation: Additional Details, Including Dual and Primal-Dual Forms

In this appendix, we derive some additional forms of the output error estimate discussed in Chapter II. In particular, we derive both "dual" and "primal-dual" forms of the error estimate, and discuss their use within the context of linear and nonlinear Galerkin methods.

A.1 Dual Form of Output Error Estimate

Assume we are solving a differential equation $Lu = f$ and are interested in an output written as

$$J = (g, u)\,. \tag{A.1}$$

If we have some approximate solution u_H, the output is approximately

$$J(u_H) = (g, u_H)\,. \tag{A.2}$$

Recall from Sec. 2.4.1 that the amount of error in this output is then given by

$$\delta J(u_H) = -(r(u_H), \psi) \;=\; -\int\limits_{\Omega} \psi\, r(u_H) dx\,, \tag{A.3}$$

where $r(u_H) = Lu_H - f$ is the residual evaluated with the approximate solution and ψ is the (exact) adjoint for J, which satisfies the adjoint equation

$$L^*\psi = g\,. \tag{A.4}$$

While Eqn. A.3 is often the most useful, we can derive a slightly different form of the output error. In Chapter II, we showed how an output $J = (g, u)$ can be equivalently written in "dual form" as $J = (f, \psi)$. It stands to reason that if we can write the output itself in two ways, we should also be able to write the output *error* in two ways.

When deriving the so-called "primal" form of the output error (Eqn. A.3), we started with the output definition $J = (g, u)$ and proceeded from there. If we instead start from the relation $J = (f, \psi)$, we can derive a corresponding "dual" form of the output error.

Note that if we knew ψ exactly, the error in the output would be zero, since we could compute the exact output value from the above dual form. However, assume that we do not know the exact adjoint, but instead know only an approximation, ψ_h. This ψ_h could come from numerically approximating the adjoint equations on a given mesh. In general, ψ_h may be computed in a different space (or on a different mesh) than u_H, so we use the subscript h rather than H to emphasize this potential difference. If ψ_h happened to be computed in the same space as u_H, we would write $\psi_h = \psi_H$.

With ψ_h in hand, the output can be computed as $J(\psi_h) = (f, \psi_h)$. The amount of error in this output can then be expressed as:

$$\begin{aligned}
\delta J(\psi_h) &= J(\psi) - J(\psi_h) \\
&= (f, \psi) - (f, \psi_h) && \text{(dual form of output)} \\
&= (f, \psi - \psi_h) && \text{(linearity of inner product)} \\
&= (Lu, \psi - \psi_h) && \text{(primal equation)} \\
&= (u, L^*(\psi - \psi_h)) && \text{(adjoint identity)} \\
&= (u, L^*\psi) - (u, L^*\psi_h) && \text{(linearity of inner product)} \\
&= (u, g) - (u, L^*\psi_h) && \text{(adjoint equation)} \\
&= -(u, L^*\psi_h - g) && \text{(linearity of inner product).}
\end{aligned} \tag{A.5}$$

Now, we know that ψ satisfies $L^*\psi - g = 0$, so the expression $L^*\psi_h - g$ is an ***adjoint residual***, in the same way that the quantitiy $Lu_H - f$ is a primal residual.

If we define this adjoint residual to be:

$$r^*(\psi_h) \equiv L^*\psi_h - g\,,$$

then we can write the above form of the output error as

$$\delta J(\psi_h) = -(u, r^*(\psi_h))\,.$$

Or, by symmetry of the inner product:

$$\boxed{\delta J(\psi_h) = -(r^*(\psi_h), u)}\,. \tag{A.6}$$

This is known as the **dual form of the output error**.

For comparison, recall that the primal form of the output error is

$$\boxed{\delta J(u_H) = -(r(u_H), \psi)}\,. \tag{A.7}$$

We see that the above expressions have the roles of ψ and u switched, but otherwise have the same form. Thus, there is a certain “duality” between them. In summary then: if we have computed a given output using ψ_h, the corresponding output error is given by Eqn. A.6. On the other hand, if we have computed the output using u_H, the corresponding output error is given by Eqn. A.7. Depending on how ψ_h and u_H were obtained, these errors could potentially differ.

In the next section, we will show that if ψ_h is computed in the same space as u_H (so that $\psi_h = \psi_H$) and a Galerkin-type finite element method is used, then the primal and dual forms of the output error are identical. In other words, $\delta J(\psi_H) = \delta J(u_H)$.

A.2 Galerkin Methods: Equivalence of Primal and Dual Forms of Output Error

Above, we derived both a primal and dual form of the output error. For a primal problem $Lu = f$ with corresponding adjoint problem $L^*\psi = g$, we have the output errors:

$$\delta J(u_H) = J(u) - J(u_H) = -(r(u_H), \psi)\,,$$

and

$$\delta J(\psi_h) = J(\psi) - J(\psi_h) = -(r^*(\psi_h), u)\,,$$

where the residuals are given by $r(u_H) = Lu_H - f$ and $r^*(\psi_h) = L^*\psi_h - g$.

In general, since the h and H spaces may differ, there is no reason to think that these error values will be the same. Even if we use the same mesh to compute both primal and adjoint solutions, Eqns. A.6 and A.7 may give different values for δJ depending on the method used. However, it turns out that for Galerkin methods in particular, the two error values *are* in fact identical. In other words, if we compute ψ_H and u_H in the same space using a Galerkin method, then we will have $\delta J(u_H) = \delta J(\psi_H)$.

This is straightforward to show. The Galerkin formulation of both primal and adjoint problems is (respectively):

$$(Lu_H, v_H) = (f, v_H) \qquad \forall v_H \in \mathcal{V}_H \tag{A.8}$$

$$(L^*\psi_H, v_H) = (g, v_H) \qquad \forall v_H \in \mathcal{V}_H, \tag{A.9}$$

where $\mathcal{V}_H$ is a discrete space of choice.

Furthermore, the *exact* solutions u and ψ will satisfy

$$(Lu, v) = (f, v) \qquad \forall v \in \mathcal{V}, \tag{A.10}$$

$$(L^*\psi, v) = (g, v) \qquad \forall v \in \mathcal{V}, \tag{A.11}$$

where $\mathcal{V}$ is an appropriate continuous space. Note that in general we will have $\mathcal{V}_H \subset \mathcal{V}$.

Define the primal error to be $e = u - u_H$ and the adjoint error to be $e^* = \psi - \psi_H$. Next, since $\mathcal{V}_H$ is contained in $\mathcal{V}$, we can choose the test functions in A.10 and A.11 to *be* the discrete ones – i.e. we can take $v = v_H$. We can then subtract equations A.8 and A.9 from A.10 and A.11, respectively. This gives, for the primal problem:

$$\begin{aligned}
(Lu, v_H) - (Lu_H, v_H) &= (f, v_H) - (f, v_H) && \forall v_H \in \mathcal{V}_H \\
(L(u - u_H), v_H) &= 0 && \forall v_H \in \mathcal{V}_H \\
(Le, v_H) &= 0 && \forall v_H \in \mathcal{V}_H, \qquad \text{(A.12)}
\end{aligned}$$

and for the adjoint problem:

$$\begin{aligned}
(L^*\psi, v_H) - (L^*\psi_H, v_H) &= (g, v_H) - (g, v_H) && \forall v_H \in \mathcal{V}_H \\
(L^*(\psi - \psi_H), v_H) &= 0 && \forall v_H \in \mathcal{V}_H \\
(L^* e^*, v_H) &= 0 && \forall v_H \in \mathcal{V}_H .
\end{aligned} \tag{A.13}$$

The relations A.12 and A.13 are statements of primal and adjoint Galerkin orthogonality. They say that the errors e and e^*, after application of the primal and adjoint operators (respectively), are orthogonal to the discrete space $\mathcal{V}_H$.

Now, our goal is to show that for Galerkin methods, we have $\delta J(u_H) = \delta J(\psi_H)$. Since $\delta J(u_H) = (g, u - u_H) = (g, e)$ and $\delta J(\psi_H) = (f, \psi - \psi_H) = (f, e^*)$, we need to show that $(g, e) = (f, e^*)$.

Noting that both e and e^* are $\in \mathcal{V}$ (and hence can be used as test functions in Eqns. A.10 and A.11), we have:

$$\begin{aligned}
(g, e) &= (L^*\psi, e) && \text{(adjoint equation)} \\
&= (Le, \psi) && \text{(adjoint identity and symmetry)} \\
&= (Le, \psi) - \underbrace{(Le, \psi_H)}_{0} && \text{(Galerkin orthogonality)} \\
&= (Le, \psi - \psi_H) && \text{(linearity of inner product)} \\
&= (Le, e^*) && \text{(definition of } e^*\text{)} \\
&= (L(u - u_H), e^*) && \text{(definition of } e\text{)} \\
&= (Lu, e^*) - (Lu_H, e^*) && \text{(linearity of inner product)} \\
&= (Lu, e^*) - \underbrace{(u_H, L^* e^*)}_{0} && \text{(adjoint identity)} \\
&= (Lu, e^*) && \text{(adjoint Galerkin orthogonality)} \\
&= (f, e^*) && \text{(primal equation)} \\
\implies \ (g, e) &= (f, e^*) \, .
\end{aligned} \tag{A.14}$$

Thus, we have shown that for Galerkin methods, both primal and dual forms of the error estimate are identical if ψ_H and u_H are computed in the same space. So we can write the single error esimate δJ as

$$\boxed{\delta J = -(r(u_H), \psi) \ = \ -(r^*(\psi_H), u)} \ . \tag{A.15}$$

Furthermore, as mentioned in Chapter II, by Galerkin orthogonality we can subtract coarse-space approximations ψ_H and u_H from the above estimates, resulting in the following equivalent forms:

$$\boxed{\delta J = -(r(u_H), \psi - \psi_H) \;=\; -(r^*(\psi_H), u - u_H)} \quad . \tag{A.16}$$

A.3 Continuous Error Estimation: Nonlinear Problems

Up until now, we have focused on linear problems. However, in practice, the most relevant problems are nonlinear. This raises the question: how do we perform error estimation for nonlinear problems? Can we use a similar adjoint-based strategy to approximate the error?

The answer is yes. We will show how this is done in the following sections.

A.3.1 A Second-Order Nonlinear Error Estimate

Recall that the generalized adjoint equation (Eqn. 2.35) defines the adjoint as the function ψ satisfying

$$J'_u(\delta u) = \int\limits_{\Omega} \psi\, r'_u(\delta u)\, d\Omega \qquad \forall\,(\text{permissible})\,\delta u\,, \tag{A.17}$$

where J'_u and r'_u denote the variations of the output and residual with respect to u, respectively. For a nonlinear problem, these variations must be taken about a particular state. If we assume that this state is an approximation denoted by u_H, then we can rewrite the adjoint equation as

$$J'_u[u_H](\delta u) = \int\limits_{\Omega} \psi\, r'_u[u_H](\delta u)\, d\Omega \qquad \forall\,(\text{permissible})\,\delta u\,. \tag{A.18}$$

We can use this adjoint definition to derive a second-order error estimate for a given output of interest, $J(u_H)$.

To obtain an output error estimate, we Taylor expand our true output $J(u)$ about the current state u_H as follows:

$$J(u) \;\approx\; J(u_H) \;+\; J'_u[u_H](\delta u) \;+\; O(\delta u^2)\,. \tag{A.19}$$

From Eqn. A.18, the first-order term in this expansion can be written as

$$\begin{aligned} J'_u[u_H](\delta u) &= (\psi,\, r'_u[u_H](\delta u)) && \text{(inner product notation)} \\ &= (r'_u[u_H](\delta u),\, \psi) && \text{(symmetry of inner product)}\,. \end{aligned} \tag{A.20}$$

Thus, from Eqns. A.19 and A.20, we can write the output error as

$$J(u) - J(u_H) \;\approx\; (r'_u[u_H](\delta u),\, \psi) \;+\; O(\delta u^2)\,. \tag{A.21}$$

Our final step is to write the quantity $r'_u[u_H](\delta u)$ in a simpler form. To do this, we expand the true residual $r(u)$ about the current solution u_H, just as we did with J:

$$\begin{aligned} r(u) = 0 \;\approx\; r(u_H) \;+\; r'_u[u_H](\delta u) \;+\; O(\delta u^2) \\ \implies\; r'_u[u_H](\delta u) \;\approx\; -r(u_H) \;+ O(\delta u^2)\,. \end{aligned} \tag{A.22}$$

Substituting Eqn. A.22 into A.21 gives the second-order error estimate:

$$\boxed{J(u) - J(u_H) \;\approx\; -(r(u_H), \psi) \;+\; R^{(2)}}\;, \tag{A.23}$$

where $R^{(2)} = O(\delta u^2)$ is the remainder.

Now, once again, for a Galerkin discretization the weak form of the primal equation is

$$\int\limits_{\Omega} r(u_H)\, v_H \, dx \;=\; (r(u_H), v_H) \;=\; 0 \qquad \forall v_H \in \mathcal{V}_H\,. \tag{A.24}$$

Therefore, if we have a coarse-space approximation to the adjoint, denoted by ψ_H, the quantity $(r(u_H), \psi_H)$ will be 0 and can be subtracted from the above error estimate to no effect, allowing us to write it as:

$$\boxed{J(u) - J(u_H) \;\approx\; -(r(u_H), \psi - \psi_H) \;+\; R^{(2)}}\;. \tag{A.25}$$

Overall, this error estimate looks exactly like the linear one derived in Chapter II, except that we now have an $O(\delta u^2)$ ***error*** in the error estimate.

A few points worth emphasizing are:

- It is important to note that the ψ in the above expressions is the exact adjoint for the *inexact* linearization $r'_u[u_H](\delta u)$. In other words, for the current state

u_H, it is the adjoint solution that would be obtained by solving the adjoint equations on an infinitely fine mesh. It is *not*, however, the "true" adjoint solution, unless the approximate solution u_H happens to be the exact solution u.

- The $O(\delta u^2)$ error that comprises the remainder is a *linearization* error, which arises due to the fact that we kept only the first two terms in the Taylor expansions of $J(u)$ and $r(u)$. If $J(u)$ and $r(u)$ are both linear, this error will disappear, and we will recover our earlier (linear) form of the error estimate. However, if either $J(u)$ or $r(u)$ is nonlinear, then $R^{(2)}$ will be nonzero, and the error estimate will be inexact.

- The question then arises: can we eliminate the $R^{(2)}$ remainder and obtain a more accurate error estimate? One obvious way to do this is to simply keep more terms in the expansions of $J(u)$ and $r(u)$. However, this is an expensive proposition and would require computation of the primal Hessian $r''_u[u_H](\delta u)$. It turns out that there is a better way to eliminate $R^{(2)}$. This is described in the next section.

A.3.2 A Third-Order Nonlinear Error Estimate for Galerkin Methods

In this section, which follows the work of Becker and Rannacher [10], we describe how to eliminate the second-order remainder $R^{(2)}$ and hence obtain a third-order error estimate. We assume that a Galerkin discretization method is used, so that the equations being solved can be written as

$$\int_{\Omega} v_H \, r(u_H) \, d\Omega = 0 \qquad \forall v_H \in \mathcal{V}_H = \mathcal{U}_H \, . \tag{A.26}$$

In order to set the stage for the derivation, we will first need to review some calculus.

From basic calculus, the change of a function $f(x)$ over an interval $[a, b]$ can be written as:

$$f(b) - f(a) = \int_a^b f'(s) ds \, , \tag{A.27}$$

where the prime denotes a standard derivative. If we take $a = x$ and $b = x + \Delta x$, this becomes:

$$f(x+\Delta x) - f(x) = \int\limits_{x}^{x+\Delta x} f'(s)ds. \tag{A.28}$$

Now, we can parameterize the variable s by a new variable t, which goes from 0 to 1 over the interval $[x, x+\Delta x]$. Taking $s = x + t\Delta x$ and rewriting the above equation in terms of t gives:

$$f(x+\Delta x) - f(x) = \left[\int\limits_0^1 f'(\underbrace{x+t\Delta x}_{s})\, dt\right] \underbrace{\Delta x}_{\frac{ds}{dt}}. \tag{A.29}$$

Thus, we get the following representation for the change in the function f:

$$\boxed{f(x+\Delta x) - f(x) = \left[\int\limits_0^1 f'(x+t\Delta x)\, dt\right] \Delta x} \quad . \tag{A.30}$$

The term in brackets is the "mean value" of the slope of f between x and $x+\Delta x$. In other words, in a plot of f vs. x, it is just the slope of the line connecting $f(x)$ and $f(x+\Delta x)$.

Above, f was a function of the single variable x. If instead f depends on two variables (say x and y), we can write a similar formula for the change of f between the points (x, y) and $(x+\Delta x, y+\Delta y)$:

$$\boxed{\begin{aligned} f(x+\Delta x, y+\Delta y) - f(x,y) &= \left[\int\limits_0^1 \frac{\partial f}{\partial x}(x+t\Delta x, y+t\Delta y)\, dt\right] \Delta x \\ &+ \left[\int\limits_0^1 \frac{\partial f}{\partial y}(x+t\Delta x, y+t\Delta y)\, dt\right] \Delta y \end{aligned}} \quad . \tag{A.31}$$

Here, the variable t again goes from 0 to 1 as we traverse the diagonal between (x, y) and $(x+\Delta x, y+\Delta y)$.

We now need to make one further generalization. What if, rather than a function, f is actually a *functional*, which depends now on two *functions* x and y (as opposed to coordinates)? Let us call this functional F rather than f. Then our new goal is to find how F changes when we perturb the functions x and y from the original "point"

(x, y) to some new "point" $(x + \delta x, y + \delta y)$. Note that we use a lowercase δ to denote that these perturbations are now *variations* rather than scalar values.

While functionals cannot be differentiated in the classic sense, they *do* have Fréchet derivatives. And we can use these Fréchet derivatives of $F(x, y)$ to determine how it changes when we go from (x, y) to $(x + \delta x, y + \delta y)$. We denote the Fréchet derivative of F with respect to x as

$$F'_x[\cdot, \cdot](\delta x), \tag{A.32}$$

and the Fréchet derivative of F with respect to y as

$$F'_y[\cdot, \cdot](\delta y), \tag{A.33}$$

where the terms in brackets denote the "point" in state space about which F is being linearized. For example, the Fréchet derivative of F with respect to x at the point (x_0, y_0) would be denoted by $F'_x[x_0, y_0](\delta x)$.

Since the Fréchet derivatives play a similar role as the multi-dimensional derivatives in Eqn. A.31, we can write our formula for the change in F as follows:

$$\boxed{\begin{aligned} F(x + \delta x, y + \delta y) - F(x, y) = & \int_0^1 F'_x \left[x + t\,\delta x, y + t\,\delta y\right](\delta x)\, dt \\ & + \int_0^1 F'_y \left[x + t\,\delta x, y + t\,\delta y\right](\delta y)\, dt \end{aligned}}\ , \tag{A.34}$$

where again t parameterizes the space between (x, y) and $(x + \delta x, y + \delta y)$. We will use this formula to obtain a **third-order output error estimate**.

Assume that we have obtained a state approximation $u_H \in \mathcal{U}_H$ and an adjoint approximation $\psi_H \in \mathcal{U}_H$, which have been computed using Galerkin discretizations of the primal and adjoint problems, respectively. Note that the Galerkin discretization of the adjoint problem (linearized about u_H) is defined by

$$J'_u[u_H](\delta v) = \int_\Omega \psi_H\, r'_u[u_H](\delta v)\, d\Omega \qquad \forall\, \delta v \in \mathcal{U}_H\,, \tag{A.35}$$

for some discrete space $\mathcal{U}_H$.

In the following derivation, we will also make use of the *true* adjoint, ψ, which is associated with the exact solution u and defined to satisfy

$$J'_u[u](\delta v) = \int_\Omega \psi\, r'_u[u](\delta v)\, d\Omega \qquad \forall\, \delta v \in \mathcal{U}\,. \tag{A.36}$$

Note that the definition of ψ here is different than in the previous section, where ψ represented the infinite-dimensional adjoint for the *inexact* state u_H.

To obtain an error estimate for some output of interest $J(u_H)$, we then construct two functionals, $F(u, \psi)$ and $F(u_H, \psi_H)$:

$$F(u,\psi) \;\equiv\; J(u) - \underbrace{\int_\Omega \psi\, r(u)\, d\Omega}_{0 \text{ since } r(u)=0} \tag{A.37}$$

$$F(u_H,\psi_H) \;\equiv\; J(u_H) - \underbrace{\int_\Omega \psi_H\, r(u_H)\, d\Omega}_{0 \text{ by Galerkin orthogonality}} \tag{A.38}$$

Since the adjoint-weighted residual terms in the above equations are exactly zero, we have that:

$$F(u,\psi) - F(u_H,\psi_H) = J(u) - J(u_H)\,.$$

Next, we define the difference between the true and approximate adjoints to be $\delta\psi = \psi - \psi_H$ and the difference between the true and approximate states to be $\delta u = u - u_H$. We then have that $\psi = \psi_H + \delta\psi$ and $u = u_H + \delta u$, which means the above formula can be written:

$$F(u_H + \delta u, \psi_H + \delta\psi) - F(u_H,\psi_H) = J(u) - J(u_H)\,.$$

We have now expressed the output error in terms of a change in some functional F, and from Eqn. A.34, we know a formula for computing this change. Just swapping

$[x, y] \to [u_H, \psi_H]$ and $[\delta x, \delta y] \to [\delta u, \delta\psi]$ in Eqn. A.34 allows us to write:

$$\begin{aligned} J(u) - J(u_H) &= \int_0^1 F'_u \left[u_H + t\,\delta u, \psi_H + t\,\delta\psi\right](\delta u)\,dt \\ &+ \int_0^1 F'_\psi \left[u_H + t\,\delta u, \psi_H + t\,\delta\psi\right](\delta\psi)\,dt \quad . \end{aligned} \tag{A.39}$$

Now the question is: how should we compute the integrals in this expression? Since in general we cannot evaluate them analytically, we will instead approximate them using the trapezoidal rule. For a classic function f, the trapezoidal rule over the interval $[a, b]$ is shown in blue below:

$$\int_a^b f(x)\,dx = \frac{1}{2}\left[f(a) + f(b)\right] \underbrace{-\frac{1}{2}\int_a^b f''(x)(b-x)(x-a)\,dx}_{\text{remainder}} \tag{A.40}$$

$$= \frac{1}{2}\left[f(a) + f(b)\right] \underbrace{-\frac{\Delta x^3}{12} f''(\bar{x})}_{\text{remainder}} \tag{A.41}$$

(Here, either form of the remainder may be used, and $\Delta x = b - a$ while $\bar{x}$ is a particular point inside the interval.) From Eqn. A.41, we see that the trapezoidal rule gives a **third-order** approximation to the integral by using only the "endpoint" values of f.

We can use the trapezoidal rule to approximate each of the Fréchet derivatives in the above output error equation. The "endpoints" of the intervals correspond to parameter values of $t = 0$ and $t = 1$, so we have:

$$\int_0^1 F'_u \left[u_H + t\,\delta u, \psi_H + t\,\delta\psi\right](\delta u)\,dt \approx \frac{1}{2}\Big[F'_u[u_H, \psi_H](\delta u) + F'_u[u, \psi](\delta u)\Big]$$

$$\int_0^1 F'_\psi \left[u_H + t\,\delta u, \psi_H + t\,\delta\psi\right](\delta\psi)\,dt \approx \frac{1}{2}\Big[F'_\psi[u_H, \psi_H](\delta\psi) + F'_\psi[u, \psi](\delta\psi)\Big]$$

With these equations, we have converted our integrals into simple point evaluations of the Fréchet derivatives at (u_H, ψ_H) and (u, ψ). The only thing that remains is to actually write out the derivatives at these points. We can obtain these by just taking variations of Eqns. A.37 and A.38 with respect to u and ψ, then inserting our particular values of δu and $\delta\psi$. The four terms in the above trapezoidal approximations then become:

$$F'_u[u_H,\psi_H](\delta u) = J'_u[u_H](\delta u) - \int\limits_{\Omega} \psi_H\, r'_u[u_H](\delta u)\, d\Omega \tag{A.42}$$

$$F'_u[u,\psi](\delta u) = J'_u[u](\delta u) - \int\limits_{\Omega} \psi\, r'_u[u](\delta u)\, d\Omega \tag{A.43}$$

$$F'_\psi[u_H,\psi_H](\delta\psi) = -\int\limits_{\Omega} r(u_H)\,\delta\psi\, d\Omega \tag{A.44}$$

$$F'_\psi[u,\psi](\delta\psi) = -\int\limits_{\Omega} r(u)\,\delta\psi\, d\Omega \tag{A.45}$$

Now, looking at these expressions, we see that the last term (Eqn. A.45) is exactly zero, since the residual $r(\cdot)$ evaluated with the exact solution u vanishes. Likewise, Eqn. A.43 is also zero. This is because the exact adjoint equation (Eqn. A.36) holds for *all* $\delta v \in \mathcal{U}$, so it must hold for our particular $\delta u \in \mathcal{U}$ as well.

So out of our four original terms, we are left with only two that are nonzero. Thus, our error estimate in Eqn. A.39, after approximation by trapezoidal rule, can be written as:

$$\begin{aligned} J(u) - J(u_H) &\approx \frac{1}{2}\Bigg[F'_u[u_H,\psi_H](\delta u) + \underbrace{F'_u[u,\psi](\delta u)}_{=0} \Bigg] \\ &+ \frac{1}{2}\Bigg[F'_\psi[u_H,\psi_H](\delta\psi) + \underbrace{F'_\psi[u,\psi](\delta\psi)}_{=0} \Bigg] \\ &\approx \frac{1}{2}\Bigg[F'_u[u_H,\psi_H](\delta u) + F'_\psi[u_H,\psi_H](\delta\psi) \Bigg] \\ &\approx \frac{1}{2} F'_u[u_H,\psi_H](\delta u) \;-\; \frac{1}{2}\left(r(u_H),\delta\psi\right) . \end{aligned} \tag{A.46}$$

Now, let us look more closely at the $F'_u[u_H,\psi_H](\delta u)$ term in the above expression, which is given by Eqn. A.42 as

$$F'_u[u_H,\psi_H](\delta u) = J'_u[u_H](\delta u) - \left(\psi_H, r'_u[u_H](\delta u)\right). \tag{A.47}$$

We will show that this term can actually be written as an adjoint residual weighted by δu.

Since the $r'_u[u_H](\delta u)$ term in the above expression represents the linearized residual, it can be rewritten as simply some linear operator L applied to the perturbation δu. Thus, we have that

$$r'_u[u_H](\delta u) = L\delta u \,. \tag{A.48}$$

Now, we can define the adjoint operator (L^*) to this linear operator L in exactly the same way as usual, i.e. via the identity

$$(Lu, v) \;=\; (u, L^*v) \qquad\qquad \forall u, v \in \mathcal{U} \,. \tag{A.49}$$

Furthermore, since $\delta u \in \mathcal{U}$, this implies that

$$(L\delta u, v) \;=\; (\delta u, L^*v) \qquad\qquad \forall v \in \mathcal{U} \,. \tag{A.50}$$

Inserting $r'_u[u_H](\delta u)$ back in for $L\delta u$ then gives

$$(r'_u[u_H](\delta u), v) \;=\; (\delta u, L^*v) \qquad\qquad \forall v \in \mathcal{U} \,. \tag{A.51}$$

Finally, since $\psi_H \in \mathcal{U}$, using the above relation, the last term in Eqn. A.47 can be rewritten as

$$(r'_u[u_H](\delta u), \psi_H) \;=\; (\delta u, L^*\psi_H) \;, \tag{A.52}$$

or

$$\boxed{(\psi_H, r'_u[u_H](\delta u)) \;=\; (L^*\psi_H, \delta u)} \;. \tag{A.53}$$

Next, let us rewrite the $J'_u[u_H](\delta u)$ term in Eqn. A.47 in a slightly different form as well. If we assume the output J is defined as

$$J(u) = \int_\Omega j(u)\, d\Omega \,, \tag{A.54}$$

where $j(u)$ is some nonlinear function of our choosing (e.g. $j(u) = u^2$), then the Fréchet linearization of this output (about u_H) can be written as

$$\boxed{J'_u[u_H](\delta u) = (j'[u_H], \delta u)} \;. \tag{A.55}$$

Recall that for linear problems, our output was defined to be $J = (g, u)$ and its variation was just $J'_u(\delta u) = (g, \delta u)$. Then, comparing to the above equation, we see that $j'[u_H]$ is effectively our "g" for a nonlinear problem.

Inserting Eqns. A.55 and A.53 back into Eqn. A.47 gives

$$\begin{aligned} F'_u[u_H, \psi_H](\delta u) &= (j'[u_h], \delta u) - (L^* \psi_H, \delta u) \\ &= -(L^* \psi_H - j'[u_H], \delta u)\,. \end{aligned} \tag{A.56}$$

Now, just as the continuous adjoint equation is defined as

$$L^* \psi = g \tag{A.57}$$

for a linear problem, it is straightforward to verify that the continuous adjoint equation is defined as

$$L^* \psi = j'_u[u_H] \tag{A.58}$$

for a nonlinear problem. (This follows from the fact that, as discussed, $j'_u[u_H]$ plays effectively the same role as g.) Thus, for a nonlinear problem, the adjoint *residual* can be defined as

$$r^*[u_H](\psi) \equiv L^* \psi - j'_u[u_H]\,. \tag{A.59}$$

Looking back at Eqn. A.56, we see then that it can be rewritten as

$$\boxed{F'_u[u_H, \psi_H](\delta u) = -(r^*[u_H](\psi_H), \delta u)}\;, \tag{A.60}$$

i.e. as an *adjoint residual* weighted by δu. This is exactly what we sought to show.

Inserting the above expression (Eqn. A.60) back into the original output error estimate (Eqn. A.46) and including the remainder term from the trapezoidal rule then yields

$$J(u) - J(u_H) \approx -\frac{1}{2}\,(r(u_H), \delta\psi) \;-\; \frac{1}{2}\,(r^*[u_H](\psi_H), \delta u) \;+\; R^{(3)}\,. \tag{A.61}$$

Finally, expanding δu and $\delta\psi$, the output error estimate becomes:

$$\boxed{J(u) - J(u_H) \approx -\frac{1}{2}\,(r(u_H), \psi - \psi_H) - \frac{1}{2}\,(r^*[u_H](\psi_H), u - u_H) + R^{(3)}}\;. \tag{A.62}$$

We have now done what we set out to do – we have derived a third-order accurate error estimate for our output of interest. By including both primal and adjoint terms in the error estimate, we have taken advantage of the duality between these two vectors, allowing us to obtain a more accurate estimate than if we had used the primal form alone.

Some further observations about the error estimate are:

- The error estimate depends on the perturbations $\psi - \psi_H$ and $u - u_H$. In practice, we do not know ψ or u, so we must approximate them in some way. This can be done, for example, by smoothing coarse approximations to the state and adjoint on a finer mesh. Note that for linear Galerkin problems, our error estimate also involved a factor of $\psi - \psi_H$. In that case, we could approximate ψ by just solving the adjoint problem on a finer mesh. However, for nonlinear problems, the adjoint equations themselves depend on the current value of the primal state. Thus, to approximate the true ψ in the $\psi - \psi_H$ term, we could consider approximating the adjoint equations on *both* a finer mesh and using a better approximation to the state.

- In the above analysis, we have claimed that the trapezoidal rule is third-order accurate. However, it is typically thought of as a second-order method. So which is correct? The answer is that it is second-order *globally*, but third-order *locally*. In other words, if we partition our domain of integration into subintervals, then the trapezoidal rule will be third-order accurate over each subinterval, but when we sum all of the errors, the *total* error will be second-order. This is the same thing that happens, for example, when numerically integrating ODEs in time. The local accuracy on each time step is one order higher than the global accuracy of the time scheme. In the case of our error estimate, the important point is that we have only one interval (from (u_H, ψ_H) to $(u_H + \delta u, \psi_H + \delta\psi)$), so we obtain third-order accuracy over that interval.

- We have said that the remainder $R^{(3)}$ is third-order in the adjoint/state perturbations, but we can give a more detailed description of this term. From the trapezoidal rule in Eqn. A.40, the remainder is given by:

$$-\frac{1}{2}\int\limits_a^b f''(x)(b-x)(x-a)\,dx.$$

Now, in our output error estimation, the term we approximated with the trapezoidal rule (in Eqn. A.39) was $F'_u[\cdot,\cdot](\delta u) + F'_\psi[\cdot,\cdot](\delta\psi)$. So this is our effective "$f(x)$" in the above formula. Thus, to compute the remainder, we need to take two more derivatives of this quantity. Noting that our interval ranges from $a = 0$ to $b = 1$, we can write our output error remainder as:

$$R^{(3)} = -\frac{1}{2}\int_0^1 \underbrace{\mathcal{D}^{(2)}\left[F'_u[\cdot,\cdot](\delta u) + F'_\psi[\cdot,\cdot](\delta\psi)\right]}_{\text{effective } f''}(1-t)\,t\,dt. \tag{A.63}$$

Here, for ease of notation, we simply use dots $[\cdot,\cdot]$ to denote the states about which the linearization is performed. These dots actually represent the states $[u_H + t\delta u, \psi_H + t\delta\psi]$. Furthermore, note that $\mathcal{D}^{(2)}$ denotes a full second Fréchet derivative, which requires linearizations with respect to both u and ψ. In other words, we can write the **first** Fréchet derivative of our "$f(x)$" term as:

$$\begin{aligned} f'[\cdot,\cdot] \equiv \mathcal{D}^{(1)}\left[F'_u[\cdot,\cdot](\delta u) + F'_\psi[\cdot,\cdot](\delta\psi)\right] =& F''_{uu}[\cdot,\cdot](\delta u^2) \\ &+ F''_{u\psi}[\cdot,\cdot](\delta u\,\delta\psi) \\ &+ F''_{\psi u}[\cdot,\cdot](\delta\psi\,\delta u) \\ &+ F''_{\psi\psi}[\cdot,\cdot](\delta\psi^2), \end{aligned} \tag{A.64}$$

and the **second** Fréchet derivative as:

$$\begin{aligned} f''[\cdot,\cdot] \equiv \mathcal{D}^{(2)}\left[F'_u[\cdot,\cdot](\delta u) + F'_\psi[\cdot,\cdot](\delta\psi)\right] =& F'''_{uuu}[\cdot,\cdot](\delta u^3) \\ &+ F'''_{uu\psi}[\cdot,\cdot](\delta u^2\,\delta\psi) \\ &+ F'''_{u\psi u}[\cdot,\cdot](\delta u^2\,\delta\psi) \\ &+ F'''_{u\psi\psi}[\cdot,\cdot](\delta u\,\delta\psi^2) \\ &+ F'''_{\psi uu}[\cdot,\cdot](\delta\psi\,\delta u^2) \\ &+ F'''_{\psi u\psi}[\cdot,\cdot](\delta\psi^2\,\delta u) \\ &+ F'''_{\psi\psi u}[\cdot,\cdot](\delta\psi^2\,\delta u) \\ &+ F'''_{\psi\psi\psi}[\cdot,\cdot](\delta\psi^3). \end{aligned} \tag{A.65}$$

This second Fréchet derivative is the term we are interested in. Now, recall that from Eqns. A.37 and A.38, our functional F depends only *linearly* on ψ. This means that any terms above containing more than one derivative with respect to ψ will be zero. Eliminating these terms and combining equivalent mixed partials causes our second derivative to reduce to:

$$f''[\cdot,\cdot] \;=\; F'''_{uuu}[\cdot,\cdot](\delta u^3) \;+\; 3F'''_{uu\psi}[\cdot,\cdot](\delta u^2\,\delta\psi). \tag{A.66}$$

Finally, writing out these variations of F explicitly (based on its definition in Eqn. A.37) gives:

$$f''[u,\psi] \;=\; \underbrace{J'''_{uuu}[u](\delta u^3) - \int_\Omega \psi\, r'''_{uuu}[u](\delta u^3)\,dx}_{F'''_{uuu}} \underbrace{-3\int_\Omega \delta\psi\, r''_{uu}[u](\delta u^2)\,dx}_{3F'''_{uu\psi}} \tag{A.67}$$

Here, we have inserted $[u,\psi]$ as the state about which we are linearizing, just to avoid confusion.

Finally, inserting this f'' into our expression for the remainder gives:

$$R^{(3)} = -\frac{1}{2}\int_0^1 \Bigg[J'''_{uuu}[u](\delta u^3) - \int_\Omega \psi\, r'''_{uuu}[u](\delta u^3)\,dx \\ -3\int_\Omega \delta\psi\, r''_{uu}[u](\delta u^2)\,dx\Bigg](1-t)\,t\,dt \tag{A.68}$$

Again, for simplicity here we are using the states $[u,\psi]$ to represent the actual parameterized states $[u_H + t\delta u, \psi_H + t\delta\psi]$.

With the form of the remainder derived, it is useful to consider what it means in practice. First, it says that if the output and primal equations are *linear* (so that the above derivatives vanish), then the remainder is zero and the output error estimate is exact. This is expected, since we previously showed that the error estimates for linear Galerkin problems are exact. Also, it follows from this (and from the equivalence of primal and dual forms for Galerkin methods) that for linear problems, the combined primal-dual error estimate will reduce precisely to the single adjoint-weighted residual estimate derived previously.

We also see that, if J is quadratic in u (so that its third derivative vanishes) then the corresponding term in the remainder will vanish. Thus, if our output is **quadratic** and the primal equations **linear**, the third-order error estimate will again be exact. However, for any nonlinear primal residual, there will always be at least one nonzero remainder term, and the error estimate will not be exact.

Note also that since all terms in the above equation involve δu, but only one involves $\delta\psi$, there is an asymmetry. This asymmetry means that if we have the exact *primal* solution, then the remainder (as well as the error estimate as a whole) will reduce to zero; however, if we have the exact *adjoint* solution, then both the output error and the remainder may still be nonzero. This reflects the fact that, while for linear problems the adjoint and primal problems were independent (and therefore of equal "importance"), for nonlinear problems the adjoint is strictly dependent on the primal problem, and hence of less fundamental importance for computing the output. (Another way of saying this is that for nonlinear problems, outputs cannot be written in an equivalent "dual" form, so more information is required to compute the output than just the adjoint.)

- Finally, after deriving the third-order error estimate, we might wonder: **is it actually worth using?** Or is the second-order form typically good enough?

  The remainder terms of both the second- and third-order error estimates depend on the perturbations $\delta u = u - u_H$ and $\delta\psi = \psi - \psi_H$. Since u_H and ψ_H are assumed to be approximated on a coarse mesh with order p, we expect them to converge at order $p+1$. Therefore, the exected rates for the remainder terms are:

$$R^{(2)} = O((\delta u)^2) = O((\delta\psi)^2) \sim ((\Delta x)^{p+1})^2 \sim (\Delta x)^{2p+2} \tag{A.69}$$

$$R^{(3)} = O((\delta u)^3) = O((\delta\psi)^3) \sim ((\Delta x)^{p+1})^3 \sim (\Delta x)^{3p+3}. \tag{A.70}$$

  Now, given some coarse-space output and corresponding error estimate, we can write "corrected" forms of the output as:

$$J^{(2)}_{\text{corrected}} = J(u_H) + \delta J^{(2)}_{\text{est}} = J(u_h) + R^{(2)} \tag{A.71}$$

$$J^{(3)}_{\text{corrected}} = J(u_H) + \delta J^{(3)}_{\text{est}} = J(u_h) + R^{(3)} \tag{A.72}$$

  Here, $\delta J^{(2)}_{\text{est}}$ refers to the second-order form of the error estimate, while $\delta J^{(3)}_{\text{est}}$ refers to the third-order form. Also, we assume that in computing these error estimates, a "fine" space has been used to approximate the true u and ψ (the fine-space state is denoted by u_h above). For this reason, our corrected output is actually an approximation to $J(u_h)$ rather than the true $J(u)$.

  Looking at the above equations, we see that in general, the convergence rate of the corrected output will be limited by the rate of *either* $J(u_h)$ or R. So the question is, which of these quantities is more restrictive?

Consider the case where the output converges at a **standard** $p+1$ rate – i.e. it does not superconverge. Also, assume that our fine space is a $p+1$ space, as is often used in practice. Then we have the following expected rates:

$$\begin{aligned} J(u_h) &\sim (\Delta x)^{p+2} \\ R^{(2)} &\sim (\Delta x)^{2p+2} \\ R^{(3)} &\sim (\Delta x)^{3p+3} \end{aligned}$$

Here, we see that both $R^{(2)}$ and $R^{(3)}$ converge at a higher order than $J(u_h)$. Thus, $J_{\text{corrected}}$ will achieve the same convergence rate as the fine-space output, regardless of whether the second- or third-order form of the error estimate is used. So we see that in this case, it is not necessarily worth using the third-order estimate.

However, now consider the case where our output **superconverges** at a rate of $2p+1$, as is often true of integral outputs. Then our new expected rates are:

$$\begin{aligned} J(u_h) &\sim (\Delta x)^{2(p+1)+1} \sim (\Delta x)^{2p+3} \\ R^{(2)} &\sim (\Delta x)^{2p+2} \\ R^{(3)} &\sim (\Delta x)^{3p+3} \end{aligned}$$

In this case, we see that the rate of $R^{(2)}$ is one order lower than $J(u_h)$. This means that if we were to use the second-order form of the error estimate, our corrected output would converge sub-optimally, at a lower order than $J(u_h)$. On the other hand, using the third-order error estimate would allow us to achieve the optimal rate of $J(u_h)$.

APPENDIX B

ALE DG Implementation

This appendix provides implementation details for the deforming-domain (ALE) DG method of Persson *et al* [78]. The following text is reproduced with minor modifications from the XFlow DG solver documentation, which is written and maintained primarily by K. Fidkowski.

B.1 Mesh Motion Implementation

As derived in Chapter 5, the PDE on the reference domain is given by

$$\left.\frac{\partial \mathbf{u}_X}{\partial t}\right|_X + \nabla_X \cdot \vec{\mathbf{F}}_X(\mathbf{u}_X, \nabla_X \mathbf{u}_X) = \mathbf{0}, \tag{B.1}$$

where

$$\begin{aligned} \mathbf{u}_X &= g\mathbf{u}, \\ \vec{\mathbf{F}}_X &= g\mathcal{G}^{-1}\vec{\mathbf{F}} - \mathbf{u}_X \mathcal{G}^{-1}\vec{v}_G, \end{aligned}$$

and ∇_X denotes the gradient with respect to the reference coordinates. The flux $\vec{\mathbf{F}}_X$ can then be broken into inviscid and viscous components by lumping the grid-velocity term into the inviscid flux,

$$\vec{\mathbf{F}}_X = \vec{\mathbf{F}}^i_X - \vec{\mathbf{F}}^v_X, \tag{B.2}$$

$$\vec{\mathbf{F}}^i_X = g\mathcal{G}^{-1}\vec{\mathbf{F}}^i - \mathbf{u}_X\mathcal{G}^{-1}\vec{v}_G, \tag{B.3}$$

$$\vec{\mathbf{F}}^v_X = g\mathcal{G}^{-1}\vec{\mathbf{F}}^v. \tag{B.4}$$

We will need to transform the gradient from reference to physical, and this is done by the chain and product rules,

$$\begin{aligned}\nabla\mathbf{u} = \frac{\partial\mathbf{u}}{\partial x_j} = \frac{\partial(g^{-1}\mathbf{u}_X)}{\partial X_d}\frac{\partial X_d}{\partial x_j} &= \left(g^{-1}\frac{\partial\mathbf{u}_X}{\partial X_d} - g^{-2}\frac{\partial g}{\partial X_d}\mathbf{u}_X\right)G_{dj}^{-1} \\ &= g^{-1}\left(\frac{\partial\mathbf{u}_X}{\partial X_d} - g^{-1}\frac{\partial g}{\partial X_d}\mathbf{u}_X\right)G_{dj}^{-1}, \qquad \text{(B.5)}\end{aligned}$$

where d and j index the reference and physical coordinates, respectively. Also, the following relationships involving $\mathcal{G}$ are useful

$$\begin{aligned}\mathcal{G} &= G_{jd} = \frac{\partial x_j}{\partial X_d}, && \text{(B.6)}\\ \delta_{ji} &= \frac{\partial x_j}{\partial x_i} = \frac{\partial x_j}{\partial X_d}\frac{\partial X_d}{\partial x_i} = G_{jd}G_{di}^{-1}, && \text{(B.7)}\\ \Rightarrow \quad \mathcal{G}^{-1} &= G_{di}^{-1} = \frac{\partial X_d}{\partial x_i}. && \text{(B.8)}\end{aligned}$$

B.2 Discretization

B.2.1 Weighted Residual Statement

Eqn. B.1 is solved on the reference domain using the DG method described in Chapter IV. The weighted residual statement is obtained from the PDE by multiplying by test functions (defined in the reference domain) and integrating over reference-domain elements, κ. The resulting terms are as follows:

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

where $\mathbf{v}$ is a test function, d indexes the reference domain spatial coordinates, and k indexes the state vector. The hats on quantities defined at the element boundaries indicate numerical fluxes, inviscid or viscous, on element interfaces or domain boundaries – these will involve information from the neighboring element or boundary condition. Also the $^+$ superscript indicates quantities taken from the element interior.

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

B.2.2 Inviscid Flux

As derived in the previous section, the inviscid flux takes the following form,

$$\begin{aligned} \vec{\mathbf{F}}^i_X &= g\mathcal{G}^{-1}\vec{\mathbf{F}}^i - \mathbf{u}_X\mathcal{G}^{-1}\vec{v}_G && \text{(B.9)} \\ &= g\mathcal{G}^{-1}\left(\vec{\mathbf{F}}^i - \mathbf{u}\vec{v}_G\right). && \text{(B.10)} \end{aligned}$$

Thus, it includes the standard Galilean transformation and a multiplication by $g\mathcal{G}^{-1}$, which is done by post-processing the equation-set specific flux.

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

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

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

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

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

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

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

With mesh motion present, the Roe flux becomes

$$\begin{aligned}\left[(F^i_{jk} - u_k v_{G,j})n_j\right]^{\text{Roe}} =& \frac{1}{2}\left(F^L_{jk}n_j + F^R_{jk}n_j\right) - \frac{1}{2}\left(u^L_k + u^R_k\right)u_G - \frac{1}{2}\Big|A_{kl}(\mathbf{u}^{\text{Roe}}) \\ & -\delta_{kl}u_G\Big|(u^R_l - u^L_l), \qquad \text{(B.11)}\end{aligned}$$

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

B.2.3 Viscous Flux

The contribution of viscous terms to the residual semilinear form is

$$\mathcal{R}^v_X(\mathbf{u}_X, \mathbf{v}) = \int_{\kappa} \partial_{X_d} v_k F^v_{X,dk} dV - \int_{\partial\kappa} v^+_k \widehat{F^v_{X,dk} N_d}\, dA. \qquad \text{(B.12)}$$

The reference-domain viscous flux is related to the physical viscous flux (which we know how to compute) through

$$F^v_{X,dk} = g G^{-1}_{di} F^v_{ik}. \qquad \text{(B.13)}$$

The physical viscous flux is calculated using a diffusion matrix and the physical state gradient,

$$F^v_{ik} = A_{ijkl}\partial_{x_j} u_l. \qquad \text{(B.14)}$$

Using Eqn. B.5 for the physical gradient, the reference-domain viscous flux is

$$\begin{aligned} F^v_{X,dk} &= g G^{-1}_{di} A_{ijkl} \partial_{x_j} u_l \\ &= g G^{-1}_{di} A_{ijkl} g^{-1}\left(\partial_{X_c} u_{X,l} - u_{X,l} g^{-1} \partial_{X_c} g\right) G^{-1}_{cj} \\ &= \underbrace{G^{-1}_{di} A_{ijkl} G^{-1}_{cj}}_{A_{X,dckl}} \left(\partial_{X_c} u_{X,l} - u_{X,l} g^{-1} \partial_{X_c} g\right), \qquad \text{(B.15)}\end{aligned}$$

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

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

The residual contribution in Eqn. B.12 then becomes

$$\begin{aligned}\mathcal{R}^v_X(\mathbf{u}_X, \mathbf{v}) &= \int\limits_{\kappa} \partial_{X_d} v_k A_{X,dckl} \partial_{X_c} u_{X,l} dV \\ &\quad - \int\limits_{\kappa} \partial_{X_d} v_k A_{X,dckl} u_{X,l} g^{-1} \partial_{X_c} g dV \\ &\quad - \int\limits_{\partial\kappa} v_k^+ \widehat{F^v_{X,dk} N_d}\, dA.\end{aligned}$$

In the above expression the middle term is new – it is specific to the reference-to-global mesh motion transformation. Of course, mesh motion is also included in the transformed diffusion matrix, but there the transformation is hidden through the definition of $A_{X,dckl}$. The new middle term is unique because it cannot be rolled into any of the other terms. Fortunately, the new term does not interfere with the viscous discretization, which is obtained by integrating the first term twice by parts. This integration pulls off a dual-consistency term (as mentioned in Chapter IV) evaluated along the element boundary. The resulting terms in the final BR2 discretization are:

$$\begin{aligned}\mathcal{R}^v_X(\mathbf{u}_X, \mathbf{v}) &= \int\limits_{\kappa} \partial_{X_d} v_k \, A_{X,dckl} \, \partial_{X_c} u_{X,l} dV \qquad \text{(interior term)} \\ &\quad - \int\limits_{\kappa} \partial_{X_d} v_k \, A_{X,dckl} \, u_{X,l} g^{-1} \partial_{X_c} g \, dV \qquad \text{(new interior term)} \\ &\quad - \int\limits_{\partial\kappa} \partial_{X_d} v_k^+ \, A^+_{X,dckl} \, (u^+_{X,l} - \hat{u}_{X,l}) N_c \, dA \qquad \text{(dual-consistency term)} \\ &\quad - \int\limits_{\partial\kappa} v_k^+ \widehat{F^v_{X,dk} N_d}\, dA \qquad \text{(viscous flux term)}\ .\end{aligned}$$

We lump the two interior terms together in the implementation. The dual-consistency term involves a unique definition of the state on the boundary, which is $\hat{u}_{X,l} = (u^+_{X,l} + u^-_{X,l})/2$ for BR2.

The viscous flux term is evaluated using Eqn. B.15,

$$\text{(viscous flux term)} = -\int\limits_{\partial\kappa} v_k^+ \widehat{F^v_{X,dk} N_d}\, dA = -\int\limits_{\partial\kappa} v_k^+ \overline{[A_{X,dckl} \, \partial_{X_c} u_{X,l} N_d]} \, dA \tag{B.16}$$

$$+ \int\limits_{\partial\kappa} v_k^+ \overline{[A_{X,dckl} u_{X,l} g^{-1} \partial_{X_c} g \, N_d]} \, dA \tag{B.17}$$

where we use an overline to denote flux averaging on long expressions because the hat symbol is not wide enough. The first term in the result is the standard viscous flux. Its evaluation involves bringing in a jump-penalty (e.g. BR2) stabilization. All expressions in this term are evaluated on the reference domain (they are the "X" quantities), but aside from that there is no difference compared to the viscous flux evaluation without mesh motion.

On the other hand, the second term is new. Noting that for an analytically specified mesh motion g – whose derivatives are continuous across interfaces – the averaging only has to be performed on the diffusion matrix and state,

$$\begin{aligned}\overline{[A_{X,dckl}u_{X,l}g^{-1}\partial_{X_c}g\,N_d]} &= \overline{[A_{X,dckl}u_{X,l}]}g^{-1}\partial_{X_c}g\,N_d \\ &= \{A_{X,dckl}u_{X,l}\}\,g^{-1}\partial_{X_c}g\,N_d.\end{aligned}$$

The curly brackets in the last line refer to equal averaging of the left and right quantities at the interface in calculating the state and the diffusion matrix. This is consistent with the definition of the state on the interface in the BR2 viscous discretization (i.e. that it is just the average).

B.3 Boundary Conditions

The physical convective boundary flux, $\vec{\mathbf{F}}^{ib}$, is modified to account for mesh motion as given in Eqn. B.10,

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

We note that the physical boundary flux must be aware of motion on the boundary, $\vec{v}_G$. For example, on a moving wall, the flow tangency boundary condition states that the normal component of the fluid velocity is equal to the normal component of the boundary motion velocity (which would be zero without mesh motion). This physical consideration is separate from the subtraction of $\mathbf{u}^b\vec{v}_G$ above – both must be included.

Calculation of the viscous contribution on a boundary requires not only the boundary state, $\mathbf{u}^b$, but also the boundary flux. Using Eqn. B.17, the transformed viscous flux on the boundary is

$$\left[\widehat{F^v_{X,dk}N_d}\right]^b = \left[\overline{A_{X,dckl}\,\partial_{X_c}u_{X,l}N_d}\right]^b - \left[\overline{A_{X,dckl}u_{X,l}g^{-1}\partial_{X_c}g\,N_d}\right]^b. \qquad \text{(B.18)}$$

Although we always have access to a boundary state, $\mathbf{u}^b$ (constructed for use in the convective flux), the gradient on the boundary is not necessarily available from the boundary conditions. In these (Dirichlet) cases, the state gradient information is taken from the interior, and we write

$$\begin{aligned}\left[\widehat{F^v_{X,dk} N_d}\right]^b &= A^b_{X,dckl}\,(\partial_{X_c} u_{X,l})^+ N_d - A^b_{X,dckl} u^b_{X,l} g^{-1} \partial_{X_c} g\, N_d \\ &= A^b_{X,dckl}\,\left[(\partial_{X_c} u_{X,l})^+ - u^b_{X,l} g^{-1} \partial_{X_c} g\right] N_d.\end{aligned}$$

In other cases, the physical viscous flux is prescribed directly on the boundary (e.g. zero heat flux for an adiabatic wall). In these cases, Eqn. B.18 is not used and instead the viscous flux contribution is added directly to the residual. Let us call Q^b_k the prescribed boundary viscous flux dotted with the physical normal. Then in our residual contribution, we will be integrating,

$$Q^b_k da = F^v_{ik} n_i da = g^{-1} G_{id} F^v_{X,dk}\, g G^{-1}_{ci} N_c dA = F^v_{X,dk} N_d dA.$$

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

Finally we note that for systems of equations, boundary conditions will generally be of the mixed type, in that for some k a boundary value of the state will be prescribed while for other k a boundary flux will be prescribed.

B.4 Geometric Conservation Law

If we wish to preserve a free-stream state, we must satisfy a Geometric Conservation Law (GCL). Persson *et al* [78] describe one technique for satisfying the GCL, which we follow here. This technique relies on approximating (in reference space) $\mathbf{u}_{\bar{X}} = \bar{g}\mathbf{u}$ instead of $\mathbf{u}_X = g\mathbf{u}$, where $\bar{g}$ is a separate variable approximated using the same basis and marched using the same unsteady solver as the state to solve the following equation:

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

Note that now a constant physical state ($\bar{\mathbf{u}}$) is representable, since $\mathbf{u}_X = \bar{g}\bar{\mathbf{u}}$ is a polynomial in the discrete approximation space.

Implementing a GCL requires integrating another vector, $\bar{g}$ in time. In addition, the formulas above need to be modified given that we now need to store $\mathbf{u}_{\bar{X}}$ instead of $\mathbf{u}_X$. Let's see how these formulas change. First, some useful relations,

$$\mathbf{u}_X = g\mathbf{u}, \quad \mathbf{u}_{\bar{X}} = \bar{g}\mathbf{u}, \quad \mathbf{u}_{\bar{X}} = \bar{g}g^{-1}\mathbf{u}_X, \quad \mathbf{u}_X = g\bar{g}^{-1}\mathbf{u}_{\bar{X}}.$$

The expression for the physical gradient in Eqn. B.5 changes,

$$\begin{aligned} \nabla\mathbf{u} = \frac{\partial\mathbf{u}}{\partial x_j} = \frac{\partial(\bar{g}^{-1}\mathbf{u}_{\bar{X}})}{\partial X_d}\frac{\partial X_d}{\partial x_j} &= \left(\bar{g}^{-1}\frac{\partial\mathbf{u}_{\bar{X}}}{\partial X_d} - \bar{g}^{-2}\frac{\partial\bar{g}}{\partial X_d}\mathbf{u}_{\bar{X}}\right)G^{-1}_{dj} \\ &= \bar{g}^{-1}\left(\frac{\partial\mathbf{u}_{\bar{X}}}{\partial X_d} - \bar{g}^{-1}\frac{\partial\bar{g}}{\partial X_d}\mathbf{u}_{\bar{X}}\right)G^{-1}_{dj}. \qquad (B.19) \end{aligned}$$

The convective flux requires no modifications as long as the physical state is computed correctly (using $\bar{g}$). The viscous flux in Eqn. B.15 does change because the gradient changes,

$$\begin{aligned} F^v_{X,dk} &= gG^{-1}_{di}A_{ijkl}\partial_{x_j}u_l \\ &= gG^{-1}_{di}A_{ijkl}\bar{g}^{-1}\left(\partial_{X_c}u_{\bar{X},l} - u_{\bar{X},l}\bar{g}^{-1}\partial_{X_c}\bar{g}\right)G^{-1}_{cj} \\ &= \underbrace{g\bar{g}^{-1}G^{-1}_{di}A_{ijkl}G^{-1}_{cj}}_{A_{\bar{X},dckl}}\left(\partial_{X_c}u_{\bar{X},l} - u_{\bar{X},l}\bar{g}^{-1}\partial_{X_c}\bar{g}\right), \qquad (B.20) \end{aligned}$$

The resulting terms in the BR2 discretization are as follows:

$$\begin{aligned} \mathcal{R}^v_X(\mathbf{u}_X,\mathbf{v}) = &\int_\kappa \partial_{X_d}v_k\, A_{\bar{X},dckl}\,\partial_{X_c}u_{\bar{X},l}dV \quad \text{(interior term)} \qquad (B.21) \\ &- \int_\kappa \partial_{X_d}v_k\, A_{\bar{X},dckl}\,u_{\bar{X},l}\bar{g}^{-1}\partial_{X_c}\bar{g}\,dV \quad \text{(new interior term)} \qquad (B.22) \\ &- \int_{\partial\kappa} \partial_{X_d}v^+_k\, A^+_{\bar{X},dckl}\,(u^+_{\bar{X},l} - \hat{u}_{\bar{X},l})N_c\,dA \quad \text{(dual-consistency term)} \quad (B.23) \\ &- \int_{\partial\kappa} \partial_{X_d}v^+_k\,\left[-A^+_{\bar{X},dckl}\,(\bar{g}^+)^{-1}u^+_{\bar{X},l}\right](\bar{g}^+ - \hat{\bar{g}})\,N_c\,dA \quad \text{(GCL d.c. term)} \\ &\qquad (B.24) \\ &- \int_{\partial\kappa} v^+_k\widehat{F^v_{X,dk}N_d}\,dA \quad \text{(viscous flux term)}\,. \qquad (B.25) \end{aligned}$$

Note that $\hat{\bar{g}} = \frac{1}{2}(\bar{g}^+ + \bar{g}^-)$. The GCL dual-consistency term accounts for the use of the gradient of $\bar{g}$ in the viscous flux. We derive it by considering an augmented state,

$[u_l; \bar{g}]$, for which the viscous flux from Eqn. B.20 is

$$
\begin{aligned}
F^v_{X,dk} &= A_{\bar{X},dckl}\left(\partial_{X_c} u_{\bar{X},l} - u_{\bar{X},l}\bar{g}^{-1}\partial_{X_c}\bar{g}\right), & \text{(B.26)}\\
\Rightarrow \begin{bmatrix} F^v_{X,dk} \\ 0 \end{bmatrix} &= \begin{bmatrix} A_{\bar{X},dckl} & A_{\bar{X},dckl} u_{\bar{X},l}\bar{g}^{-1} \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \partial_{X_c} u_{\bar{X},l} \\ \partial_{X_c}\bar{g} \end{bmatrix} & \text{(B.27)}
\end{aligned}
$$

Note that the GCL component of the viscous flux is zero, hence the zeros in the bottom row of the matrix. Applying the BR2 discretization to the above system results in two dual-consistency terms, as written above. These can be combined to yield:

$$
\begin{aligned}
\mathcal{R}^v_X(\mathbf{u}_X, \mathbf{v}) = &\int_\kappa \partial_{X_d} v_k \, A_{\bar{X},dckl} \, \partial_{X_c} u_{\bar{X},l} dV \quad \text{(interior term)} & \text{(B.28)}\\
&- \int_\kappa \partial_{X_d} v_k \, A_{\bar{X},dckl} \, u_{\bar{X},l}\bar{g}^{-1}\partial_{X_c}\bar{g} \, dV \quad \text{(new interior term)} & \text{(B.29)}\\
&- \int_{\partial\kappa} \partial_{X_d} v^+_k \, A^+_{\bar{X},dckl} \left(u^+_{\bar{X},l} \frac{\hat{\bar{g}}}{\bar{g}^+} - \hat{u}_{\bar{X},l} \right) N_c \, dA \quad \text{(combined d.c. term)} & \\
& & \text{(B.30)}\\
&- \int_{\partial\kappa} v^+_k \widehat{F^v_{X,dk} N_d} \, dA \quad \text{(viscous flux term)} \; . & \text{(B.31)}
\end{aligned}
$$

The viscous flux term, Eqn. B.17 also changes,

$$
\begin{aligned}
\text{(viscous flux term)} \; = -\int_{\partial\kappa} v^+_k \widehat{F^v_{X,dk} N_d} \, dA = &-\int_{\partial\kappa} v^+_k \overline{\left[A_{\bar{X},dckl} \, \partial_{X_c} u_{\bar{X},l} N_d\right]} \, dA & \text{(B.32)}\\
&+ \int_{\partial\kappa} v^+_k \overline{\left[A_{\bar{X},dckl} u_{\bar{X},l} \bar{g}^{-1} \partial_{X_c}\bar{g} \, N_d\right]} \, dA & \text{(B.33)}
\end{aligned}
$$

Note that now quantities involving $\bar{g}$ need to be averaged across element interfaces since $\bar{g}$ may be discontinuous at an element interface.

$$
\begin{aligned}
\overline{\left[A_{\bar{X},dckl} u_{\bar{X},l} \bar{g}^{-1} \partial_{X_c}\bar{g} \, N_d\right]} &= \overline{\left[A_{\bar{X},dckl} u_{\bar{X},l} \bar{g}^{-1} \partial_{X_c}\bar{g}\right]} \, N_d \\
&= \left\{A_{\bar{X},dckl} u_{\bar{X},l} \bar{g}^{-1} \partial_{X_c}\bar{g}\right\} \, N_d .
\end{aligned}
$$

There is no boundary condition on $\bar{g}$, since there is no PDE for this variable, and so when imposing boundary conditions, we take the interior state: $\bar{g}^b = \bar{g}^+$. We sometimes need to linearize the boundary state with respect to $\bar{g}^+$, and the required equations read,

$$\begin{aligned} u_l^b &= u_l^b(u_l^+) = u_l^b\left(\frac{u_{\bar{X},l}^+}{\bar{g}^+}\right) \\ \Rightarrow \frac{\partial u_l^b}{\partial \bar{g}^+} &= \frac{\partial u_l^b}{\partial u_l^+}\left(-\frac{u_{\bar{X},l}^+}{(\bar{g}^+)^2}\right) \end{aligned}$$

When penalizing boundary jumps (e.g. in the BR2 viscous discretization), we take the difference of the interior reference state, $u_{\bar{X},l}^+$ and the boundary *reference state*, $u_{\bar{X},l}^b \equiv \bar{g}^+ u_l^b$.

APPENDIX C

Relevance of GCL for Shock Tube Problem

In this appendix, we investigate how important satisfaction of the Geometric Conservation Law (GCL) is for a one-dimensional shock-tube problem, within the context of the DG ALE method discussed in Chapter V.

C.1 Relevance of the GCL for a 1D Shock Tube Problem

It is known that a lack of strict conservation can cause certain numerical methods to give incorrect shock speeds and/or strengths, even upon mesh refinement [52, 64]. For simulations with mesh motion, we know that if we do not satisfy the GCL, conservation errors can (and do) arise. An important question is whether these conservation errors actually lead to incorrect shock properties for the DG ALE method. If they do, then satisfying the GCL would be a necessity for problems with shocks.

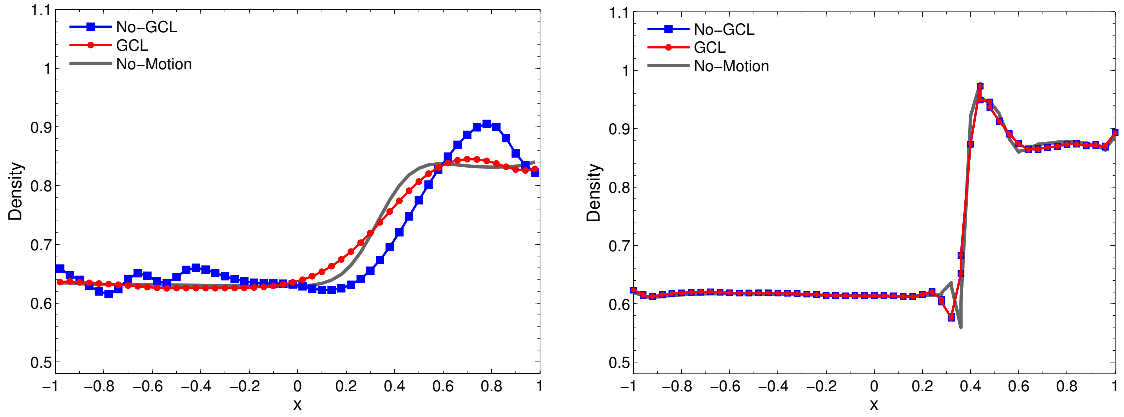
To investigate this issue, we solve a one-dimensional Euler shock tube problem. The initial condition is a standard Riemann problem, with a discontinuity located at the center of the tube, and the left and right states shown in Table C.1. We solve the problem with (i) no mesh motion, (ii) mesh motion and GCL, and (iii) mesh motion without the GCL. For the motion cases, a strong left-right "plunge" motion is introduced into the 1D mesh, which consists of 50 elements and spans the domain $[-1, 1]$. The temporal discretization consists of a DG1 scheme with 400 time slabs. The motion itself is defined by the mapping

$$x = X + 0.45 \sin(2\pi t) \tag{C.1}$$

and is blended to zero at the domain boundaries using a quintic blending polynomial and parameters $X_C = 0$, $R_C = 0.01$, and $D = 0.98$ (as defined in Chapter V). Theoretically, this motion should have no influence on the physics of the problem.

State	Left	Right
ρ	1.0	0.4
u	0.0	0.0
p	1.0	0.32

Table C.1: Left and right states for the shock tube initial condition. ρ, u, and p refer to density, velocity, and pressure, respectively.



(a) $p = 0$

(b) $p = 1$

Figure C.1: Density profiles for a 1D shock tube problem, solved at (a) $p = 0$ and (b) $p = 1$. The profiles shown are snapshots taken at time $t = 5$, after the shock has reflected several times off of the domain boundaries.

By the end of the simulation, the initial shock has reflected several times off of the domain boundaries, and the mesh has undergone 5 periods of motion. Snapshots of density at the final time are shown in Figure C.1. We see that satisfaction of the GCL makes a large difference for the under-resolved (i.e. $p = 0$) runs, but essentially no difference at all as soon as $p = 1$ is used. For the $p = 0$ case, the no-GCL run gives an incorrect shock speed, as expected, while the run satisfying the GCL gives approximately correct shock properties. However, for $p = 1$, both GCL and no-GCL runs give the correct shock position and strength, suggesting that satisfying the GCL is not critical for this method so long as the solution is sufficiently resolved.

Of course, a more rigorous study could be performed to ensure that this trend holds for even stronger shocks, and this conclusion about the GCL may not hold for other methods. The fact that the current (DG ALE) method is conservative on the reference domain – combined with the fact that it approximates the integral form of the conservation law – likely explains its correct placement of shocks despite the lack of strict conservation on the physical domain.

APPENDIX D

Optimal Test Functions

In this section, we show that for a one-dimensional advection-reaction problem, the localized optimal test functions discussed in Chapter VI are in fact globally optimal with respect to boundary accuracy.

D.1 Localization of the Test Space (One-Dimensional Advection-Reaction)

Our primary goal in this work is to achieve accuracy in the fluxes through the domain boundaries. For the advection-reaction problem in Section 6.5, this means that we would like the flux on the right boundary,

$$J = au_H(x_R)\,, \tag{D.1}$$

to be accurate. Ideally we would like it to have zero error.

From *a posteriori* error estimation, the error in a certain output (including the above J) can be represented as the inner product of a corresponding adjoint solution and the residual of the governing PDE [10, 47, 36]. For our one-dimensional advection-reaction problem, it is straightforward to show that the adjoint solution v corresponding to J satisfies the following equation:

$$L^*v \;=\; -a\frac{\partial v}{\partial x} + cv \;=\; 0 \qquad\quad v(x_R) = 1\,. \tag{D.2}$$

This is a global differential equation, which can be solved analytically to obtain

$$v(x) = \underbrace{e^{-cx_R/a}}_{\text{const.}}\, e^{cx/a}\,. \tag{D.3}$$

The output error $\delta J = J(u_H) - J(u)$ can then be written as a product of this v and the residual of the PDE:

$$\delta J \;=\; \int\limits_{\Omega} v\, r(u_H)\, dx \;=\; b(u_H, v) - l(v)\,. \tag{D.4}$$

From this expression, it is clear that if the adjoint v happens to lie in the test space of our finite element method, then the error in the flux J will be zero. This is because if v were in the test space, one of our finite element equations would be

$$b(u_H, v) \;=\; l(v) \quad\Longrightarrow\quad b(u_H, v) - l(v) \;=\; 0 \quad\Longrightarrow\quad \delta J = 0\,. \tag{D.5}$$

Our goal in this appendix is to show that, when using the local optimal test functions defined in Section 6.5, this v *is* in fact contained in the test space, and therefore the local test space *can* in fact deliver zero error in the domain-boundary flux.

As described in Section 6.5, to compute the local optimal test functions, we solve elementwise adjoint problems for the following outputs on each element Ω_e:

$$J_i = \int\limits_{\Omega_e} \phi_i u\, dx \;+\; w_R \phi_i u\Bigg|_{\partial\Omega_{e,R}}. \tag{D.6}$$

Now, for a trial basis ϕ_i that is nonzero on the right boundary, taking w_R large will make the boundary term in J_i dominate the interior term. Therefore, in the limit of large w_R, the interior term can be neglected and the output effectively becomes

$$J_i = w_R \phi_i u\Bigg|_{\partial\Omega_{e,R}}. \tag{D.7}$$

This is just a constant multiple of the flux through the downwind boundary of the element. Since the constant makes no difference to the final test space, for purposes of analysis we can treat the above J_i as a pure (local) flux output. This means that, on any given element, one of the local optimal test functions (denoted by $\bar{v}(\bar{x})$) satisfies the following adjoint equation:

$$-a\frac{\partial \bar{v}}{\partial \bar{x}} + c\bar{v} \;=\; 0 \qquad\qquad \bar{v}(\Delta x) \;= 1\,. \tag{D.8}$$

This equation is analogous to that for a global flux output (i.e. Eqn. D.2), but is defined over an individual element rather than the entire domain. Here, $\bar{x}$ is a local coordinate associated with the element in question (see Figure D.1 for a definition of the relevant quantities). Solving this equation for the test function $\bar{v}$ gives

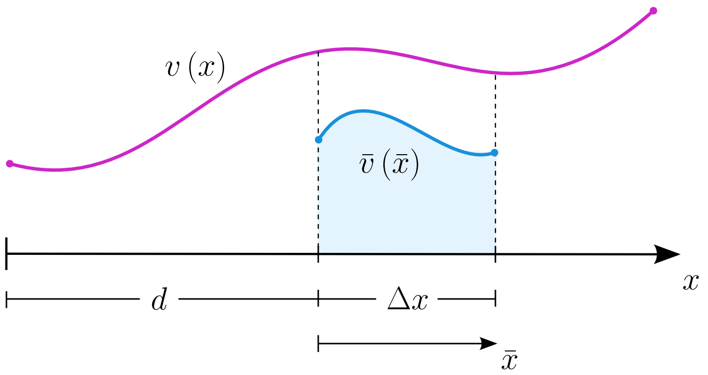


Figure D.1: Local and global coordinate systems and test functions. Note that a bar designates local quantities.

$$\bar{v}(\bar{x}) = \underbrace{e^{-c\Delta x/a}}_{\text{const.}}\, e^{c\bar{x}/a}\,. \tag{D.9}$$

Finally, writing $\bar{x}$ in terms of x using the transformation $\bar{x} = x - d$ results in

$$\bar{v}(x) = \underbrace{e^{-c\Delta x/a}\, e^{-cd/a}}_{\text{const.}}\, e^{cx/a}\,. \tag{D.10}$$

Comparing this local test function $\bar{v}$ to the global adjoint v in Eqn. D.3, we see these functions are indeed just constant multiples of each other. Thus, regardless of where a given element is located (i.e. regardless of d) and regardless of the size of the element (i.e. regardless of Δx), one of the local test functions on each element satisfies

$$\bar{v}(x) = C\, v(x) \tag{D.11}$$

for some constant C. Since by definition the test space includes all constant multiples of $\bar{v}$, this means that the global adjoint v is in fact contained in the test space. From our earlier discussion, this then implies that the error in the domain-boundary flux J is zero. Thus, the local optimal test space is in fact globally optimal with respect to the domain-boundary flux. While we have focused on an advection-reaction problem here, similar logic holds for more general problems (such as advection-diffusion).

APPENDIX E

Trial Space Optimization

In this section, we show some proof-of-concept results for a trial basis "optimization" on element boundaries. The problem is two-dimensional advection. We show that if the trial space is optimized near element boundaries (while simultaneously employing optimal test functions) the errors in the domain-boundary fluxes can be driven toward zero while keeping the total number of trial space basis functions fixed.

E.1 2D Advection: Trial Basis Optimization with Optimal Test Functions

As discussed previously, if BDPG is to provide an advantage over standard HDG in multiple dimensions, some type of trial space optimization on element boundaries is required. If this trial-space optimization were performed, the number of globally coupled degrees-of-freedom could be reduced relative to standard HDG.

In the end, our goal is to have the trial basis functions near element boundaries be capable of representing the shape of the true fluxes well. One simple attempt at accomplishing this consists of the following strategy:

1. On a given mesh (consisting of e.g. quadrilateral elements), solve a standard DG problem at order p.

2. Next, in an element-local manner, obtain an estimate for the solution on a *finer* space. (Here, we inject the order-p solution on each element to a $p+1$ space, then apply a single iteration of an elementwise block-Jacobi smoother. This provides a cheap estimate/surrogate for the solution in the full $p+1$ space.)

3. Next, along each element boundary, pull off the "trace" of this $p+1$ surrogate and add this trace shape to the trial space basis. (Since we need each basis function to be two-dimensional, these one-dimensional traces are multiplied by a linear function that smoothly blends them to zero at the opposite face of the quadrilateral element.)

4. Delete any undesired interior modes in the original order-p trial space, since these will not be relevant to achieving boundary accuracy.

5. Compute optimal test functions for the new trial space basis, which contains the blended trace shapes. Use $p_{\text{test}} = p+1$ to match the order of these trace functions.

6. Solve the problem again, now using the optimal test functions and the "optimized" trial functions.

7. After solving again, the errors in the boundary fluxes should be lower than before, since the optimal test functions are now requesting accuracy in the fluxes, and the trial space actually has the ability to *provide* that accuracy (since we have added basis functions that approximate the true boundary/flux modes).

This process can be repeated indefinitely to drive the flux errors lower and lower. Note that after each iteration, we can simply update the shape of the "trace modes" in the trial basis. We do not have to keep adding *more* trace modes. Thus, the number of trial space degrees-of-freedom can be kept fixed as the flux errors are driven lower.

Another important point is that it is not always necessary to increase the order of the "fine space" after every iteration. If we keep, say, 4 trace modes in the trial basis (one corresponding to each edge of a quadrilateral element), then even if these modes are, say, a $p=3$ function at a certain iteration, they will still not *span* a full $p=3$ space. Thus, taking a full $p=3$ space as the fine space even when the current trace modes already contain some $p=3$ information can still improve the solution. This is the strategy we employ for the results shown below.

To test this preliminary trial space optimization, we solve a two-dimensional advection problem with veclocity $\vec{a} = [1.0 \;\; 0.7]$, using a BDPG method based on the DG formulation. The solution is manufactured to be

$$u(x,y) = 2\sin^2(\frac{3}{5}\pi x)\sin^2(\frac{3}{5}\pi y) \tag{E.1}$$

(shown in Fig. E.1) and the mesh is chosen to consist of 64 quadrilateral elements.

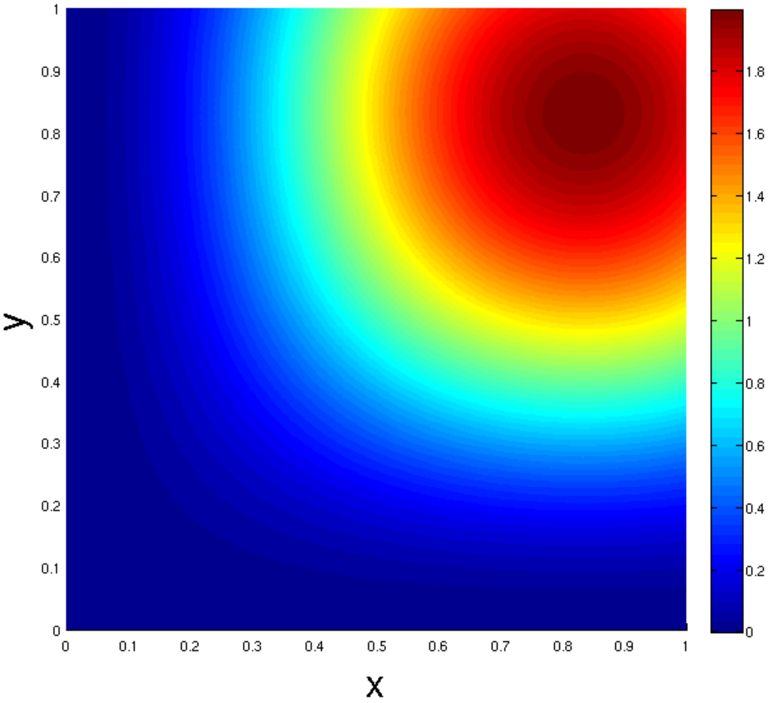


Figure E.1: Two-dimensional advection manufactured solution.

To compute the optimal test functions for BDPG, we use a large flux weight in the error norm: specifically, $w = 10^9$.

We then perform the strategy outlined above. Specifically, for this problem, we:

1. Solve a $p = 1$ DG problem.

2. Use a $p = 2$ fine space for computing the surrogate/approximate solution for the first "iteration" of the above algorithm.

3. Delete all undesired interior modes from the trial space, leaving only a constant mode along with 4 blended "trace" basis functions. (From this point on, the number of trial space basis functions is kept fixed at 5, and the trace modes are simply updated during each iteration of the algorithm.)

4. For all subsequent iterations of the algorithm, keep the fine-space fixed at $p = 3$ and continue performing a block Jacobi smoothing/trace basis update after every iteration.

Now, theoretically, if we perform enough block Jacobi smoothings (over the course of the above algorithm), we should eventually reach the accuracy of the full $p = 3$ DG solution, since this is the fine space we are smoothing on. However, the question is: does updating the trial basis itself, combined with the use of optimal test functions, cause us to reach this $p = 3$ level of accuracy *before* a more standard strategy? In this case, a "standard" strategy would be to solve the $p = 1$ DG problem above, inject this solution to a $p = 3$ space, then (rather than modifying the trial/test basis) simply perform block Jacobi smoothings on this $p = 3$ space until the solution converges.

Fig. E.2 shows the convergence of the right-boundary flux error for both the test/trial basis optimization strategy and the standard Jacobi smoothing. As we see, both strategies eventually reach the same $p = 3$ level of accuracy (which happens to be approximately 10^{-9} in this case), but the method using an optimized test/trial space converges with only half the number of iterations. This is a promising result, and verifies the idea of optimizing both test and trial functions simultaneously.

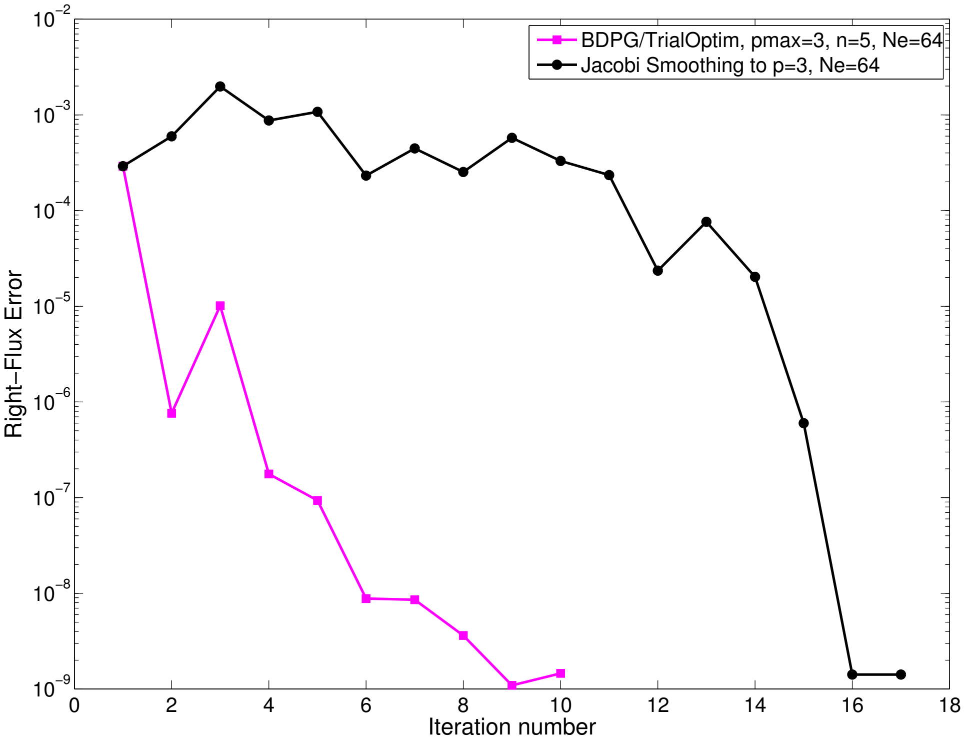


Figure E.2: Convergence of the flux along the right boundary of the domain for both (1) a standard block-Jacobi smoothing procedure, starting from a $p = 1$ DG solution and smoothing to a $p = 3$ solution, and (2) a trial space "optimization" procedure, in which the number of trial space basis functions n is kept fixed at 5 during later iterations, while only the *shape* of these functions is updated. Optimal test functions are computed simultaneously to request accuracy in the fluxes.

Finally, while the above problem was solved on a 64-element mesh, we can investigate how the same strategy performs on meshes with various numbers of elements, N_e. This performance is shown in Fig. E.3, along with the performance of standard block-Jacobi smoothing. We see that the optimal test/trial function strategy scales more favorably with mesh size compared to the Jacobi method. Indeed, it is nearly (though not quite) h-independent, so that the finer the mesh it is employed on, the more efficient it is compared to block-Jacobi smoothing. Of course, block-Jacobi smoothing is not the most efficient standard to compare against, but preliminary comparisons to a p-multigrid technique show that the optimal test/trial functions can outperform this method as well.

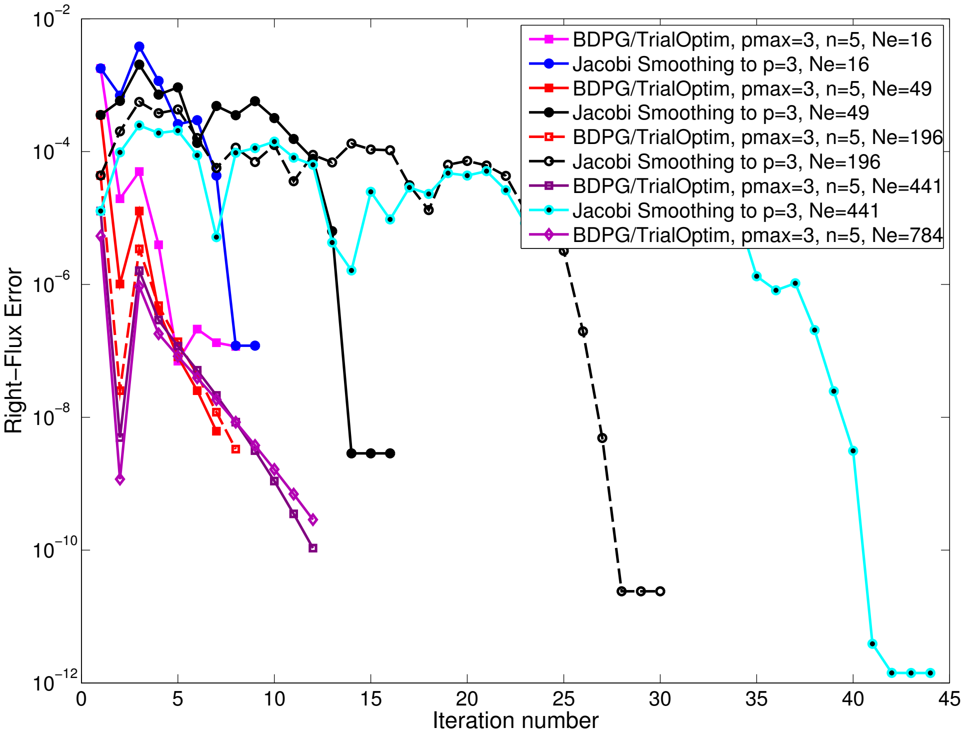


Figure E.3: Performance of test/trial function optimization compared to elementwise block Jacobi smoothing. Convergence is shown for various mesh sizes, each with a given number of elements N_e. The optimal test/trial functions strategy scales better than Jacobi smoothing with increasing mesh size.

While these results are preliminary, they suggest that a similar strategy of test/trial function optimizaiton could be worth pursuing for more general problems.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] Slimane Adjerid, Karen D. Devine, Joseph E. Flaherty, and Lilia Krivodonova. A posteriori error estimation for discontinuous Galerkin solutions of hyperbolic problems. *Computer Methods in Applied Mechanics and Engineering*, 191(11):1097–1112, 2002.

[2] Slimane Adjerid and Thomas C. Massey. A posteriori discontinuous finite element error estimation for two-dimensional hyperbolic problems. *Computer Methods in Applied Mechanics and Engineering*, 191(5152):5877 – 5897, 2002.

[3] Douglas N. Arnold, Franco Brezzi, Bernardo Cockburn, and L. Donatella Marini. Unified analysis of discontinuous Galerkin methods for elliptic problems. *SIAM Journal on Numerical Analysis*, 39(5):1749–1779, 2002.

[4] P. Barbone and I. Harari. Nearly H^1-optimal finite element methods. *Computer Methods in Applied Mechanics and Engineering*, 190:5679–5690, November 2000.

[5] JW Barrett and Karen W Morton. Approximate symmetrization and Petrov-Galerkin methods for diffusion-convection problems. *Computer Methods in Applied Mechanics and Engineering*, 45(1):97–122, 1984.

[6] Timothy Barth and Mats Larson. A posteriori error estimates for higher order Godunov finite volume methods on unstructured meshes. In R. Herban and D. Kröner, editors, *Finite Volumes for Complex Applications III*, pages 41–63, London, 2002. Hermes Penton.

[7] Timothy J. Barth. Space-time error representation and estimation in Navier-Stokes calculations. In Stavros C. Kassinos, Carlos A. Langer, Gianluca Iaccarino, and Parviz Moin, editors, *Complex Effects in Large Eddy Simulations*, pages 29–48. Springer Berlin Heidelberg, Lecture Notes in Computational Science and Engineering Vol 26, 2007.

[8] F. Bassi and S. Rebay. A high–order discontinuous finite element method for the numerical solution of the compressible Navier-Stokes equations. *Journal of Computational Physics*, 131:267–279, 1997.

[9] F. Bassi and S. Rebay. GMRES discontinuous Galerkin solution of the compressible Navier-Stokes equations. In Cockburn, Karniadakis, and Shu, editors,

Discontinuous Galerkin Methods: Theory, Computation and Applications, pages 197–208. Springer, Berlin, 2000.

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

[11] A. Belme, A. Dervieux, and F. Alauzet. Error estimation and adaptation for functional outputs in time-dependent flow problems. *Journal of Computational Physics*, 231:6323–6348, 2012.

[12] Michael Besier and Rolf Rannacher. Goal-oriented space-time adaptivity in the finite element galerkin method for the compution of nonstationary incompressible flow. *International Journal for Numerical Methods in Fluids*, 70:1139–1166, 2012.

[13] F. Brezzi, L.P. Franca, and A. Russo. Further considerations on residual-free bubbles for advective-diffusive equations. *Computer Methods in Applied Mechanics and Engineering*, 166:25–33, January 1998.

[14] Franco Brezzi, Bernardo Cockburn, L Donatella Marini, and Endre Süli. Stabilization mechanisms in discontinuous Galerkin finite element methods. *Computer Methods in Applied Mechanics and Engineering*, 195(25):3293–3310, 2006.

[15] A. Brooks and T.J.R Hughes. Streamline upwind/Petrov-Galerkin formulations for convection dominated flows with particular emphasis on the incompressible Navier-Stokes equations. *Computer Methods in Applied Mechanics and Engineering*, 32:199–259, September 1982.

[16] Tan Bui-Thanh, Leszek Demkowicz, and Omar Ghattas. A unified discontinuous Petrov–Galerkin method and its analysis for Friedrichs' systems. *SIAM Journal on Numerical Analysis*, 51(4):1933–1958, 2013.

[17] M. Celia, T. Russell, I. Herrera, and R. Ewing. An Eulerian-Lagrangian localized adjoint method for the advection-diffusion equation. *Advances in Water Resources*, 13:187–206, December 1990.

[18] Marco Ceze and Krzysztof J Fidkowski. Anisotropic hp-adaptation framework for functional prediction. *AIAA journal*, 51(2):492–509, 2012.

[19] Marco Ceze and Krzysztof J Fidkowski. Drag prediction using adaptive discontinuous finite elements. *Journal of Aircraft*, 51(4):1284–1294, 2014.

[20] Jesse Chan, Leszek Demkowicz, Robert Moser, and Nate Roberts. A new discontinuous Petrov-Galerkin method with optimal test functions. Part V: Solution of 1d Burgers and Navier–Stokes equations. The Institute for Computational Engineering and Sciences, The University of Texas at Austin, Tech Report No. 10-25, 2010.

[21] Jesse Chan, Norbert Heuer, Tan Bui-Thanh, and Leszek Demkowicz. A robust DPG method for convection-dominated diffusion problems II: Adjoint boundary conditions and mesh-dependent test norms. *Computers & Mathematics with Applications*, 67(4):771–795, 2014.

[22] Bernardo Cockburn, George E Karniadakis, and Chi-Wang Shu. *The development of discontinuous Galerkin methods*. Springer, 2000.

[23] Bernardo Cockburn, Mitchell Luskin, Chi-Wang Shu, and Endre Süli. Enhanced accuracy by post-processing for finite element methods for hyperbolic equations. *Mathematics of Computation*, 72(242):577–606, 2003.

[24] Johann P.S. Dahm and Krzysztof J. Fidkowski. Error estimation and adaptation in hybridized discontinous Galerkin methods. *Proceedings of the 52nd Aerospace Sciences Meeting*, 2014.

[25] L. Demkowicz and J. Gopalakrishnan. A class of discontinuous Petrov-Galerkin methods. Part I: The transport equation. *Computer Methods in Applied Mechanics and Engineering*, 199:1558–1572, April 2010.

[26] L. Demkowicz and J. Gopalakrishnan. A class of discontinuous Petrov-Galerkin methods. II. Optimal test functions. *Numerical Methods for Partial Differential Equations*, 27:70–105, 2011.

[27] L Demkowicz and JT Oden. An adaptive characteristic Petrov-Galerkin finite element method for convection-dominated linear and nonlinear parabolic problems in one space variable. *Journal of Computational Physics*, 67:188–213, 1986.

[28] Haibo Dong, Zongxian Liang, and Michael Harff. Optimal settings of aerodynamic performance parameters in hovering flight. *International Journal of Micro Air Vehicles*, 1(3):173–181, 2009.

[29] Donald Estep. A short course on duality, adjoint operators, Greens functions, and a posteriori error analysis. *Lecture Notes*, 2004.

[30] Donald Estep, Michael Holst, and Mats Larson. Generalized Green's functions and the effective domain of influence. *SIAM Journal on Scientific Computing*, 26(4):1314–1339, 2005.

[31] K. Fidkowski. High-order output-based adaptive methods for steady and unsteady aerodynamics. In H. Deconinck and R. Abgrall, editors, *37th Advanced VKI CFD Lecture Series*. von Karman Institute, 2013.

[32] K. Fidkowski. Output-based error estimation and mesh adaptation for steady and unsteady flow problems. In H. Deconinck and T. Horvath, editors, *38th Advanced VKI CFD Lecture Series*. von Karman Institute, 2015.

[33] Krzysztof J. Fidkowski. *A Simplex Cut-Cell Adaptive Method for High–order Discretizations of the Compressible Navier-Stokes Equations*. PhD thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, 2007.

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

[35] Krzysztof J. Fidkowski. An output-based dynamic order refinement strategy for unsteady aerodynamics. AIAA Paper 2012-77, 2012.

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

[37] Krzysztof J. Fidkowski and Yuxing Luo. Output-based space-time mesh adaptation for the compressible Navier-Stokes equations. *Journal of Computational Physics*, 230:5753–5773, 2011.

[38] Bryan T. Flynt and Dimitri J. Mavriplis. Discrete adjoint based adaptive error control in unsteady flow problems. AIAA Paper 2012-0078, 2012.

[39] Leopoldo P Franca, Sergio L Frey, and Thomas JR Hughes. Stabilized finite element methods: I. Application to the advective-diffusive model. *Computer Methods in Applied Mechanics and Engineering*, 95(2):253–276, 1992.

[40] Neal T. Frink. Test case results from the 3rd AIAA drag prediction workshop. NASA Langley, 2007. `http://aaac.larc.nasa.gov/tsab/cfdlarc/aiaa-dpw/Workshop3/final_results_jm.tar.gz`.

[41] Haiyang Gao and ZJ Wang. A conservative correction procedure via reconstruction formulation with the chain-rule divergence evaluation. *Journal of Computational Physics*, 232(1):7–13, 2013.

[42] M. B. Giles and E. Süli. Adjoint methods for PDEs: a posteriori error analysis and postprocessing by duality. In *Acta Numerica*, volume 11, pages 145–236, 2002.

[43] M.B. Giles and N.A. Pierce. Analytic adjoint solutions for the quasi-one-dimensional Euler equations. *Journal of Fluid Mechanics*, 426:327–345, 2001.

[44] D. Givoli. Non-local and semi-local optimal weighting functions for symmetric problems involving a small parameter. *International Journal for Numerical Methods in Engineering*, 26:1281–1298, 1988.

[45] Andreas Griewank and Andrea Walther. Revolve: An implementation of checkpointing for the reverse or adjoint mode of computational differentiation. *ACM Transactions on Mathematical Software*, 26(1):19–45, 2000.

[46] K. Harriman, D. Gavaghan, and E. Süli. The importance of adjoint consistency in the approximation of linear functionals using the discontinuous Galerkin finite element method. Technical Report Technical Report NA 04/18, Oxford University Computing Lab Numerical Analysis Group, 2004.

[47] R. Hartmann and P. Houston. Error estimation and adaptive mesh refinement for aerodynamic flows. In H. Deconinck, editor, *36^{th} CFD/ADIGMA course on hp-adaptive and hp-multigrid methods: VKI Lecture Series 2010-01 (Oct. 26-30, 2009)*. von Karman Institute for Fluid Dynamics, 2010.

[48] Ralf Hartmann. Adjoint consistency analysis of discontinuous Galerkin discretizations. *SIAM Journal on Numerical Analysis*, 45(6):2671–2696, 2007.

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

[50] Ralf Hartmann and Paul Houston. An optimal order interior penalty discontinuous Galerkin discretization of the compressible Navier-Stokes equations. *Journal of Computational Physics*, 227:9670–9685, 2008.

[51] I. Herrera. Trefftz method: A general theory. *Numerical Methods for Partial Differential Equations*, 16:561–580, November 2000.

[52] T.Y. Hou and P. LeFloch. Why non-conservative schemes converge to the wrong solutions: Error analysis. *Mathematics of Computation*, 62:497–530, 1994.

[53] Thomas JR Hughes, Gonzalo R Feijóo, Luca Mazzei, and Jean-Baptiste Quincy. The variational multiscale method – a paradigm for computational mechanics. *Computer Methods in Applied Mechanics and Engineering*, 166(1):3–24, 1998.

[54] T.J.R. Hughes. Multiscale phenomena: Green's functions, the Dirichlet-to-Neumann formulation, subgrid scale models, bubbles and the origins of stabilized methods. *Computer Methods in Applied Mechanics and Engineering*, 127:387–401, March 1995.

[55] TJR Hughes and G Sangalli. Variational multiscale analysis: the fine-scale Green's function, projection, optimization, localization, and stabilized methods. *SIAM Journal on Numerical Analysis*, 45(2):539–557, 2007.

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

[57] Antony Jameson. Aerodynamic design via control theory. *Journal of Scientific Computing*, 3:233–260, 1988.

[58] Steven M. Kast, Marco A. Ceze, and Krzysztof J. Fidkowski. Output-adaptive solution strategies for unsteady aerodynamics on deformable domains. ICCFD7 -3802, 2012.

[59] Steven M Kast, Johann P.S. Dahm, and Krzysztof J Fidkowski. A hybrid Petrov-Galerkin method for optimal output prediction. In *53rd AIAA Aerospace Sciences Meeting*. AIAA, 2015.

[60] Steven M. Kast, Johann P.S. Dahm, and Krzysztof J. Fidkowski. Optimal test functions for boundary accuracy in discontinuous finite element methods. *Journal of Computational Physics*, 298:360–386, 2015.

[61] Steven M. Kast and Krzysztof J. Fidkowski. Output-based mesh adaptation for high order Navier–Stokes simulations on deformable domains. *Journal of Computational Physics*, 252(0):468–494, 2013.

[62] Lilia Krivodonova and Joseph E Flaherty. Error estimation for discontinuous Galerkin solutions of two-dimensional hyperbolic problems. *Advances in Computational Mathematics*, 19(1-3):57–71, 2003.

[63] Ki-Hwan Lee, Juan J. Alonso, and Edwin van der Weide. Mesh adaptation criteria for unsteady periodic flows using a discrete adjoint time-spectral formulation. AIAA Paper 2006-692, 2006.

[64] Randall J. LeVeque. Numerical methods for conservation laws. In *Lecture Notes in Mathematics*, pages 122–129. Birkhauser, 1992.

[65] James Lu. *An a Posteriori Error Control Framework for Adaptive Precision Optimization Using Discontinuous Galerkin Finite Element Method*. PhD thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, 2005.

[66] Y. Luo and K.J. Fidkowski. Output-based space time mesh adaptation for unsteady aerodynamics. AIAA Paper 2011-491, 2011.

[67] Karthik Mani and Dimitri J. Mavriplis. Discrete adjoint based time-step adaptation and error reduction in unsteady flow problems. AIAA Paper 2007-3944, 2007.

[68] Karthik Mani and Dimitri J. Mavriplis. Error estimation and adaptation for functional outputs in time-dependent flow problems. *Journal of Computational Physics*, 229:415–440, 2010.

[69] D. J. Mavriplis. Results from the 3rd drag prediction workshop using the NSU3D unstructured mesh solver. AIAA Paper 2007-256, 2007.

[70] D. Meidner and B. Vexler. Adaptive space-time finite element methods for parabolic optimization problems. *SIAM Journal on Control Optimization*, 46(1):116–142, 2007.

[71] D Moro, NC Nguyen, and J Peraire. A hybridized discontinuous Petrov–Galerkin scheme for scalar conservation laws. *International Journal for Numerical Methods in Engineering*, 91(9):950–970, 2012.

[72] Siva K. Nadarajah and Antony Jameson. Optimum shape design for unsteady flows with time-accurate continuous and discrete adjoint methods. *AIAA Journal*, 45(7):1478–1491, 2007.

[73] Siva K. Nadarajah and Antony Jameson. Optimum shape design for unsteady three-dimensional viscous flows using a nonlinear frequency-domain method. *AIAA Journal of Aircraft*, 44(5):1513–1527, 2007.

[74] Marian Nemec and Michael J. Aftosmis. Error estimation and adpative refinement for embedded-boundary Cartesian meshes. AIAA Paper 2007-4187, 2007.

[75] NC Nguyen, J Peraire, and B Cockburn. Hybridizable discontinuous galerkin methods. In *Spectral and High Order Methods for Partial Differential Equations*, pages 63–84. Springer, 2011.

[76] Michael Ol and et al. Unsteady aerodynamics for micro air vehicles. NATO Research and Technology Organisation AVT-149, 2009.

[77] J. Peraire, N. C. Nguyen, and B. Cockburn. An embedded discontinuous Galerkin method for the compressible Euler and Navier-Stokes equations. AIAA Paper 2011-3228, 2011.

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

[79] Niles A. Pierce and Michael B. Giles. Adjoint recovery of superconvergent functionals from PDE approximations. *SIAM Review*, 42(2):247–264, 2000.

[80] W. Reed and T. Hill. Triangular mesh methods for the neutron transport equation. Los Alamos Scientific Laboratory Technical Report LA-UR-73-479, 1973.

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

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

[83] Markus P. Rumpfkeil and David W. Zingg. A general framework for the optimal control of unsteady flows with applications. AIAA Paper 2007-1128, 2007.

[84] Adrian Sandu. On the properties of Runge-Kutta discrete adjoints. In *Computational Science–ICCS 2006*, pages 550–557. Springer, 2006.

[85] Adrian Sandu. On consistency properties of discrete adjoint linear multistep methods. *Technical Report TR-07-40, Department of Computer Science, Virginia Polytechnic Institute and State University*, 2007.

[86] Michael Schmich and Boris Vexler. Adaptivity with dynamic meshes for space-time finite element discretizations of parabolic equations. *SIAM Journal on Scientific Computing*, 30(1):369–393, 2008.

[87] Jochen Schütz and Georg May. A hybrid mixed method for the compressible Navier–Stokes equations. *Journal of Computational Physics*, 240:58–75, 2013.

[88] S. Sen, K. Veroy, D.B.P. Huynh, S. Deparis, N.C. Nguyen, and A.T. Patera. “Natural norm” a posteriori error estimators for reduced basis approximations. *Journal of Computational Physics*, 217:37–62, 2006.

[89] W. Shyy, Y. Lian, J. Tang, H. Liu, P. Trizila, B. Stanford, L. Bernal, C. Cesnik, and P. Friedmann. Computational aerodynamics of low Reynolds number plunging, pitching and flexible wings for MAV applications. *Acta mechanica Sinica*, 24(4):351–373, 2008.

[90] Wei Shyy, Mats Berg, and Daniel Ljungqvist. Flapping and flexible wings for biological and micro air vehicles. *Progress in Aerospace Sciences*, 35:455–505, 1999.

[91] Pavel Solín, Karel Segeth, and Ivo DoleŽel. *Higher–Order Finite Element Methods*. Chapman and Hall, 2003.

[92] D.N. Srinath and Sanjay Mittal. An adjoint method for shape optimization in unsteady viscous flows. *Journal of Computational Physics*, 229:1994–2008, 2010.

[93] Ed Tinoco, David Levy, and Olaf Brodersen. Test case results from the 5th AIAA drag prediction workshop. NASA Langley, 2012. `http://aiaa-dpw.larc.nasa.gov/Workshop5/presentations/DPW5_Presentation_Files/14_DPW5%20Summary-Draft_V7.pdf`.

[94] John C. Vassberg, Mark A. DeHaan, and Tony J. Sclafani. Grid generation requirements for accurate drag predictions based on OVERFLOW calculations. AIAA Paper 2003-4124, 2003.

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

[96] Peter E Vincent, Patrice Castonguay, and Antony Jameson. A new class of high-order energy stable flux reconstruction schemes. *Journal of Scientific Computing*, 47(1):50–72, 2011.

[97] Li Wang, Dimitri Mavriplis, and W. Kyle Anderson. Adjoint sensitivity formulation for discontinuous galerkin discretizations in unsteady inviscid flow problems. *AIAA Journal*, 48(12):2867–2883, 2010.

[98] Qiqi Wang. Forward and adjoint sensitivity computation of chaotic dynamical systems. *Journal of Computational Physics*, 235:1–13, 2013.

[99] Qiqi Wang, Rui Hu, and Patrick Blonigan. Least Squares Shadowing sensitivity analysis of chaotic limit cycle oscillations. *Journal of Computational Physics*, 267:210–224, 2014.

[100] David J. Willis, Emily R. Israeli, Per-Olof Persson, Mark Drela, Jaime Peraire, Sharon M. Swartz, and Kenneth S. Breuer. A computational framework for fluid structure interaction in biologically inspired flapping flight. AIAA Paper 2007-3803, 2007.

[101] Michael Woopen, Georg May, and Jochen Schütz. Adjoint-based error estimation and mesh adaptation for hybridized discontinuous Galerkin methods. *International Journal for Numerical Methods in Fluids*, 76(11):811–834, 2014.

[102] Nail K. Yamaleev, Boris Diskin, and Eric J. Nielsen. Local-in-time adjoint-based method for design optimization of unsteady flows. *Journal of Computational Physics*, 229:5394–5407, 2010.

[103] Masayuki Yano. *An Optimization Framework for Adaptive Higher-Order Discretizations of Partial Differential Equations on Anisotropic Simplex Meshes*. PhD thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, 2012.

[104] Masayuki Yano and David L Darmofal. An optimization-based framework for anisotropic simplex mesh adaptation. *Journal of Computational Physics*, 231(22):7626–7649, 2012.

[105] J. Zitelli, I. Muga, L. Demkowicz, J. Gopalakrishnan, D. Pardo, and V. M. Calo. A class of discontinuous Petrov-Galerkin methods. Part IV: The optimal test norm and time-harmonic wave propagation in 1D. *Journal of Computational Physics*, 230:2406–2432, 2011.