SMOOTHNESS-INCREASING ACCURACY-CONSERVING FILTERS (SIAC) FOR DISCONTINUOUS GALERKIN SOLUTIONS

by

Hanieh Mirzaee

A dissertation submitted to the faculty of
The University of Utah
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

in

Computing

School of Computing

The University of Utah

May 2013

The University of Utah Graduate School

STATEMENT OF DISSERTATION APPROVAL

The dissertation of **Hanieh Mirzaee**
has been approved by the following supervisory committee members:

Robert M. Kirby	, Chair	**8/21/2012** Date Approved
Jennifer K. Ryan	, Member	Date Approved
Martin Berzins	, Member	**8/22/2012** Date Approved
Spencer J. Sherwin	, Member	Date Approved
Christopher R. Johnson	, Member	**8/22/2012** Date Approved

and by **Alan L. Davis** , Chair of
the Department of **School of Computing**

and by Charles A. Wight, Dean of The Graduate School.

ABSTRACT

Smoothness-increasing accuracy-conserving (SIAC) filters were introduced as a class of postprocessing techniques to ameliorate the quality of numerical solutions of discontinuous Galerkin (DG) simulations. SIAC filtering works to eliminate the oscillations in the error by introducing smoothness back to the DG field and raises the accuracy in the L^2-norm up to its natural superconvergent accuracy in the negative-order norm. The increased smoothness in the filtered DG solutions can then be exploited by simulation postprocessing tools such as streamline integrators where the absence of continuity in the data can lead to erroneous visualizations. However, lack of extension of this filtering technique, both theoretically and computationally, to nontrivial mesh structures along with the expensive core operators have been a hindrance towards the application of the SIAC filters to more realistic simulations.

In this dissertation, we focus on the numerical solutions of linear hyperbolic equations solved with the discontinuous Galerkin scheme and provide a thorough analysis of SIAC filtering applied to such solution data. In particular, we investigate how the use of different quadrature techniques could mitigate the extensive processing required when filtering over the whole computational field. Moreover, we provide detailed and efficient algorithms that a numerical practitioner requires to know in order to implement this filtering technique effectively. In our first attempt to expand the application scope of this filtering technique, we demonstrate both mathematically and through numerical examples that it is indeed possible to observe SIAC filtering characteristics when applied to numerical solutions obtained over structured triangular meshes. We further provide a thorough investigation of the interplay between mesh geometry and filtering. Building upon these promising results, we present how SIAC filtering could be applied to gain higher accuracy and smoothness when dealing with totally unstructured triangular meshes. Lastly, we provide the extension of our filtering scheme to structured tetrahedral meshes. Guidelines and future work regarding the application of the SIAC filter in the visualization domain are also presented. We further note that throughout this document, the terms *postprocessing* and *filtering* will be used interchangeably.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

It would not have been possible to write this doctoral thesis without the help and support of the kind people around me, to only some of whom it is possible to give particular mention here.

I wish to thank, first and foremost, my advisor Mike Kirby, for his supervision, advice, and guidance from the very early stages of this research. Above all and the most needed, he provided me unflinching encouragement and support in various ways. His truly scientific intuition along with his wide knowledge and logical way of thinking have exceptionally inspired and enriched my growth as a student, a researcher, and a scientist. Numerous unexpected circumstances could arise during a span of five years, and Mike showed more sympathy and patience than anyone could have asked for. For that, I am also extremely grateful.

I would also like to express my immense gratitude to Jennifer Ryan, whose role during the course of my PhD studies went above and beyond just that of a committee member. This thesis would not have been possible, had it not been for her support and valuable and insightful scientific discussions. I also wish to thank her and Kees Vuik, for kindly hosting me within the numerical modeling and simulation group at TU Delft in the Netherlands during the summers of 2010 and 2011.

I wish to acknowledge the rest of my committee members, Martin Berzins for helpful conversations, Spencer Sherwin for his support of the Nektar++ library, and Chris Johnson for providing an exceptional research environment at the Scientific Computing and Imaging (SCI) institute. I am thankful that in the midst of all their activities, they accepted to be members of my committee.

I am also grateful to the administrative staff at the SCI institute for their generous help and support. In particular, I wish to thank Deb Zemek, Magali Coburn, and Ed Cask.

Many friends have helped me stay sane during the times of difficulty. From my friends of almost two decades to more recent ones I made in Utah, their support and care helped me overcome setbacks and stay focused on my graduate study. I greatly value their friendship and I deeply appreciate their belief in me. I particularly thank Vincent Pegoraro, Samar

Emami, Neda Sadeghi, and Pooyan Amini for their presence, the joyful moments they created for me, and their limitless moral support.

Most importantly, none of this would have been possible without the love and patience of my family. My mother Maryam Azima, my father Karam Mirzaee, my brother Hesam, and my sister Hoda, to whom this dissertation is dedicated, have been a constant source of love, concern, support, and strength all these years. No mere paragraph can do justice to the appreciation I have for them. They are my everything.

Finally, I wish to acknowledge the financial support provided for this work by the Air Force Office of Scientific Research (AFOSR), Computational Mathematics program (Program Manager: Dr. Fariba Fahroo), under grant number FA9550-08-1-0156.

CHAPTER 1

INTRODUCTION

The discontinuous Galerkin (DG) method provides a high-order extension of the finite volume method and has been under rapid development during the past few years. The DG methodology allows for a dual path to convergence through both elemental h and polynomial p refinement. Due to this discretization flexibility, the discontinuous Galerkin method has increased in use steadily in such diverse applications as computational solid mechanics, fluid mechanics, acoustics, and electromagnetics (e.g., [9, 6, 58, 16]).

The primary mathematical advantage of DG is that unlike classic continuous Galerkin finite element method (FEM) which seeks approximations that are piecewise continuous, the DG methodology merely requires weak constraints on the fluxes between elements. This feature provides a flexibility which is difficult to match with conventional continuous Galerkin methods. However, lack of smoothness across elements can hamper simulation postprocessings like feature extraction and visualization. Many commonly used visualization techniques explicitly (or tacitly) assume that the field upon which they are acting is smooth. Applying such techniques under the nonideal cases of nonsmooth solutions can in the best case result only in a loss of convergence rate (or accuracy) and in the worst case can lead to erroneous visualization results.

To illustrate this point, we draw the reader's attention to streamline integration of fields produced by computational fluid mechanics simulations, which is a commonly used tool for the investigation and analysis of fluid flow phenomena. Integration is often accomplished through the application of ordinary differential equation (ODE) integrators– integrators whose error characteristics are predicted on the smoothness of the field through which the streamline is being integrated. This smoothness is not available at the interelement level of DG approximation data.

A class of postprocessing techniques were introduced in [19, 61] as a means of gaining increased accuracy from DG solutions through the exploitation of the superior convergence rates of DG in the negative-order norm; these filters have as a secondary consequence that they increase the smoothness of the output solution. Building upon these concepts, in [67, 75], SIAC filters were proposed as a means of ameliorating the challenges introduced by the lack of regularity at element interfaces while at the same time maintaining accuracy constraints that are consistent with the verification process used in the original simulation. In essence, in the application domain, one seeks to increase the smoothness *without destroying* (*i.e.,* by maintaining) the order of accuracy of the original input DG solution.

1.1 Contributions

The purpose of this dissertation is to further develop smoothness-increasing accuracy-conserving filters which respect the mathematical properties of the data while providing levels of smoothness so that commonly-used visualization tools can be used appropriately, accurately, and efficiently. The goals of this effort are to address the technical obstacles inherent in visualization of data derived from high-order discontinuous Galerkin methods and to provide robust and easy to use algorithms to overcome the difficulties that arise due to lack of smoothness. In particular, we contribute both mathematically and algorithmically to the class of smoothness-increasing and accuracy-conserving (SIAC) methods and provide a means to make this technique more suitable for real-life engineering problems. In meeting the goals, the following major contributions have been made:

- *A study of the numerical quadrature approximations used for evaluating the convolution operator in SIAC filters.* Theoretical estimates as well as empirical results that demonstrate the efficacy of the SIAC postprocessing approach when different levels and types of quadrature approximation are used is presented. This study is primarily for engineering circumstances when the trade-offs between time, resources, and accuracy are important. Here, we focus mainly on one-dimensional and two-dimensional quadrilateral implementations of the postprocessor over periodic domains, and we use as our gold-standard the solving of the convolution operation with *consistent integration* (integration that partitions the domain so as to respect all breaks in regularity) combined with Guassian integration that integrates the kernel times the DG-based polynomial exactly to double-precision machine zero. Alternatively, we quantify the impact of inexact quadrature on the filtering process and we investigate whether it greatly impacts the usage of the postprocessor as an intermediary stage between simulation and visualization in the scientific pipeline. These contributions are documented with permission in Chapter 4 and reported in the published peer-reviewed journal article: "Quantification of errors introduced in the numerical approximation and implementation of smoothness-increasing accuracy-conserving (SIAC) filtering of discontinuous Galerkin (DG) fields," H. Mirzaee, J. K. Ryan, and R. M. Kirby, Journal of Scientific Computing, Volume 45, Pages 447-470, 2010.

- *Application of the SIAC filters to structured triangular meshes.* The basic theoretical assumption in the previous implementations of the postprocessor limits the use to numerical solutions solved over a quadrilateral mesh. However, this assumption is restrictive, which in turn complicates the application of this postprocessing technique to general tessellations. We extend the current theoretical results to variable coefficient hyperbolic equations solved over structured triangular meshes and demonstrate the effectiveness of the application of this postprocessor to structured triangular meshes. We show that there is a direct theoretical extension to structured triangular meshes for hyperbolic equations with bounded coefficients. These contributions are documented with permission in Chapter 5 and reported in the published peer-reviewed journal article: "Smoothness-increasing accuracy-conserving (SIAC) postprocessing for discontinuous Galerkin solutions over structured triangular meshes," H. Mirzaee, L. Ji, J. K. Ryan, and R. M. Kirby, SIAM Journal of Numerical Analysis, Volume 49, Pages 1899-1920, 2011.

- *Improved errors versus higher order accuracy in applications of SIAC filters to DG solutions.* Smoothness-increasing accuracy-conserving (SIAC) filtering has demonstrated its effectiveness in raising the convergence rate for discontinuous Galerkin solutions from order $k + \frac{1}{2}$ to order $2k + 1$ for specific types of translation invariant meshes [19, 46]. Additionally, it improves the weak continuity in the discontinuous Galerkin method to $k - 1$ continuity. Typically, this improvement has a positive impact on the error quantity in the sense that it also reduces the absolute errors in the solution. However, not enough emphasis has been placed on the difference between superconvergent accuracy and improved errors. This distinction is particularly important when it comes to interpreting the interplay between geometry and filtering as introduced through meshing. The underlying mesh over which the DG solution is built is important because the tool used in SIAC filtering – convolution – is scaled by the geometric mesh size. This scaling heavily contributes to the effectiveness of the postprocessor. Although the choice of this scaling is straightforward when dealing with a uniform mesh, it is not clear what the impact of either a global or local scaling will be on either the absolute error or on the superconvergence properties of the postprocessor.

We present a study of this mesh scaling used in the SIAC filter and how it factors into the theoretical errors. These contributions are documented with permission in Chapter 6 and reported in the peer-reviewed journal article: "Smoothness-increasing accuracy-conserving (SIAC) filtering for discontinuous Galerkin solutions: Improved errors versus higher-order accuracy," J. King, H. Mirzaee , J. K. Ryan and, R. M. Kirby, Journal of Scientific Computing, In press, 2012.

- *Application of the SIAC filters to unstructured triangular meshes.* Although the DG methodology can be applied to arbitrary triangulations, the typical application of SIAC filters has been to discontinuous Galerkin solutions obtained over translation invariant meshes such as structured quadrilaterals and triangles. As the assumption of any sort of regularity, including the translation invariance of the mesh, is a hindrance towards making the SIAC filter applicable to real-life simulations, we demonstrate for the first time the behavior and complexity of the computational extension of this filtering technique to fully *unstructured* tessellations. We consider different types of unstructured triangulations and show that it is indeed possible to get reduced errors and improved smoothness in the filtered solution. These results are promising as they pave the way towards a more generalized SIAC filtering technique. These contributions are documented with permission in Chapter 7 and reported in the accepted journal article: "Smoothness-increasing accuracy-conserving (SIAC) filters for discontinuous Galerkin solutions over unstructured triangular meshes," H. Mirzaee, J. King, J. K. Ryan and, R. M. Kirby, SIAM Journal of Scientific Computing, accepted upon revision, 2012.

- *Application of the SIAC filters to structured tetrahedral meshes.* While there have been several attempts to demonstrate the usefulness of the SIAC filtering technique to nontrivial mesh structures, the application of the SIAC filter never exceeded beyond two-space dimensions. Thereby, we consider this contribution to be the very first attempt of its kind in demonstrating the potential usefulness of SIAC filtering when applied to real-world simulations. Here, we examine the effect of filtering over three-dimensional structured tetrahedral meshes. These types of meshes are generated by tetrahedralizing uniform hexahedra and therefore, while maintaining the structured nature of a hexahedral mesh, they exhibit an unstructured tessellation within each hexahedral element. We consider two examples of a hyperbolic equation and demonstrate that it is indeed possible to obtain the superconvergence accuracy of $2k+1$ through the application of the SIAC filter. These contributions are documented with permission in Chapter 8 and reported in the submitted journal article:"Smoothness-increasing accuracy-conserving (SIAC) filters for discontinuous Galerkin solutions: Application to structured tetrahedral meshes," H. Mirzaee , J. K. Ryan and, R. M. Kirby, SIAM Journal of Numerical Analysis, submitted, 2012.

The following contribution has been made in order to provide the necessary steps and algorithms used to obtain the results in the above contributions:

- *Efficient implementation of SIAC filtering for DG solutions.* Quite often, a numerical practitioner is interested in explicit steps to make a numerical scheme applicable. We explicitly define the steps to efficient computation of the postprocessor applied to different structured mesh tessellations. In addition, we explain how well the inexact postprocessor [48] performs computationally comparing to the exact scheme. Furthermore, as the SIAC filter is a good candidate for parallelization, we provide, for the first time, results that confirm anticipated performance scaling when parallelized on a shared-memory multiprocessor machine. These contributions are documented with permission in Chapter 3 and reported in the published peer-reviewed journal article: "Efficient implementation of smoothness-increasing accuracy-conserving (SIAC) filters for discontinuous Galerkin solutions," H. Mirzaee, J. K. Ryan, and R. M. Kirby, Journal of Scientific Computing, Volume 52, Pages 85-112, 2011.

1.2 Organization

We proceed in this dissertation as follows: In Chapter 2, we start by a brief introduction of classical choices for numerically solving partial differential equations (PDEs). We then continue by discussing the detail of the discontinuous Galerkin scheme and present the background and relevant work in filtering of numerical solutions. Chapter 3 provides the definition and properties of the SIAC filters. In this chapter, we provide error analysis that demonstrates the usefulness of SIAC filtering in extracting the higher order accuracy in the negative-order norm and obtaining superconvergence in the L^2-norm. Moreover, we present the detail of the implementation of SIAC filtering over structured meshes in 2D. Efficient algorithms along with timing results exhibiting the perfect parallelization of the SIAC filters on shared-memory multiprocessors are given. Chapter 4 discusses the various numerical quadrature strategies one may use in filtering numerical solutions. The purpose of this chapter is to demonstrate the trade-offs in gaining higher order accuracy and computational efficiency. In Chapter 5, we explain the numerical behavior of SIAC filtering for structured triangulations. In this chapter, we provide numerical proofs confirming the effectiveness of SIAC filtering of the numerical solutions of variable coefficient hyperbolic PDEs obtained over structured triangular meshes. Numerical results are also provided. In Chapter 6, we examine more general structured triangular meshes and demonstrate that it is indeed possible to obtain superconvergence of order $2k+1$ for these mesh types when the proper scaling of the filter, related to the translation invariant properties of the mesh, is employed. Furthermore, we introduce theoretical proof that these results can be extended to adaptive meshes that are constructed in a hierarchical manner – in particular, adaptive meshes whose elements are defined by hierarchical (integer) splitting of elements of size H, where H represents both the macro-element spacing used in the generation of the mesh and the minimum scaling used for the SIAC filters. Chapter 7 presents the computational extension of SIAC filtering to unstructured triangulations. In this chapter, we demonstrate for the first time the mathematical behavior and computational complexity of the extension of this filter to *unstructured* tessellations. We consider four examples: a simple Delaunay triangulation, a Delaunay triangulation with obvious change in element sizes, a Delaunay triangulation with splitting, and a stretched (anisotropic) triangulation. We show that it is indeed possible to obtain order improvement and accuracy enhancement through a proper choice of kernel scaling. Chapter 8 discusses the extension of SIAC filtering for structured tetrahedral meshes. While there have been several attempts to demonstrate the usefulness of this filtering technique to nontrivial mesh structures, the application of the SIAC filter never exceeded beyond two-space dimensions. Thereby, we consider the contribution of this chapter to be the very first attempt of its kind in demonstrating the potential usefulness of SIAC filtering when applied to real-world simulations. Lastly, Chapter 9 discusses ongoing and future research.

CHAPTER 2

NUMERICAL SCHEMES AND PREVIOUS WORK

As [59] puts it, there are three important steps in the computational modeling of any physical process: (i) problem definition, (ii) mathematical model, and (iii) computer simulation.

Typically, the starting point is a given mathematical model which has been formulated in an attempt to explain and understand an observed phenomenon in biology, chemistry, physics, economics, or any other scientific or engineering discipline. In defining such a model, we expect to gain a well-posed problem that has a unique solution for a given set of parameters. Normally, we concentrate on those mathematical models which are (piecewise) continuous and are difficult or impossible to solve analytically; this is usually the case in practice. Relevant application areas in scientific computing and computer science include the Navier-Stokes equations in fluid dynamics which provide an accurate representation of the fluid motion and the equations of elasticity in structural mechanics that govern the deformation of a solid object due to applied external forces. These are complex general equations that are very difficult to solve both analytically and computationally.

In order to solve such a model approximately on a computer, the continuous or piecewise continuous problem is approximated by a discrete one. Functions are approximated by finite arrays of values. Algorithms are then sought which approximately solve the mathematical problem efficiently, accurately, and reliably. Throughout this dissertation, we consider the numerical solutions of mathematical models of conservation laws which are described by partial differential equations (PDEs). The three classical choices for the numerical solution of PDEs are the finite difference method (FDM), the finite element method (FEM), and the finite volume method (FVM).

In this chapter, we provide a brief overview of conservation laws in their integral and differential forms in Section 2.1. Section 2.2 discusses the salient features of finite difference, finite volume, and finite element methods and provides the necessary background which will lead us to the discontinuous Galerkin method in Section 2.3. Finally, in Section 2.4, we briefly review the development of postprocessing techniques devised to improve the quality of numerical approximations.

2.1 Conservation Laws: Integral and Differential Forms

If a system does not interact with its environment in any way, then certain mechanical properties of the system cannot change. These quantities are said to be *conserved* and the corresponding conservation laws state that this particular measurable property of our isolated physical system does not change as the system evolves. Conservation laws are considered to be the most fundamental principles of mechanics. Examples of such conserved quantities include energy, momentum, and angular momentum.

As stated in [59], the general principle behind the derivation of conservation laws is that the rate of change of $u(x,t)$ within a volume V plus the flux of u denoted by $f(u)$ through the boundary A of the volume is equal to the rate of production of u denoted by $S(u,x,t)$ which can be written as

$$\frac{\partial}{\partial t}\int_V u(x,t)\mathrm{d}V + \int_A f(u)\cdot\mathbf{n}\mathrm{d}A - \int_V S(u,x,t)\mathrm{d}V = 0, \tag{2.1}$$

where n is the unit outward normal to the boundary A. Equation (2.1) is referred to as the *integral form* of the conservation law. For a fixed volume and (independent of t), under suitable conditions of smoothness of the intervening quantities, we can apply Gauss' theorem

$$\int_V \nabla\cdot f\mathrm{d}V = \int_A f\cdot\mathbf{n}\mathrm{d}A \tag{2.2}$$

to obtain

$$\int_V \left(\frac{\partial u}{\partial t} + \nabla\cdot f(u) - S\right)\mathrm{d}V = 0. \tag{2.3}$$

For the integral expression to be zero for any volume V, the integrand must be zero. This results in the *strong* or differential form of the equation

$$\frac{\partial u}{\partial t} + \nabla\cdot f(u) - S = 0. \tag{2.4}$$

As mentioned in [31], the construction of any numerical method for solving a partial differential equation requires one to consider the two choices:

- How does one represent the solution $u(x,t)$ by an approximate solution $u_h(x,t)$?
- In which sense will the approximate solution $u_h(x,t)$ satisfy the partial differential equation?

These two choices separate the different methods and define the properties of each method.

2.2 Finite Difference, Finite Volume, and Finite Element Schemes

The simplest and historically oldest method for numerically solving PDEs is known as the *finite difference method*, which is based upon the application of a local Taylor expansion to approximate the differential equations. In this approach, a grid, x_k, $k = 1, \cdots, K$, is laid down in space and spacial derivatives are approximated by difference methods; that is, the conservation law in the strong form given by Equation (2.4) in one-space dimension is approximated as

$$\frac{du_h(x_k, t)}{dt} + \frac{f_h(x_{k+1}, t) - f_h(x_{k-1}, t)}{h_k + h_k - 1} = S(x_k, t) \tag{2.5}$$

where u_h and f_h are the numerical approximations to the solution and the flux, respectively, and $h_k = x_{k+1} - x_k$ is the local grid size. Here we have used the central difference approximation of $\nabla \cdot f(u)$. Inserting these local approximations into Equation (2.4) results in the residual

$$x \in [x_{k-1}, x_{k+1}] : R_h(x, t) = \frac{\partial u_h}{\partial t} + \frac{\partial f_h}{\partial x} - S(x, t). \tag{2.6}$$

Clearly, $R_h(x, t)$ is not zero, as in that case, $u_h(x, t)$ would satisfy Equation (2.4) exactly and would be the solution $u(x, t)$. Therefore, we need to specify in which way u_h must satisfy the equation, which leads to a statement about the residual. If we have a total of K grid points and, thus, K unknown grid point values $u_h(x_k, t)$, a natural choice is to require that the residual vanishes exactly at these grid points. This results in exactly K finite difference equations of the type in Equation (2.5) for the K unknowns, completing the scheme.

One of the most appealing aspects of this method is its simplicity; that is, the discretization of general problems and operators is often intuitive and, for many problems, leads to very efficient schemes. Furthermore, the explicit semidiscrete form gives flexibility on the choice of timestepping methods if needed. Finally, these methods are supported by an extensive body of theory, they are sufficiently robust and efficient to be used for a variety of problems, and extensions to higher order approximations by using a local polynomial approximation of higher degree is relatively straightforward. However, the finite difference method uses a topologically square network of lines to construct the discretization of the PDE. Additional complications caused by the simple underlying structure are introduced around boundaries and discontinuous internal layers. These make the native finite difference method not suitable to deal with complex geometries, both in terms of general computational domains and internal discontinuities as well as for local order and grid size changes to reflect local features of the solution.

The above discussion highlights that to ensure geometric flexibility, one needs to abandon the simple one-dimensional approximation in favor of something more general. This issue motivated the use of an integral form of the PDEs and subsequently the development of the *finite element* and *finite volume* techniques. These schemes are better suited to real-world applications than the standard finite difference method as the integral formulations do not rely on any special mesh structure. [1]

The finite element and finite volume schemes use the integral form of the equation as the starting point of the discretization process. For example, if the strong form of the PDE is $\mathcal{L}(u) = s$, the integral form is given by [59]

$$\int_0^1 \mathcal{L}(u)w(x)\,dx = \int_0^1 s\,w(x)dx, \tag{2.7}$$

where the choice of the weight function $w(x)$ defines the type of the scheme.

A method closely related to the finite difference method, but with added geometric flexibility, is the finite volume method. Here the region of integration is taken to be a control volume Ω_i, associated with the point coordinate x_i, represented by $x_{i-(1/2)} \le x \le x_{i+(1/2)}$, and the conservation law given in Equation (2.4) in one-space dimension in integral form becomes

$$\int_{x_{i-(1/2)}}^{x_{i+(1/2)}} u_t\,dt + \int_{x_{i-(1/2)}}^{x_{i+(1/2)}} f_x(u)\,dx = 0, \tag{2.8}$$

where we assumed $S = 0$. This expression could also be obtained from the weighted residual form given in Equation (2.7) by selecting a weight $w(x)$ such that $w(x) = 1$ for $x_{i-(1/2)} \le x \le x_{i+(1/2)}$ and 0 elsewhere. The last term in Equation (2.8) can be evaluated analytically to obtain

$$\int_{x_{i-(1/2)}}^{x_{i+(1/2)}} f_x(u)dx = f(u_{i+(1/2)}) - f(u_{i-(1/2)})$$

and if we approximate the first integral using the midpoint rule, we get the following semi-discrete form

$$u_t|_i(x_{i+(1/2)} - x_{i-(1/2)}) + f(u_{i+(1/2)}) - f(u_{i-(1/2)}) = 0, \tag{2.9}$$

where $u_t|_i$ is the value of the solution at x_i. This approach produces a *conservative* scheme if the flux on the boundary of one cell equals the flux on the boundary of the adjacent cell. For linear problems and equidistant grids, this method reduces to the finite difference method. However, one easily realizes that the formulation is less restrictive in terms of the grid structure; that is, the reconstruction of the solution at the interfaces is a local procedure

[1]We should note that there has been extensive ongoing research to make the finite difference method work on unstructured meshes. See [2] for an example.

and generalizes straightforwardly to unstructured grids in high dimensions, thus ensuring the desired geometric flexibility. If, however, we need to increase the order of accuracy of the method, a fundamental problem emerges. To reconstruct the interface values at a higher accuracy, we need information from more adjacent cells. In the simple one-dimensional case, this can be done similarly, as for the finite difference scheme. However, the need for a high-order reconstruction reintroduces the need for a particular grid structure and thus destroys the geometric flexibility of the finite volume method in higher dimensions. On unstructured grids, this approach requires a reconstruction based on genuinely multivariate polynomials with general cell center locations which is both complex and prone to stability problems [31]. The main limitation of the finite volume method is found in its inability to extend to higher order accuracy on general unstructured grids. This motivates the development of the next class of schemes known as the finite element schemes.

Let us first redefine the element Ω_i as the interval bounded by the grid points $[x_i, x_{i+1}]$ and with a total of K elements and $K+1$ grid points. Note that this is slightly different from the finite volume scheme where the element was defined by staggered grid points as $\left[x_{i-(1/2)}, x_{i+(1/2)}\right]$. Here we assume that the solution is expressed globally in the form

$$u_h(x,t) = \sum_{i=1}^{N} u_i(t) N_i(x)$$

where we have introduced the use of a locally defined basis function, $N_i(x)$. In the finite element method, we use expansion bases with compact support which are piecewise continuous polynomials within each element. In the simplest case, we can take these basis functions to be linear [59]. To recover the scheme to solve the conservation law given in Equation (2.4) in one-space dimension, following the weighted residual form given in Equation (2.7), we set the weight function $w(x)$ to be the same as the basis function $N_i(x)$, i.e., $w(x) = N_i(x)$, and we arrive at the following integral form

$$\int_{\Omega} \left(\frac{\partial u_h}{\partial t} + \frac{\partial f_h}{\partial x} - S_h \right) N_j(x) dx = 0, \tag{2.10}$$

for $j = 1 \cdots K$. Straightforward manipulations yield the scheme

$$M \frac{du_h}{dt} + S f_h = M S_h, \tag{2.11}$$

where M_{ij} and S_{ij} reflect the globally defined mass and stiffness matrices, respectively.

This approach, which presents the essence of the classic finite element method [33, 68, 78, 79], clearly allows different element sizes. Furthermore, we recall that a main motivation for considering methods beyond the finite volume approach was the interest in higher order approximations. Such extensions are relatively simple in the finite element setting and can be achieved by adding additional degrees of freedom to the element. In particular, one can have different orders of approximations in each element, thereby enabling local changes in both size and order, known as *hp*-adaptivity [24, 25].

However, the above discussion also highlights disadvantages [31]. First, we see that the globally defined basis functions and the requirement that the residual be orthogonal to the same set of globally defined test functions implies that the semidiscrete scheme becomes implicit and M must be inverted. For time-dependent problems, this is a clear disadvantage compared to finite difference and finite volume methods. On the other hand, for problems with no explicit time dependence, this is less of a concern.

There is an additional subtle issue that is related to the structure of the basis. From the discussion above, we recognize that the basis functions are symmetric in space. For many types of problems (*e.g.,* the heat equation), this is a natural choice. However, for problems such as wave problems and conservation laws, in which information flows in specific directions, this is less natural and can cause stability problems if left unchanged [33, 77]. In finite difference and finite volume methods, this problem is addressed by the use of upwinding, either through the stencil choice or through the design of the reconstruction approach.

Reflecting on the previous discussion, one realizes that to ensure geometric flexibility and support for locally adopted resolution, we must strive for an element-based method where high-order accuracy is enabled through the local approximation. However, the global Galerkin statement, introduced by the globally defined basis and test (weight) functions, destroys the locality of the scheme and introduces potential problems with the stability for wave-dominated problems. On the other hand, this is precisely the regime where the finite volume method has several attractive features.

An intelligent combination of the finite element and the finite volume methods, utilizing a space of basis and test functions that mimics the finite element method but satisfying the equation in a sense closer to the finite volume method, appears to offer many of the desired properties. This combination is exactly what leads to the discontinuous Galerkin finite element method that will be discussed next.

2.3 The Discontinuous Galerkin Method

Problems of particular interest in which *convection* plays an important role arise in applications as diverse as meteorology, weather-forecasting, oceanography, gas dynamics, aeroacoustics, turbomachinery, turbulent flows, granular flows, oil recovery simulation, modeling of shallow water, transport of contaminant in porous media, viscoelastic flows, semiconductor device simulation, magnetohydrodynamics, and electromagnetism, among many others [15]. This is why devising robust, accurate, and efficient methods for numerically solving these problems is of considerable importance and, as expected, has attracted the interest of many researchers and practitioners.

The discontinuous Galerkin method makes use of the same function space as the continuous method, but with relaxed continuity at the interelement boundaries. It was first introduced by Reed and Hill [60] for the solution of the neutron transport equation, and its history and recent development have been reviewed by Cockburn *et al.* [15, 14]. The essential idea of the method is derived from the fact that basis functions can be chosen that either the field variable or its derivatives, or generally both, are considered discontinuous across the element boundaries, while the computational domain continuity is maintained. From this point of view, the discontinuous finite element method includes, as its subset, both the finite element method and the finite difference method (or finite volume) method [43]. Therefore, it has the advantages of both finite difference and finite element methods, in that it can be effectively used in convection-dominant applications, while maintaining geometric flexibility and higher local approximations through the use of higher order elements. This feature makes it uniquely useful for computational dynamics and heat transfer. Because of the local nature of a discontinuous formulation, no global matrix needs to be assembled; thus, this reduces the demand on the in-core memory. The effects of the boundary conditions on the interior field distributions then gradually propagate through the element-by-element connection. This is another important feature that makes this method useful for fluid flow calculations.

To illustrate the basic ideas of the discontinuous Galerkin method, we consider the transport equation given below

$$\begin{aligned} u_t + \nabla \cdot (au) &= 0 \qquad \text{in}\, \mathbb{R}^d \times (0,T), \\ u(t=0) &= u_0 \qquad \text{on}\, \mathbb{R}^d. \end{aligned} \tag{2.12}$$

We consider only the discretization of this equation in space. For full discretization of this equation, please consult [13]. Note also that the Equation in (2.12) is the conservation law given in Equation (2.4) where $f(u) = au$, a being a constant coefficient, and $S = 0$.

To discretize the transport equation in space by using a DG method, we first triangulate the domain $\mathbb{R}^d$; we denote such triangulation by $\mathcal{T}_h$. We then seek a discontinuous approximate solution u_h which, in each element K of the triangulation $\mathcal{T}_h$, belongs to the space $V(K)$. There is no restriction on how to choose the space $V(K)$, though a typical choice is the space of polynomials of degree at most k, $P^k(K)$. We determine the approximate solution on the element K, we multiply Equation (2.12) by a weight function $v(x) \in V(K)$, and substitute the approximate solution u_h to get the following weak form

$$\int_K (u_h)_t v - \int_K a u_h \cdot \nabla v + \int_{\partial K} \widehat{a u_h} \cdot \mathbf{n} ds = 0, \tag{2.13}$$

where we have have used the Gauss (divergence) theorem to obtain the last two terms, n is the unit outward normal, and $\widehat{au_h}$ is the numerical flux.

The difference between this scheme and the previously discussed finite element method is that V_h here is a broken space– space of piecewise discontinuous polynomials– which leads to local mass and stiffness matrices. At first, the locality also appears problematic as this statement does not allow one to recover a meaningful global solution. Furthermore, how does one ensure uniqueness of solution at element interfaces? This is where the concept of numerical flux comes handy. The main purpose of the numerical flux term $\widehat{au_h}$ is to connect the elements. In what follows, we give a more detailed description of the terms involved in one-space dimension.

2.3.1 Revisiting the Transport Problem in One Dimension

Let us consider the linear scalar transport (or wave) equation in one dimension

$$\frac{\partial u}{\partial t} + \frac{\partial f(u)}{\partial x} = 0, \qquad x \in [L, R] = \Omega, \tag{2.14}$$

where the linear flux is given as $f(u) = au$. This is subject to the appropriate initial condition

$$u(x, 0) = u_0(x).$$

Boundary conditions are given when the boundary is an inflow boundary [31], that is

$$\begin{aligned} u(L,t) &= g(t) \qquad \text{if } a \geq 0, \\ u(R,t) &= g(t) \qquad \text{if } a \leq 0. \end{aligned}$$

As we mentioned earlier, we approximate Ω by K nonoverlapping elements, $K = [x_l^k, x_r^k]$. On each of these elements, we express the local solution as a polynomial of order $N = N_p - 1$

$$x \in K : u_h^k(x,t) = \sum_{n=1}^{N_p} \hat{u}_n^k(t)\, \psi_n(x) = \sum_{i=1}^{N_p} u_h^k(x_i^k, t)\, \ell_i^k(x).$$

These are two complementary expressions for the local solution. In the first one, known as the *modal* form, we have used a local polynomial basis, $\psi_n(x)$. In the alternative form, which is known as the *nodal* form, we use N_p local grid points, and express the polynomial through the associated interpolating Lagrange polynomial, $\ell_i^k(x)$. We have used the modal representation throughout this dissertation.

The global solution $u(x,t)$ is then assumed to be approximated by the piecewise N-th order polynomial approximation $u_h(x,t)$ defined as the direct sum of the K local polynomial solutions $u_h^k(x,t)$. We continue by forming the residual

$$\mathcal{R}_h(x,t) = \frac{\partial u_h}{\partial t} + \frac{\partial a u_h}{\partial x}, \tag{2.15}$$

and going back to the two main questions presented in Section 2.1, we must decide in which sense this residual should vanish. To answer this question, we continue by introducing a globally defined space V_h of test functions, $\bigcup_{k=1}^{K} V_h^k$, where the locally defined spaces are defined as $V_h^k = span\ \psi_n(K)_{n=1}^{N_p}$. We recognize V_h as the space of piecewise smooth functions defined on Ω_h. We now require that the residual is orthogonal to all test functions in V_h, resulting in the local statement

$$\int_K \mathcal{R}_h(x,t)\psi_n(x)dx = 0,\ 1 \leq n \leq N_p, \tag{2.16}$$

on all K elements. This yields exactly N_p equations for the N_p local unknowns on each element. However, we have not imposed any particular constraints on the basis or test functions, and thus, we have neglected the issue of how to impose boundary conditions and how to recover the global solution from the K local solutions. Assume that the test functions in V_h are smooth but not continuous or otherwise constrained across interfaces. Spatial integration by parts of Equation (2.16) yields

$$\int_K \left(\frac{\partial u_h^k}{\partial t}\psi_n - a u_h^k \frac{d\psi_n}{dx} \right) dx = -\left[a u_h^k \psi_n \right]_{x_l^k}^{x_r^k} = -\int_{\partial K} \mathbf{n} \cdot a u_h^k \psi_n dx,\ 1 \leq n \leq N_p,$$

where $\mathbf{n}$ represents the local outward pointing normal. The use of a surface integral may seem a bit artificial in this simple example, but it makes generalization very natural. In this one-dimensional case, n is simply a scalar and takes the value $+1$ and -1 at the right and left interface, respectively.

We see that as a consequence of the lack of conditions on the local solution and the test functions, the solution at the interface between elements is multiply defined and we will need to choose which solution, or combination of solutions, is correct. We refer to this solution as $\widehat{au_h}$, known as the numerical flux. This leads to the semidiscrete scheme

$$\int_K \left(\frac{\partial u_h^k}{\partial t}\psi_n - a u_h^k \frac{d\psi_n}{dx} \right) dx = -\int_{\partial K} n.\widehat{au_h}\psi_n dx,\ 1 \leq n \leq N_p. \tag{2.17}$$

We recover a total of $K \times N_p$ equations for the same number of unknowns; that is, we have defined a method for obtaining the globally defined solution. This is also the so-called weak formulation.

It is understood from Equation (2.17), it is the right-hand side that is responsible for recovering the global solution from the local solutions and imposing the boundary conditions. This emphasizes the key role played by the numerical flux, $\widehat{au_h}$.

In the classical discontinuous Galerkin method, as we mentioned previously, the space of test functions are the same as the solution space. The weak form in Equation (2.17) can also be written as

$$\mathcal{M}^k \frac{d}{dt}\hat{u}_h^k - \left(\hat{S}^k\right)^T a\hat{u}_h^k = -(\widehat{au_h})\psi(x_r^k) + (\widehat{au_h})\psi(x_l^k), \tag{2.18}$$

where

$$\hat{\mathcal{M}}_{ij}^k = (\psi_i, \psi_j)_K, \qquad \hat{S}_{ij}^k = \left(\psi_i, \frac{d\psi_j}{dx}\right)_K, \tag{2.19}$$

are the local and stiffness mass matrices, respectively. Furthermore, we have

$$\hat{u}_h^k = \left[\hat{u}_1^k, \cdots, \hat{u}_{N_p}^k\right]^T, \psi = \left[\psi_1(x), \cdots, \psi(x)_{N_p}\right]^T, \tag{2.20}$$

as the vector of the local solution and the local test functions, respectively. The scheme in Equation (2.18) is the classical discontinuous Galerkin method. To complete the definition of the DG method, it only remains to define the numerical flux $\widehat{au_h}$.

The specification of the numerical flux is most naturally related to the the dynamics of the partial differential equation being solved. At the left end of the local domain K, this numerical flux should be a function of $\left[au_h^{k-1}(x_r^{k-1}), au_h^k(x_l^k)\right]$ while the right end depends on $\left[au_h^k(x_r^k), au_h^{k+1}(x_l^{k+1})\right]$. A simple interpretation is that $\widehat{au_h}$ is the flux one would wish to know at the interface. Alternatively, the role of the flux is to guarantee the stability of the formulation by mimicking the flow of information in the underlying partial differential equation.

To derive the stability condition for our DG scheme, one would follow the approach of the energy method [3], replace ψ_n with u_h in the weak formulation in Equation (2.17), and add on the elements K. We refer the interested reader to [13, 31] for the detailed mathematical formulas. After this step, we find out that the numerical flux will have the following form in order to guarantee stability

$$\widehat{au_h} = au_h + C[\![u_h]\!], \tag{2.21}$$

where

$$u_h = \frac{1}{2}(u_h^+ + u_h^-), \qquad [\![u_h]\!] = u_h^- n^- + u_h^+ n^+.$$

u_h^+ and u_h^- denote the left and right value at the interface and C is a non-negative definite matrix. This completes the definition of our DG scheme.

There are two main examples of DG methods considering the flux definition in Equation (2.21) [13]. The first uses the following choice for the parameter C: $\frac{1}{2}|a \cdot \mathbf{n}|\, I$, where I is the identity matrix. This implies that the numerical flux is

$$\widehat{au_h}(x) = \lim_{\epsilon \to 0} u_h(x - \epsilon a),$$

which is nothing but the classical *upwinding numerical flux.*

The second example is when we take $C = \frac{1}{2}|a|\, I$. For this choice, we have

$$\widehat{au_h}(x) = au_h + \frac{1}{2}|a|\, [\![u_h]\!], \tag{2.22}$$

which is the so-called *Lax-Friedrichs numerical flux* [42].

From the two examples above, we see that the DG methods are strongly related to finite volume methods. Indeed, the method of lines, that is, the discretization in space, for the upwinding scheme and the local Lax-Freidrichs scheme coincide with the corresponding DG method under consideration when the local space $V(K)$ is taken to be the space of constant functions. Moreover, the DG methods, like finite volume methods, can easily handle complex computational domains. However, unlike finite volume methods, DG achieves higher order accuracy easily. Indeed, a theoretical order of convergence of $k + 1/2$ can be proven simply by requiring that the local spaces $V(K)$ contain all polynomials of degree at most k. Moreover, this is achieved while keeping a high degree of locality since to increase the degrees of freedom of the approximate solution u_h in an element, only the degrees of freedom of u_h in the immediate neighbors are involved. We also add the fact that the mass matrix is block diagonal, and hence easily invertible, which renders the DG schemes extremely parallelizable when they are discretized in time by, for example, an explicit Runge-Kutta method.

2.4 Previous Work

In order to introduce the basic ideas of this work and to put them into proper perspective, we briefly review the development of postprocessing techniques devised to improve the quality of numerical approximations. For further detail, the reader should consult lecture notes of Wahlbin [74] on superconvergence in Galerkin finite element methods as well as [19].

In 1978, Mock and Lax [52] showed that for a difference scheme of any formal order of accuracy μ, for linear hyperbolic systems, the *moments* of the exact solution converge with order μ provided that the initial data were suitably preprocessed. This result holds even when the exact solution contains discontinuities. They also showed how to postprocess the approximate solution by a simple convolution to enhance its accuracy over regions of smoothness of the exact solution. If the solution were sufficiently smooth locally, they could obtain nearly the full order of convergence μ provided that the support of the kernel was of order almost one. This seems to have been the first instance when the ideas of (i) preprocessing the initial data, (ii) obtaining an error estimate for the moments, and (iii) postprocessing the approximation appear clearly delineated. Gottlieb and Tadmor [30], motivated by the work of Mock and Lax [52], found a spectrally accurate postprocessing kernel for spectral methods (see also [45]). Again, the full spectral accuracy could be recovered by using a convolution; the measure of the support of the kernel had to be of order one.

Quite independently of the review above, in 1977, Bramble and Schatz [11] considered linear elliptic problems and showed how to postprocess the finite element solution by means of a simple convolution to enhance the quality of the approximation. They showed that the order of convergence could be doubled if the exact solution were locally smooth. In 1977, Thomée [70] extended the work of Bramble and Schatz [11] to include superconvergence of the derivatives and gave an elegant proof of their approximation results by using Fourier analysis. In 1980, he extended these results [71] to semidiscrete Galerkin finite element methods for parabolic problems.

It is important to point out that, just like Mock and Lax, Bramble and Schatz proved a *negative-order norm* error estimate (an error estimate of the moments in Mock and Lax's terminology) and then showed how to use it to enhance the approximation by convolution. However, unlike Mock and Lax's convolution kernel, for locally translation invariant grids, the Bramble-Schatz kernel has support in a cube whose diameter is of order h (mesh characteristic) only; this fact represents a considerable advantage from the computational point of view.

In 1981, Johnson and Nävert [37] applied this technique to steady-state advection-diffusion problems with small diffusion; they considered the standard Galerkin and the streamline-diffusion methods. An application of this technique to the simulation of miscible displacement was devised and analyzed in 1985 by Douglas [38]. Other applications can be found in the book of Wahlbin [74].

According to Cockburn *et al.* [19] it seems that the first (and only) attempt to apply this technique to finite element methods for hyperbolic problems was carried out in 1993 by Bales [10] who considered a fourth-order accurate finite element method applied to a one-dimensional wave equation. Built upon the framework initially established by Bramble and Schatz [11] and Mock and Lax [52], Cockburn, Luskin, Shu, and Süli [19, 17] introduced a class of postprocessors for hyperbolic PDEs using the discontinuous Galerkin method. In [19, 17], the authors considered the postprocessing of the discontinuous Galerkin approximation to time-dependent linear hyperbolic systems. In this case, they show that the postprocessor improves the accuracy from order $k + 1$ to order $2k + 1$ for linear hyperbolic systems solved over a locally uniform mesh, namely $h = \Delta x_i$ for all i in the support of the postprocessor. Moreover, the postprocessor consists of a convolution kernel applied to the approximation only once, at the final time, and is independent of the partial differential equation under consideration as long as the necessary negative-order norm error estimate can be proven. The negative-order norm error estimates give us information on the oscillatory nature of the error and should be of higher order than the L^2-norm error estimates for the postprocessor to be applicable. The postprocessor extracts this information and works to filter out oscillations in the error and to enhance the accuracy in the usual L^2-norm, up to the order of the error estimates in the negative-order norm. Our work in this dissertation is based on the filtering technique introduced by Cockburn *et al.* in [19] applied to linear hyperbolic equations.

CHAPTER 3

OVERVIEW OF THE SIAC FILTERS

The Smoothness-Increasing Accuracy-Conserving (SIAC) filters were first introduced as a class of postprocessors for the discontinuous Galerkin method applied to hyperbolic equations in [18, 19]. This filtering technique was extended to a broader set of applications such as being used for filtering within streamline visualization algorithms in [20, 22, 62, 61, 67, 75]. Here we provide an overview of the structure and properties of this filtering technique. Furthermore, quite often a numerical practitioner is interested in explicit steps to make a numerical scheme applicable. In this chapter, we explicitly define the steps to efficient computation of the postprocessor applied to different mesh tessellations. In addition, as the SIAC filter is a good candidate for parallelization, we provide, for the first time, results that confirm anticipated performance scaling when parallelized on a shared-memory multiprocessor machine. We add that postprocessing over unstructured triangulations will be discussed in Chapter 7.

We proceed in this chapter by providing the definition and properties of the convolution kernel in Section 3.1. We then continue by demonstrating how SIAC filtering works to extract the higher order accuracy hidden in the DG solution in Section 3.2. The detail of the construction of the convolution kernel is given in Section 3.3. The implementation of the SIAC filter will be discussed in Section 3.4 in one dimension. Moving on to higher dimensions, we provide implementation details for quadrilateral and hexahedral meshes in Section 3.4.1 and for triangular mesh structures in Section 3.4.2. In Section 3.5, we provide performance analysis as well as the parallel implementation of the postprocessor. The result of this contribution has been published in [49].

3.1 The Convolution Kernel

The postprocessor itself is simply the discontinuous Galerkin solution at the final time T, convolved against a B-splines kernel $K^{r+1,k+1}$. That is, in one dimension,

$$
\begin{aligned}
u^\star(x) =& K^{r+1,k+1} \star u_h, \\
=& \frac{1}{h}\int_{-\infty}^{\infty} K^{r+1,k+1}\left(\frac{y-x}{h}\right) u_h(y)dy,
\end{aligned} \tag{3.1}
$$

where $u^\star$ is the postprocessed solution and h is the mesh characteristic length. The superscript $r+1, k+1$ typically represents the number of B-splines used in the convolution kernel as well as the B-spline order. In the following discussions, we shall drop this superscript for the sake of a less cluttered explanation.

The convolution kernel in the SIAC filter is a linear combination of B-splines. We note that filtering in most visualization applications has as its goal the reconstruction of a continuous function from a given (discrete) data set. For example, assume that f_k is the given set of evenly sampled points of some function $f(x)$. A filter might take this set of points and introduce some type of continuity assumption to create the reconstructed solution $f^\star(x)$. Filtering for visualization based upon discrete data is often done using convolution with some type of spline, often a cubic B-splines [57, 56, 28, 32, 51, 63]. Much of the literature concentrates on the specific use in image processing, though there has also been work in graphic visualization [12, 72] and computer animation [32].

There are many filtering techniques that rely on the use of splines in filtering. A good overview of the evaluation of filtering techniques is presented in [53]. In [54], Möller *et al.* further discuss imposing a smoothness requirement for interpolation and derivative filtering. In [57], the methods of nearest neighbor interpolation and cubic splines are compared. Hou and Andrew [32] specifically discuss the use of cubic B-splines for interpolation with applications to image processing. The interpolation consists of using a linear combination of five B-splines with the coefficients determined by the input data. This is a similar approach to the one discussed throughout this dissertation. Another method of filtering for visualization via convolution is presented in [54]. This methods chooses an even number of filter weights for the convolution kernel to design a filter based on smoothness requirements. The authors also discuss classifying filters and extend the analysis to the spatial domain. We can also relate our filter to those evaluated by Mitchell and Netraveli [51], where they design a reconstruction filter for images based on piecewise cubic filters, with the B-spline filter falling into this class. In [51], it is noted that a 2D separable filter is desirable, as is the case with the B-spline filter discussed in this work. Further discussion on spline filters can be found in [34, 65, 66]. Here, we focus on a type of spline filter that is used to improve the numerical solutions obtained from a discontinuous Galerkin scheme.

The kernel in Equation (3.1) has the following form

$$K(x) = \sum_{\gamma=0}^{r} c_\gamma \psi^{(k+1)}(x - x_\gamma), \tag{3.2}$$

where $\psi^{(k+1)}$ is the B-spline of order $k+1$, c_γ are the kernel coefficients, and $r = 2k$ represents the number of B-splines used in the kernel. Furthermore, x_γ represent the positions of the kernel nodes and are given by

$$x_\gamma = -\frac{r}{2} + \gamma, \qquad \gamma = 0, \cdots, r. \tag{3.3}$$

The kernel in Equation (3.2) has a *symmetric* form (Figure 3.1 left) that can be used for postprocessing in the interior of the domain. This type of kernel requires information from both sides of an evaluation point at which we wish to calculate the postprocessed value. To perform filtering near boundaries or shocks, we need to use a *one-sided* form of the kernel that requires information only from one side of the boundary or shock (Figure 3.1 right). In general, we can generate such a kernel by shifting the positions of the kernel nodes to one side. This shifting can be done by using a shift function $\lambda(x)$ so that the new kernel positions $x_\gamma + \lambda(x)$ all reside on the one side of the boundary or shock. We note that in this work, we always consider periodic meshes and solutions; hence, only the symmetric kernel is implemented. However, the ideas presented in this document can easily adapt to the one-sided kernel. For more information on one-sided postprocessing, consult [62, 73].

The convolution kernel given in Equation (3.2) has three mains properties [19]:

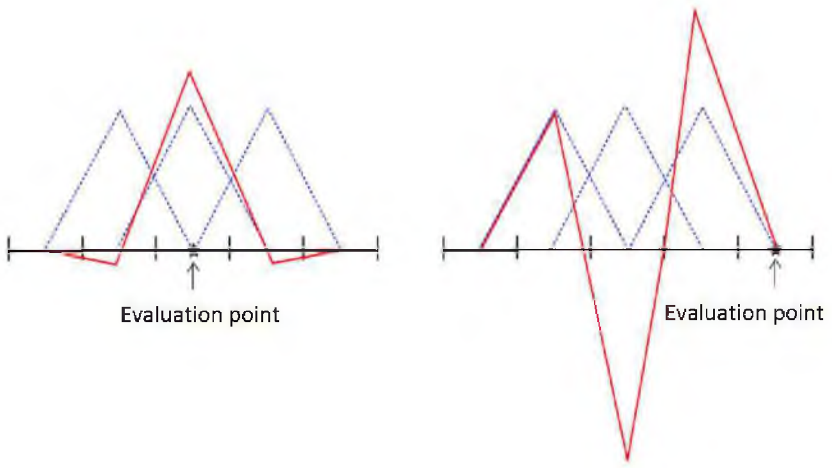


Figure 3.1. Symmetric (left) versus one-sided (right) kernel. Symmetric kernel uses information from both sides of the evaluation point. Solid red line represents the kernel and dashed blue lines depict the constructing B-splines of order two ($k = 1$).

1. K has a compact support.

2. It reproduces polynomials of degree up to r by convolution. For example, considering monomials we get
$$K \star x^p = x^p \qquad p = 0, 1, \cdots, r. \tag{3.4}$$
This guarantees that the accuracy of order $r+1$ is not destroyed. Moreover, it provides a mechanism for calculating the kernel coefficients c_γ. More detail is given in Section 3.3.

3. It allows us to express derivatives of the convolution with the kernel in terms of simple difference quotients. This is the consequence of the kernel being a linear combination of B-splines. Indeed it is not difficult to verify that for multi-indices α and β such that $\beta_i \geq \alpha_i$ for $i = 1, \cdots, d$, we have
$$D^\alpha(\psi_H^\beta \star u) = \psi^{\beta-\alpha} \star \partial_H^{\boldsymbol{\alpha}} u, \tag{3.5}$$
where $\psi_H^{\boldsymbol{\alpha}}(x) = \psi^{(\alpha/H)}/H^d$, $\partial_H^{\boldsymbol{\alpha}} := \partial_{H,1}^{\alpha_1} \cdots \partial_{H,d}^{\alpha_d}$ and
$$\partial_{H,j} u(x) = \frac{1}{H}(u(x + \frac{1}{2}He_j) - u(x - \frac{1}{2}He_j)). \tag{3.6}$$
For $\alpha = 1$, $\partial_H^{\boldsymbol{\alpha}}$ is simply the central difference operator. This property can be exploited in the finite element framework and in the theoretical proofs, as will be seen in the next section.

Given the necessary background on the convolution kernel, we continue by demonstrating how we can extend the higher order accuracy in the negative-order norm to the L^2-norm through the application of the SIAC filter.

3.2 Extracting the Higher Order Accuracy in DG Solutions

We begin this section by stating the main theorem in filtering numerical solutions from Bramble and Schatz [11] valid for locally uniform mesh structures:

Theorem 3.2.1 *Let u_h be any numerical approximation to u and $K_h^{2k+1,k+1}$ the kernel given in Equation (3.2). For $T > 0$ and sufficient smoothness of u we have*
$$\|u(T) - K_h^{2k+1,k+1} \star u_h\|_{0,\Omega} \leq \mathsf{C}_1 h^{2k+1} + \mathsf{C}_2 \sum_{|\alpha| \leq k+1} \|\partial_h^\alpha (u - u_h)\|_{-(k+1),\Omega} \tag{3.7}$$
where C_1 and C_2 are independent of h.

What we would ultimately like to show is

$$\|u(T) - K_h^{2k+1,k+1} \star u_h\|_{0,\Omega} \leq \mathsf{C}h^{2k+1}, \tag{3.8}$$

where C depends solely on the smoothness of the solution and is independent of h.

We note here that the notation $\|u\|_{0,\Omega}$ represents the standard L^2-norm of a function u on Ω. In general, for any natural number ℓ, we consider the norm of the Sobolev space $H^\ell(\Omega)$, defined by

$$\|u\|_{\ell,\Omega} = \left\{ \sum_{|\alpha|\leq\ell} \|D^\alpha u\|^2{}_{0,\Omega} \right\}^{1/2}. \tag{3.9}$$

Moreover, $\|u\|_{-\ell,\Omega}$ denotes the negative-order norm of a function u and its definition will be given later in this section.

In this work, we consider u_h to be an approximation obtained by the DG methodology. In this case, Equation (3.8) indicates that through the application of the SIAC filter, we can pass from $O(h^{k+1})$ accuracy in the L^2-norm of DG solutions to $O(h^{2k+1})$ in the L^2-norm of filtered DG solutions. According to Theorem 3.2.1, we can bound the error in the L^2-norm by a higher order term of h^{2k+1} accuracy and the negative-order norm of the divided differences of the error. If this second term, *i.e.*, negative-order norm of the divided differences of the error, is also of higher order h^{2k+1}, we arrive at Equation (3.8) and hence the effectiveness of SIAC filtering in raising the order of accuracy in the L^2-norm.

Let us now demonstrate how we can bound the L^2-norm, as given in Theorem 3.2.1. Note that we can rewrite the estimate in Equation (3.7) as

$$\|u - K_h^{2k+1,k+1} \star u_h\|_{0,\Omega} \leq \|u - K_h^{2k+1,k+1} \star u\|_{0,\Omega} + \|K_h^{2k+1,k+1} \star (u - u_h)\|_{0,\Omega}. \tag{3.10}$$

To estimate the first term in Equation (3.10), we consider a Taylor series expansion of $u(x,T)$ around a point y and of degree $2k$. We denote the Taylor polynomial by $T^{2k+1}u(y,x)$ which is given by

$$T^{2k+1}u(y,x) = \sum_{|\alpha|=0}^{2k} \frac{D^\alpha u(y)}{\alpha!}(x-y)^\alpha$$

and

$$R^{2k+1}u(y,x) = u(x) - T^{2k+1}u(y,x) = \sum_{|\alpha|=2k+1}^{\infty} \frac{D^\alpha u(y)}{\alpha!}(x-y)^\alpha.$$

Consequently, we arrive at

$$\begin{aligned} u(x) - K_h \star u(x) &= u(x) - K_h \star (T^{2k+1}u(y,.) + R^{2k+1}u(y,.))(x) \\ &= u(x) - K_h \star T^{2k+1}u(y,x) - K_h \star R^{2k+1}u(y,x) \\ &= R^{2k+1}u(y,x) - K_h \star R^{2k+1}u(y,x), \end{aligned}$$

where the third equation is resulted from the second property of the kernel mentioned in the previous section which indicates that the kernel reproduces polynomials of degree up to r (here $r = 2k$) by convolution.

For $y = x$, the above expression becomes

$$\begin{aligned} u(x) - K_h \star u(x) &= -\int_{R^d} K_h(z) R^{2k+1}(x, x-z)dz \\ &= -\int_{\mathcal{I}} K(z) R^{2k+1}u(x, x-hz)dz, \end{aligned}$$

where the second equation is resulted from $K_h = \frac{1}{h}K(\frac{x}{h})$ and $\mathcal{I}$ is the support of the kernel.

Consequently, we obtain the error bound on the first term of Equation (3.10) as

$$\|u - K_h^{2k+1,k+1} \star u\|_{0,\Omega} \leq \|K^{2k+1,k+1}\|_{L1(\mathbb{R}^d)} \sup_{z\in\mathcal{I}} \|R^{2k+1}u(x, x-hz)\|_{0,\Omega_0}.$$

We note that $R^{2k+1}u(x, x-hz)$ in the above equation is given by

$$R^{2k+1}u(x, x-hz) = \frac{D^\alpha u(\epsilon)}{\alpha!}(-hz)^\alpha = (-1)^{2k+1}\frac{h^{2k+1}}{2k+1!}D^\alpha u(\epsilon)z^\alpha, \tag{3.11}$$

where $|\alpha| = 2k+1$. Ultimately, we arrive at the following error bound

$$\|u - K_h^{2k+1,k+1} \star u\|_{0,\Omega} \leq \mathsf{C}_1 h^{2k+1}. \tag{3.12}$$

At this stage, it only remains to show that the second term in Equation (3.10) is of higher order which then results in the error estimate in Equation (3.8).

If u is a function in $L^2(\Omega)$, it can be shown that (Bramble and Schatz [11], Lemma 4.2)

$$\|u\|_{0,\Omega} \leq \mathsf{C} \sum_{|\alpha|\leq\ell} \|D^\alpha f\|_{-\ell}. \tag{3.13}$$

This indicates that we can derive error estimates for the standard L^2- norm in terms of the negative-order norm of the divided differences. Substituting u with $K_h^{2k+1,k+1} \star (u - u_h)$ and ℓ with $k+1$ we obtain:

$$\|K_h^{2k+1,k+1} \star (u - u_h)\|_{0,\Omega} \leq \mathsf{C}_2 \overbrace{\sum_{|\alpha|\leq k+1} \|D^\alpha(K_h^{2k+1,k+1} \star (u-u_h))\|_{-(k+1),\Omega}}^{\Theta}. \tag{3.14}$$

We further emphasize that this is an important step to pass from the usual L^2-norm to the negative-order Sobolev norm.

To bound the error in Equation (3.14), we make use of the third property of the kernel presented in the previous section to get

$$\begin{aligned}\Theta \leq & C_2 \sum_{|\alpha|\leq k+1} \|K_h^{2k+1,k+1-\alpha} \star \partial_h^\alpha(u-u_h)\|_{-(k+1),\Omega} \\ \leq & C_2 \sum_{|\alpha|\leq k+1} \|K_h^{2k+1,k+1-\alpha}\|_{L^1(\mathbb{R}^d)} \|\partial_h^\alpha(u-u_h)\|_{-(k+1,\Omega)} \\ \leq & C_2 \sum_{|\alpha|\leq k+1} \|\partial_h^\alpha(u-u_h)\|_{-(k+1),\Omega}. \end{aligned} \tag{3.15}$$

In other words, by using the property of the kernel to rewrite the derivative of the convolution as the convolution with the divided differences, we are able to bound Θ by the negative-order norm of the divided differences of the error. Consequently, it is now clear how the inequality relation given in Theorem 3.2.1 is obtained. Moreover, if we demonstrate that this negative-order norm of the divided differences of the error is of higher order, our initial claim given in Equation (3.8) is realized. For this, we continue by providing the definition of the negative-order norm.

A negative-order norm is a norm equipped with the dual space of the Sobolev space $H^\ell(\Omega)$. A dual space of a vector space consists of all linear functionals (linear maps) of that vector space. For example, for $u \in H^\ell$ and $\phi \in \mathcal{C}_0^\infty(\Omega)$, $\varphi = (u, \phi)$, *i.e.*, the inner product of u and ϕ, denotes a linear functional on u and thereby φ belongs to the dual space of $H^\ell(\Omega)$. We denote this space by $H^{-\ell}(\Omega)$ and define its associated norm as

$$\|u\|_{-\ell,\Omega} = \sup_{\phi \in \mathcal{C}_0^\infty(\Omega)} \frac{(u,\phi)}{\|\phi\|_{\ell,\Omega}}. \tag{3.16}$$

Equation (3.16) is what we refer to as the negative-order norm. We emphasize that although $\|u\|_{-\ell,\Omega}$ is associated with $H^{-\ell}(\Omega)$, here we only use this as a quantitative measure for functions in L^2.

In Equation (3.16), if we consider the simplified domain $\Omega = [-\pi, \pi]$, then the complex exponential functions $e^{inx} = \cos(nx) + i\sin(nx)$, $n \in \mathcal{Z}$ is an orthonormal basis for $L^2([-\pi,\pi])$. By assigning $\phi = e^{inx}$, $\frac{(u,\phi)}{\|\phi\|_{\ell,\Omega}}$ will result in normalized Fourier coefficients in the Fourier series expansion of u. Consequently, the negative-order norm in this case will yield the supremum Fourier coefficient.

Similarly, the negative-order norm of the divided differences of the error as given in Equation (3.15) will be

$$\|\partial_h^\alpha(u-u_h)\|_{-\ell,\Omega} = \sup_{\phi \in \mathcal{C}_0^\infty(\Omega)} \frac{(\partial_h^\alpha(u-u_h),\phi)}{\|\phi\|_{\ell,\Omega}}. \tag{3.17}$$

The norm given in Equation (3.17) needs to be of higher order accuracy (using any numerical approximation u_h) in order to achieve higher order accuracy in the L^2-norm through convolution. Normally what happens in practice is that we demonstrate, through mathematical proofs, the higher order accuracy of $O(h^{2k+1})$ in the negative-order norm of the error $u-u_h$ by applying a duality argument for locally uniform meshes. Proofs for the divided differences will be similar. Cockburn *et al.* have sketched the detail of the proof of higher order accuracy in the negative-order norm of the error for a DG scheme of linear hyperbolic equations in [19], therefore, we avoid repeating that discussion here. Details of the proof for variable coefficient linear hyperbolic equations will be given in Section 5.2.

By demonstrating that Θ in Equation (3.15) can be estimated with high accuracy, the estimate in Equation (3.10) will also be of higher accuracy and therefore, we provided in this section how one can raise the order of accuracy in the L^2-norm through the application of the SIAC filter.

3.3 Construction of the Kernel

We remind the reader that the postprocessor is simply the discontinuous Galerkin solution at the final time T, convolved against a linear combination of B-splines. That is, in one dimension,

$$u^\star(x) = \frac{1}{h}\int_{-\infty}^{\infty} K^{r+1,k+1}\left(\frac{y-x}{h}\right) u_h(y)dy, \tag{3.18}$$

where $u^\star$ is the postprocessed solution, h is the characteristic length and

$$K^{r+1,k+1}(x) = \sum_{\gamma=0}^{r} c_\gamma^{r+1,k+1}\psi^{(k+1)}(x-x_\gamma), \tag{3.19}$$

is the convolution kernel. $\psi^{(k+1)}$ is the B-spline of order $k+1$ and $c_\gamma^{r+1,k+1}$ represent the kernel coefficients. x_γ represent the positions of the kernel nodes and are defined as:

$$x_\gamma = -\frac{r}{2}+\gamma, \qquad\qquad \gamma = 0,\cdots,r=2k. \tag{3.20}$$

B-splines form the core elements of spline filters. The B-splines that we consider for our convolution kernel are more specifically referred to as *central B-splines.* A central B-spline of degree k is a B-spline with knots $x_j = -\frac{k+1}{2}+j_{j=0,\cdots,k+1}$ and it can be defined by recursive convolutions of the characteristic function with itself.

$$\begin{aligned}\psi^{(1)}(x) =& \chi_{[-1/2,1/2]},\\ \psi^{(k+1)}(x) =& (\psi^{(1)}\star\psi^{(k)})(x) := \int_{-\infty}^{\infty}\psi^{(1)}(\xi)\psi^{(k)}(x-\xi)d\xi, \qquad k=1,2,3,\cdots.\end{aligned} \tag{3.21}$$

By evaluating the integral in Equation (3.21), we arrive at the following recursion relations:

$$\begin{aligned}\psi^{(1)}(x) =& \chi_{[-1/2,1/2]},\\ \psi^{(k+1)}(x) =& \frac{1}{k}\left(\left(\frac{k+1}{2}+x\right)\psi^{(k)}\left(x+\frac{1}{2}\right)+\left(\frac{k+1}{2}-x\right)\psi^{(k)}\left(x-\frac{1}{2}\right)\right),\end{aligned} \tag{3.22}$$

Figure 3.2 depicts the B-splines of different orders. From these plots, we observe that as the order increases, the smoothness and support increase and the maximum value of the B-spline decreases.

A B-spline of order $k+1$ is a piecewise polynomial of degree k over each individual interval separated by the B-spline knots. Using Equation (3.21), one can also calculate the polynomial coefficients as fractions, *a priori*, store them in a matrix, and then use some polynomial evaluation scheme such as Horner's method to evaluate the B-spline at some arbitrary point. As an example, for $k=2$, the B-spline has the form

$$\psi^{(3)}(x) = \begin{cases} \frac{1}{2}x^2 + \frac{3}{2}x + \frac{9}{8}, & x \in [-\frac{3}{2}, -\frac{1}{2}), \\ -x^2 + \frac{3}{4}, & x \in [-\frac{1}{2}, \frac{1}{2}), \\ \frac{1}{2}x^2 - \frac{3}{2}x + \frac{9}{8}, & x \in [\frac{1}{2}, \frac{3}{2}], \\ 0, & \text{otherwise}. \end{cases} \tag{3.23}$$

We should also note that from the aforementioned definitions, it is obvious that the B-splines have compact support, meaning that a B-spline $\psi^{(k+1)}(x)$ of degree k with knots $x_0 < \cdots < x_{k+1}$ is zero outside of $[x_0, x_{k+1}]$, where $x_0 = -\frac{(k+1)}{2}$ and $x_{k+1} = \frac{(k+1)}{2}$. This leads to a more efficient scheme for B-spline evaluations since the points outside this interval simply result in a zero value. Algorithm 1 depicts the pseudo-code for evaluating the B-spline at a particular point x. We further note that B-splines have been well-studied, and we refer the interested reader to [35, 64, 23] for a more thorough discussion.

Now that the B-splines are defined, they can be used to construct the convolution kernel in the SIAC filters. However, the kernel coefficients c_γ remain to be defined.

One of the important properties of the kernel, as mentioned in the previous chapter, is that it reproduces polynomials up to a certain degree r, which equals $2k$ for the symmetric kernel postprocessing we consider in this work. This means that the convolution of the

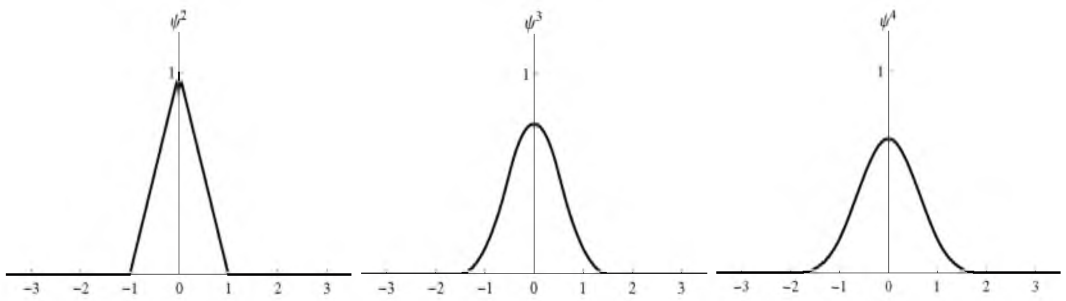


Figure 3.2. B-splines of order 2, 3, and 4. Note that as the order increases, the smoothness and support increase and the maximum value of the B-spline decreases.

Algorithm 1 Evaluating $\psi^{(k+1)}(x)$

```
1: if $x$ in $[x_0, x_{k+1}]$ then
2:     Find the interval of $x$, call it $[x_j, x_{j+1}]$
3:     $\psi^{(k+1)}(x)$ = value of the corresponding polynomial over $[x_j, x_{j+1}]$ (as in Equation
       3.21) at $x$
4: else
5:     $\psi^{(k+1)}(x) = 0$
6: end if
```

kernel with a polynomial of degree less than or equal to r is equal to that polynomial itself. In addition, it guarantees that the accuracy of the DG approximation is not destroyed by the convolution. Using the monomials, we obtain the following linear system for the kernel coefficients:

$$\sum_{\gamma=0}^{r} c_\gamma \int_{\mathbb{R}} \psi^{(k+1)}(y)(y - x - x_\gamma)^m dy = x^m, \quad m = 0, 1, \cdots, r. \tag{3.24}$$

To calculate the integral in Equation (3.24), we use Gaussian quadrature with $\frac{k+m}{2}+1$ quadrature points [40]. As an example for $k = 1$, Equation (3.24) gives

$$\begin{bmatrix} 1 & 1 & 1 \\ x+1 & x & x-1 \\ x^2+2x+\frac{7}{6} & x^2+\frac{1}{6} & x^2-2x+\frac{7}{6} \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} 1 \\ x \\ x^2 \end{bmatrix}. \tag{3.25}$$

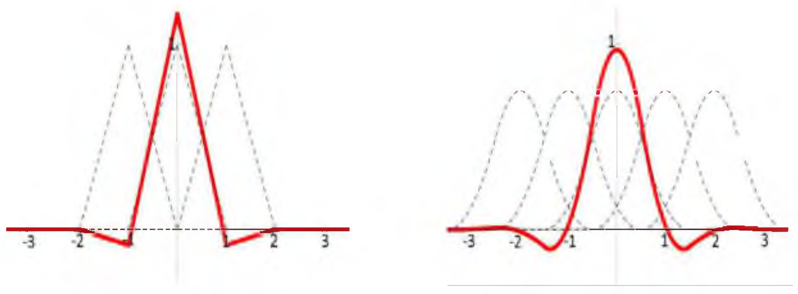
Equation (3.25) must hold for all x; we simply set $x = 0$ and obtain the coefficients

$$\begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} -\frac{1}{12} \\ \frac{7}{6} \\ -\frac{1}{12} \end{bmatrix}. \tag{3.26}$$

The linear system in Equation (3.24) is a nonsingular system. Hence, the existence and uniqueness of the kernel coefficients is guaranteed (see [19] for a proof). Linear algebra routines provided by the LAPACK library can be used for solving the system (visit www.netlib.org/lapack/). Figure 3.3 provides a schematic of the kernel function for $k = 1$ and $k = 2$.

Algorithm 2 provides the pseudo-code for constructing the matrix of the linear system mentioned in Equation (3.24). Note that for calculating the integral in Equation (3.24) (Line 12 in Algorithm 2), the integration region $\mathbb{R}$ actually reduces to $[-(k+1)/2, (k+1)/2]$, and that we need to divide this region into subintervals that respect the continuity breaks in the B-splines. This is required for the integral to be evaluated exactly to machine precision using Gaussian quadrature.

Having defined the kernel, we continue by demonstrating how to implement the post-processor operator by evaluating the integral in Equation (3.18).



(a) $k = 1$ (b) $k = 2$

Figure 3.3. The convolution kernels for $k = 1$ (a) and $k = 2$ (b). Dashed lines represent the constructing B-splines.

Algorithm 2 Constructing the B-spline coefficient matrix

```
$rowSize = r + 1$
$colSize = r + 1$
$LinMatrix[rowSize][colSize]$
$bsplineKnots = [-\frac{k+1}{2}, -\frac{k+1}{2} + 1, \cdots, \frac{k+1}{2}]$
for $row = 0$ to $rowSize$ do
  for $col = 0$ to $colSize$ do
    $LinMatrix[row][col] = 0$
    $\gamma = col$
    $x_\gamma$=-$\frac{r}{2}$+$\gamma$
    {Evaluate the integral in Equation (3.24)}
    for $i = 0$ to $size(bsplineKnots) - 1$ do
      $\Omega = [bsplineKnots[i], bsplineKnots[i + 1]]$
      $x$ = map the Gaussian quadrature points obtained over $[-1, 1]$ to $\Omega$
      $LinMatrix[row][col] += \int_\Omega \psi^{(k+1)}(x)(x + x_\gamma)^{row} dx$ using Gaussian quadrature
    end for
  end for
end for
```

3.4 Evaluation of the Convolution Operator

Traditionally, SIAC filters are implemented as small matrix-vector multiplications [61]. That is, considering a fixed number of evaluation points per element, a number of coefficient matrices are produced. These are computed one time and stored for future use. The postprocessing is then implemented in a simple manner via these small matrix-vector multiplications of the prestored coefficient matrices and the coefficients of the numerical solution. However, as this approach is not suitable for the more general case of unstructured meshes, we discuss in this section how the integral in Equation (3.18) can be evaluated directly.

We begin by introducing the notion of a *standard region* (sometimes referred to as the reference element). In order to evaluate a DG approximation at an arbitrary point or to compute an integral using Gaussian quadrature, we often first need to map the points to a standard region (see [40]). In this section, we introduce the 1D standard element, Ω_{st} such that

$$\Omega_{st} = \{\xi | -1 \leq \xi \leq 1\}\,. \tag{3.27}$$

Therefore, to evaluate our DG approximation $u_h(x)$ in Equation (3.18) at a point x defined on the interval $I_i = [x_a, x_b]$, which we refer to as the *local region*, we have

$$u_h(x) = \sum_{l=0}^{k} u_i^{(\ell)} \phi^{\ell}(\mu^{-1}(x)) \tag{3.28}$$

where $u_i^{(\ell)}$ are the local polynomial modes on I_i resulting from a discontinuous Galerkin approximation, ϕ^{ℓ} are the polynomial basis functions of degree ℓ defined over the standard region, and $\mu(\xi)$ is the affine mapping from the standard to local region given by

$$\mu(\xi) = x_a \frac{1+\xi}{2} + x_b \frac{1+\xi}{2}. \tag{3.29}$$

To evaluate the postprocessed solution at an isolated point $x \in I_i$ by directly evaluating the integral in the convolution operator, we have

$$u^{\star}(x) = \frac{1}{h}\int_{-\infty}^{\infty} K\left(\frac{y-x}{h}\right) u_h(y)dy = \frac{1}{h}\sum_{I_{i+j}\in Supp\{K\}} \int_{I_{i+j}} K\left(\frac{y-x}{h}\right) u_h(y)dy, \tag{3.30}$$

where the second equation is due to the compact support property of the kernel. In order to evaluate the integral in Equation (3.30) exactly, we need to divide the interval I_{i+j} to subintervals over which there is no break in regularity in the integrand. We then use Gaussian quadrature with sufficient quadrature points to evaluate the integration.

Figure 3.4 shows how the integration regions are constructed from the intersection of the kernel knots and the DG element interfaces. As the figure demonstrates, in the final integration mesh (red line), each DG element is divided into two subintervals so that there

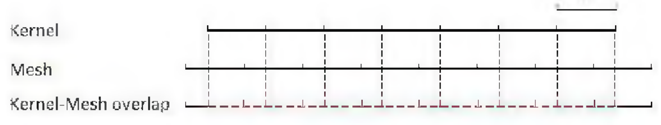


Figure 3.4. A possible kernel-mesh overlap in one dimension. Upper line represents the kernel, middle line depicts a DG mesh, and the lower line (dashed red) represents the integration mesh.

is no break in continuity. The integration will then be carried out over these subintervals, that is,

$$\begin{aligned} u^\star(x) =& \frac{1}{h}\int_{-\infty}^{\infty} K\left(\frac{y-x}{h}\right) u_h(y)dy, \\ =& \frac{1}{h}\sum_{I_{i+j}}\int_{I_{i+j}} K\left(\frac{y-x}{h}\right) u_h(y)dy, \\ =& \frac{1}{h}\sum_{I_{i+j}}\left[\int_{-1}^{1} K\left(\frac{\mu_{s_1}(\xi)-x}{h}\right) u_h(\mu_{s_1}(\xi))|J_1|d\xi + \int_{-1}^{1} K\left(\frac{\mu_{s_2}(\xi)-x}{h}\right) u_h(\mu_{s_2}(\xi))|J_2|d\xi\right], \end{aligned} \tag{3.31}$$

where s_1 and s_2 are the two aforementioned subintervals within each DG element, *i.e.*, $s_1 \cup s_2 = I_{i+j}$. In addition, $\mu_{s_1}(\xi)$ and $\mu_{s_2}(\xi)$ represent the mappings from the standard to local regions s_1 and s_2 and $|J_1|$ and $|J_2|$ are the Jacobians of these mappings. Moreover, the number of quadrature points should be enough to integrate polynomials of degree $2k$. In addition, similar to Equation (3.28), we evaluate the DG approximation in Equation (3.31) as

$$u_h(x) = \sum_{l=0}^{k} u_{I_{i+j}}^{(\ell)} \phi^\ell(\mu_{I_{i+j}}^{-1}(x)) \tag{3.32}$$

where in this case, $x = \mu_{s_{1,2}}(\xi)$ and belongs to element I_{i+j}.

We note that the support of the kernel is given by

$$\left[K_a = x + h\left(-\frac{(r+\ell)}{2}\right), K_b = x + h\left(\frac{(r+\ell)}{2}\right)\right], \tag{3.33}$$

where x is the evaluation point and $\ell = k+1$. Consequently, the position of the *kernel breaks* are given by

$$K_a + h, K_a + 2h, \cdots, K_b. \tag{3.34}$$

Algorithm 3 provides a pseudo-code for implementing the convolution operator in a one-dimensional field. In Line 5, it is stated that the subintervals are the result of the kernel and DG mesh intersection, which is a geometric problem. This will be addressed thoroughly in Chapter 6. In one dimension, this problem is fairly straightforward as it is the result of the sorted merge of the kernel breaks and the mesh element interfaces that are (partially) covered by the support of the kernel (note that kernel breaks and element interfaces are already sorted lists). For a uniform mesh, the footprint of the kernel can be found in constant computational time. For a nonuniform mesh structure, where the positions of the element interfaces in the mesh are defined by a smooth function, the footprint can be found in $O(log(N))$, N being the number of elements in one direction (or total number of elements in 1D). Moreover, for uniform meshes, h represents the uniform mesh spacing. For

Algorithm 3 1D-Convolution

```
1: for each evaluation point $x$ do
2:     $I_i$ = the element to which $x$ belongs
3:     $h$ = the size of the element $I_i$
4:     {Find the integration subintervals}
5:     $S$ = kernel and mesh intersection
6:     {The following for loop implements the third line in Equation 3.31}
7:     for each $s$ in $S$ do
8:         $intg$ += Evaluate $\int_{-1}^{1} K\left(\frac{\mu_s(\xi)-x}{h}\right) u_h(\mu_s(\xi))|J|d\xi$
9:     end for
10:    $u^\star(x) = intg/h$
11: end for
```

nonuniform meshes, we simply consider h as being the size of the element (length of element in 1D and length of element sides in 2D).

We further note that the postprocessed polynomial is of degree $2k+1$. Therefore, if we want to postprocess a DG approximation of degree k over the entire field so that a transformation to a modal representation is feasible, we need to evaluate the postprocessor at $2k+2$ collocating points per element. Moreover, in our initial DG approximation space, we have $N \times (k+1)$ degrees of freedom (N being the number of elements in the field), whereas in the postprocessed solution space, there are $N \times (2k+2) - N \times (k-1)$ degrees of freedom. The first term is due to higher order polynomials and the second term is due to what is removed (constrained) due to continuity.

3.4.1 Quadrilateral and Hexahedral Meshes

In two dimensions, the convolution kernel is a tensor product of the one-dimensional kernels

$$\begin{aligned} \bar{K}(x,y) =& \sum_{\gamma_1=0}^{r_1}\sum_{\gamma_2=0}^{r_2} c_{\gamma_1}c_{\gamma_2}\psi^{(k+1)}(x-x_{\gamma_1})\psi^{(k+1)}(y-x_{\gamma_2}) \\ =& K(x)\times K(y), \end{aligned} \tag{3.35}$$

where x_{γ_1} and x_{γ_2} are the position of the kernel nodes in the x_1- and x_2-directions and the two-dimensional coordinate system is denoted with (x_1, x_2). Furthermore, r_1, r_2 equal $2k$.

The two-dimensional convolution over quadrilateral mesh structures is therefore,

$$
\begin{aligned}
u^\star(x,y) =& \frac{1}{h_1h_2}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} K\left(\frac{x_1-x}{h_1}\right)K\left(\frac{x_2-y}{h_2}\right)u_{h_1,h_2}(x_1,x_2)dx_1dx_2,\\
=& \frac{1}{h_1h_2}\sum_{I_{i+d_1,j+d_2}\in Supp\{K\}}\int\int_{I_{i+d_1,j+d_2}} K\left(\frac{x_1-x}{h_1}\right)K\left(\frac{x_2-y}{h_2}\right)u_{h_1,h_2}(x_1,x_2)dx_1dx_2
\end{aligned}
\tag{3.36}
$$

which evaluates the postprocessed solution at $(x,y)\in I_{i,j}$. Notice that the scaling of the kernel does not require $h_1 = h_2$.

Once again to evaluate the integral in Equation (3.36) exactly to machine precision, we need to divide the integration region $I_{i+d_1,j+d_2}$ which has a quadrilateral shape to subregions over which there is no break in regularity.

Figure 3.5 demonstrates a possible kernel-mesh intersection over a quadrilateral mesh. As shown in the figure, the postprocessing kernel can be viewed as a two-dimensional patch which is the immediate result of the tensor product of the one-dimensional kernels on each direction.

For the example in Figure 3.5, we can see that it is necessary to break down the DG element to four subelements in order to evaluate the integration in Equation (3.36) exactly, *i.e.*,

$$
\begin{aligned}
&\int\int_{I_{i+d_1,j+d_2}} K\left(\frac{x_1-x}{h_1}\right)K\left(\frac{x_2-y}{h_2}\right)u_{h_1,h_2}(x_1,x_2)dx_1dx_2\\
&=\sum_{n=0}^{3}\int\int_{s_n} K\left(\frac{x_1-x}{h_1}\right)K\left(\frac{x_2-y}{h_2}\right)u_{h_1,h_2}(x_1,x_2)dx_1dx_2\\
&=\sum_{n=0}^{3}\int_{-1}^{1} K\left(\frac{\mu_{s_{n_2}}(\xi_2)-y}{h_2}\right)|J_2|\left(\int_{-1}^{1} K\left(\frac{\mu_{s_{n_1}}(\xi_1)-x}{h_1}\right)u_{h_1,h_2}(\mu_{s_{n_1}}(\xi_1),\mu_{s_{n_2}}(\xi_2))|J_1|d\xi_1\right)d\xi_2,
\end{aligned}
\tag{3.37}
$$

where s_n indicates the integration region resulting from the kernel-mesh intersection. Furthermore, the third equation is obtained using the tensor product property of the two-dimensional kernel. This allows us to write the two-dimensional integration as a product of one-dimensional integrations. $\mu_{s_{n_1}}(\xi_1)$ and $\mu_{s_{n_2}}(\xi_2)$ are used to denote the one-dimensional mappings from the standard element to the local regions, which in this case are the sides of the subelement s_n in the x_1 and x_2 directions. $|J_1|$ and $|J_2|$ are the Jacobians of these transformations. In addition, the number of quadrature points required for integration should be chosen to exactly integrate polynomials of degree $2k$ in each direction.

Another point that we would like to mention here is the evaluation of our DG approximation, $u_{h_1,h_2}(x,y)$. To evaluate the DG approximation at an arbitrary point$(x,y)\in I_{i,j}$, we have

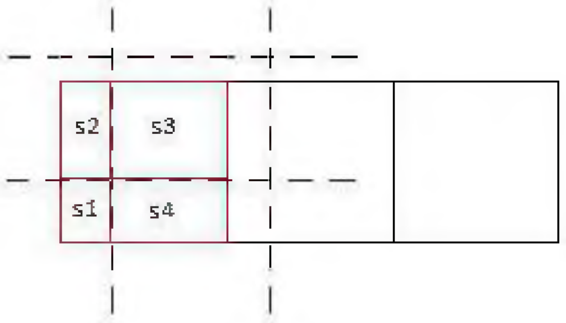


Figure 3.5. A possible kernel-mesh overlap in two dimensions. Dashed lines represent the kernel patch and solid line represent the DG mesh. Red lines depict possible integration regions over one element.

$$u_{h_1,h_2}(x,y) = \sum_{p=0}^{k}\sum_{q=0}^{k} u_{I_{i,j}}^{(pq)} \phi^{(pq)}(\xi_1,\xi_2) \tag{3.38}$$

where ξ_1 and ξ_2 are obtained using the appropriate one-dimensional inverse mappings from local to standard regions (see Section 3.4).

If we define our two-dimensional basis function $\phi^{(pq)}(\xi_1,\xi_2)$ as the tensor product of one-dimensional basis functions (see [40])

$$\phi^{pq}(\xi_1,\xi_2) = \psi_p^a(\xi_1)\psi_q^b(\xi_2), \tag{3.39}$$

with ψ_p^a and ψ_q^b being the modified or orthogonal basis functions as in [40], then we can evaluate the DG approximation in Equation (3.38) as

$$u_{h_1,h_2}(x,y) = \sum_{p=0}^{k}\psi_p^a(\xi_1)\sum_{q=0}^{k} u_{I_{i,j}}^{(pq)}\psi_q^b(\xi_2). \tag{3.40}$$

This is the so-called *sum-factorization* technique introduced in [40]. Using this approach, the number of operations needed to evaluate the DG approximation at $O(k^2)$ quadrature points formed by the tensor product of one-dimensional points reduces from $O(k^4)$ to $O(k^3)$. This aids the numerical practitioner in saving on the overall computational cost of the postprocessor.

Algorithm 4 provides a pseudo-code for the two-dimensional convolution over quadrilateral mesh structures. Lines 6 and 7 are implemented the same way as the one-dimensional case explained in the previous section. Consequently, S_1 and S_2 are integration sets that represent the one-dimensional integration regions on each direction. This means that the tensor product of these two sets produces the two-dimensional integration regions, as shown in Figure 3.5. In addition, in Lines 3 and 4, if we are dealing with a uniform mesh, $h_1 = h_2$

Algorithm 4 2D Quadrilateral Mesh Convolution

```
1:  for each evaluation point $(x, y)$ do
2:    $I_{i,j}$ = the element to which $(x, y)$ belongs
3:    $h_1$ = length of $I_{i,j}$ side in direction $x_1$
4:    $h_2$ = length of $I_{i,j}$ side in direction $x_2$
5:    {Find the 1D integration subintervals on each direction}
6:    $S_1$=kernel and mesh intersection in direction $x_1$
7:    $S_2$=kernel and mesh intersection in direction $x_2$
8:    for $s_1$ in $S_1$ do
9:      for $s_2$ in $S_2$ do
10:       $intg+=$ Evaluate the outer integral in Equation (3.37)
11:     end for
12:   end for
13:   $u^\star(x, y) = intg/(h_1 h_2)$
14: end for
```

and is equal to the uniform mesh spacing on each direction. Otherwise as we mentioned in the previous section, we choose the length of the element size on each direction for h_1 and h_2, respectively (Figure 3.5).

Similar to the two-dimensional case, the convolution kernel in three dimensions can also be formed by performing the tensor product of one-dimensional kernels. That is,

$$\begin{aligned}\hat{K}(x,y,z) =& \sum_{\gamma_1=0}^{r_1}\sum_{\gamma_2=0}^{r_2}\sum_{\gamma_3=0}^{r_3} c_{\gamma_1}c_{\gamma_2}c_{\gamma_3}\psi(x-\gamma_1)\psi(y-\gamma_2)\psi(z-\gamma_3), \\ =& K(x)\times K(y)\times K(z),\end{aligned} \tag{3.41}$$

where x_{γ_1}, x_{γ_2}, and x_{γ_3} are the position of the kernel nodes in x_1-, x_2- and x_3-directions and we have denoted the three-dimensional coordinate system with (x_1, x_2, x_3). Furthermore, $r_d,\ d = 1, 2, 3$ and equals $2k$.

From Equation (3.41), it is clear that the three-dimensional postprocessor over hexahedral meshes will be a natural extension of the two-dimensional quadrilateral postprocessor given above and therefore, we will not provide further detail for this type of the postprocessing.

3.4.2 Structured Triangular Meshes

In this section, we discuss postprocessing over structured triangular meshes. For this, we simply take the quadrilateral mesh implementation and apply the same kernel for structured triangular meshes. This means that we still use the kernel definition given in Equation (3.35) and evaluate the integral in Equation (3.36). However, now this is over a triangular region. The accuracy-enhancement capabilities of the SIAC filter over structured triangular regions will be discussed thoroughly in Chapter 5. Here, we explain the details of the implementation.

Figure 3.6 depicts a possible kernel-mesh intersection for a structured triangular mesh. As was done in the quadrilateral mesh case, we note that it is necessary to divide the element into subregions that respect both the element interfaces and kernel breaks, in order to perform the integrations exactly to machine precision. Moreover, we choose to further divide these subregions into triangles, as shown in red in Figure 3.6. Therefore, the postprocessed solution at $(x, y) \in I_{i,j}$ becomes

$$\begin{aligned} u^\star(x,y) =& \frac{1}{h_1 h_2}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} K\left(\frac{x_1 - x}{h_1}\right) K\left(\frac{x_2 - y}{h_2}\right) u_{h_1,h_2}(x_1,x_2)dx_1dx_2, \\ =& \frac{1}{h_1 h_2} \sum_{I_{i+d_1,j+d_2}\in Supp\{K\}} \left[\int\int_{U(I_{i+d_1,j+d_2})} K\left(\frac{x_1 - x}{h_1}\right) K\left(\frac{x_2 - y}{h_2}\right) u_{h_1,h_2}(x_1,x_2)dx_1dx_2 + \right. \\ & \left. \int\int_{L(I_{i+d_1,j+d_2})} K\left(\frac{x_1 - x}{h_1}\right) K\left(\frac{x_2 - y}{h_2}\right) u_{h_1,h_2}(x_1,x_2)dx_1dx_2\right]. \end{aligned} \tag{3.42}$$

In Equation (3.42), we have simply modified Equation (3.36) in Section 3.4.1 by dividing the integration over a quadrilateral element into two triangular elements $U(I_{i+d_1,j+d_2})$ and $L(I_{i+d_1,j+d_2})$. We refer to the quadrilateral element in this case as a *super-element*, to indicate the quadrilateral combination of the two diagonally aligned triangles in the DG structured mesh.

We consider one of the integrals in Equation (3.42),and explain how this integration can be evaluated exactly to machine precision. After triangulating the integration regions as shown in Figure 3.6, we arrive at

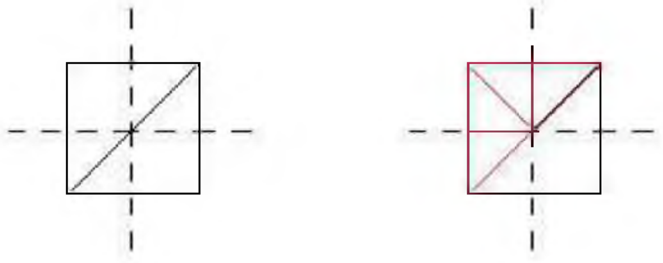


Figure 3.6. A possible kernel-mesh overlap in two dimensions over a structured triangular mesh (left). Dashed lines represent the kernel patch and solid lines represent the DG mesh. In the right image, red lines depict possible integration regions over the upper element.

$$\begin{aligned}
&\int\int_{U(I_{i+d_1,j+d_2})} K\left(\frac{x_1-x}{h_1}\right)K\left(\frac{x_2-y}{h_2}\right)u_{h_1,h_2}(x_1,x_2)dx_1dx_2\\
&=\sum_{n=0}^{3}\int\int_{\tau_n} K\left(\frac{x_1-x}{h_1}\right)K\left(\frac{x_2-y}{h_2}\right)u_{h_1,h_2}(x_1,x_2)dx_1dx_2\\
&=\int_{-1}^{1}\int_{-1}^{-\xi_2} K\left(\frac{\mu_1(\xi_1,\xi_2)-x}{h_1}\right)K\left(\frac{\mu_2(\xi_1,\xi_2)-y}{h_2}\right)u_{h_1,h_2}(\mu_1(\xi_1,\xi_2),\mu_2(\xi_1,\xi_2))|J_\xi|d\xi_1d\xi_2
\end{aligned} \tag{3.43}$$

where τ_n is the triangular subelement in $U(I_{i+d_1,j+d_2})$, μ_1 and μ_2 are the appropriate mappings from the the standard to local triangular region which are defined later in the section, and $|J_\xi| = |\frac{\partial(x_1,x_2)}{\partial(\xi_1,\xi_2)}|$.

In Equation (3.43), the third equation is derived by mapping the standard triangular element defined as

$$T_{st} = \{(\xi_1,\xi_2)| -1 \le \xi_1,\xi_2; \xi_1+\xi_2 \le 0\} \tag{3.44}$$

to the local region τ_n. We also note that in order to have the triangular expansion be as efficient as the quadrilateral one, we want to be able to define the two-dimensional basis functions used to evaluate our DG approximation in terms of a tensor product of one-dimensional basis functions. Consequently, we define a mapping from the Cartesian coordinate system to the so-called *collapsed coordinate system* such that

$$\eta_1 = 2\frac{1+\xi_1}{1-\xi_2} - 1, \qquad \eta_2 = \xi_2, \tag{3.45}$$

with the inverse transformation

$$\xi_1 = \frac{(1+\eta_1)(1-\eta_2)}{2} - 1, \qquad \xi_2 = \eta_2. \tag{3.46}$$

These new local coordinates (η_1,η_2) define the standard triangular region by

$$T_{st} = \{(\eta_1,\eta_2)| -1 \le \eta_1,\eta_2 \le 1\}. \tag{3.47}$$

The transformation in Equation (3.45) can be interpreted as a mapping from the triangular region to a rectangular one, as seen in Figure 3.7. This transformation ia also known as the *Duffy transformation* (see [27]). Furthermore, for more information regarding tensorial basis functions and the collapsed coordinate system, we refer the interested reader to [40].

Using the collapsed coordinate system, the integral in Equation (3.43) now becomes

$$\begin{aligned}
&\int_{-1}^{1}\int_{-1}^{-\xi_2} K\left(\frac{\mu_1(\xi_1,\xi_2)-x}{h_1}\right)K\left(\frac{\mu_2(\xi_1,\xi_2)-y}{h_2}\right)u_{h_1,h_2}(\mu_1(\xi_1,\xi_2),\mu_2(\xi_1,\xi_2))|J_\xi|d\xi_1d\xi_2\\
&=\int_{-1}^{1}\int_{-1}^{1} K\left(\frac{\mu_1^e(\eta_1,\eta_2)-x}{h_1}\right)K\left(\frac{\mu_2^e(\eta_1,\eta_2)-y}{h_2}\right)u_{h_1,h_2}(\mu_1^e(\eta_1,\eta_2),\mu_2^e(\eta_1,\eta_2))|J_\xi||J_\eta|d\eta_1d\eta_2
\end{aligned} \tag{3.48}$$

where $|J_\eta| = |\frac{\partial(\xi_1,\xi_2)}{\partial(\eta_1,\eta_2)}|$.

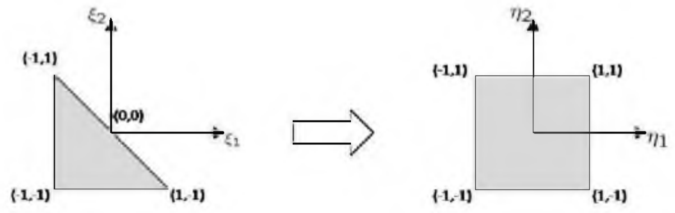


Figure 3.7. Triangle to rectangle transformation.

Equation (3.48) results in the value of the two-dimensional integral over the triangular region τ_n. We note that since the kernel is a function of both variables in the standard as well as the collapsed coordinate systems, we can not separate the two-dimensional integration in terms of one-dimensional integrations as we did in Equation (3.37). Consequently, the number of quadrature points required for integration should be enough to integrate polynomials of degree $3k$ exactly. Furthermore, if we denote the vertices of τ_n as x_i^A, x_i^B and x_i^C then we have

$$x_i = \mu_i(\xi_1, \xi_2) = x_i^A \frac{-\xi_2 - \xi_1}{2} + x_i^B \frac{1+\xi_1}{2} + x_i^C \frac{1+\xi_2}{2}, \quad i = 1, 2. \tag{3.49}$$

Substituting ξ_1 and ξ_2 using Equation (3.46), we arrive at

$$x_i = \mu_i^e(\eta_1, \eta_2) = x_i^A \frac{1-\eta_1}{2}\frac{1-\eta_2}{2} + x_i^B \frac{1+\eta_1}{2}\frac{1-\eta_2}{2} + x_i^C \frac{1+\eta_2}{2}, \quad i = 1, 2. \tag{3.50}$$

We can also use a similar procedure to Equation (3.38) to evaluate the DG approximation at $(x, y) \in U(I_{i,j})$. That is,

$$u_{h_1,h_2}(x, y) = \sum_{p=0}^{k} \sum_{q=0}^{k-p} u_{U(I_{i,j})}^{pq} \phi^{pq}(\xi_1, \xi_2), \tag{3.51}$$

with the difference that the basis functions are given by

$$\phi^{pq}(\xi_1, \xi_2) = \psi_p^a(\eta_1)\psi_{pq}^b(\eta_2), \tag{3.52}$$

with ψ_p^a and ψ_{pq}^b being the orthogonal or modified basis functions for triangular elements defined in [40]. In Equation (3.51), $U(I_{i,j})$ represents the DG triangular element that contains (x, y) and η_1 and η_2 are obtained by first applying an inverse mapping that maps $U(I_{i,j})$ to the standard triangular region given in Equation (7.4), and then using Equation (3.45). The use of the sum-factorization technique mentioned in the previous section is not beneficial here, since the quadrature points obtained as a result of several mappings do not necessarily follow a tensor-product form.

Algorithm 5 provides a pseudo-code for implementing the convolution operator over structured triangular meshes. As we mentioned earlier, for implementation purposes, we

Algorithm 5 2D Triangular Mesh Convolution

```
1: for each evaluation point $(x, y)$ do
2:   $I_{i,j}$ = the super-element to which $(x, y)$ belongs
3:   $h_1$ = size of $I_{i,j}$ in direction $x_1$
4:   $h_2$ = size of $I_{i,j}$ in direction $x_2$
5:   {This simply gives the super-elements (partially) covered by the 2D kernel}
6:   $kFootPrint$= the footprint of the 2D kernel on the DG mesh
7:   for each super-element $I$ in $kFootPrint$ do
8:     {lower triangle}
9:     $L(I)$=lower triangle
10:    $intgRegions$ = intersection of $L(I)$ with each square in the 2D kernel patch
11:    for each triangle $\tau$ in $intgRegions$ do
12:      intg += Result of the integral in Equation (3.48)
13:    end for
14:    {upper triangle}
15:    $U(I)$=upper triangle
16:    $intgRegions$ = intersection of $U(I)$ with each square in the 2D kernel patch
17:    for each triangle $\tau$ in $intgRegions$ do
18:      intg += Result of the integral in Equation (3.48)
19:    end for
20:  end for
21:  $u^\star(x, y) = intg/(h_1 h_2)$
22: end for
```

consider the kernel in two dimensions as a patch or a two-dimensional matrix of squares. Therefore, to find the intersection region of a triangle with the kernel, we simply find the intersection of the triangle with these squares. For structured triangulations, we can identify all the possible cases for kernel-mesh intersection, as shown in Figure 3.8. Moreover, similar ideas discussed in the previous section apply to the scaling parameters h_1 and h_2 for the nonuniform mesh structure. For general unstructured grids, these parameters need to be modified properly to gain optimal error convergence. However, the general implementation scheme for unstructured triangular grids will be similar to that of the structured ones. Once we identify the elements covered by the kernel support, we solve a series of geometric intersections (similar to Figure 3.6). As the unstructured mesh will be more complex, we are likely to get more integration regions comparing to the structured mesh. Investigation of the SIAC filter for unstructured triangular meshes will be discussed in Chapter 7.

We further note that in this document, when choosing a distribution of points for integration, we prefer the *Lobatto*-type quadrature. Particularly, for triangular regions, we choose *Gauss-Lobatto-Legendre* (GLL) points on the x_1-direction and *Gauss-Radau-Legendre* (GRL) on the x_2-direction. GRL points absorb the Jacobian of the collapsed coordinate transformation and do not include the singularity at the collapsed vertex. For

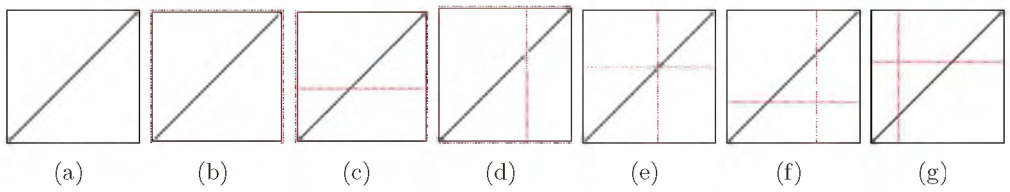


Figure 3.8. One super-element of the DG mesh (black) with possible kernel breaks (red). The number of integration regions increases with triangular meshes, with possibility of two to seven different regions.

more information on these types of quadrature points, we refer the reader to [40].

3.4.3 Note on the Computational Complexity

In this section, we discuss the complexity of the computational cost for postprocessing.

As mentioned in Section 3.4, finding the intersection of the kernel mesh with the DG mesh is a geometric intersection problem that in general can be quite complex in two and three dimensions. The complexity increases due to the elements being further tessellated into subelements over which numerical integrations are performed. In our analysis thus far, we have only considered uniform triangular meshes where at most one kernel break per direction lies within a super-element. However, in the case of totally unstructured triangular meshes, the number of breaks can be up to several breaks within an element, which will result in more integration regions and therefore more numerical quadratures. Here, we consider the cost of the convolution operator, assuming that the integration subintervals have already been found.

The number of elements that are covered by the kernel support is dependent on the extent (width) of the convolution kernel, which is a function of the polynomial order per element. Therefore, if we denote the polynomial order as k and assume that all elements have the same polynomial order in both directions, the number of elements that need to be considered for every evaluation point will be $O(k^2)$ in two dimensions. Furthermore, for each of these elements, depending on the number of integration regions within each element, a series of numerical quadratures must be performed. After transforming to the collapsed coordinate system, we evaluate integrals as shown in Equation (3.48). Gaussian quadrature in two dimensions will be performed in $O(k^2)$ operations. However, we need to evaluate the kernel as well as the DG approximation at each of the quadrature points used in the integration. The DG approximation can be calculated at $O(k^2)$ quadrature points in $O(k^3)$ floating point operations using the so-called sum-factorization technique when possible, and in $O(k^4)$ operations otherwise. Kernel evaluation can also be performed in $O(k^3)$ operations in the case of quadrilateral elements or in $Q(k^4)$ for triangular elements. Hence, the overall cost of performing one numerical integration will be $O(k^4)$. Consequently, from Equation (3.43), the cost of numerical quadrature on a single triangular element will be $O(Mk^4)$, with M being the number of integration regions within the element. When performing the exact postprocessor scheme, $M \geq 1$ and is at most seven per triangular element in the case of a uniform mesh. However, this upper bound increases in the case of totally unstructured grids and will play a significant role in the overall performance of the algorithm, especially when postprocessing the entire DG field such that a transform to the modal representation is feasible. In that case, the total computational cost is $O(MNk^8)$, with N being the total number of elements in the field.

3.5 Performance Analysis of the Postprocessor

In this section, we provide performance results for postprocessing DG fields using the implementation strategies in Section 3.4. The postprocessor is a good candidate for parallelization because when filtering an entire computational field, evaluating the postprocessed value at one quadrature point is independent of the other. Therefore, having access to a multiprocessor machine, we can have separate threads evaluate the postprocessed value at different points without any communications among them. Using *OpenMP*, only a few compiler directives are required to parallelize the execution of the postprocessor and gain proper scaling in the performance on a multiprocessor shared-memory machine.

We note that although we are only considering the performance of the SIAC filter, we provide the performance when applied to a discontinuous Galerkin solution. For our results, we consider the traditional second-order wave equation,

$$\eta_{tt} - \eta_{xx} - \eta_{yy} = 0, \qquad (x, y) \in (0,1) \times (0,1), \qquad T = 6.28. \tag{3.53}$$

We rewrite Equation (3.53) as a system of first-order linear equations,

$$\begin{aligned} \eta_t + u_x + v_y &= 0 \\ u_t + \eta_x &= 0 \\ v_t + \eta_y &= 0, \end{aligned} \tag{3.54}$$

with initial conditions

$$\begin{aligned} \eta(x, y, 0) &= 0.01 \times (\sin(2\pi x) + \sin(2\pi y)) \\ u(x, y, 0) &= 0.01 \times (\sin(2\pi x)) \\ v(x, y, 0) &= 0.01 \times (\sin(2\pi y)), \end{aligned} \tag{3.55}$$

and 2π periodic boundary conditions in both directions. We apply the postprocessor to the solutions of this DG problem for the η variable, after one period in time over triangular mesh structures. The numerical behavior of the SIAC filter for Equation (3.53) is examined in Chapter 5. Here, we provide a thorough performance analysis of the parallelization.

Algorithm 6 depicts the condensed version of Algorithm 5 presented in Section 3.4.2. The OpenMP directives in lines 1 and 3 are used to parallelize the execution of the postprocessor. As displayed in Algorithm 6, there exist three principle nested for loops in the code, and we choose to parallelize the outer most one to minimize the overhead due to initiation of OpenMP directives.

The performance results for postprocessing an entire triangular DG field provided in this section consider six evaluation points per element. Results are provided for both the uniform and the smoothly varying triangular meshes shown in Figure 3.9. Moreover, we provide a performance comparison between the filtering approaches using exact and inexact integration. Note that the timing results have been gathered on a SGI multiprocessor machine with 2.67 MHz CPUs, using up to 16 threads.

The timing results for postprocessing are given in Table 3.1 (uniform mesh) and Table 3.2 (smoothly varying mesh). We note that the workload for the uniform mesh is statically assigned to each thread, as each thread performs an equal amount of work. However, for the smoothly varying mesh, we have used dynamic scheduling to simulate an equal workload for each thread. Figures 3.10 and 3.11 demonstrate the performance scaling plots for the uniform and smoothly varying meshes, respectively. The scaling results have been calculated from the following,

$$scaling = \frac{T_{serial}}{T_{parallel}} \tag{3.56}$$

where T_{serial} represents the serial execution time and $T_{parallel}$ is the parallel execution time. Ideally, this parameter should result in the number of threads used in the parallel execution. We see that as we increase the order of the polynomial, the scaling approaches the theoretically desired.

3.6 Summary and Conclusions

This chapter presents the explicit steps a numerical practitioner should use in order to implement the Smoothness-Increasing Accuracy-Conserving (SIAC) filters in an efficient manner. We consider quadrilateral and triangular element shapes in two dimensions and hexahedra in three dimensions for both uniform and smoothly varying mesh structures. We

Algorithm 6 Parallel-2D-Tri-Postprocessor

```
♯ pragma omp parallel
{
♯ pragma omp for schedule(static/dynamic)
for each evaluation point $(x, y)$ do
    for each super-element $I$ in $kFootPrint$ do
        {lower triangle}
        for each triangle in $intgRegions$ do
            intg += Result of the integral in Equation 3.48
        end for
        {upper triangle}
        for each triangle in $intgRegions$ do
            intg += Result of the integral in Equation 3.48
        end for
    end for
    {Lines 5-14 will be repeated here if convex combination is needed.}
    $u^\star(x, y) = intg/(h_1 h_2)$
end for
}
```

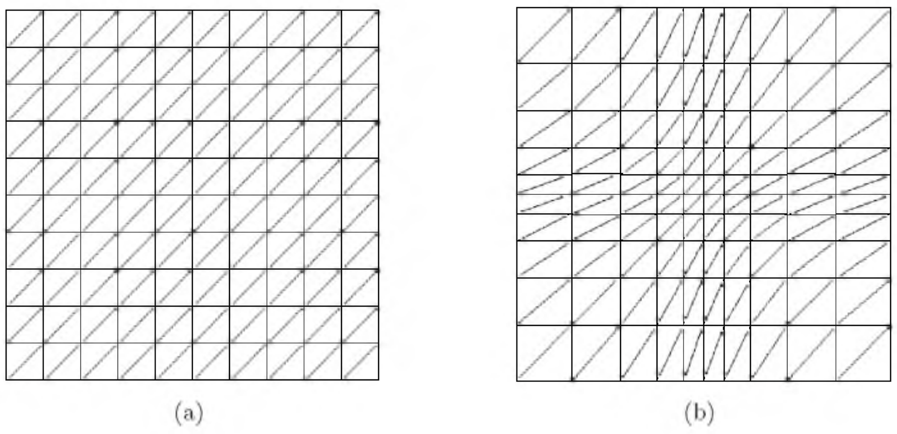


Figure 3.9. Examples of the (a) uniform and the (b) smoothly varying triangular meshes used in calculations.

Table 3.1. Timing results in seconds for postprocessing over the entire domain for the uniform triangular mesh considering $\mathbb{P}^2$, $\mathbb{P}^3$, and $\mathbb{P}^4$ polynomials. th represents the number of threads used in the parallel execution.

$\mathbb{P}^2$					
mesh	$th = 1$	$th = 2$	$th = 4$	$th = 8$	$th = 16$
$20^2 \times 2$	8.68	4.39	2.19	1.13	0.63
$40^2 \times 2$	34.74	17.70	8.85	4.48	2.47
$80^2 \times 2$	137.85	68.92	34.61	17.68	9.53
$\mathbb{P}^3$					
$20^2 \times 2$	39.76	19.98	10.00	5.02	2.64
$40^2 \times 2$	159.68	79.68	40.04	20.22	10.47
$80^2 \times 2$	632.34	316.43	158.77	80.45	40.39
$\mathbb{P}^4$					
$20^2 \times 2$	154.76	77.05	38.69	19.35	9.81
$40^2 \times 2$	617.32	310.56	155.65	78.27	39.43
$80^2 \times 2$	2455.01	1238.38	622.12	311.82	157.08

Table 3.2. Timing results in seconds for postprocessing over the entire domain for the smoothly varying triangular mesh considering $\mathbb{P}^2$, $\mathbb{P}^3$ and $\mathbb{P}^4$ polynomials. th represents the number of threads used in the parallel execution.

$\mathbb{P}^2$					
mesh	$th = 1$	$th = 2$	$th = 4$	$th = 8$	$th = 16$
$20^2 \times 2$	13.74	6.95	3.46	1.78	0.94
$40^2 \times 2$	52.77	26.51	13.28	6.80	3.51
$80^2 \times 2$	208.10	104.59	52.40	26.84	13.92
$\mathbb{P}^3$					
$20^2 \times 2$	59.39	29.77	14.90	7.59	3.84
$40^2 \times 2$	223.59	112.04	56.14	28.57	14.64
$80^2 \times 2$	853.65	426.84	213.43	106.72	53.37
$\mathbb{P}^4$					
$20^2 \times 2$	202.85	101.81	51.06	25.72	12.98
$40^2 \times 2$	765.83	384.75	192.09	96.83	48.71
$80^2 \times 2$	2960.00	1483.83	742.29	372.28	188.58

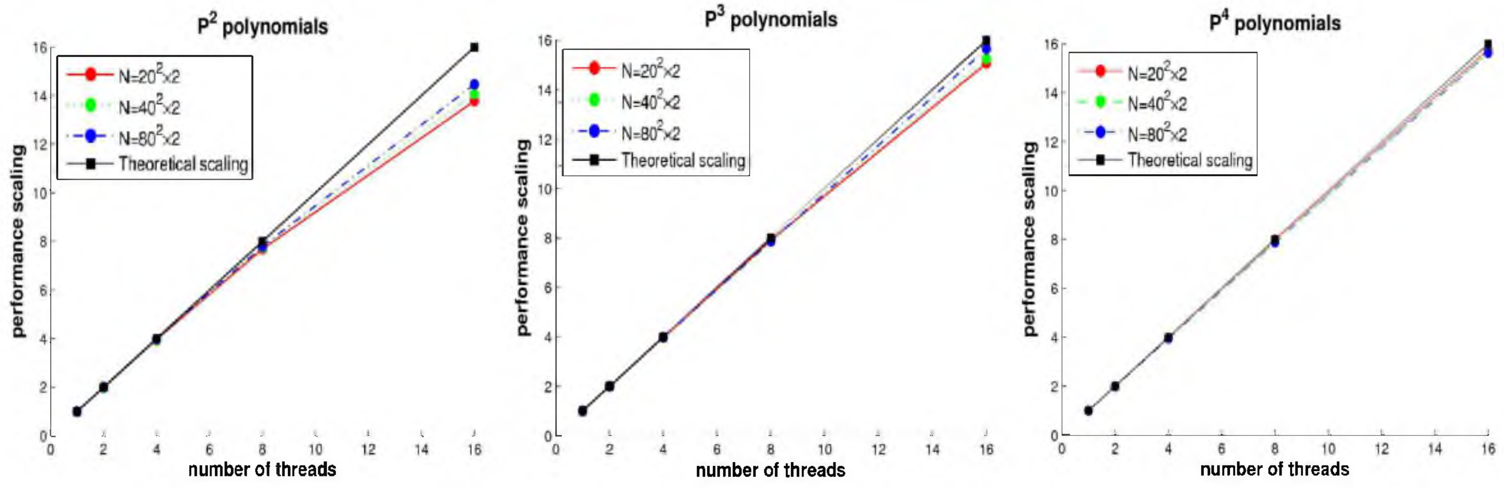


Figure 3.10. Postprocessor performance scaling for the uniform triangular mesh. N represents the number of elements in the field.

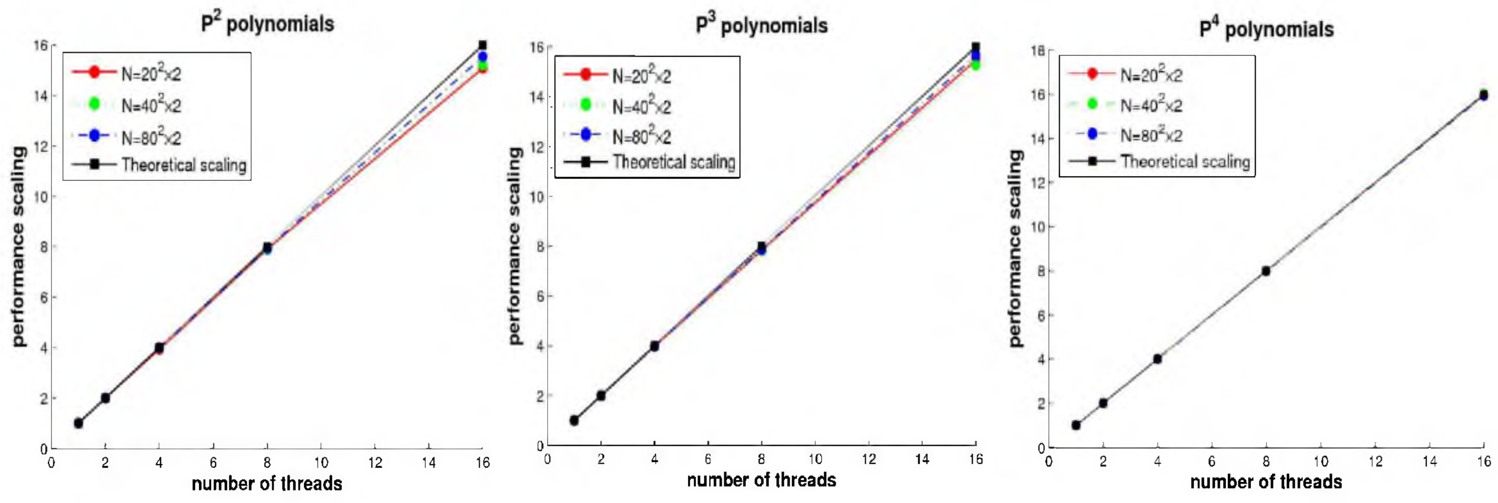


Figure 3.11. Postprocessor performance scaling for the smoothly varying triangular mesh. N represents the number of elements in the field.

address the computational tasks performed when postprocessing discontinuous Galerkin (DG)fields. As the conventional way of postprocessing through the use of matrix-vector multiplications has limited applicability, we provide a more general scheme to calculate the postprocessed value, by directly evaluating the convolution operator of the postprocessor. We demonstrate that when evaluating the convolution operator exactly to machine precision, we need to respect the breaks in continuity over the integration regions which are due to the element interfaces and kernel breaks. Consequently, the number of numerical quadratures can increase significantly when dealing with general mesh structures. In an attempt to overcome the cost of several numerical quadratures, we provide results for the first time that demonstrate the efficiency of the postprocessor when parallelized on a shared-memory multiprocessor machine.

CHAPTER 4

QUADRATURE APPROXIMATIONS FOR EVALUATING THE CONVOLUTION OPERATOR IN THE SIAC FILTERS

As discussed in the previous chapters, the basic operation performed to gain the smoothness and accuracy benefits is convolution of the DG solution against a judiciously constructed B-spline-based kernel. The goal of this chapter is to ascertain and quantify the impact of quadrature errors within this convolution process. All of the mathematical proofs concerning accuracy and smoothness assume exact integration. All the empirical numerical examples both in the mathematical literature [19, 61] and the engineering literature [67, 75] employed consistent integration with Gaussian quadrature to guarantee that the numerical errors within the convolution operator could be driven below machine precision. In this chapter, we seek to quantify the impact of inexact quadrature on the filtering process and to assess whether it greatly impacts its use as an intermediary stage between simulation and visualization in the scientific pipeline.

Here, we examine a collection of common scenarios that might arise when one seeks to implement the aforementioned postprocessing algorithms in the engineering context. We focus on one-dimensional and two-dimensional quadrilateral implementations and use as our gold-standard the solving of the convolution operation with consistent integration (integration that partitions the domain so as to respect all breaks in regularity) combined with Gaussian integration that integrates the kernel times the DG-based polynomial exactly to double-precision machine zero. We first examine the case when consistent integration with inexact Gaussian quadrature (under-integration) is used. Because consistent integration requires solving the geometric problem of finding all the places in which regularity is decreased and generating a super-mesh based on these data, we consider what happens when only the original DG mesh is used as the underlying support mesh for integration. Under this scenario, we examine the use of Gaussian quadrature and of midpoint quadrature. This choice highlights the difference between polynomial-based high-order and adaptive low-order quadrature implementations. We emphasis that this study is primarily for engineering circumstances when the trade-offs between time, resources, and accuracy are important. Although the case against committing such numerical crimes is well-known, the repercussions have not been well documented for the use of this filter as a visualisation tool. It is this specific crime that we wish to address.

The chapter is organized as follows. In Section 4.1, we present the different implementation strategies one might employ, in particular: (1) the consistent integration approach with exact and inexact Gaussian quadrature, (2) the input mesh-based Gaussian quadrature approach, and (3) the input mesh-based midpoint quadrature approach. In Section 4.2.2, we present analysis which provides theoretical estimates which bound the numerical crimes committed when using the three aforementioned approaches. In Section 4.3, we present an empirical study which corroborates the error estimates we have derived. In Section 4.4, we summarize our results and provide guidelines based upon our study concerning under which circumstances one technique should be used versus another. We further add that the result of these contributions has been published in [48].

4.1 Numerical Quadrature Approaches

In this section, we present the different implementation strategies used to calculate the convolution operator.

4.1.1 Gaussian Quadrature Approaches

We remind the reader that the postprocessor is simply the discontinuous Galerkin solution convolved against a linear combination of B-splines at the final simulation time. That is, in one dimension,

$$u^{\star}(x) = \frac{1}{h}\int_{-\infty}^{\infty} K^{2k+1,k+1}\left(\frac{y-x}{h}\right) u_h(y)dy, \tag{4.1}$$

where $u^{\star}$ is the postprocessed solution, h is the mesh characteristic length, and u_h the DG solution of degree k.

The postprocessed solution, $u^{\star}(x)$, which is a piecewise polynomial of degree $2k+1$, can be evaluated exactly. As we mentioned in Chapter 3, $u_h(y) = \sum_{\ell=0}^{k} u_i{}^{(\ell)}\phi_i^{(\ell)}(y)$, and $\phi_i^{(\ell)}$ are the basis functions of the projected function on cell I_i. Therefore, for $x \in I_i$, we have

$$u^{\star}(x) = \frac{1}{h}\int_{-\infty}^{\infty} K\left(\frac{y-x}{h}\right) u_h(y)dy = \frac{1}{h}\sum_{I_{i+j}\in Supp\{K\}} \int_{I_{i+j}} K\left(\frac{y-x}{h}\right) u_h(y)dy. \tag{4.2}$$

The kernel in the expression above consists of a linear combination of B-splines. Therefore, in order to calculate the above integral exactly, we need to decompose the interval I_{i+j} into subintervals that respect the kernel knots (which we refer to as *breaks*) (see also Section 3.4); the resulting integral is calculated as the summation of the integrals over each subinterval. The term *consistent integration* is used to denote integration that respects these breaks by finding the necessary subintervals such that the integral on each subinterval can be done exactly to machine precision. To compute the postprocessed solution for $x \in I_i$, the algorithm is as follows:
For $j = -k', \cdots, k'$:

- Find the kernel breaks (if any) that lie in the interval I_{i+j}. Use the kernel breaks to identify the subintervals. Note that in the general case, the number of breaks can be zero up to several breaks within an element. In the case of uniform meshes, one can show that there is at most one break per input mesh element.

- Evaluate the integral over each of the subintervals using Gaussian quadrature. For the case of an exact quadrature, we are required to evaluate the integrand at $k + 1$ Gauss points where k is the approximation degree of the DG solution. We have used k Gauss points (one less than the required) for the inexact quadrature experiments presented in the results section.

- Sum the resulting values from each subinterval to gain the overall value of the integral on element I_{i+j}.

This general algorithm can be used in one of several ways which we will mention here. First, this algorithm holds for any $x \in I_i$, and hence can be used for isolated postprocessing of the solution at some arbitrary point. The only additional cost not previously mentioned is the search time needed to find the element I_i containing the point x of interest. Building upon this usage, the second strategy is to postprocess an entire element (*i.e.,* find the postprocessed polynomial of degree $2k+1$ on an element) by repeating the above procedure for a collection of collocation points or at quadrature points so that a transform to a modal representation can be done. The third usage, which is often implemented for uniform meshes, is to rewrite the above equation using small matrix multiplications,

$$u^*(x) = \sum_{j=-k'}^{k'} \sum_{l=0}^{k} u_{i+j}^{(l)} C_{j,l,k}(x) \tag{4.3}$$

where $C_{j,l,k}(x)$ is a polynomial of degree $2k + 1$ and $u_{i+j}^{(l)}$ are the coefficients in the discontinuous Galerkin approximation.

The purpose of this chapter is to quantify the numerical crimes committed when using different quadrature schemes. Therefore, it is useful to understand the dominant costs in the above algorithm so that one can appreciate why different engineering implementation choices might be made. The possibly dominating cost in the above algorithm is performing the numerical quadrature on the consistent integration mesh. When filtering based upon the *input mesh*, we disregard the position of the kernel breaks and evaluate the integrand over the entire element. That is, we skip the first step in the consistent integration approach (and hence its associated cost); in the second step, there is only one interval which is the entire element. We then proceed by using $Q \geq k+1$ Gauss points for computing the Gaussian quadrature. Considering the computational cost, as we increase the number of quadrature points, Q, filtering based upon the input mesh will use more floating point operations for $Q > 2k+2$ to calculate the integral over I_{i+j} compared to the consistent integration approach; however, the algorithmic scaling is still $O(M)$ where M is a number related to the extent of the local filter.

Two further notes are worth mentioning before proceeding. The first is that the algorithm above extends easily to the case of two-dimensional and three-dimensional postprocessing as the convolution kernel is merely a tensor-product of the one-dimensional kernels. Secondly, in the case of a nonuniform mesh, since the kernel is no longer translation invariant, the postprocessing coefficients need to be recomputed for each element as is mentioned in [22]. In order to avoid this recomputation, Curtis *et al.* proposed two strategies for postprocessing over nonuniform meshes: one based upon the local $L^2 - projection$ of the solution to a uniform "scratch-pad" mesh and one based upon the characteristic length. The algorithmic scaling for both algorithms is $O(N)$, with N being the size of the domain.

4.1.2 Midpoint Quadrature Approach

In this section, we examine an alternative strategy to using Gaussian quadrature for the approximation of the convolution integral. For a given $x \in I_i$ we try to approximate the integral

$$u^*(x) = \frac{1}{h}\int_{-\infty}^{\infty} K\left(\frac{y-x}{h}\right) u_h(y)dy \tag{4.4}$$

using midpoint integration. For a complete overview of the derivation and implementation of the midpoint rule, we refer the reader to [39]. In this case, we evaluate the postprocessed solution $u^*(x)$ using the midpoint rule to compute the integral in Equation (4.4) over the entire kernel support at once, *i.e.*, we are not following the element-by-element approach mentioned in the previous section. In other words, we proceed as follows:

- For $x \in I_i$, determine where the limits of the kernel lie on the DG mesh. That specifies the integration area, which we denote by $[x_{left}, x_{right}]$.
- Set the level of the midpoint integration. Assuming Δx is the size of each of the n equal cells involved in the discretization, we have $\Delta x = (x_{right} - x_{left})/2^{level}$ with 2^{level} being the number of evaluation points used in the midpoint rule. Typically, the support of the kernel centered around zero will be the case where $x_{left} = -\frac{3k+1}{2}$ and $x_{right} = \frac{3k+1}{2}$. This gives $\Delta x = \frac{3k+1}{2^{level}}$.
- Perform the midpoint integration,

$$\begin{aligned} u^*(x) &= \frac{1}{h}\int_{-\infty}^{\infty} K\left(\frac{y-x}{h}\right) u_h(y)dy \\ &= \frac{1}{h}\int_{x_{left}}^{x_{right}} K\left(\frac{y-x}{h}\right) u_h(y)dy \\ &= \frac{1}{h}\left(\Delta x \sum_{i=1}^{n} K_{i-1/2} u_{h,i-1/2}\right). \end{aligned} \tag{4.5}$$

Again, we follow a similar process in case of a 2D postprocessor along each direction. As it is understood from the aforementioned steps, we both disregard the kernel breaks and the element interfaces in the input mesh for this quadrature approach.

4.2 Quadrature Approximations of the Convolution Operator

In this section, we analyze the crimes committed when performing a nonconsistent and/or inexact quadrature. Although we discuss the simplified case of a uniform one-dimensional mesh, many of the concepts extend to postprocessing over nonuniform meshes. We begin by discussing the ideal case of an exact, consistent quadrature, then move on to discuss an inexact quadrature on a consistent integration mesh. We secondly discuss implementing an inconsistent quadrature which takes only the DG mesh into account and lastly, the midpoint quadrature on the DG mesh.

4.2.1 Gaussian Quadrature on a Consistent Integration Mesh

4.2.1.1 Exact, Consistent Gaussian Quadrature

Gaussian quadrature is well-known to integrate polynomials of degree $2m-1$ exactly by using m points, $x_1, x_0, \cdots, x_m$, and m weights $w_1, w_2, \cdots, w_m$ [21]. The formula for the quadrature is given by

$$\int_a^b f(x)\,dx = \sum_{j=1}^{m} f(y_j)w_j, \tag{4.6}$$

where $y_j = \frac{b-a}{2}x_j + \frac{b+a}{2}$, and w_j is the associated weight function. This implies that for our convolution, which consists of integrals of polynomial degree at most $2k$, we must employ $k+1$ points and weights in our quadrature in order to be exact to machine precision.

The main point of this discussion is to emphasize that in order to postprocess one element in a uniform mesh construction that contains a discontinuous Galerkin approximation of degree k, we have a support size of $2k'+1$ elements for the postprocessor, where $k' = \lceil \frac{3k+1}{2} \rceil$. There are two integral evaluations per element. Therefore, we are performing $4k' + 2$ Gaussian quadrature evaluations of degree $2k+1$. Although Schumaker gives a simplified formula for integrating B-splines against a polynomial [64], we should point out that we are convolving the B-splines against a piecewise polynomial.

To begin, let us examine the kernel performed using exact integration over a uniform mesh. In this case, to postprocess element i, we have

$$u^\star(x) = \sum_{j=-k'}^{k'} \sum_{\ell=0}^{k} u_{i+j} \sum_{\gamma=-k}^{k} c_\gamma \frac{1}{h} \int_{I_{i+j}} \psi^{(k+1)}\left(\frac{y-x}{h} - \gamma\right) \phi^{(\ell)}\left(\frac{y - x_{i+j}}{h}\right) dy. \quad (4.7)$$

Setting $\eta = \frac{y-x}{h}$ and $\xi_i = \frac{y-x_i}{h}$, this becomes

$$u^\star(x) = \sum_{j=-k'}^{k'} \sum_{\ell=0}^{k} u_{i+j} \sum_{\gamma=-k}^{k} c_\gamma \int_{-\xi_i+j-1/2}^{-\xi_i+j+1/2} \psi^{(k+1)}(\eta-\gamma)\, \phi^{(\ell)}(\xi_i+\eta-j)\, d\eta. \quad (4.8)$$

On each element, I_{i+j}, define the function in our integral to be

$$f(\eta, \xi_i) = \sum_{\ell=0}^{k} \psi^{(k+1)}(\eta-\gamma)\, \phi^{(\ell)}(\xi_i+\eta-j). \quad (4.9)$$

For purposes of error analysis, we note that this function is $\mathcal{C}^{k-1}$ over each DG element and discontinuous at element boundaries. This stems from the properties of the B-splines and the basis of our DG approximation and is easily seen by examining the case of piecewise monomials. In this case, $\phi^{(\ell)}(\xi_i + \eta - j) = (\xi_i + \eta - j)^\ell$. If we consider the boundary where $\eta \to (-\xi_i - 1/2)$, then we have, from the left of the elemental boundary:

$$\lim_{\eta \to -(\xi_i+1/2)^-} f(\eta, \xi_i) = \sum_{\ell=0}^{k} \psi^{(k+1)}(-\xi_i - 1/2 - \gamma) \left(-\frac{3}{2}\right)^\ell,$$

since we are approaching from the element I_{i-1}. However, approaching the limit from the right, we would have

$$\lim_{\eta \to -(\xi_i+1/2)^+} f(\eta, \xi_i) = \sum_{\ell=0}^{k} \psi^{(k+1)}(-\xi_i - 1/2 - \gamma) \left(-\frac{1}{2}\right)^\ell,$$

as we are approaching from element I_i. The limits of these two functions are continuous for piecewise constants, but otherwise they are discontinuous.

In order to analyze the error for the case of consistent integration with reduced quadrature and quadrature based on the input DG mesh, it is useful to review the existing literature. Here, we follow the work of de Boor [21] for $w(x) = 1$. That is, if we were to use exact quadrature over a consistent integration mesh using $k+1$ Gauss points per integral, the error is given by

$$\int_a^b f(\eta, \xi_i)\, d\eta - \int_a^b p_k(\eta)\, d\eta = \int_a^b f[x_0, \cdots, x_{2k+1}] g_{2k+1}(x)\, dx \tag{4.10}$$

where p_k is some polynomial that interpolates $f(\eta, \xi_i)$ at $k+1$ points and

$$g_{2k+1}(x) = [(x - x_0) \cdots (x - x_k)]^2$$

If $f(\eta, \xi_i) \in \mathcal{C}^{2k+2}$, then we have the error estimate

$$\int_a^b f(\eta, \xi_i)\, d\eta - \int_a^b p_k(\eta)\, d\eta = C(\xi_i, y_0, \cdots, y_{k+1}) f^{(2k+2)}(z, \xi_i), \quad z \in (a, b). \tag{4.11}$$

Note that in our case, $f(\eta, \xi_i)$ is a polynomial of degree $2k$, and the integral is exact.

4.2.1.2 Inexact, Consistent Gaussian Quadrature

Now that we have discussed the necessary components for exact, consistent quadrature, let us examine the case where we simply use fewer Gauss points. In this case, we are respecting both the kernel breaks and the DG elemental boundaries and use only k Gauss points. From the error formula above (Equation 4.11), we have the following error estimate for one integral

$$\int_a^b f(\eta, \xi_i)\, d\eta - \int_a^b p_k(\eta)\, d\eta = C(\xi_i, y_0, \cdots, y_{k+1}) f^{(2k)}(z, \xi_i) = \mathsf{C}. \tag{4.12}$$

The constant in the error is not necessarily less than one, but is certainly bounded due to the smoothness of $f(\eta, \xi_i)$. We also note that the constant depends upon k.

4.2.2 Gaussian Quadrature on the DG Mesh

Next, we examine the case were we are using exact quadrature over a discontinuous Galerkin mesh. That is, we are ignoring the knots of the B-splines and disregarding the level of smoothness. For this case, we simply note that it is only possible to use the error estimate given in Equation (4.10) since the function on a given element is only $\mathcal{C}^{k-1}$. That is, the error is given by

$$\int_a^b f(\eta, \xi_i)\, d\eta - \int_a^b p_k(\eta)\, d\eta = \int_a^b f[x_0, \cdots, x_{2k-1}] g_{2k+1}(x)\, dx. \tag{4.13}$$

4.2.3 Midpoint Quadrature on the DG Mesh

Lastly, we consider the problem of using midpoint integration. In this case, our formula is given by

$$\int_a^b f(x)\,dx = (b-a)f(c) \tag{4.14}$$

where $c = (a+b)/2$ and f is as given in Equation 4.9. It is well known that the midpoint error is given by

$$Error(midpoint) = \frac{1}{24}f''(z)(b-a)^3, \quad z \in (a,b)$$

[21]. In our case, we are carrying out $m = level$ midpoint quadratures. Therefore, we have for the error

$$\sum_{j=0}^{m-1} \frac{1}{24}f''(z)\triangle x^3 \le \mathsf{C}\,\triangle x^3 \tag{4.15}$$

where $\triangle x = \frac{3k+1}{2^m}$ since $x_{left} = -\frac{3k+1}{2}$, $x_{right} = \frac{3k+1}{2}$, and $\triangle x = (x_{right} - x_{left})/2^m$. We should note two things: first, that the constant that bounds $f''(z)/24$ depends upon the polynomial order, secondly, that this estimate of the error does not improve for higher polynomial orders. Therefore, it does not matter whether we increase the polynomial order of our B-spline or our DG solution. The only time where implementing the midpoint quadrature may be effective is for piecewise linear polynomial approximations.

4.3 Results

In this section, we present the results for the various choices of quadrature to see the computational effect of the crimes committed in our quadrature. We examine the L^2-errors, and in the case of one dimension, the smoothness of the error. Lastly, we present a test example to demonstrate the usefulness for visualisation purposes.

4.3.1 Consistent Integration with Inexact Gaussian Quadrature Approach

We consider the one-dimensional projection of the function

$$u(x) = \sin(2\pi x), \qquad x \in (0,1) \tag{4.16}$$

onto a uniform mesh. We assume that we have a periodic interval in order to simplify the application of the postprocessor. Consistent integration is used, but the number of quadrature points used in the computation of the integrals is one less than what is required for exact integration. The results are presented in Figures 4.1, 4.2, and 4.3. In this set of plots, we can see that the aliasing error is more sensitive at low orders and that the higher

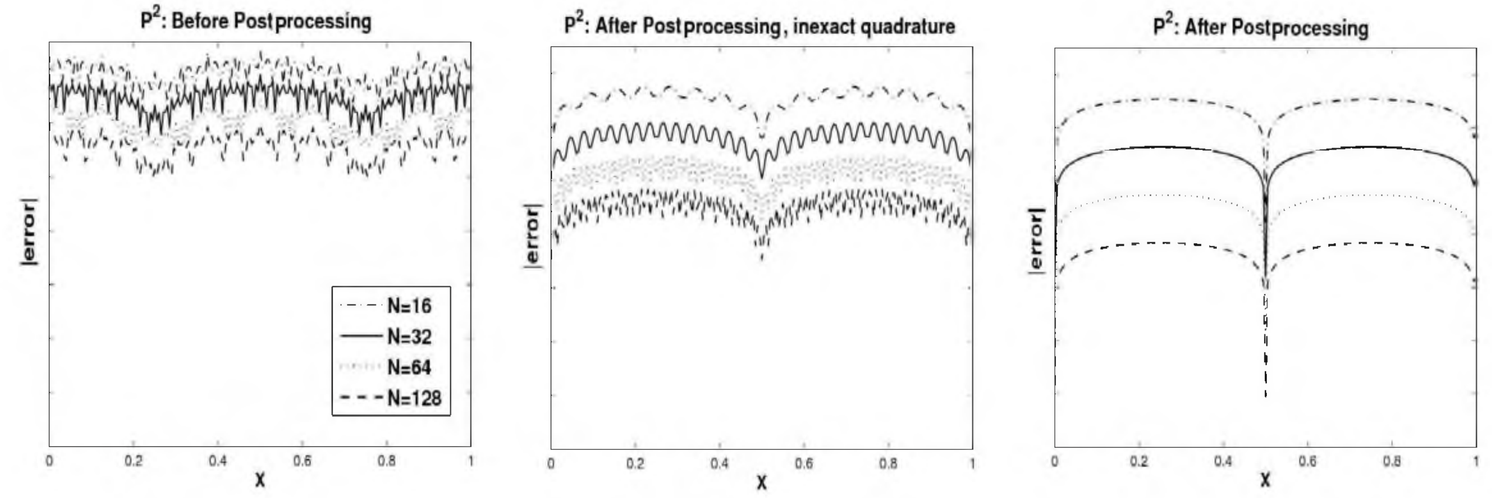


Figure 4.1. Point-wise errors on a logarithmic scale before postprocessing (left), after postprocessing on the consistent integration mesh with inexact quadrature (middle), and after postprocessing with exact quadrature (right). P^2 polynomials.

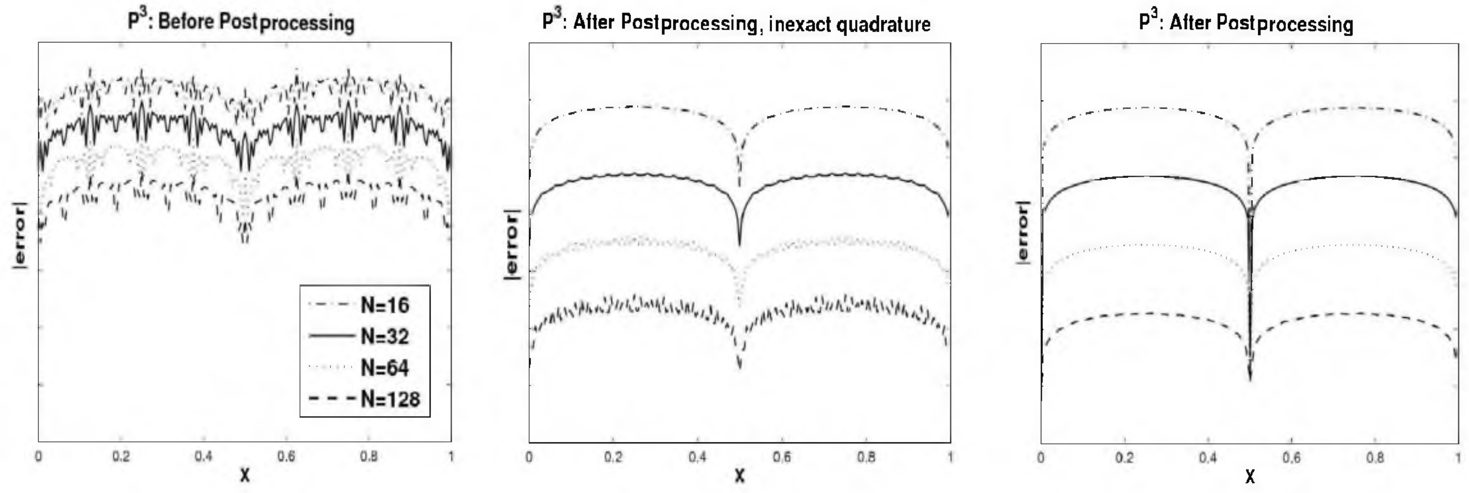


Figure 4.2. Point-wise errors on a logarithmic scale before postprocessing (left), after postprocessing on the consistent integration mesh with inexact quadrature (middle), and after postprocessing with exact quadrature (right). P^3 polynomials.

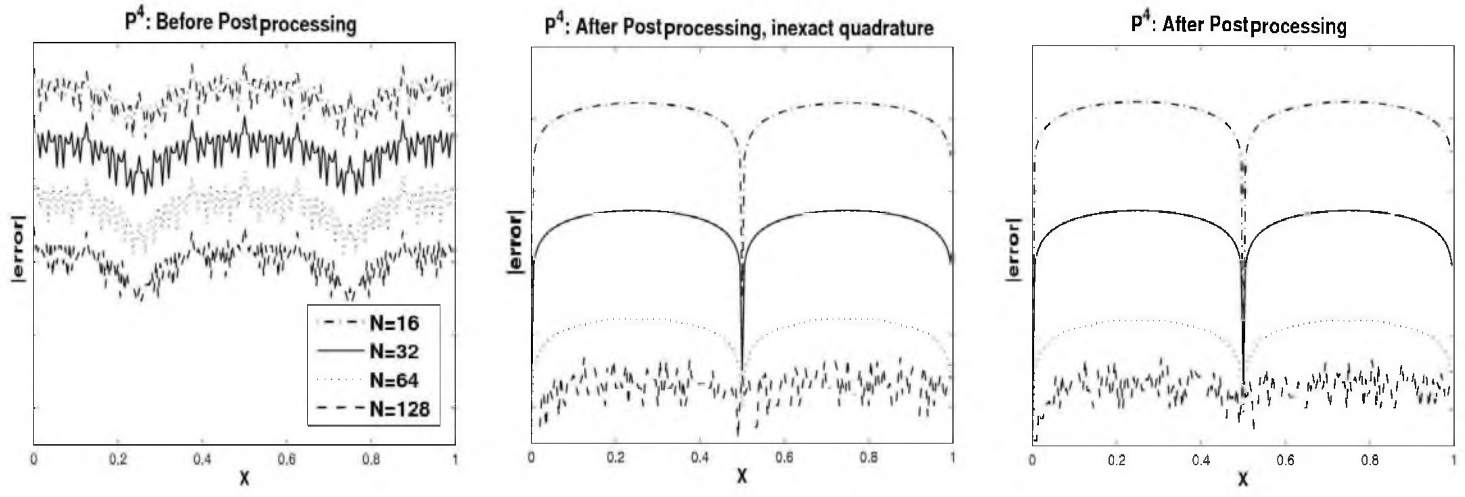


Figure 4.3. Point-wise errors on a logarithmic scale before postprocessing (left), after postprocessing on the consistent integration mesh with inexact quadrature (middle), and after postprocessing with exact quadrature (right). P^4 polynomials.

order coefficients in the approximation are important. Additionally, notice that increasing N for $\mathbb{P}^2$-polynomial approximations does not lower the convergence rate as expected.

4.3.2 Input Mesh-Based Gaussian Quadrature Approach

In this section, we examine the results of not using a consistent integration mesh, but instead attempt to overcome the numerical crimes committed by integrating over the jump by increasing the number of quadrature points.

4.3.2.1 One-Dimensional DG

For this section, we again consider the L^2-projection of the function

$$u(x) = \sin(2\pi x), \qquad x \in (0,1) \tag{4.17}$$

to a uniform mesh. We assume a periodic interval. Figures 4.4, 4.5, 4.6 4.7, 4.8, 4.9, and 4.10 show the point-wise errors when postprocessing on the DG mesh for $\mathbb{P}^2$, $\mathbb{P}^3$, and $\mathbb{P}^4$ polynomials. Considering $\mathbb{P}^k$ polynomials, we need $k+1$ Gauss points to exactly evaluate the inner products involved in postprocessing of a DG solution on a consistent integration mesh. Using the same number of points when postprocessing on the DG input mesh, we observe that the accuracy in terms of error is improved but that oscillations still exist.

4.3.2.2 One-Dimensional DG – Nonuniform Mesh

In this section, we examine the L^2-projection of the function

$$u(x) = \sin(2\pi x), \qquad x \in (0,1) \tag{4.18}$$

to smoothly varying mesh. The smoothly varying mesh is defined by $x = \xi + \frac{1}{2}\sin\xi$, so that the element sizes vary by at most 50% from each other. We use the characteristic length-based postprocessor implementation introduced in [22]. In the case of a nonuniform mesh, Table 4.1 demonstrates that the lowest quadrature order for piecewise quadratic polynomials is not sufficient for removing the error, unlike the uniform mesh case. In the case of a nonuniform mesh, a quadrature that uses at least nine Gauss points seems to be required. For piecewise cubic, the errors using the lowest value of the quadrature are even worse (Table 4.2). And, unless exact quadrature is implemented, the errors are always worse when using ten elements.

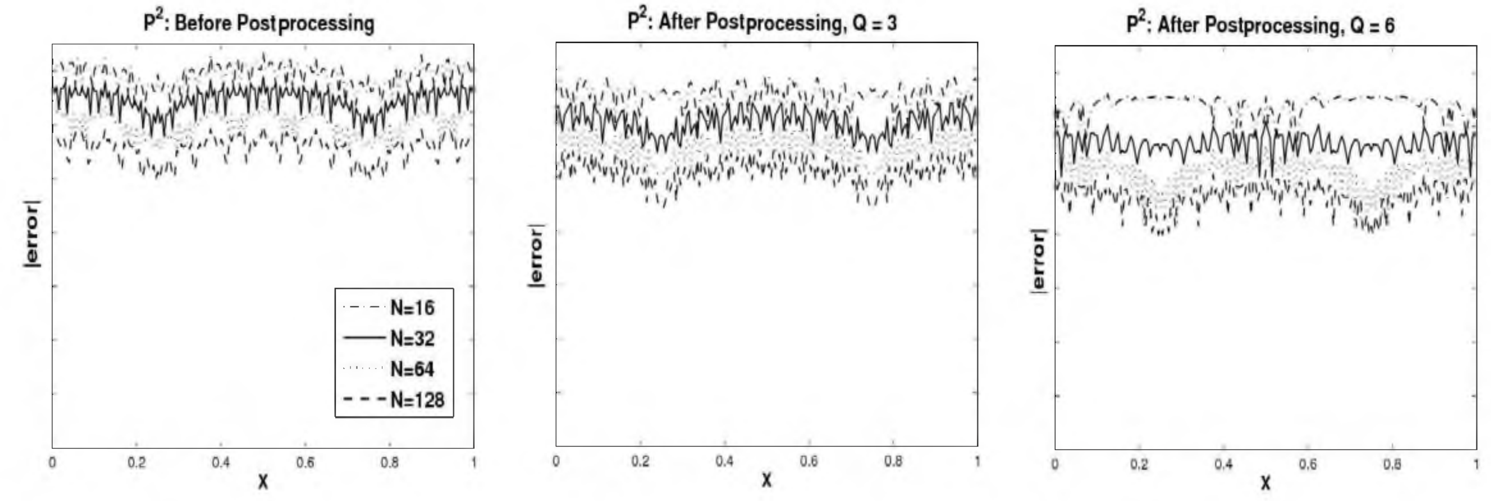


Figure 4.4. Point-wise errors on a logarithmic scale when postprocessing on the DG input mesh using $\mathbb{P}^2$ polynomials. Left: errors before postprocessing. Q indicates the number of quadrature points.

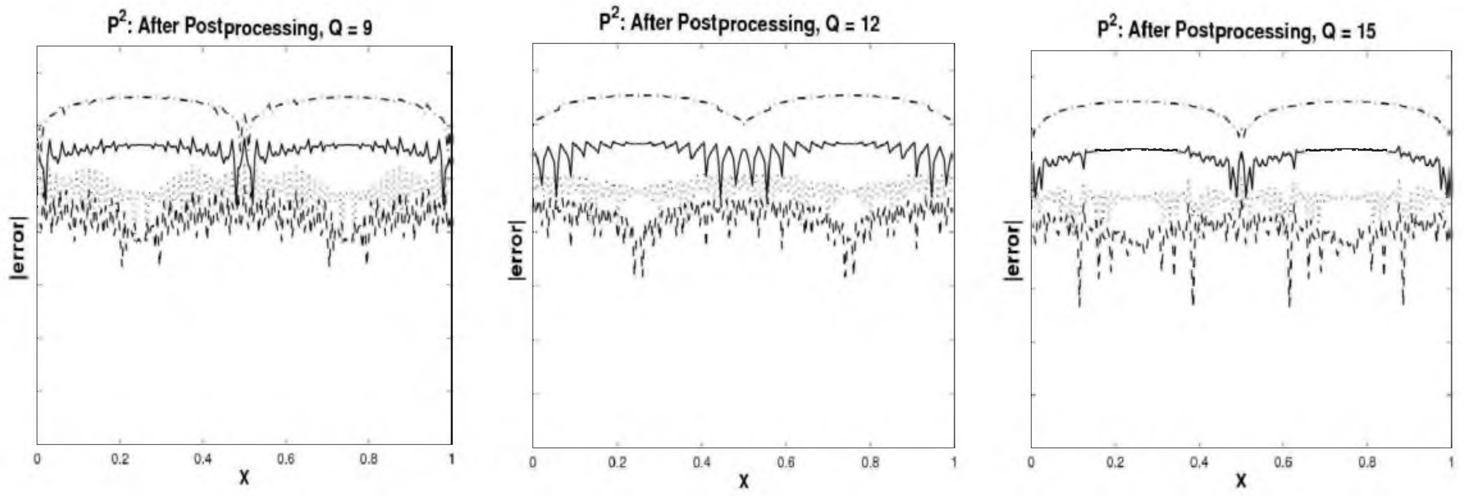


Figure 4.5. Point-wise errors on a logarithmic scale when postprocessing on the DG input mesh using $\mathbb{P}^2$ polynomials. Q indicates the number of quadrature points.

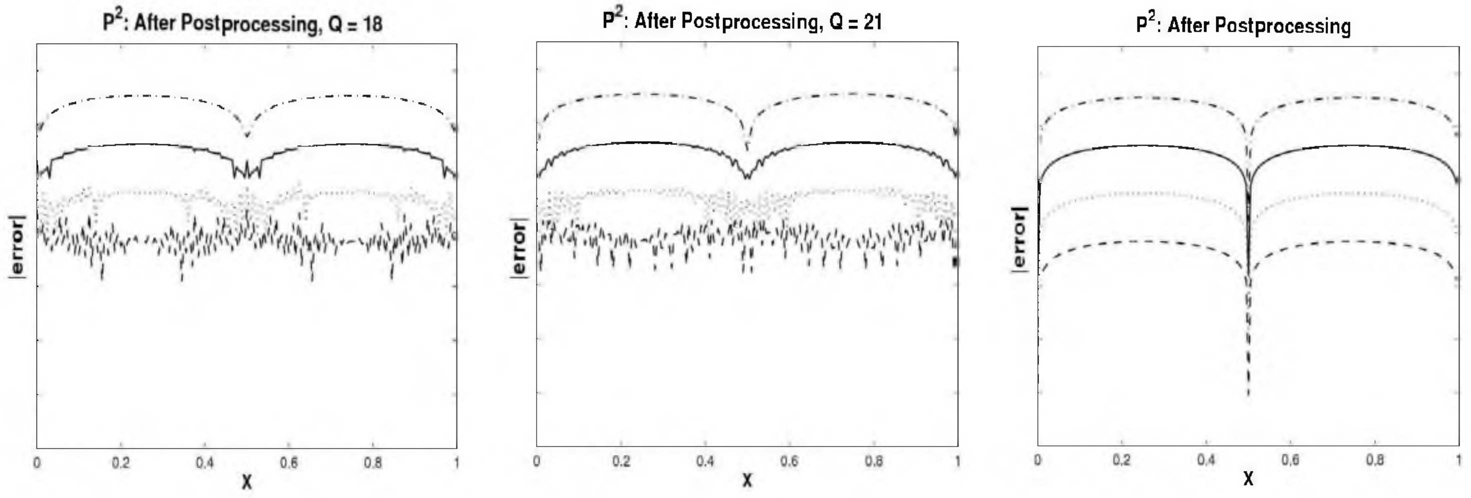


Figure 4.6. Point-wise errors on a logarithmic scale when postprocessing on the DG input mesh using $\mathbb{P}^2$ polynomials. Right: errors after postprocessing on the consistent integration mesh. Q indicates the number of quadrature points.

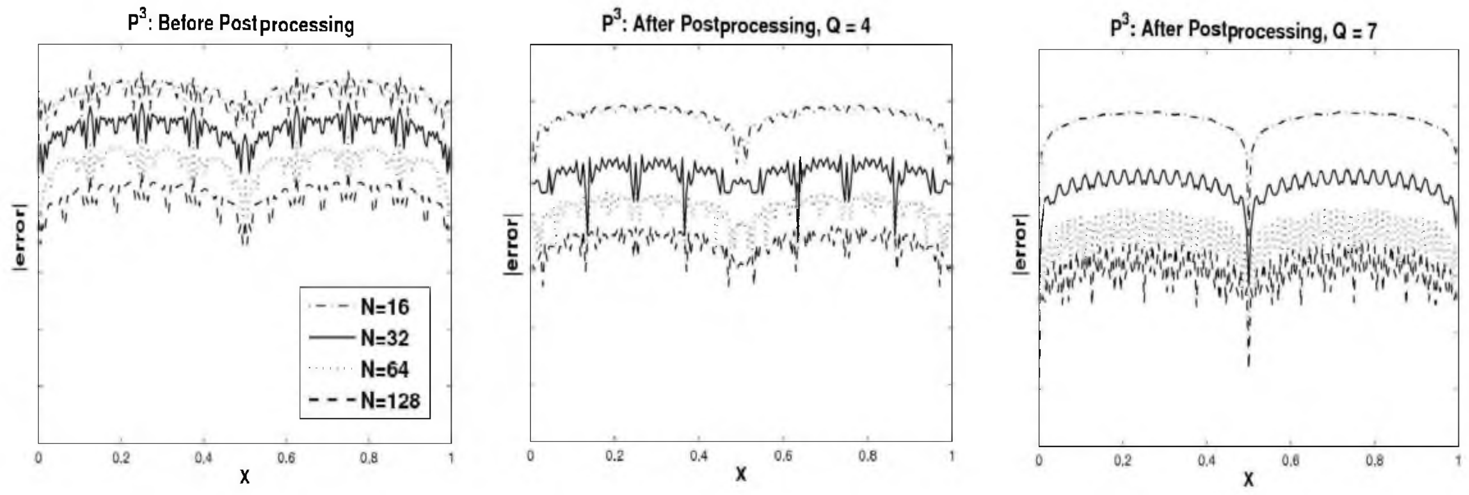


Figure 4.7. Point-wise errors on a logarithmic scale using $\mathbb{P}^3$ polynomials.

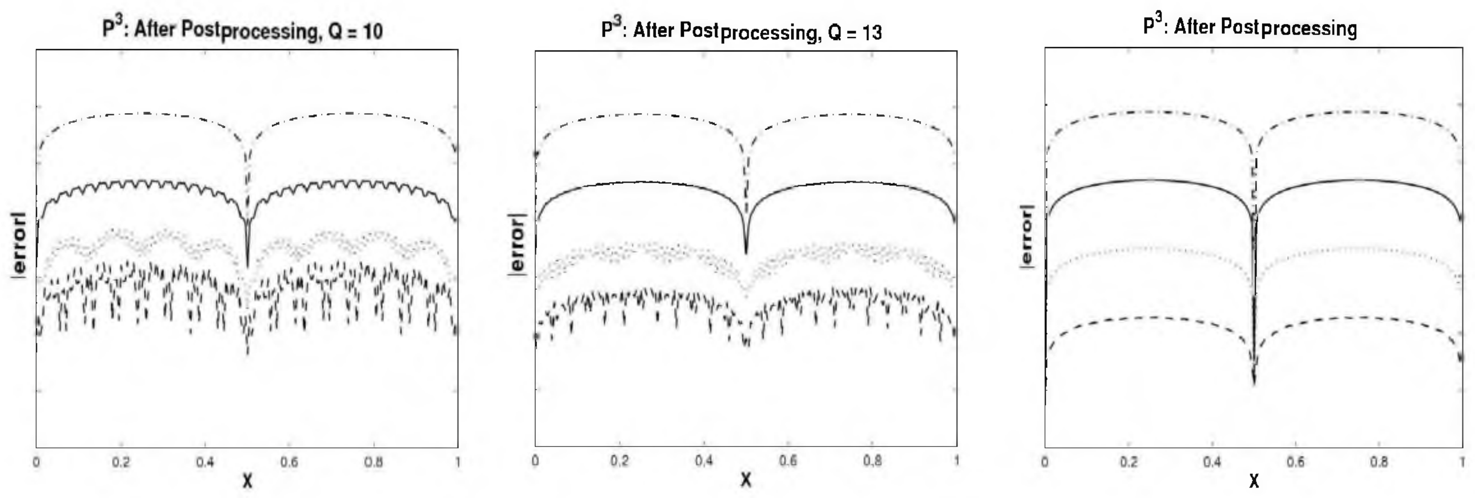


Figure 4.8. Point-wise errors on a logarithmic scale using $\mathbb{P}^3$ polynomials.

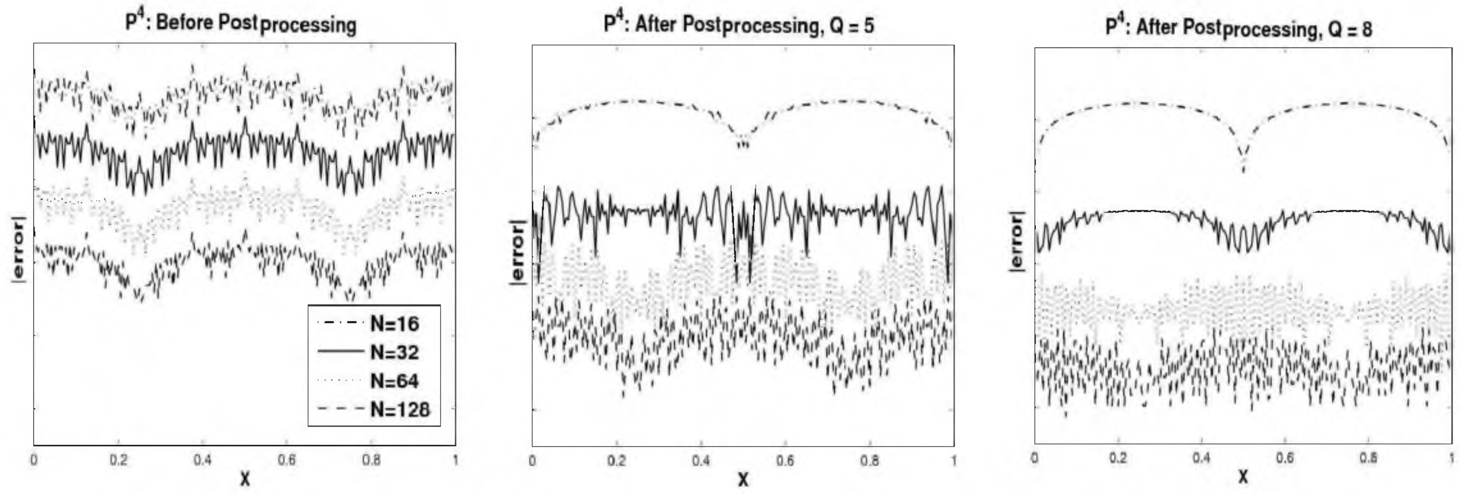


Figure 4.9. Point-wise errors on a logarithmic scale using $\mathbb{P}^4$ polynomials.

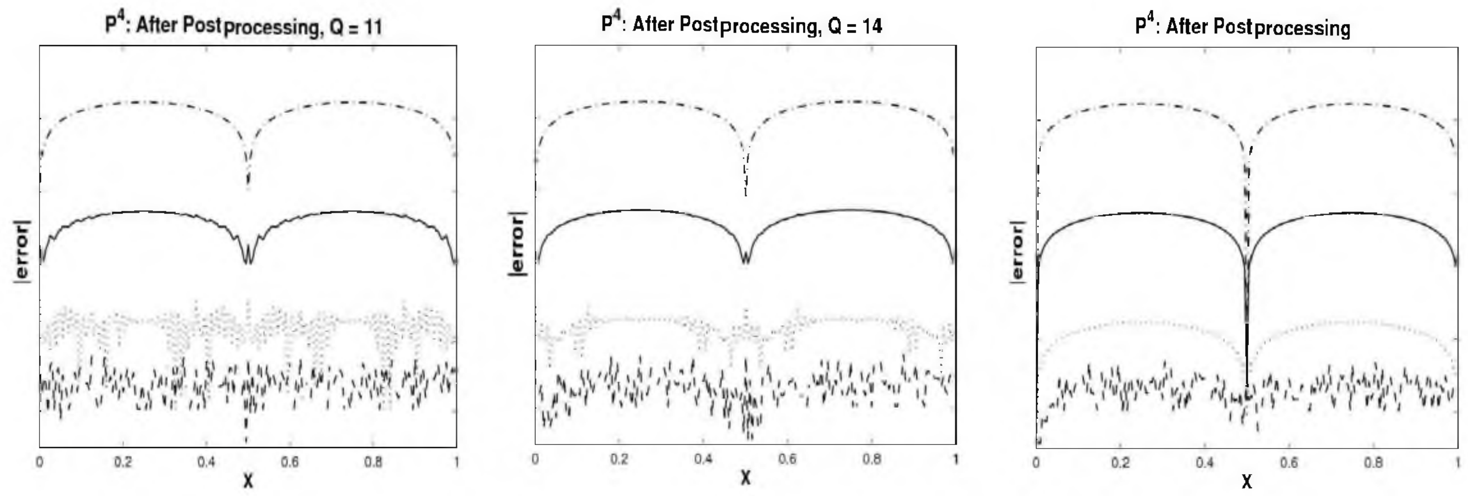


Figure 4.10. Point-wise errors on a logarithmic scale using $\mathbb{P}^4$ polynomials.

Table 4.1. Errors for one-dimensional DG using $\mathbb{P}^2$ polynomials for the nonuniform mesh. Before postprocessing, after postprocessing on the DG mesh where Q is the number of quadrature points, and finally after postprocessing on the consistent integration mesh. CI stands for consistent integration mesh.

				$\mathbb{P}^2$				
mesh	L^2 error	order	L^∞ error	order	L^2 error	order	L^∞ error	order
	Before postprocessing				After postprocessing, $Q = 3$			
10	1.22E-03	–	5.60E-03	–	2.67E-03	–	1.04E-02	–
20	1.55E-04	2.98	7.74E-04	2.85	1.58E-03	0.76	6.16E-03	0.76
40	1.94E-05	3.00	9.93E-05	2.96	4.76E-04	1.73	2.22E-03	1.46
80	2.43E-06	3.00	1.25E-05	2.99	1.05E-04	2.17	6.06E-04	1.87
	After postprocessing, $Q = 9$				After postprocessing, $Q = 12$			
10	6.23E-04	–	1.44E-03	–	6.30E-04	–	1.49E-03	–
20	1.03E-04	2.60	3.68E-04	1.97	5.99E-05	3.39	1.94E-04	2.94
40	9.76E-05	0.08	3.26E-04	0.18	4.01E-05	0.58	1.67E-04	0.22
80	4.09E-05	1.26	1.81E-04	0.85	2.11E-05	0.93	1.12E-04	0.57
	After postprocessing, $Q = 21$				After postprocessing, $Q = 35$			
10	6.17E-04	–	1.42E-03	–	6.21E-04	–	1.41E-03	–
20	1.07E-05	5.85	2.95E-05	5.59	1.15E-09	5.76	2.57E-05	5.78
40	9.79E-06	0.12	3.50E-05	-0.24	2.14E-06	2.43	8.76E-06	1.55
80	8.03E-06	0.28	3.43E-05	0.03	2.01E-06	0.09	8.03E-06	0.13
	After postprocessing, $Q = 45$				After postprocessing, CI			
10	6.22E-04	–	1.42E-03	–	6.21E-04	–	1.42E-04	–
20	1.08E-05	5.84	2.30E-05	5.95	1.08E-05	5.84	2.27E-05	5.96
40	8.99E-07	3.59	3.65E-06	2.65	1.73E-07	5.96	3.36E-07	6.08
80	8.69E-07	0.05	4.34E-06	-0.25	2.74E-09	5.99	5.31E-09	5.99

Table 4.2. Errors for one-dimensional DG using $\mathbb{P}^3$ polynomials for the nonuniform mesh. Before postprocessing, after postprocessing on the DG mesh where Q is the number of quadrature points, and finally after postprocessing on the consistent integration mesh. CI stands for consistent integration mesh.

				$\mathbb{P}^3$				
mesh	L^2 error	order	L^∞ error	order	L^2 error	order	L^∞ error	order
	Before postprocessing				After postprocessing, $Q = 4$			
10	5.56E-05	–	2.18E-04	–	2.97E-04	–	8.94E-04	–
20	3.55E-05	3.97	1.55E-05	3.81	1.88E-04	0.66	6.08E-04	0.55
40	2.24E-07	3.99	9.89E-07	3.97	6.85E-05	1.46	2.75E-04	1.15
80	1.40E-08	4.00	6.16E-08	4.00	1.32E-05	2.38	3.94E-05	2.80
	Before postprocessing, $Q = 7$				After postprocessing, $Q = 16$			
10	1.42E-04	–	3.39E-04	–	1.38E-04	–	3.40E-04	–
20	1.86E-05	2.94	6.16E-05	2.46	9.62E-07	7.16	2.74	6.95
40	1.98E-05	-0.09	6.07E-05	0.02	7.84E-07	0.30	2.82E-06	-0.04
80	4.63E-06	2.10	1.62E-05	1.90	6.21E-07	0.34	1.96E-06	0.53
	Before postprocessing, $Q = 31$				After postprocessing, CI			
10	1.38E-04	–	3.40E-04	–	1.38E-04	–	3.40E-04	–
20	6.34E-07	7.76	1.42E-06	7.91	6.32E-07	7.77	1.41E-06	7.91
40	5.96e-08	3.41	2.56E-07	2.47	2.57E-09	7.94	5.24E-09	8.08
80	5.68E-08	0.07	1.95E-07	0.39	1.02E-11	7.98	2.00E-11	8.04

4.3.2.3 Two-Dimensional DG

In this case, we consider the L^2-projection of the function

$$u(x,y) = \sin(2\pi(x+y)), \qquad x \in (0,1)\,, y \in (0,1) \tag{4.19}$$

to a uniform mesh. Again, we assume a periodic domain. Additionally, we see in Tables 4.3 and 4.4 that although the convergence rates with the postprocessor are not always improved, the errors are always lower than for the input DG solution.

4.3.2.4 Two-Dimensional DG – Constant Coefficient Linear Advection Equation

For this example, we consider solutions of the equation

$$u_t + u_x + u_y = 0, \qquad (x,y) \in (0,2\pi) \times (0,2\pi)\,, \qquad T = 12.5 \tag{4.20}$$

with initial condition $u(0,x,y) = \sin(x+y)$. Observe in Table 4.5 that when we use two Gauss points for linear polynomials, we immediately obtain the desired convergence rate. Additionally, the errors are slightly better than what is observed for the original DG solution. For the quadratic polynomial approximation as presented in Table 4.6, we also immediately improve both the errors and the convergence rate using only three Gauss points.

Table 4.3. Errors for two-dimensional DG using $\mathbb{P}^2$ polynomials. Before postprocessing, after postprocessing on the DG mesh where Q is the number of quadrature points, and finally after postprocessing on the consistent integration mesh. CI stands for consistent integration mesh.

	$\mathbb{P}^2$							
mesh	L^2 error	order	L^∞ error	order	L^2 error	order	L^∞ error	order
	Before postprocessing				After postprocessing, $Q = 3$			
16^2	1.90E-04	–	9.16E-04	–	3.15E-05		1.00E-04	–
32^2	2.38E-05	3.00	1.16E-04	2.99	3.31E-06	3.25	1.18E-05	3.08
64^2	2.98E-06	3.00	1.45E-05	3.00	4.11E-07	3.01	1.48E-06	3.00
128^2	3.72E-07	3.00	1.81E-06	3.00	5.15E-08	3.00	1.85E-07	3.00
	After postprocessing, $Q = 6$				After postprocessing, $Q = 9$			
16^2	1.72E-05	–	2.71E-05	–	1.68E-05	–	2.41E-05	–
32^2	5.05E-07	5.09	1.67E-06	4.02	2.88E-07	5.87	6.75E-07	5.16
64^2	5.35E-08	3.24	2.03E-07	3.04	1.37E-08	4.40	6.97E-08	3.27
128^2	6.65E-09	3.00	2.54E-08	3.00	1.63E-09	3.07	8.68E-09	3.00
	After postprocessing, $Q = 12$				After postprocessing, CI			
16^2	1.68E-05	–	2.40E-05	–	1.68E-05		2.39E-05	–
32^2	2.77E-07	5.92	4.53E-07	5.73	2.69E-07	5.97	3.81E-07	5.97
64^2	9.45E-09	4.87	3.10E-08	3.87	4.22E-09	5.99	6.00E-09	5.99
128^2	1.06E-09	3.15	3.80E-09	3.02	6.60E-11	6.00	3.38E-11	6.00

Table 4.4. Errors for two-dimensional DG using $\mathbb{P}^3$ polynomials. Before postprocessing, after postprocessing on the DG mesh where Q is the number of quadrature points, and finally after postprocessing on the consistent integration mesh. CI stands for consistent integration mesh

$\mathbb{P}^3$								
mesh	L^2 error	order	L^∞ error	order	L^2 error	order	L^∞ error	order
	Before postprocessing				After postprocessing, $Q = 4$			
16^2	4.71E-06	–	2.42E-05	–	8.49E-07	–	1.91E-06	–
32^2	2.95E-07	4.00	1.53E-06	3.99	1.72E-08	5.62	5.21E-08	5.19
64^2	1.84E-08	4.00	9.58E-08	4.00	1.06E-09	4.03	3.21E-09	4.02
128^2	1.15E-09	4.00	5.99E-09	4.00	6.60E-11	4.00	2.01E-10	3.99
	After postprocessing, $Q = 7$				After postprocessing, $Q = 10$			
16^2	8.09E-07	–	1.15E-06	–	8.07E-07	–	1.16E-06	–
32^2	3.43E-09	7.88	5.37E-09	7.75	3.29E-09	7.94	6.05E-09	7.57
64^2	2.69E-11	7.00	6.56E-11	6.36	3.02E-11	6.77	1.08E-10	5.81
128^2	1.10E-12	4.61	3.03E-12	4.44	1.71E-12	4.15	5.70E-12	4.25
	After postprocessing, $Q = 13$				After postprocessing, CI			
16^2	8.09E-07	–	1.15E06	–	8.07E-07	–	1.14E-06	–
32^2	3.42E-09	7.88	5.26E-09	7.77	3.26E-09	7.95	4.61E-09	7.95
64^2	2.44E-11	7.13	5.84E-11	6.49	1.29E-11	7.99	1.82E-11	7.99
128^2	8.62E-13	4.82	2.59E-12	4.50	5.04E-14	7.99	7.74E-14	7.88

Table 4.5. Errors for one-dimensional DG using $\mathbb{P}^1$ polynomials for the linear advection equation. Before postprocessing, after postprocessing on the DG mesh where Q is the number of quadrature points, and finally after postprocessing on the consistent integration mesh. CI stands for consistent integration mesh.

$\mathbb{P}^1$								
mesh	L^2 error	order	L^∞ error	order	L^2 error	order	L^∞ error	order
	Before postprocessing				After postprocessing, $Q = 2$			
10^2	1.92E-01	–	2.93E-01	–	1.92E-01	–	2.78E-01	–
20^2	3.02E-02	2.67	4.98E-02	2.56	2.93E-02	2.71	4.35E-02	2.68
40^2	4.31E-03	2.81	7.69E-03	2.69	3.81E-03	2.94	5.95E-03	2.87
80^2	7.06E-04	2.61	2.45E-03	1.65	4.85E-04	2.97	8.24E-04	2.85
	After postprocessing, $Q = 5$				After postprocessing, $Q = 8$			
10^2	1.92E-01	–	2.73E-01	–	1.92E-01	–	2.71E-01	–
20^2	2.94E-02	2.71	4.18E-02	2.71	2.92E-02	2.71	4.15E-02	2.71
40^2	3.83E-03	2.94	5.50E-03	2.93	3.78E-03	2.95	5.39E-03	2.94
80^2	4.88E-04	2.97	7.10E-04	2.95	4.76E-04	2.99	6.82E-04	2.98
	After postprocessing, $Q = 11$				After postprocessing, CI			
10^2	1.92E-01	–	2.71E-01	–	1.92E-01	–	2.71E-01	–
20^2	2.92E-02	2.71	4.14E-02	2.71	2.92E-02	2.71	4.14E-02	2.71
40^2	3.78E-03	2.95	5.37E-03	2.95	3.78E-03	2.95	5.35E-03	2.95
80^2	4.75E-04	2.99	6.77E-04	2.99	4.76E-04	2.99	6.73E-04	2.99

Table 4.6. Errors for two-dimensional DG using $\mathbb{P}^2$ polynomials for the linear advection equation. Before postprocessing, after postprocessing on the DG mesh where Q is the number of quadrature points, and finally after postprocessing on the consistent integration mesh. CI stands for consistent integration mesh.

$\mathbb{P}^2$								
mesh	L^2 error	order	L^∞ error	order	L^2 error	order	L^∞ error	order
	Before postprocessing				After postprocessing, $Q = 3$			
10^2	4.95E-03	–	2.78E-02	–	3.48E-03	–	4.99E-03	–
20^2	4.87E-04	3.34	3.72E-03	2.90	1.11E-04	4.97	1.64E-04	4.92
40^2	5.96E-05	3.03	4.74E-04	2.97	3.83E-06	4.86	6.57E-06	4.64
80^2	7.44E-06	3.00	5.94E-05	3.00	2.38E-07	4.01	5.75E-07	3.51
	Before postprocessing, $Q = 6$				After postprocessing, CI			
10^2	3.48E-03	–	4.92E-03	–	3.48E-03	–	4.92E-03	–
20^2	1.10E-04	4.98	1.56E-04	4.98	1.10E-04	4.98	1.56E-04	4.98
40^2	3.44E-06	5.00	4.91E-06	4.99	3.42E-06	5.01	4.84E-06	5.01
80^2	1.12E-07	4.94	1.78E-07	4.79	1.07E-07	5.01	1.51E-07	5.01

4.3.2.5 Two-Dimensional DG – Variable Coefficient Linear Advection Equation

In this example, we consider solutions to a two-dimensional variable coefficient equation,

$$u_t + (au)_x + (au)_y = f(x,y,t), \quad (x,y) \in (0,2\pi) \times (0,2\pi), \quad T = 12.5 \tag{4.21}$$

with the variable coefficient function $a(x,y) = 2 + \sin(x+y)$. We set the forcing function so that the solution is $u(x,y,t) = \sin(x+y-2t)$. We observe in Tables 4.7 and 4.8 that the convergence rate is not always optimal, but that indeed, with a minimum number of Gauss points, we improve the errors.

4.3.3 Input Mesh-Based Midpoint Quadrature Approach

In this section, the effect of using midpoint integration while approximating the convolution operator is examined for the 1D and 2D cases.

4.3.3.1 One-Dimensional DG – Midpoint Quadrature

In this example, we again consider the case of projecting

$$\sin(2\pi x), \quad x \in (0,1)$$

onto a piecewise polynomial space. The results are displayed in Figures 4.11, 4.12, and 4.13. The plots show that it takes many applications of the midpoint rule to get better errors for the postprocessed solution than the initial projection, and that the midpoint rule is effective when we use 128 points or greater. However, the point-wise errors are smoother than the errors for the piecewise polynomial projection. This is because the breakpoints for the midpoint rule align with the element boundaries on our mesh. Additionally, in Figure 4.14, we can see that the lines level off to the same error as that for consistent integration error, which is the best possible scenario.

Table 4.7. Errors for two-dimensional DG using $\mathbb{P}^1$ polynomials for the variable coefficient advection equation. Before postprocessing, after postprocessing on the DG mesh where Q is the number of quadrature points, and finally after postprocessing on the consistent integration mesh. CI stands for consistent integration mesh.

				$\mathbb{P}^1$				
mesh	L^2 error	order	L^∞ error	order	L^2 error	order	L^∞ error	order
	Before postprocessing				After postprocessing, $Q = 2$			
10^2	3.52E-02	–	2.06E-01	–	1.50E-02	–	2.51E-02	–
20^2	8.42E-03	2.06	5.10E-02	2.01	1.85E-03	3.01	4.90E-03	2.36
40^2	2.09E-03	2.01	1.28E-02	2.00	3.48E-04	2.41	9.61E-04	2.35
80^2	5.23E-04	2.00	3.17E-03	2.00	7.78E-05	2.16	2.16E-04	2.15
	After postprocessing, $Q = 5$				After postprocessing, $Q = 8$			
10^2	1.42E-02	–	2.21E-02	–	1.41E-02	–	2.20E-02	–
20^2	1.36E-03	3.38	2.68E-03	3.04	1.44E-03	3.29	2.66E-03	3.05
40^2	1.66E-04	3.04	3.73E-04	2.85	1.95E-04	2.88	3.74E-04	2.83
80^2	2.11E-05	2.97	5.57E-05	2.74	2.71E-05	2.85	5.67E-05	2.72
	After postprocessing, $Q = 11$				After postprocessing, CI			
10^2	1.41E-02	–	2.20E-02	–	1.41E-02	–	2.20E-02	–
20^2	1.44E-03	3.29	2.64E-03	3.06	1.44E-03	3.29	2.63E-03	3.06
40^2	1.97E-04	2.87	3.69E-04	2.84	1.94E-04	2.89	3.53E-04	2.90
80^2	2.75E-05	2.84	5.53E-05	2.74	2.66E-05	2.86	5.00E-05	2.82

Table 4.8. Errors for two-dimensional DG using $\mathbb{P}^2$ polynomials for the variable coefficient advection equation. Before postprocessing, after postprocessing on the DG mesh where Q is the number of quadrature points, and finally after postprocessing on the consistent integration mesh. CI stands for consistent integration mesh.

				$\mathbb{P}^2$				
mesh	L^2 error	order	L^∞ error	order	L^2 error	order	L^∞ error	order
	Before postprocessing				After postprocessing, $Q = 3$			
10^2	3.87E-03	–	3.39E-03	–	2.85E-04	–	6.30E-04	–
20^2	4.79E-04	3.02	4.06E-03	3.06	1.52E-05	4.22	4.36E-05	3.85
40^2	5.97E-05	3.01	4.93E-04	3.04	1.72E-06	3.14	4.73E-06	3.20
80^2	7.45E-06	3.00	6.05E-05	3.03	2.12E-07	3.02	5.64E-07	3.07
	After postprocessing, $Q = 6$				After postprocessing, $Q = 9$			
10^2	2.62E-04	–	4.91E-04	–	2.61E-04		4.69E-04	–
20^2	6.91E-06	5.24	1.43E-05	5.10	6.59E-06	5.31	1.44E-05	5.03
40^2	3.59E-07	4.27	1.05E-06	3.77	2.49E-07	4.73	6.30E-07	4.51
80^2	3.43E-08	3.39	1.02E-07	3.36	1.12E-08	4.47	4.53E-08	3.80
	After postprocessing, $Q = 12$				After postprocessing, $Q = 15$			
10^2	2.61E-04	–	4.63E-04	–	2.61E-04		4.62E-04	–
20^2	6.58E-06	5.31	1.06E-05	5.45	6.60E-06	5.31	1.06E-05	5.45
40^2	2.44E-07	4.75	4.92E-07	4.43	2.41E-07	4.76	4.47E-07	4.57
80^2	9.40E-09	4.70	2.64E-08	4.22	8.37E-09	4.85	1.87E-08	4.58
	After postprocessing, $Q = 18$				After postprocessing, CI			
10^2	2.61E-04	–	4.62E-04	–	2.61E-04		4.62E-04	–
20^2	6.57E-06	5.31	1.06E-05	5.45	6.57E-06	5.31	1.06E-05	5.45
40^2	2.41E-07	4.77	4.34E-07	4.61	2.41E-07	4.77	4.18E-07	4.66
80^2	8.08E-09	4.90	1.70E-08	5.34	8.04E-09	4.91	1.49E-08	4.81

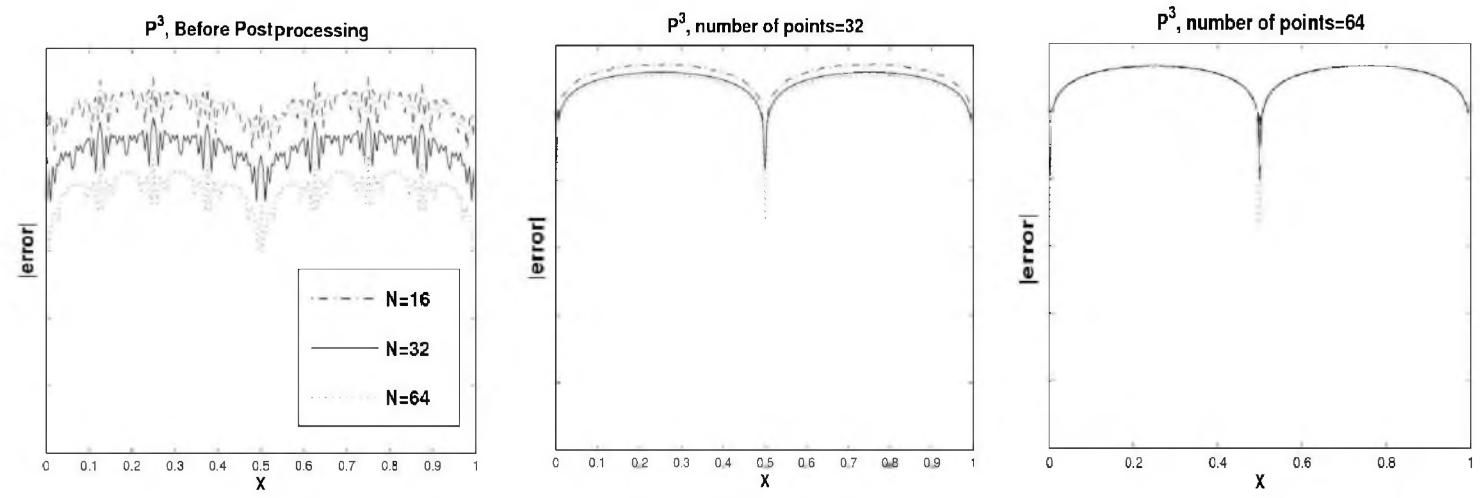


Figure 4.11. Point-wise errors on a logarithmic scale when using midpoint integration for postprocessing with different number of evaluation points. Left: before postprocessing.

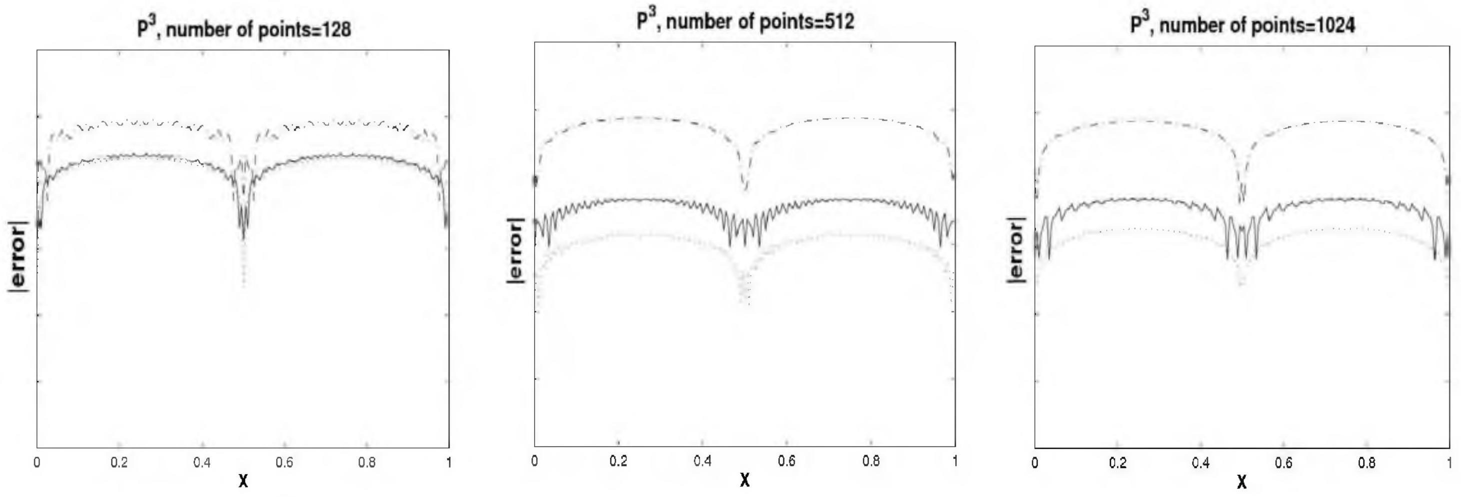


Figure 4.12. Point-wise errors on a logarithmic scale when using midpoint integration for postprocessing with different number of evaluation points.

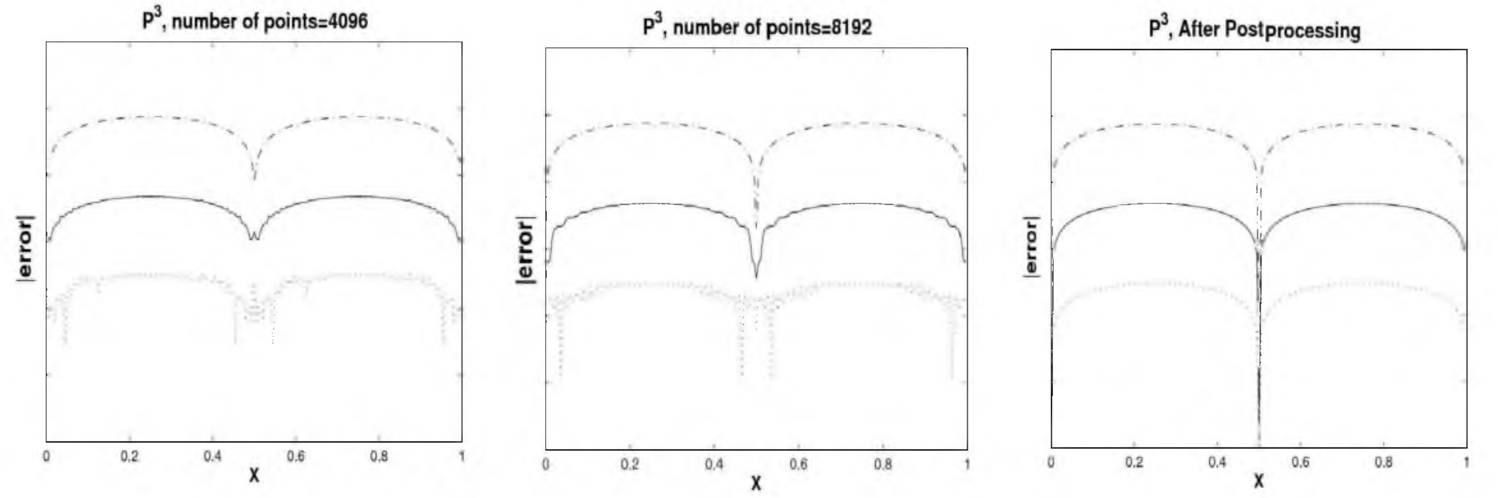


Figure 4.13. Point-wise errors on a logarithmic scale when using midpoint integration for postprocessing with different number of evaluation points. Right: after postprocessing on the consistent integration mesh.

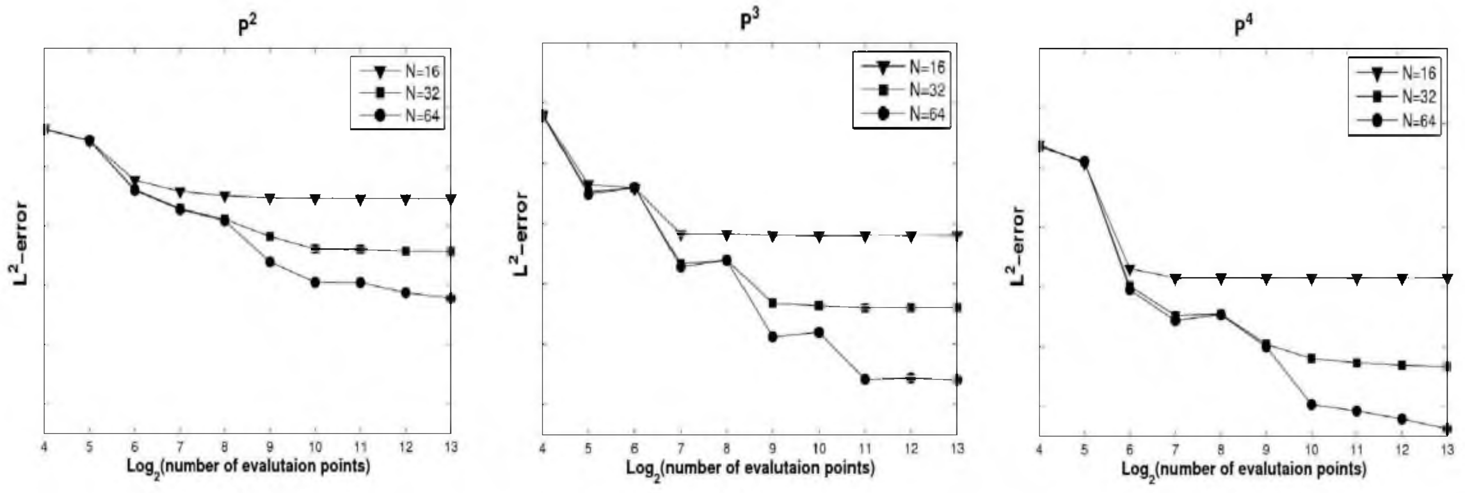


Figure 4.14. Convergence of the L^2 errors when using midpoint integration for postprocessing.

4.3.3.2 Two-Dimensional DG – Midpoint Quadrature

In this section, we again consider the two-dimensional projection problem given by Equation (4.19). It is observed that in the case of the 2D midpoint rule, the convergence rate in the L^2-norm is linear, as is shown in the sample plot in Figure 4.15. We observe this linear trend as we double the number of evaluation points. For this particular example and based upon the slope of our convergence diagram, we would need approximately 2^{13} evaluation points to get an error level similar to the DG solution. The errors for the higher degree polynomials are not shown as they do not provide any new information given the computational time required to compute them.

4.3.4 Two-Dimensional Vector Field

As it is mentioned in [67, 75], smoothness-increasing, accuracy-conserving filtering can be applied to discontinuous Galerkin vector fields to enhance streamline integration. In this section, we examine the impact of *input mesh-based filtering* of a 2D vector field on streamline calculations for visualization purposes.

A two-dimensional vector field was created from

$$\begin{bmatrix} u(r,\theta) \\ v(r,\theta) \end{bmatrix} = \begin{bmatrix} \frac{1}{2}\cos(20\theta)\cos(\theta) - r\sin(\theta) \\ \frac{1}{2}\cos(20\theta)\sin(\theta) + r\cos(\theta) \end{bmatrix}. \tag{4.22}$$

This has streamlines which are oscillating closed circuits. In Figures 4.16, 4.17, and 4.18,

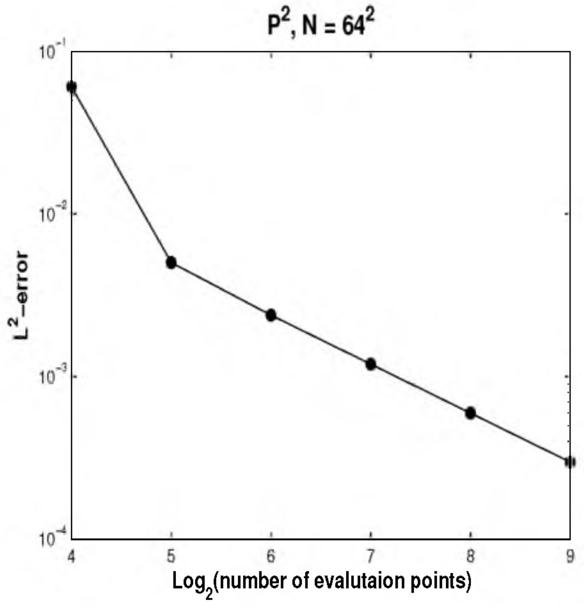


Figure 4.15. Convergence of the L^2 errors when using midpoint integration for postprocessing.

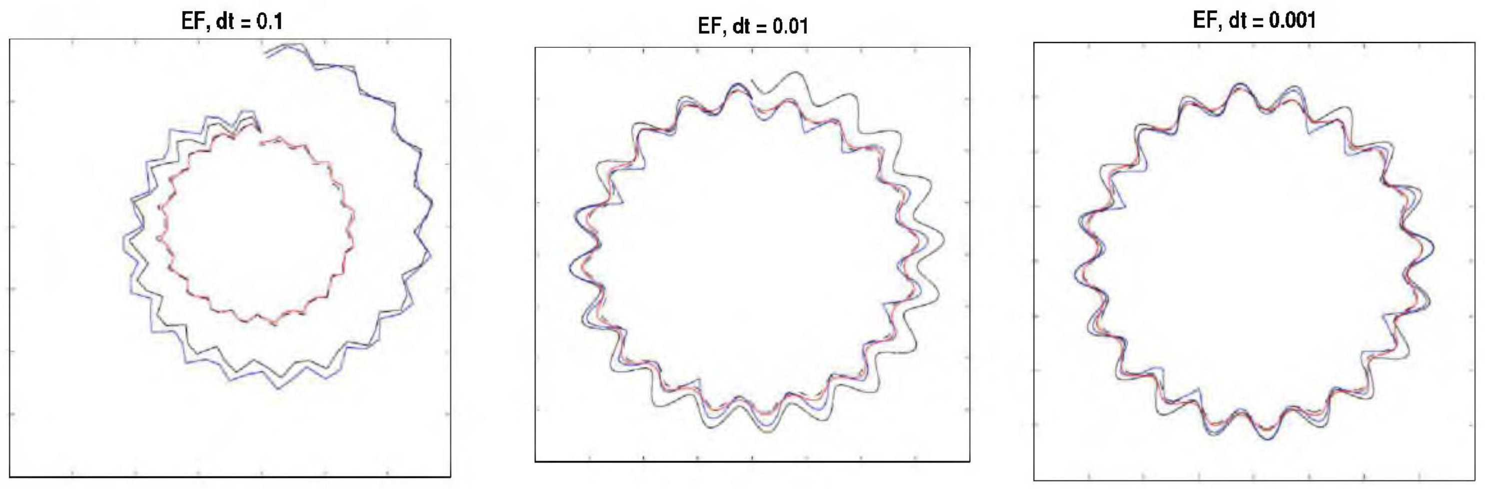


Figure 4.16. Streamline integration example based upon vector field mentioned in Equation (4.22). Solid black streamlines denote "true" solution; blue streamlines were created based upon integration on an L_2 projected field; red streamlines were created based upon integration on a filtered field using consistent integration approach, and dashed black streamlines were created based upon integration on a filtered field using the input mesh-based approach. Euler forward time integration.

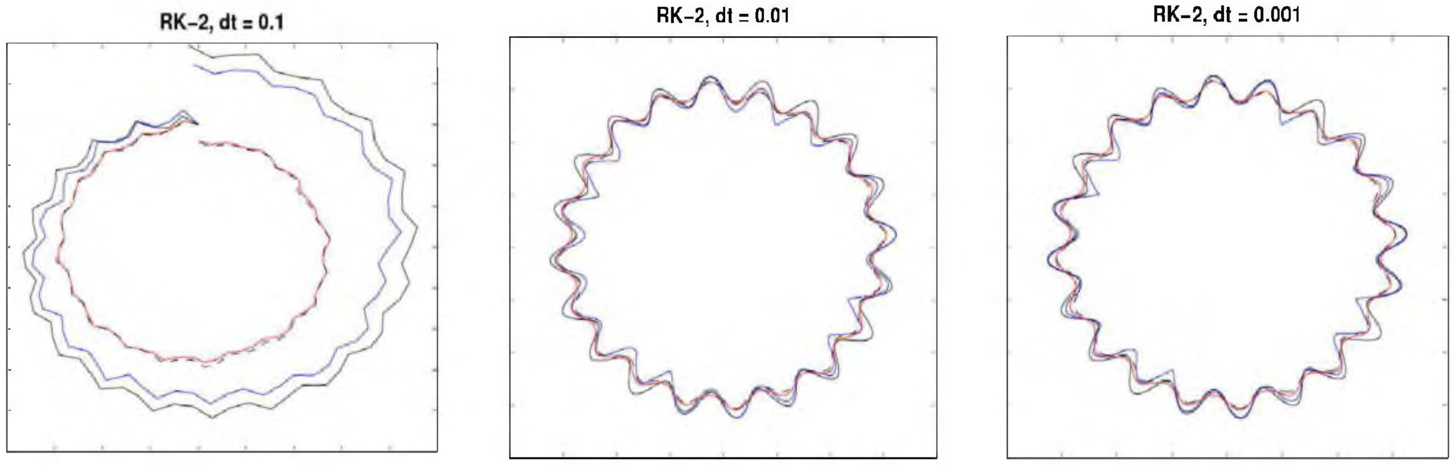


Figure 4.17. Streamline integration example based upon vector field mentioned in Equation (4.22). Solid black streamlines denote "true" solution; blue streamlines were created based upon integration on an L_2 projected field; red streamlines were created based upon integration on a filtered field using consistent integration approach, and dashed black streamlines were created based upon integration on a filtered field using the input mesh-based approach. Euler forward time integration. Rk-2 time integration.

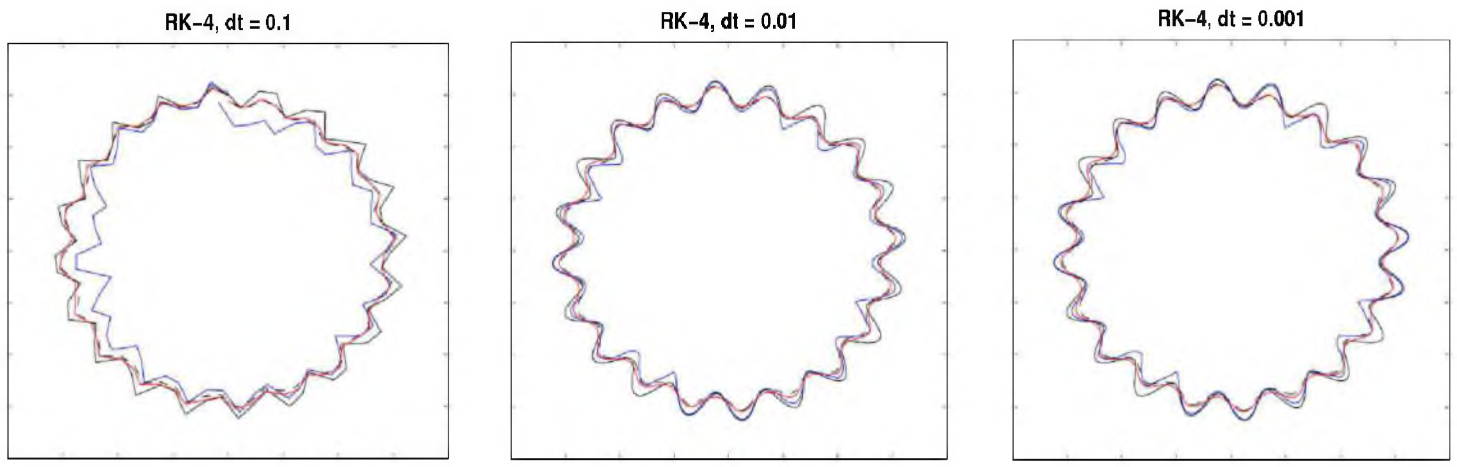


Figure 4.18. Streamline integration example based upon vector field mentioned in Equation (4.22). Solid black streamlines denote "true" solution; blue streamlines were created based upon integration on an L_2 projected field; red streamlines were created based upon integration on a filtered field using consistent integration approach, and dashed black streamlines were created based upon integration on a filtered field using the input mesh-based approach. RK-4 time integration.

we present a sample streamline of this vector field by projecting the function above over a 40×40 uniform mesh on the interval $[-1, 1] \times [-1, 1]$ with a starting location of $(0.0, 0.3)$. The field approximations are linear in both the $x-$ and $y-$ directions. Streamlines were calculated using three different time integration schemes, Euler Forward, 2nd-order Runge-Kutta (RK-2), and 4th-order Runge-Kutta (RK-4), with three different time steps $dt = 0.1, 0.01, 0.001$. The "true solution" streamlines (denoted as a solid black line in all the images) are calculated by performing RK-4 on the analytical function.

These results corroborate that input mesh-based integration with sufficient quadrature provides sufficient postprocessing benefit in terms of smoothness and accuracy to be of use in data processing and visualization.

4.4 Summary and Conclusions

In this chapter, we presented a atudy of the impact of numerical quadrature approximations used for evaluating the convolution operator in the smoothness-increasing accuracy-conserving (SIAC) filter. We provided both theoretical estimates as well as empirical results which demonstrated the efficacy of the postprocessing approach when different levels and types of quadrature approximation are used. We first examined the case when consistent integration with inexact Gaussian quadrature (under-integration) is used. Because consistent integration requires performing up to several numerical quadratures within an element, we considered what happens when only the original DG mesh is used as the underlying support mesh for integration. Under this scenario, we examined the use of Gaussian quadrature and midpoint quadrature. This choice highlighted the differences between polynomial-based high-order and adaptive low-order quadrature implementations.

There are several points that can be drawn from our results and discussions:

- The major uncontrollable cost in the postprocessing algorithm is performing several numerical quadratures within a DG element. In the case of uniform meshes, many things simplify to drastically cut down the cost; however, uniform meshes are not often used in general engineering practice.

- Because the postprocessor consists of integrating a B-spline kernel against a DG solution, there are certain things we can state about the integrand being integrated. As the DG element discontinuities and the B-spline knot lines cannot overlap, we know that in the worst case, the integrands contain a reduction in regularity due to the product of the DG discontinuity with the polynomial on a B-spline knot-segment.

Although Gauss quadrature over such a region is not exact, it can nonetheless be very effective.

- If the cost of performing integration on the consistent mesh is prohibitive, postprocessing with input mesh-based postprocessing provides in many cases benefits comparable with consistent integration. The error introduced can be controlled by increasing the number of quadrature points.

- Alternatives to Gaussian quadrature can be used for evaluating the convolution integrals; however, a large number of samples are needed to obtain comparable results.

- When examined in light of an application area as visualization of DG solutions, input mesh-based postprocessing appears to provide a convenient means of obtaining smooth solutions with controllable accuracy.

We emphasize again that our study is primarily for engineering circumstances when trade-offs between time, resources, and accuracy are important. Although the case against committing such numerical crimes is well-known, the repercussions have not been well documented for the use of this filter as a visualization tool. It is concerning this specific crime to which we have attempted to provide both theoretical and empirical insight.

CHAPTER 5

NUMERICAL BEHAVIOR OF SIAC FILTERING FOR STRUCTURED TRIANGULATIONS

One drawback of the previous implementations of the postprocessor is that there is a basic assumption that the data are obtained over a uniform quadrilateral mesh. However, this assumption is restrictive, which makes the application of this postprocessing technique to general tessellations a challenging task. In this chapter, we demonstrate the behavior and complexity of the computational extension of this SIAC filter to structured triangular meshes. We furthermore show that the theoretical extension to variable coefficient equations over structured triangular meshes is straightforward. Moving from quadrilateral meshes to triangulated ones introduces more complexity in the calculations as the number of required integrations increases. This is a challenging first step toward implementing smoothness-increasing accuracy-conserving filters for unstructured tessellations. By using the usual B-spline implementation, we are able to improve on the order of accuracy as well as decrease the magnitude of the errors. We are essentially able to increase the order of accuracy from $\mathcal{O}(h^{k+1})$ to approximately $\mathcal{O}(h^{2k+1})$ for structured triangular meshes and show accuracy enhancement for smoothly varying meshes and Union-Jack meshes. These results are valid regardless of whether we employ exact or inexact integration.

The detail of implementation of the SIAC filter for structured triangular meshes was discussed in Section 3.4.2. Here, we present the numerical behaviour of this filtering technique over these type of meshes. We begin by reviewing the key concepts of discontinuous Galerkin methods over triangulations. In Section 5.2, we show how there is a natural theoretical extension of the accuracy enhancing capabilities of the SIAC filter for structured triangular meshes. In Section 5.3, we give numerical results confirming the usefulness of our smoothness-increasing accuracy-conserving filter for triangulated meshes. The result of these contributions has been published in [46].

5.1 The Discontinuous Galerkin Formulation for Triangular Mesh Structures

In this section, we provide an overview of the discontinuous Galerkin method over a domain that is subdivided into triangular elements. A more thorough study of the DG formulation is presented in Section 2.3. Here, we introduce the notations we consider throughout this chapter.

We consider the numerical solution of a two-dimensional linear hyperbolic equation of the form

$$\begin{aligned} u_t + \sum_{i=1}^{2} \frac{\partial}{\partial x_i}(A_i(\mathbf{x})u) &= 0, \qquad \mathbf{x} \in \Omega \times [0,T], \\ u(\mathbf{x},0) &= u_o(\mathbf{x}), \qquad \mathbf{x} \in \Omega, \end{aligned} \tag{5.1}$$

where $\mathbf{x} \in \Omega,\ \ t \in \mathbb{R}$, and $A_i(\mathbf{x}),\ i = 1,2$ is bounded in the $L^\infty-$norm. We also assume smooth initial conditions are given along with periodic boundary conditions.

We begin by defining our mesh such that the computational domain, Ω, consists of N nonoverlapping triangular elements. We designate these by τ_e and assume that their characteristic length is h. The tessellated computational domain then is given by $\widetilde{\Omega} = \bigcup_{e=1}^{N} \tau_e$. We also define V_h to be the approximation space consisting of piecewise polynomials of degree at most k on element τ_e,

$$V_h = \left\{ \varphi \in L^2(\widetilde{\Omega}) : \quad \varphi|_{\tau_e} \in \mathbb{P}^k(\tau_e), \qquad \forall \tau_e \in \widetilde{\Omega} \right\}. \tag{5.2}$$

We will approximate the exact solution $u(\mathbf{x})$ with $u_h(\mathbf{x}) \in V_h$.

To begin defining our numerical method, we consider the weak form of Equation (5.2) in order to derive our discontinuous Galerkin approximation. That is, we multiply Equation (5.2) by a smooth function $v(x)$ and integrate over τ_e. We then make use of Green's theorem in the second term to obtain a formulation that includes boundary terms. Next, the test function, v, is replaced with a piecewise polynomial function, $v_h \in \mathbf{V}_h$, in the approximation space. The flux function is defined to be $f_i(\mathbf{x},t) = A_i(\mathbf{x})u_h(\mathbf{x},t),\ i = 1,2$. The DG formulation is then given by

$$\int_{\tau_e} \frac{\partial u_h}{\partial t} v_h \, d\mathbf{x} - \sum_{i=1}^{2} \int_{\tau_e} f_i(\mathbf{x},t) \frac{\partial v_h}{\partial x_i} \, d\mathbf{x} + \sum_{i=1}^{2} \int_{\partial\tau_e} \hat{f}_i \hat{n}_i v_h \, ds = 0, \tag{5.3}$$

where $\partial\tau_e$ is the boundary of the element τ_e, and $\hat{n}_i$ denotes the unit outward normal to the element boundary in the i^{th} direction. Notice that the flux is multiply defined at element interfaces and therefore, we impose the definition that $\hat{f}_i\hat{n}_i = h(u_h(\mathbf{x}^{exterior},t), u_h(\mathbf{x}^{interior},t))$

is a consistent two-point monotone Lipschitz flux as in [13]. We note that if we sum over all the elements, τ_e, we can rewrite Equation (5.3) as

$$\left(\frac{\partial u_h}{\partial t}, v_h\right)_{\widetilde{\Omega}} + B(\mathsf{A}, u_h; v_h)_{\widetilde{\Omega}} = 0, \tag{5.4}$$

where

$$B(\mathsf{A}, u_h; v_h) = -\sum_{i=1}^{2}\left(A_i(\mathbf{x})u_h(\mathbf{x},t), \frac{\partial v_h}{\partial x_i}\right)_{\widetilde{\Omega}} + \sum_{\tau_e}\sum_{i=1}^{2}\int_{\partial\tau_e} A_i(\mathbf{x})\hat{u_h} v_h \hat{n_i}\, ds, \tag{5.5}$$

and we have denoted $A_i(\mathbf{x}), i = 1, 2$ by A for simplicity.

The approximate solution, u_h, within an element is given by

$$u_h(\mathbf{x}, t) = \sum_{p=0}^{k}\sum_{q=0}^{k-p} u^{(p,q)}_{\tau_e(t)}\phi^{(p,q)}(\mathbf{x}), \qquad \mathbf{x} \in \tau_e, \tag{5.6}$$

where $u^{(p,q)}_{\tau_e}(t)$ represent the expansion coefficients and $\phi^{(p,q)}(\mathbf{x})$ are the given basis functions. We note here that for our DG solvers, we are using the NEKTAR++ implementation given in [40] and available at `http://www.nektar.info`.

5.2 Higher Order Accuracy in DG Solutions

We remind the reader that the two-dimensional postprocessor has the following form:

$$u^\star(x, y) = \frac{1}{h_1 h_2}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} K^{v,\ell}\left(\frac{x_1 - x}{h_1}\right) K^{v,\ell}\left(\frac{x_2 - y}{h_2}\right) u_{h_1,h_2}(x_1, x_2)dx_1 dx_2.$$

In this section, we discuss the theoretical extension of the smoothness-increasing accuracy-conserving filter to two dimensions. We build on previous proofs in [19, 36] and account for a variable coefficient as well as the triangular mesh structure. This allows us to arrive at an estimate that demonstrates that the order accuracy of the DG solution can be improved from $\mathcal{O}(h^{k+1})$ to $\mathcal{O}(h^{2k+1})$ for structured triangular meshes such as that in Figure 5.1.

We begin by stating our main theorem:

Theorem 5.2.1 *Let u_h be the discontinuous Galerkin solution for the variable coefficient problem* (5.2)*, where $A_i(\mathbf{x}),\ i = 1, 2$ is bounded in the $L^\infty-$norm, and $u_h^\star = K_H^{\nu,\ell} \star u_h$, where $K_H^{\nu,\ell} = \frac{1}{H}K^{\nu,\ell}(\frac{\mathbf{x}}{H})$ is the position-dependent SIAC kernel given in* (3.19)*. Given sufficient smoothness in the initial data, we have that*

$$\|u - K_H^{\nu,\ell} \star u_h\|_\Omega \leq \mathsf{C}h^{2k+1}, \tag{5.7}$$

where C depends upon the smoothness of the solution as well as the boundedness of $A_i(\mathbf{x}), i = 1, 2,$ and $K_H^{\nu,\ell}$ a B-spline kernel as given in Equation (3.19)*.*

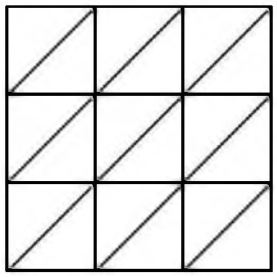


Figure 5.1. A structured triangular mesh.

In this statement, we see that it is possible to improve the order of accuracy of the approximation through the use of a B-spline kernel. This improvement is possible *provided the negative-order norm of the solution and the divided differences are of higher order.* Since we are considering the simplest case of structured triangular mesh, we have translation invariance of the mesh. It only remains to prove that we have higher order accuracy in the negative-order norm for the solution and the divided differences. To show this, we follow the proofs in [19, 36]. We note that in those proofs, the coefficients multiplying the convection term are constant. Here, we consider the variable-coefficient case.

Proof: We note that we can rewrite the estimate in Equation (5.7) as

$$\|u - K_H^{\nu,\ell\mathbf{1}} \star u_h\|_\Omega \le \|u - K_H^{\nu,\ell\mathbf{1}} \star u\|_\Omega + \|K_H^{\nu,\ell\mathbf{1}} \star (u - u_h)\|_\Omega. \tag{5.8}$$

The estimate for the first term on the right side was shown in Section 3.2. For the second term, we need to demonstrate that the divided differences of the error in a negative-order norm are of higher order for the variable coefficient equation solved over a structured triangular mesh. We show this by considering the inner product of the errors from the DG solution with a smooth function, $\Phi(x)$, that is the final time condition to the dual problem. We can define this dual problem by

$$\varphi_t + \sum_{i=1}^{2} A_i(\mathbf{x})\varphi_{x_i} = 0, \qquad (\mathbf{x}, t) \in \Omega \times (0, T], \tag{5.9}$$

$$\varphi(x, T) = \Phi(x), \tag{5.10}$$

with φ having periodic boundary conditions. Multiplying the equation for the exact solution, (5.2), by φ and the dual equation, (5.9), by the exact solution, we see that we have $\frac{d}{dt}(u, \varphi) = 0$. We can use this to begin estimating $((u - u_h)(T), \Phi)_\Omega$, which would become

$$\begin{aligned}((u-u_h)(T),\Phi)_\Omega =&(u-u_h,\varphi)_\Omega(0)-\int_0^T\{((u_h)_t,\varphi)_\Omega+(u_h,\varphi_t)_\Omega\}\,dt\\ =&(u-u_h,\varphi)_\Omega(0)-\int_0^T\{((u_h)_t,\varphi-\chi)_\Omega+B(\mathsf{A},u_h;\varphi-\chi)_\Omega\}\,dt\\ =&\Theta_1+\Theta_2,\end{aligned} \tag{5.11}$$

where $B(\mathsf{A},u_h;\varphi-\chi)$ is the bilinear form given in (5.5) and φ is the solution to the dual equation (5.9).

We note that the proof of Θ_1 being of higher order was given in [19] and is $|\Theta_1|\leq C_1h^{2k+2}\|u_o\|_{k+1}\|\varphi(0)\|_{k+1}$. The proof for Θ_2 uses $\chi=P\varphi$, the projection of φ, and Lipschitz continuity to obtain:

$$\begin{aligned}|\Theta_2|\leq&\left|-\int_0^T\{((u_h)_t,\varphi-\chi)_\Omega+B(\mathsf{A},u_h;\varphi-\chi)_\Omega\}\,dt\right|\leq\left|-\int_0^T\{B(\mathsf{A},u_h;\varphi-\chi)_\Omega\}\,dt\right|\\ \leq&\left|\sum_{i=1}^2\int_0^T\left\{(A_i(\mathbf{x})u_h(\mathbf{x},t),(\varphi-P\varphi)_{x_i})-\sum_{\tau_e}\int_{\partial\tau_e}A_i(\mathbf{x})\hat{u}_h\hat{n}_i(\varphi-P\varphi)\,ds\right\}dt\right|.\end{aligned} \tag{5.12}$$

Adding and subtracting the term $((A_i(x)u)_{x_i},\varphi-P\varphi)_\Omega$, and using simple inequalities gives the bound on Θ_2 as

$$|\Theta_2|\leq Ch^{2k+1}\left(\int_0^T\|u_h-u\|_\Omega^2\,dt\right)^{1/2}\left(\int_0^T\|\varphi\|_{k+1}^2\,dt\right)^{1/2}+Ch^{2k+2}\left(\int_0^T\|\varphi\|_{k+1}^2\,dt\right)^{1/2}. \tag{5.13}$$

For the negative-order norm of the solution, we then have the estimate

$$\|u-u_h\|_{-(k+1),\Omega}=\mathsf{C}\,h^{2k+1}, \tag{5.14}$$

where C depends on the smoothness of the solution and the bound on $\|\mathsf{A}\|_{L^\infty(\Omega)}$. We have obtained this by using the above bounds on $|\Theta_1|$, $|\Theta_2|$, as well as the definition of the negative-order norm together with the regularity for the variable coefficient equation, $\|\varphi(\mathbf{x},t)\|_{\ell,\Omega}\leq C\|\varphi(\mathbf{x},0)\|_{\ell,\Omega}$, where C is a constant that depends on $\|\mathsf{A}\|_{L^\infty(\Omega)}$.

We emphasize that the proof of the negative-order norm estimate of the DG solution did not rely on the mesh from the DG solution. The mesh type only plays a role in the estimation of the divided differences that help to bound the error. However, because we are assuming a structured triangular mesh, we maintain the translation invariance property. The constant now relies on the bound on the variable coefficient $A_i,\ i=1,2$. To show that this is indeed the case, we first define the DG method for the divided differences of the variable coefficient equation,

$$\partial_H^\alpha u_t + \sum_{i=1}^{2} \partial_H^\alpha (A_i(\mathbf{x})u)_{x_i} = 0, \qquad \mathbf{x} \in \mathbb{R}^2 \times (0,T], \tag{5.15}$$

$$\partial_H^\alpha u(\mathbf{x},0) = \partial_H^\alpha u_o(\mathbf{x}), \tag{5.16}$$

where α is the order of the divided difference, $\partial_H^\alpha = \partial_{H_1}^{\alpha_1}\partial_{H_2}^{\alpha_2}$ and

$$\partial_{H_j}^m v(x) = \frac{1}{H^m}\sum_{i=0}^{m}(-1)^i \binom{m}{i} v\left(x + \left(\frac{m}{2} - i\right) He_j\right).$$

The DG scheme is given by

$$(\partial_H^\alpha u_h, v_h)_\Omega + B_\alpha(\mathsf{A}, u_h; v_h)_\Omega = 0, \tag{5.17}$$

for all $v_h \in V_h$, where

$$B_\alpha(\mathsf{A}, u_h; v_h)_\Omega = -\sum_{i=1}^{2}\left(\partial_H^\alpha (A_i(\mathbf{x})u_h), \frac{\partial v_h}{\partial x_i}\right)_\Omega + \sum_{\tau_e}\sum_{i=1}^{2}\int_{\partial\tau_e} \partial_H^\alpha (A_i(\mathbf{x})\widehat{u_h}) v_h \widehat{n_i}\, ds. \tag{5.18}$$

For the dual equation, we need to factor in the divided differences of the error:

$$\partial_H^\alpha (\varphi_\alpha)_t + \sum_{i=1}^{2} A_i(\mathbf{x})\partial_H^\alpha (\varphi_\alpha)x_i = 0, \qquad (\mathbf{x},t) \in \Omega \times (0,T], \tag{5.19}$$

$$\varphi_\alpha(\mathbf{x},T) = \widetilde{\Phi}_\alpha(\mathbf{x}). \tag{5.20}$$

Defining the dual equation in this way gives the relation $\frac{d}{dt}(\partial_H^\alpha u, \varphi_\alpha)_\Omega = 0$. Having this relation is an important step in the proof of the higher order accuracy in the negative-order norms.

We can perform a similar rearrangement of terms as above to obtain

$$\begin{aligned}(\partial_H^\alpha (u-u_h)(T), \widetilde{\Phi}_\alpha)_\Omega =& (\partial_H^\alpha (u-u_h), \varphi_\alpha)_\Omega(0) \\ & - \int_0^T \{(\partial_H^\alpha (u_h)_t, \varphi_\alpha - \chi)_\Omega + B_\alpha(\mathsf{A}, \partial_H^\alpha u_h; \varphi_\alpha - \chi)_\Omega\}\, dt \\ & - \int_0^T \{(\partial_H^\alpha (u_h), (\varphi_\alpha)_t)_\Omega - B_\alpha(\mathsf{A}, \partial_H^\alpha u_h; \varphi_\alpha)_\Omega\}\, dt \\ =& d\Theta_1 + d\Theta_2 + d\Theta_3.\end{aligned} \tag{5.21}$$

Here, $d\Theta_1$, $d\Theta_2$, and $d\Theta_3$ are

$$d\Theta_1 = (\partial_H^\alpha (u-u_h), \varphi_\alpha)_\Omega(0) \tag{5.22}$$

$$d\Theta_2 = -\int_0^T \{(\partial_H^\alpha (u_h)_t, \varphi_\alpha - \chi)_\Omega + B_\alpha(\mathsf{A}, \partial_H^\alpha u_h; \varphi_\alpha - \chi)_\Omega\}\, dt \tag{5.23}$$

$$d\Theta_3 = -\int_0^T \{(\partial_H^\alpha (u_h), (\varphi_\alpha)_t)_\Omega - B_\alpha(\mathsf{A}, \partial_H^\alpha u_h; \varphi_\alpha)_\Omega\}\, dt. \tag{5.24}$$

The bounds on $d\Theta_1$ and $d\Theta_2$ provide a similar estimate to that appearing above and therefore, we only concentrate on bounding $d\Theta_3$. For the interior part of the integral of $d\Theta_3$, we have

$$\begin{aligned}
INT(d\Theta_3) =&(\partial_H^\alpha(u_h),(\varphi_\alpha)_t)_\Omega - B_\alpha(\mathsf{A},\partial_H^\alpha u_h;\varphi_\alpha)_\Omega \\
&= (-1)^\alpha(u_h,\partial_H^\alpha(\varphi_\alpha)_t)_\Omega + \sum_{i=1}^{2}(\partial_H^\alpha(A_i(\mathbf{x})u_h),(\varphi_\alpha)_{x_i})_\Omega \\
&-\sum_{\tau_e}\sum_{i=1}^{2}\int_{\partial\tau_e}\partial_H^\alpha(A_i(\mathbf{x})\widehat{u_h})\varphi_\alpha\widehat{n}_i\,ds \\
&= (-1)^\alpha(u_h,\partial_H^\alpha(\varphi_\alpha)_t)_\Omega + \sum_{i=1}^{2}(-1)^\alpha(u_h,A_i(\mathbf{x})\partial_H^\alpha(\varphi_\alpha)_{x_i})_\Omega = 0. \qquad (5.25)
\end{aligned}$$

Combining the estimates and using the regularity of φ, we then have

$$(\partial_H^\alpha(u-u_h)(T),\widetilde{\Phi})_\Omega \le Ch^{2k+1}, \qquad (5.26)$$

proving our theorem.

We point out that the negative-order norm estimates for the solution did not rely on the mesh type, and that in fact, we do see higher order convergence; however, the divided differences do. In order to extend this to unstructured meshes, we will need to focus our attention on improving the estimates for the divided differences of the error. The proofs for the accuracy extracting capabilities of the SIAC filter do not change.

5.3 Results

The main contribution of this section is the demonstration of the effectiveness of the quadrilateral B-spline postprocessor applied to discontinuous Galerkin solutions calculated over a structured triangular mesh and its accuracy enhancing capabilities. Furthermore, we present the results for the various choices of quadrature to see the computational effect of the numerical crimes committed in our integration. We examine both the L^2- and $L^\infty-$errors, but note that the theory presented in this paper only applies to the L^2-errors. We note that all examples have been implemented using NEKTAR++ for the discontinuous Galerkin solution to the partial differential equation under consideration. The classical Runge-Kutta 4 time-stepping scheme was used for the time discretization and the cfl was taken so that spatial errors dominate. We note that this restriction is not necessary in practical applications - the errors will still be improved over the DG errors. However, in order to see the higher rate of convergence, it is necessary to have such a restriction. For the consistent integration approach, we always use exact integration to calculate the L_2

projections, i.e., the number of quadrature points is what is required for exact integration. However, when performing integration considering only the DG mesh, we increase the number of quadrature points. In order to isolate the applicability of this filter, we assume that we have periodic boundary conditions for our test cases to simplify the application of the postprocessor so that it is only necessary to implement the symmetric filter with $r + 1 = 2k + 1$ B-splines. In addition, Gauss-Lobatto-Legendre (GLL) points are used in one direction, and Gauss-Radau-Legendre (GRL) points are used on the other. In all examples, we calculate the error for six quadrature points on each element, *i.e.*, three GLL points on one direction and two GRL points on the other.

5.3.1 Constant Coefficient Linear Advection Equation

For this example, we consider solutions of the equation

$$u_t + u_x + u_y = 0, \qquad (x, y) \in (0, 1) \times (0, 1), \qquad T = 12.5 \tag{5.27}$$

with initial condition $u(0, x, y) = \sin(2\pi(x + y))$.

Table 5.1 shows the errors when postprocessing using inexact quadrature that only respects the DG element breaks and postprocessing on a mesh that respects both kernel breaks and DG element breaks, for a uniformly structured triangular mesh. For this case, we clearly see improvements from $k{+}1$ to $2k{+}1$ in the order of the errors. Furthermore, we see a decrease in the magnitude of the errors. We note that the errors for inexact postprocessing on the DG mesh are represented for two different sets of quadrature points. The first two sets of errors are for inexact integration, where only the DG element breaks are respected. The second set of errors for inexact quadrature demonstrates our attempt to overcome the numerical crimes with increased quadrature points. These sets of quadrature points are designated "inexact postprocesssing" and the number of quadrature points are given in the tables. We see that even in the first set of quadrature points, the results of the DG mesh are similar to the results of the consistent-integration approach. In addition, increasing the number of quadrature points results in decreased error and improved order of convergence. The final set of errors is what is required to calculate the integrals exactly when applying the postprocessor on the consistent integration mesh. In the tables, we designate this "consistent postprocessing." We see that for this set of errors, we obtain the desired order accuracy as given by the theory in Equation (5.7), except for $\mathbb{P}^4$ and the finest mesh, where we expect that roundoff errors have begun to dominate.

The errors for the smoothly varying triangular mesh shown in Figure 5.2 are provided in Table 5.2. We note that technically, the theory does not cover this case. We see that

Table 5.1. Errors for the linear constant coefficient advection equation using $\mathbb{P}^2$, $\mathbb{P}^3$, and $\mathbb{P}^4$ polynomials for the structured (uniform) triangular mesh. Before postprocessing and after postprocessing on the DG mesh, where Q_0 represents GLL points on one direction and Q_1 represents GRL points on the other direction, CI indicates consistent integration.

mesh	L^2 error	order	L^∞ error	order	L^2 error	order	L^∞ error	order
	$\mathbb{P}^2$							
	Before postprocessing				Inexact postprocessing, $Q0 = 5, Q1 = 4$			
$10^2 \times 2$	6.86E-03	–	3.27E-02	–	2.32E-03	–	3.28E-03	–
$20^2 \times 2$	9.86E-04	2.78	4.35E-03	2.91	7.15E-05	5.02	1.02E-04	5.01
$40^2 \times 2$	1.36E-04	2.86	5.45E-04	3.00	2.22E-06	5.01	3.21E-06	4.99
	Inexact postprocessing, $Q0 = 34, Q1 = 33$				Consistent postprocessing, CI			
$10^2 \times 2$	2.32E-03	–	3.28E-03	–	2.32E-03	–	3.28E-03	–
$20^2 \times 2$	7.14E-05	5.02	1.01E-04	5.02	7.14E-05	5.02	1.01E-04	5.02
$40^2 \times 2$	2.18E-06	5.03	3.09E-06	5.03	2.18E-06	5.03	3.09E-06	5.03
	$\mathbb{P}^3$							
	Before postprocessing				Inexact postprocessing, $Q0 = 6, Q1 = 6$			
$10^2 \times 2$	4.95E-04	–	2.60E-03	–	4.58E-05	–	6.52E-05	–
$20^2 \times 2$	2.44E-05	4.34	1.74E-04	3.90	2.68E-07	7.42	4.21E-07	7.27
$40^2 \times 2$	2.01E-06	3.60	1.11E-05	3.97	3.17E-09	6.40	6.70E-09	5.97
	Inexact postprocessing, $Q0 = 45, Q1 = 44$				Consistent postprocessing, CI			
$10^2 \times 2$	4.54E-05	–	6.42E-05	–	4.54E-05	–	6.42E-05	–
$20^2 \times 2$	2.44E-07	7.54	3.51E-07	7.51	2.44E-07	7.54	3.51E-07	7.51
$40^2 \times 2$	1.41E-09	7.44	2.65E-09	7.05	1.41E-09	7.44	2.65E-09	7.05
	$\mathbb{P}^4$							
	Before postprocessing				Inexact postprocessing, $Q0 = 8, Q1 = 7$			
$10^2 \times 2$	3.62E-05	–	2.05E-04	–	4.02E-06	–	5.69E-06	–
$20^2 \times 2$	1.14E-06	4.99	6.47E-06	4.99	5.81E-09	9.44	8.27E-09	9.43
$40^2 \times 2$	3.56E-08	5.00	2.00E-07	5.02	2.30E-10	4.66	3.27E-10	4.66
	Inexact postprocessing, $Q0 = 56, Q1 = 55$				Consistent postprocessing, CI			
$10^2 \times 2$	4.02E-06	–	5.68E-06	–	4.02E-06	–	5.68E-06	–
$20^2 \times 2$	5.84E-09	9.43	9.35E-09	9.25	5.84E-09	9.43	9.35E-09	9.25
$40^2 \times 2$	2.31E-10	4.66	4.74E-10	4.30	2.31E-10	4.66	4.74E-10	4.30

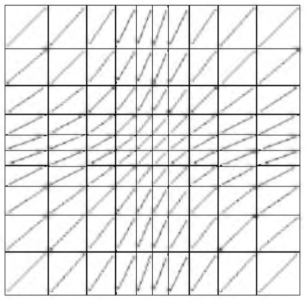


Figure 5.2. Smoothly varying triangular mesh.

Table 5.2. Errors for the linear constant coefficient advection equation using $\mathbb{P}^2$, $\mathbb{P}^3$, and $\mathbb{P}^4$ polynomials for the smoothly varying triangular mesh. Before postprocessing and after postprocessing on the DG mesh, where Q_0 represents GLL points on one direction and Q_1 represents GRL points on the other direction, CI indicates consistent integration.

$\mathbb{P}^2$								
mesh	L^2 error	order	L^∞ error	order	L^2 error	order	L^∞ error	order
	Before postprocessing				Inexact postprocessing, $Q0 = 5, Q1 = 4$			
$10^2 \times 2$	1.11E-02	–	9.15E-02	–	6.32E-03	–	1.21E-02	–
$20^2 \times 2$	1.37E-03	3.01	1.31E-02	2.80	5.97E-04	3.40	3.75E-03	1.69
$40^2 \times 2$	1.71E-04	3.00	1.66E-03	2.98	2.53E-04	1.24	1.60E-03	1.23
	Inexact postprocessing, $Q0 = 34, Q1 = 33$				Consistent postprocessing, CI			
$10^2 \times 2$	6.17E-03	–	1.14E-02	–	6.17E-03	–	1.14E-02	–
$20^2 \times 2$	1.95E-04	4.98	5.53E-04	4.37	1.96E-04	4.98	5.49E-04	4.38
$40^2 \times 2$	7.02-E06	4.80	6.04E-05	3.19	7.08E-06	4.78	6.26E-05	3.14
$\mathbb{P}^3$								
	Before postprocessing				Inexact postprocessing, $Q0 = 6, Q1 = 6$			
$10^2 \times 2$	1.22E-03	–	1.05E-02	–	3.81E-04	–	9.88E-04	–
$20^2 \times 2$	7.82E-05	3.96	7.49E-04	3.81	5.00-E05	2.93	2.64E-04	1.90
$40^2 \times 2$	4.97E-06	3.98	4.75E-05	3.98	2.37e-05	1.08	1.07E-04	1.30
	Inexact postprocessing, $Q0 = 45, Q1 = 44$				Consistent postprocessing, CI			
$10^2 \times 2$	3.45E-04	–	9.37E-04	–	3.45E-04	–	9.37E-04	–
$20^2 \times 2$	1.86E-06	7.54	4.83E-06	7.60	1.86E-06	7.54	4.83E-06	7.60
$40^2 \times 2$	1.24E-08	7.23	1.02E-07	5.57	1.24E-08	7.23	1.02E-07	5.57
$\mathbb{P}^4$								
	Before postprocessing				Inexact postprocessing, $Q0 = 8, Q1 = 7$			
$10^2 \times 2$	1.11E-04	–	1.34E-03	–	7.31E-05	–	2.13E-04	–
$20^2 \times 2$	3.66E-06	4.92	4.73E-05	4.82	2.83E-06	4.69	1.63E-05	3.71
$40^2 \times 2$	1.17E-07	4.97	1.52E-06	4.96	1.87E-06	0.60	1.38E-05	0.24
	Inexact postprocessing, $Q0 = 56, Q1 = 55$				Consistent postprocessing, CI			
$10^2 \times 2$	7.25E-05	–	2.13E-04	–	7.25E-05	–	2.13E-04	–
$20^2 \times 2$	9.55E-08	9.57	2.76E-07	9.59	9.55E-08	9.57	2.76E-07	9.59
$40^2 \times 2$	3.70E-09	4.69	6.28E-09	5.46	3.70E-09	4.69	6.28E-09	5.46

errors have decreased and the order of convergence in the L^2-norm is generally better when postprocessing on the consistent integration mesh and improves to approximately $2k+1$. For the smoothly varying structured triangular mesh, when using quartic polynomials and the finest mesh, the error values improved compared to the initial solution; however, the convergence rate decreased. Using the smoothly varying triangular mesh, more quadrature points are required to simulate the exact results. Furthermore, Figures 5.3 and 5.4 depict the point-wise errors in logarithmic scale when using quadratic polynomials over a $20^2 \times 2$ mesh. From these plots, we observe that the error has decreased for the postprocessed solution, and the postprocessor has filtered out the oscillations. We note that the error plots for the inexact integration give the same results as for the consistent integration and therefore, we neglect to include these.

5.3.2 Variable Coefficient Linear Advection Equation

For this example, we consider solutions of the equation

$$u_t + (au)_x + (au)_y = f, \qquad (x, y) \in (0,1) \times (0,1), \qquad T = 12.5. \tag{5.28}$$

We implement a smooth coefficient $a(x,y) = 2 + \sin(2\pi(x+y))$, with an initial condition of $u(x,y,0) = \sin(2\pi(x+y))$. Periodic boundary conditions are implemented in both directions and the forcing function, $f(x,y,t)$, is chosen so that the exact solution is $u(x,y,t) = \sin(2\pi(x+y-2t))$. By examining the results of this equation, we can gain insight into how effective the filter might be for nonlinear equations.

Table 5.3 shows the errors when postprocessing on both the consistent-integration mesh and the DG mesh for the structured triangular mesh. There is clear improvement from $k+1$ to approximately $2k$ for $\mathbb{P}^2$ and $\mathbb{P}^3$ polynomials. For $\mathbb{P}^4-$polynomials, the improvement is diminished, but the magnitude of the errors improves significantly, especially for fine-mesh structures. The errors for the smoothly varying triangular mesh are similar, and provided in Table 5.4.

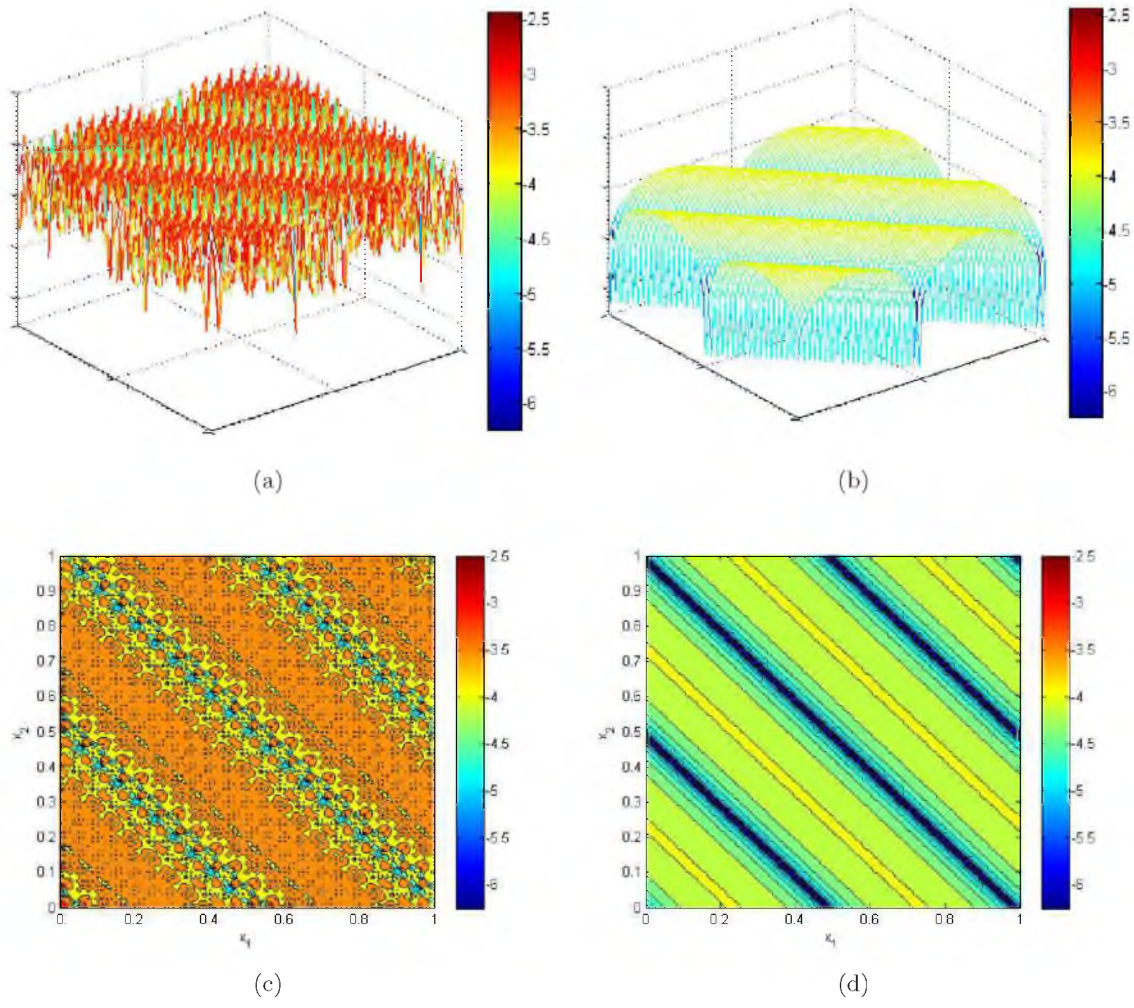


Figure 5.3. Three- and two-dimensional view of point-wise errors in logarithmic scale for constant coefficient advection equation when $\mathbb{P}^2$ discontinuous Galerkin method is used over a structured (uniform) triangular mesh. (a) and (c) demonstrate the initial DG approximation errors; (b) and (d) represent the errors after the application of the postprocessor on the consistent-integration mesh. Filled contour plots have been used to visualize the data for (c) and (d). The SIAC filter works to reduce the oscillations in the error.

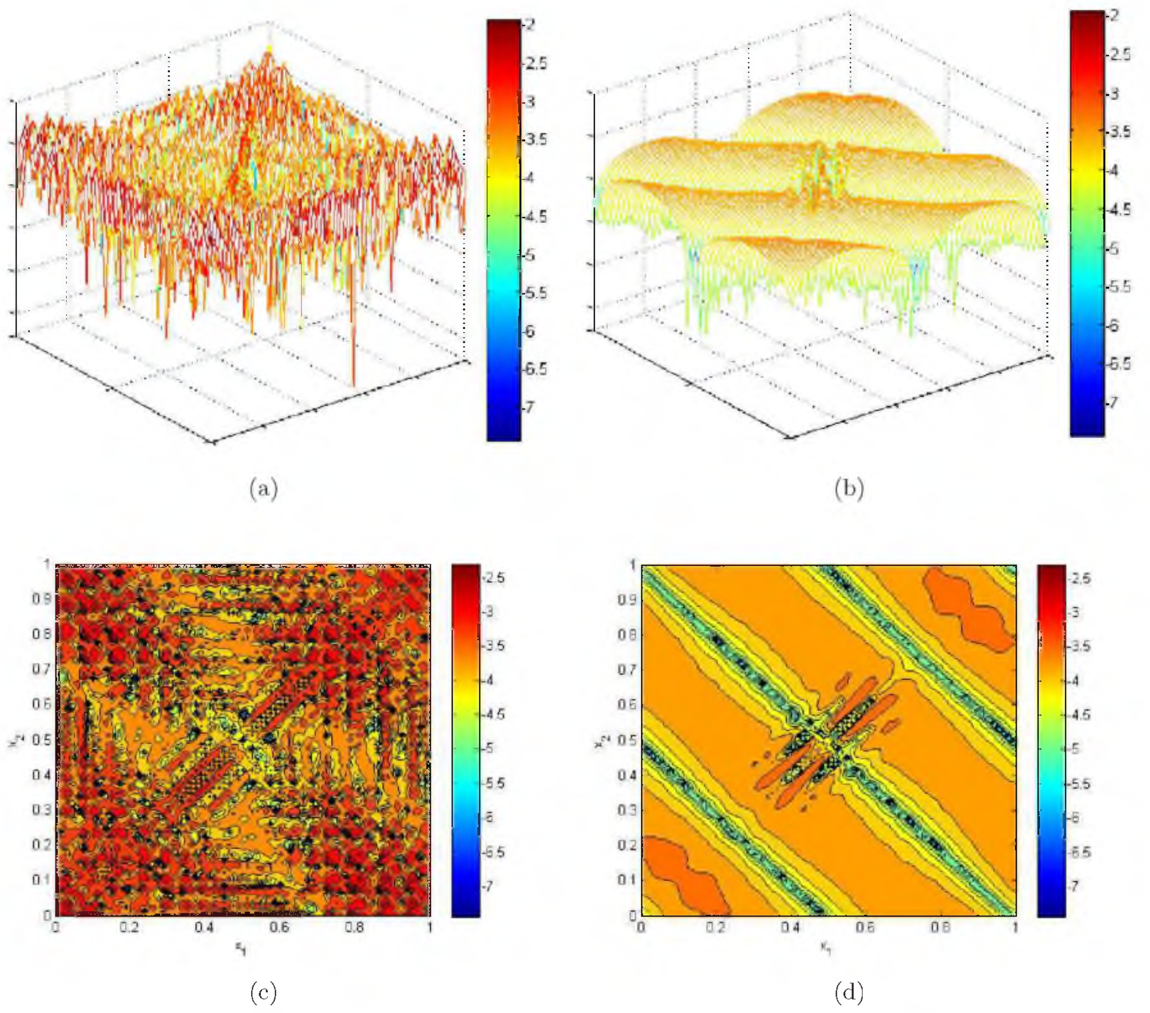


Figure 5.4. Three- and two-dimensional view of point-wise errors in logarithmic scale for constant coefficient advection equation when a $\mathbb{P}^2$ discontinuous Galerkin method is used over a smoothly varying triangular mesh. (a) and (c) demonstrate the initial DG approximation errors and (b) and (d) represent the errors after the application of the postprocessor on the consistent-integration mesh. We can clearly see how the SIAC filter reduces the oscillations in the error.

Table 5.3. Errors for variable coefficient advection equation using $\mathbb{P}^2$, $\mathbb{P}^3$, and $\mathbb{P}^4$ polynomials. Before postprocessing and after postprocessing on the consistent-integration (CI) and DG meshes. We have assumed a structured (uniform) triangular mesh.

$\mathbb{P}^2$								
mesh	L^2 error	order	L^∞ error	order	L^2 error	order	L^∞ error	order
	Before postprocessing				After postprocessing, $Q0 = 5, Q1 = 4$			
$10^2 \times 2$	6.16E-03	–	3.39E-02	–	2.11E-04	–	4.19E-04	
$20^2 \times 2$	7.43E-04	3.05	4.11E-03	3.04	5.68E-06	5.22	1.54E-05	4.77
$40^2 \times 2$	9.13E-05	3.02	4.98E-04	3.04	3.98E-07	3.84	1.05E-06	3.87
	After postprocessing, $Q0 = 34, Q1 = 33$				After postprocessing, CI			
$10^2 \times 2$	2.09E-04	–	4.05E-04	–	2.09E-04	–	4.05E-04	
$20^2 \times 2$	4.89E-06	5.42	9.66E-06	5.39	4.89E-06	5.42	9.66E-06	5.39
$40^2 \times 2$	1.74E-07	4.81	3.87E-07	4.64	1.74E-07	4.81	3.87E-07	4.64
$\mathbb{P}^3$								
	Before postprocessing				After postprocessing, $Q0 = 6, Q1 = 6$			
$10^2 \times 2$	5.50E-04	–	3.34E-03	–	3.11E-05	–	4.75E-05	
$20^2 \times 2$	3.23E-05	4.09	2.00E-04	4.06	1.55E-07	7.65	2.98E-07	7.32
$40^2 \times 2$	2.05E-06	3.98	1.20E-05	4.06	2.39E-09	6.02	5.64E-09	5.72
	After postprocessing, $Q0 = 45, Q1 = 44$				After postprocessing, CI			
$10^2 \times 2$	3.08E-05	–	4.66E-05	–	3.08E-05	–	4.66E-05	
$20^2 \times 2$	1.55E-07	7.63	2.98E-07	7.29	1.55E-07	7.63	2.98E-07	7.29
$40^2 \times 2$	2.39E-09	6.01	5.64E-09	5.72	2.39E-09	6.01	5.64E-09	5.72
$\mathbb{P}^4$								
	Before postprocessing				After postprocessing, $Q0 = 8, Q1 = 7$			
$10^2 \times 2$	4.34E-05	–	2.82E-04	–	3.95E-06	–	5.61E-06	
$20^2 \times 2$	1.20E-06	5.18	7.75E-06	5.19	4.44E-09	9.80	7.78E-09	9.49
$40^2 \times 2$	3.55E-08	5.08	2.20E-07	5.14	1.19E-10	5.22	2.73E-10	4.83
	After postprocessing, $Q0 = 56, Q1 = 55$				After postprocessing, CI			
$10^2 \times 2$	3.95E-06	–	5.61E-06	–	3.95E-06	–	5.61E-06	
$20^2 \times 2$	4.43E-09	9.80	8.30E-09	9.40	4.43E-09	9.80	8.30E-09	9.40
$40^2 \times 2$	1.19E-10	5.22	4.65E-10	4.16	1.19E-10	5.22	4.65E-10	4.16

Table 5.4. Errors for variable coefficient advection equation using $\mathbb{P}^2$, $\mathbb{P}^3$, and $\mathbb{P}^4$ polynomials for the smoothly varying triangular mesh. Before postprocessing and after postprocessing on the consistent-integration (CI) and DG meshes.

			$\mathbb{P}^2$					
mesh	L^2 error	order	L^∞ error	order	L^2 error	order	L^∞ error	order
	Before postprocessing				After postprocessing, $Q0 = 5, Q1 = 4$			
$10^2 \times 2$	1.19E-02	–	9.97E-02	–	2.78E-03	–	7.67E-03	–
$20^2 \times 2$	1.40E-03	3.09	1.32E-02	2.92	5.67E-04	2.29	3.25E-03	2.36
$40^2 \times 2$	1.70E-04	3.04	1.66E-03	2.99	2.54E-04	1.16	1.59E-03	1.03
	After postprocessing, $Q0 = 34, Q1 = 33$				After postprocessing, CI			
$10^2 \times 2$	2.63E-03	–	6.83E-03	–	2.63E-03	–	6.83E-03	–
$20^2 \times 2$	8.77E-05	4.91	3.95E-04	4.11	8.77E-05	4.91	3.95E-04	4.11
$40^2 \times 2$	4.17E-06	4.39	4.55E-05	3.12	4.03E-06	4.44	4.55E-05	3.12
			$\mathbb{P}^3$					
	Before postprocessing				After postprocessing, $Q0 = 6, Q1 = 6$			
$10^2 \times 2$	1.30E-03	–	1.22E-02	–	3.59E-04	–	1.01E-03	–
$20^2 \times 2$	7.81E-05	4.06	8.48E-04	3.85	4.97E-05	2.85	2.63E-04	1.94
$40^2 \times 2$	4.91E-06	3.99	4.96E-05	4.10	2.37E-05	2.10	1.02E-04	1.37
	After postprocessing, $Q0 = 45, Q1 = 44$				After postprocessing, CI			
$10^2 \times 2$	3.30E-04	–	9.99E-04	–	3.30E-04	–	9.99E-04	–
$20^2 \times 2$	2.19E-06	7.24	1.55E-05	6.01	2.19E-06	7.24	1.55E-05	6.01
$40^2 \times 2$	5.86E-08	5.22	8.26E-07	4.23	5.86E-08	5.22	8.26E-07	4.23
			$\mathbb{P}^4$					
	Before postprocessing				After postprocessing, $Q0 = 8, Q1 = 7$			
$10^2 \times 2$	1.48E-04	–	1.57E-03	–	7.36E-05	–	2.16E-04	–
$20^2 \times 2$	4.02E-06	5.20	5.01E-05	4.97	2.83E-06	4.70	1.59E-05	3.76
$40^2 \times 2$	1.17E-07	5.10	1.54E-06	5.02	1.87E-06	0.60	1.37E-05	0.21
	After postprocessing, $Q0 = 56, Q1 = 55$				After postprocessing, CI			
$10^2 \times 2$	7.30E-05	–	2.16E-04	–	7.30E-05	–	2.16E-04	–
$20^2 \times 2$	1.22E-07	9.22	5.84E-07	8.53	1.22E-07	9.22	5.84E-07	8.53
$40^2 \times 2$	1.25E-09	6.61	1.26E-08	5.53	1.25E-09	6.61	1.26E-08	5.53

5.3.3 Two-Dimensional Wave Equation as a System

We now consider the traditional second-order wave equation.

$$\eta_{tt} - \eta_{xx} - \eta_{yy} = 0, \qquad (x, y) \in (0, 1) \times (0, 1), \qquad T = 6.28. \tag{5.29}$$

We rewrite Equation (5.29) as a system of first-order linear equations shown below

$$\begin{aligned} \eta_t + u_x + v_y &= 0 \\ u_t + \eta_x &= 0 \\ v_t + \eta_y &= 0 \end{aligned} \tag{5.30}$$

with initial conditions

$$\begin{aligned} \eta(x, y, 0) &= 0.01 \times (\sin(2\pi x) + \sin(2\pi y)) \\ u(x, y, 0) &= 0.01 \times (\sin(2\pi x)) \\ v(x, y, 0) &= 0.01 \times (\sin(2\pi y)) \end{aligned} \tag{5.31}$$

and 2π periodic boundary conditions in both directions. In Tables 5.5 and 5.6, we have provided the errors after one period in time for the η variable. As it is presented, the value of the error and the order of convergence are improved after the application of the postprocessor. For both the structured triangular mesh and the smoothly varying mesh, this is the expected improvement to $\mathcal{O}(h^{2k+1})$.

Table 5.5. Errors for 2D system using $\mathbb{P}^2$, $\mathbb{P}^3$, and $\mathbb{P}^4$ polynomials. Before postprocessing and after postprocessing on the consistent-integration (CI) and DG meshes. We have assumed a structured (uniform) triangular mesh.

$\mathbb{P}^2$								
mesh	L^2 error	order	L^∞ error	order	L^2 error	order	L^∞ error	order
	Before postprocessing				After postprocessing, $Q0 = 5, Q1 = 4$			
$10^2 \times 2$	1.24E-05	–	6.14E-05	–	2.35E-06	–	4.78E-06	–
$20^2 \times 2$	1.56E-06	2.99	7.97E-06	2.95	5.42E-08	5.44	1.22E-07	5.29
$40^2 \times 2$	1.95E-07	3.00	1.01E-06	2.98	3.85E-09	3.82	9.42E-09	3.70
	After postprocessing, $Q0 = 34, Q1 = 33$				After postprocessing, CI			
$10^2 \times 2$	2.33E-06	–	4.66E-06	–	2.33e-06	–	4.66e-06	–
$20^2 \times 2$	4.48E-08	5.70	9.03E-08	5.69	4.48e-08	5.70	9.03e-08	5.69
$40^2 \times 2$	9.17E-10	5.61	1.85E-09	5.61	9.17E-10	5.61	1.85E-09	5.61
$\mathbb{P}^3$								
	Before postprocessing				After postprocessing, $Q0 = 6, Q1 = 6$			
$10^2 \times 2$	6.58E-07	–	3.36E-06	–	2.31E-07	–	4.70E-07	–
$20^2 \times 2$	4.12E-08	4.00	2.09E-07	4.01	1.17E-09	7.63	2.69E-09	7.45
$40^2 \times 2$	2.58E-09	4.00	1.32E-08	3.98	2.13E-11	5.78	5.51E-11	5.61
	After postprocessing, $Q0 = 45, Q1 = 44$				After postprocessing, CI			
$10^2 \times 2$	2.29E-07	–	4.58E-07	–	2.29E-07	–	3.36E-07	–
$20^2 \times 2$	9.80E-10	7.87	1.96E-09	7.87	9.80E-10	7.87	1.96E-09	7.42
$40^2 \times 2$	4.02E-12	7.93	8.05E-12	7.93	4.03E-12	7.93	1.37E-11	7.16
$\mathbb{P}^4$								
	Before postprocessing				After postprocessing, $Q0 = 8, Q1 = 7$			
$10^2 \times 2$	2.44E-08	–	1.29E–07	–	2.80E-08	–	5.61E-08	–
$20^2 \times 2$	7.64E-10	5.00	4.16E-09	4.95	3.10E-11	9.82	6.22E-11	9.82
$40^2 \times 2$	2.40E-11	4.99	1.32E-10	4.98	8.07E-13	5.26	1.64E-12	5.25
	After postprocessing, $Q0 = 56, Q1 = 55$				After postprocessing, CI			
$10^2 \times 2$	2.80E-08	–	5.61E-08	–	2.80E-08	–	5.61E-08	–
$20^2 \times 2$	3.10E-11	9.82	6.19E-11	9.82	3.10E-11	9.82	6.19E-11	9.82
$40^2 \times 2$	8.17E-13	5.24	1.78E-12	5.12	8.17E-13	5.24	1.78E-12	5.24

Table 5.6. Errors for 2D system using $\mathbb{P}^2$, $\mathbb{P}^3$, and $\mathbb{P}^4$ polynomials for the smoothly varying triangular mesh. Before postprocessing and after postprocessing on the consistent-integration (CI) and DG meshes.

$\mathbb{P}^2$								
mesh	L^2 error	order	L^∞ error	order	L^2 error	order	L^∞ error	order
	Before postprocessing				After postprocessing, $Q0 = 5, Q1 = 4$			
$10^2 \times 2$	2.79E-05	–	1.81E-04	–	1.61E-05	–	9.08E-05	–
$20^2 \times 2$	3.57E-06	2.97	2.60E-05	2.80	8.14E-06	0.98	5.37E-05	0.76
$40^2 \times 2$	4.48E-07	2.99	3.37E-06	2.95	3.59E-06	1.18	2.59E-05	1.05
	After postprocessing, $Q0 = 34, Q1 = 33$				After postprocessing, CI			
$10^2 \times 2$	9.92E-06	–	3.14E-05	–	9.92E-06	–	3.14E-05	–
$20^2 \times 2$	1.90E-07	5.71	5.30E-07	5.89	1.92E-07	5.69	5.46E-07	5.85
$40^2 \times 2$	1.48E-08	3.68	9.27E-08	2.52	3.85E-09	5.64	2.72E-08	4.33
$\mathbb{P}^3$								
	Before postprocessing				After postprocessing, $Q0 = 6, Q1 = 6$			
$10^2 \times 2$	1.70E-06	–	8.97E-06	–	2.67E-06	–	8.35E-06	–
$20^2 \times 2$	1.09E-07	3.96	6.38E-07	3.81	7.07E-07	1.92	4.20E-06	0.99
$40^2 \times 2$	6.85E-09	3.99	4.11E-08	3.96	3.55E-07	0.99	1.85E-06	1.18
	After postprocessing, $Q0 = 45, Q1 = 44$				After postprocessing, CI			
$10^2 \times 2$	1.95E-06	–	6.84E-06	–	1.95E-06	–	6.84E-06	–
$20^2 \times 2$	8.93E-09	7.77	2.84E-08	7.91	9.01E-09	7.76	2.84E-08	7.91
$40^2 \times 2$	1.11E-10	6.33	5.55E-10	5.68	3.69E-11	7.93	1.06E-10	8.07
$\mathbb{P}^4$								
	Before postprocessing				After postprocessing, $Q0 = 8, Q1 = 7$			
$10^2 \times 2$	1.21E-07	–	8.29E-07	–	4.72E-07	–	1.70E-06	–
$20^2 \times 2$	3.87E-09	4.97	3.04E-08	4.77	4.34E-08	3.44	2.72E-07	2.64
$40^2 \times 2$	1.22E-10	4.99	9.88E-10	4.94	2.70E-08	1.61	2.25E-07	0.27
	After postprocessing, $Q0 = 56, Q1 = 55$				After postprocessing, CI			
$10^2 \times 2$	4.58E-07	–	1.72E-06	–	4.58E-07	–	1.72E-06	–
$20^2 \times 2$	5.58E-10	9.68	1.86E-09	9.85	5.58E-10	9.68	1.86E-09	9.85
$40^2 \times 2$	7.67E-13	9.51	2.98E-12	9.29	5.79E-13	9.91	1.77E-12	10.04

5.3.4 Constant Coefficient Linear Advection Equation Over Union-Jack Mesh

We conclude this section by demonstrating the application of the SIAC filter on the solutions of the linear advection equation (Equation (8.7)) over a different type of triangular mesh, known as the criss-cross mesh, shown in Figure 5.5. We consider

$$u_t + u_x + u_y = 0, \quad (x, y) \in (0,1) \times (0,1), \quad T = 6.28 \tag{5.32}$$

with initial condition $u(0, x, y) = \sin(2\pi(x + y))$ and periodic boundary conditions. Tables 5.7 and 5.8 present the error values for the uniform and smoothly varying criss-cross meshes after one period in time. The error plots for the uniform structured mesh are displayed in Figure 5.6. For quartic polynomials and the finest mesh, we see that the error values improved but the convergence rate decreased. In addition, in the smoothly varying mesh case, the L^2-error value for the quartic polynomial and the coarsest mesh has increased and this is because the kernel support covers the entire area of the mesh. In general, however, we see that the error values and the convergence rate have improved. Moreover, the inexact approach when using enough quadrature points yields similar results to the exact scheme.

5.4 Summary and Conclusions

The most pressing issue in accuracy enhancement is to formulate a suitable technique for extracting extra accuracy from the discontinuous Galerkin solution solved over an unstructured triangular mesh. In this chapter, we make a significant contribution to addressing this problem by demonstrating that there is a direct extension of the theory to structure triangular meshes as well as investigating the performance of the smoothness-increasing accuracy-conserving filter to two-dimensional hyperbolic equations solved over structured

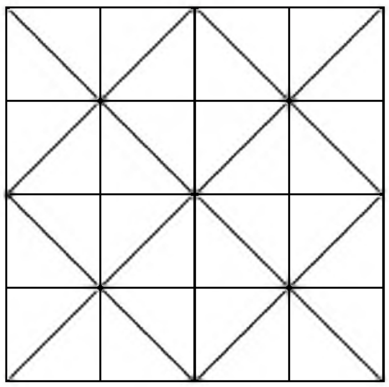


Figure 5.5. Union-Jack mesh.

Table 5.7. Errors for constant coefficient advection equation over a uniformly structured criss-cross mesh using $\mathbb{P}^2$, $\mathbb{P}^3$, and $\mathbb{P}^4$ polynomials. Before postprocessing and after postprocessing on the consistent-integration (CI) and DG meshes.

$\mathbb{P}^2$								
mesh	L^2 error	order	L^∞ error	order	L^2 error	order	L^∞ error	order
	Before postprocessing				After postprocessing, $Q0 = 5, Q1 = 4$			
$10^2 \times 2$	2.76E-03	–	1.78E-02	–	1.14E-03	–	3.61E-03	–
$20^2 \times 2$	3.47E-04	2.99	2.31E-03	2.94	6.74E-04	0.75	2.72E-03	0.40
$40^2 \times 2$	4.35E-05	2.99	3.03E-04	2.93	6.76E-04	-0.004	2.83E-03	-0.05
	After postprocessing, $Q0 = 34, Q1 = 33$				After postprocessing, CI			
$10^2 \times 2$	8.36E-04	–	1.30E-03	–	8.36E-04	–	1.30E-03	–
$20^2 \times 2$	2.88E-05	4.86	5.55E-05	4.55	2.87E-05	4.86	5.55E-05	4.55
$40^2 \times 2$	2.43E-06	3.57	5.05E-06	3.46	2.36E-06	3.60	4.63E-06	3.58
$\mathbb{P}^3$								
	Before postprocessing				After postprocessing, $Q0 = 6, Q1 = 6$			
$10^2 \times 2$	2.44E-04	–	1.59E-03	–	4.22E-05	–	1.12E-04	–
$20^2 \times 2$	1.54E-05	3.98	1.03E-04	3.94	2.61E-05	0.69	6.82E-05	0.72
$40^2 \times 2$	9.72E-07	3.98	6.44E-06	3.99	2.55E-05	0.03	7.12E-05	-0.06
	After postprocessing, $Q0 = 65, Q1 = 64$				After postprocessing, CI			
$10^2 \times 2$	3.57E-05	–	5.48E-05	–	3.57E-05	–	5.49E-05	–
$20^2 \times 2$	2.40E-07	7.22	4.99E-07	6.78	2.40E-07	7.22	4.99E-07	6.78
$40^2 \times 2$	9.30E-09	4.69	1.83E-08	4.78	9.97E-09	4.60	1.83E-08	4.78
$\mathbb{P}^4$								
	Before postprocessing				After postprocessing, $Q0 = 8, Q1 = 7$			
$10^2 \times 2$	1.48E-05	–	1.12E-04	–	3.98E-06	–	6.36E-06	–
$20^2 \times 2$	4.71E-07	4.97	3.40E-06	5.04	3.18E-07	3.65	1.19E-06	2.42
$40^2 \times 2$	1.47E-08	5.00	1.07E-07	4.99	3.09E-07	0.04	1.23E-06	-0.04
	After postprocessing, $Q0 = 56, Q1 = 55$				After postprocessing, CI			
$10^2 \times 2$	3.98E-06	–	5.68E-06	–	3.98E-06	–	5.68E-06	–
$20^2 \times 2$	4.95E-09	9.64	8.08E-09	9.45	4.95E-09	9.64	8.08E-09	9.45
$40^2 \times 2$	1.84E-09	1.43	2.65E-09	1.61	1.84E-09	1.43	2.65E-09	1.61

Table 5.8. Errors for constant coefficient advection equation over a criss-cross mesh using $\mathbb{P}^2$, $\mathbb{P}^3$, and $\mathbb{P}^4$ polynomials. Before postprocessing and after postprocessing on the consistent-integration (CI) and DG meshes. We have assumed a smoothly varying criss-cross mesh.

			$\mathbb{P}^2$					
mesh	L^2 error	order	L^∞ error	order	L^2 error	order	L^∞ error	order
	Before postprocessing				After postprocessing, $Q0 = 5, Q1 = 4$			
$10^2 \times 2$	5.57E-03	–	5.12E-02	–	4.10E-03	–	1.57E-02	–
$20^2 \times 2$	7.25E-04	2.94	8.09E-03	2.66	2.91E-03	0.49	1.42E-02	0.14
$40^2 \times 2$	9.16E-05	2.98	1.17E-03	2.78	2.88E-03	0.01	1.46E-02	0.04
	After postprocessing, $Q0 = 34, Q1 = 33$				After postprocessing, CI			
$10^2 \times 2$	2.67E-03	–	6.52E-03	–	2.67E-03	–	6.52E-03	–
$20^2 \times 2$	1.32E-04	4.34	5.77E-04	3.50	1.31E-04	4.35	5.78E-04	3.50
$40^2 \times 2$	1.57E-05	3.07	8.99E-05	2.68	1.35E-05	3.28	9.00E-05	2.68
			$\mathbb{P}^3$					
	Before postprocessing				After postprocessing, $Q0 = 6, Q1 = 6$			
$10^2 \times 2$	6.51E-04	–	6.63E-03	–	4.01E-04	–	2.43E-03	–
$20^2 \times 2$	4.18E-05	3.96	5.34E-04	3.63	3.72E-04	0.10	2.96E-03	-0.28
$40^2 \times 2$	2.64E-06	3.98	3.15E-05	4.08	3.87E-04	-0.05	3.17E-03	-.09
	After postprocessing, $Q0 = 45, Q1 = 44$				After postprocessing, CI			
$10^2 \times 2$	3.12E-04	–	8.80E-04	–	3.11E-04	–	8.80E-04	–
$20^2 \times 2$	3.00E-06	6.70	2.14E-05	5.36	2.97E-06	6.71	2.14E-05	5.36
$40^2 \times 2$	2.38E-07	3.66	1.63E-06	3.71	2.20E-07	3.75	1.63E-06	3.71
			$\mathbb{P}^4$					
	Before postprocessing				After postprocessing, $Q0 = 8, Q1 = 7$			
$10^2 \times 2$	5.52E-05	–	7.03E-04	–	7.59E-05	–	2.29E-04	–
$20^2 \times 2$	1.77E-06	4.96	2.67E-05	4.71	3.59E-05	1.08	3.20E-04	-0.48
$40^2 \times 2$	5.62E-08	4.97	9.03E-07	4.88	3.69E-05	-0.03	3.60E-04	-0.16
	After postprocessing, $Q0 = 56, Q1 = 55$				After postprocessing, CI			
$10^2 \times 2$	7.23E-05	–	2.12E-04	–	7.23E-05	–	2.12E-04	–
$20^2 \times 2$	1.07E-07	9.40	3.24E-07	9.35	1.07E-07	9.40	3.24E-07	9.35
$40^2 \times 2$	2.58E-09	5.37	1.73E-08	4.23	2.35E-09	5.51	1.73E-08	4.23

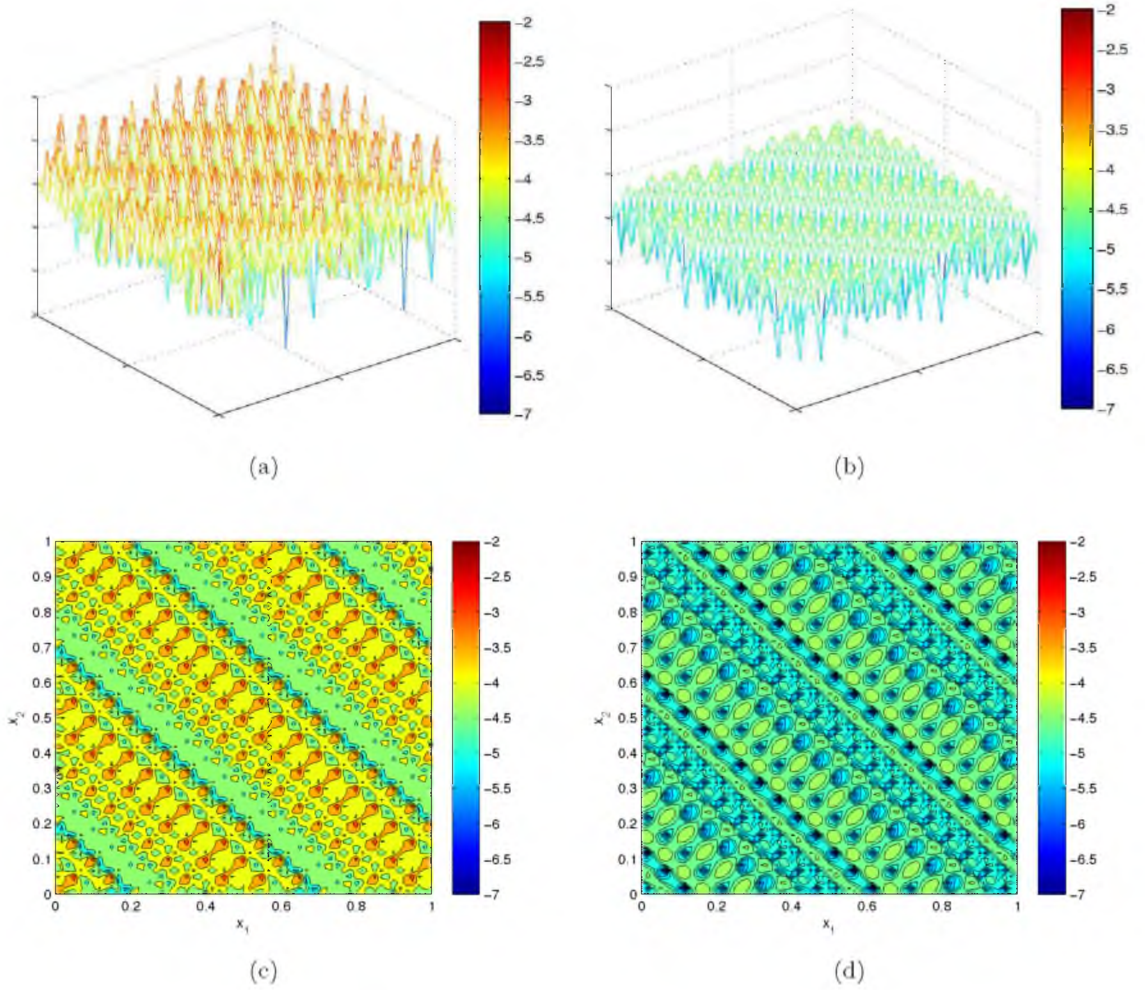


Figure 5.6. Three- and two-dimensional view of point-wise errors in logarithmic scale for constant coefficient advection equation when a $\mathbb{P}^2$ discontinuous Galerkin method is used over a Union-Jack triangular mesh. (a) and (c) demonstrate the initial DG approximation errors and (b) and (d) represent the errors after the application of the postprocessor on the consistent-integration mesh. We can clearly see how the SIAC filter reduces the oscillations in the error.

triangular meshes. The implementation of this method leads to formidable computational challenges because of the many (computationally intensive) integrations involved. For exact integration, the technique must respect both the DG mesh breaks as well as the B-spline kernel breaks. However, we have demonstrated that we can obtain optimal convergence even though we perform inconsistent integration and ignore the B-spline kernel breaks. We have given these elements both a uniform size structure and a smoothly varying structure. In both cases, we are able to improve the order of accuracy of the discontinuous Galerkin solution from $k+1$ to better then $2k$, and in some cases $2k+1$. The results obtained here are exciting because previously accuracy enhancement on triangles has been restricted to $\mathcal{O}(h^{k+2})$. Furthermore, we have addressed one of the computational bottlenecks of the multidimensional implementation of this filter by reducing the number of integrations required.

Forming the filter for general unstructured triangular meshes is a challenging task and requires visiting the issue of the appropriate interpolation function to use for the convolution kernel. The proofs of the higher order accuracy in the negative-order norm of the discontinuous Galerkin method do not rely on the mesh assumption; however, the accuracy-extracting capabilities of the kernel rely on the translation invariance of the mesh. Once this is accomplished, the postprocessor can then also be utilized for determining mesh adaptivity.

CHAPTER 6

IMPROVED ERRORS VERSUS HIGHER ORDER ACCURACY IN APPLICATIONS OF SIAC FILTERS TO DG SOLUTIONS

Smoothness-increasing accuracy-conserving (SIAC) filtering has demonstrated its effectiveness in raising the convergence rate for discontinuous Galerkin solutions from order $k+\frac{1}{2}$ to order $2k+1$ for specific types of translation invariant meshes [19, 22, 46]. Additionally, it improves the weak continuity in the discontinuous Galerkin method to $k-1$ continuity. Typically, this improvement has a positive impact on the error quantity in the sense that it also reduces the absolute errors in the solution. However, not enough emphasis has been placed on the difference between superconvergent accuracy and improved errors. This distinction is particularly important when it comes to interpreting the interplay introduced through meshing, between geometry and filtering. The underlying mesh over which the DG solution is built is important because the tool used in SIAC filtering – convolution – is scaled by the geometric mesh size. This scaling heavily contributes to the effectiveness of the postprocessor. Although the choice of this scaling is straightforward when dealing with a uniform mesh, it is not clear what the impact of either a global or local scaling will be on either the absolute error or on the superconvergence properties of the postprocessor. In this chapter, we present a study of the mesh scaling used in the SIAC filter and how it factors into the theoretical errors.

As mentioned, the typical application of SIAC filters has been to discontinuous Galerkin solutions on translation invariant meshes. The most common means of generating translation invariant meshes is by constructing a base tessellation of size H and repeatedly tiling in a nonoverlapping fashion the base tessellation until the volume of interest is filled [7, 1]. The effectiveness of such a translation invariant filter for discontinuous Galerkin solutions of linear hyperbolic equations was initially demonstrated by Cockburn, Luskin, Shu, and Süli [19]. A computational extension to smoothly varying meshes as well as random meshes, where a scaling equal to the largest element size was used, was given in [22]. For smoothly varying meshes, the improvement to order $2k+1$ was observed. For random meshes, there was no clear order improvement, which could be due to an incorrect kernel scaling. These results were theoretically and numerically extended to translation invariant structured triangular meshes in Chapter 5 and published in [46]. However, the outlook for triangular meshes is actually much better than those presented in [46]. Indeed, the order improvement was not clear for filtering over a Union-Jack mesh when a filter scaling equal $\frac{H}{2}$ was used. In this chapter, we revisit the Union-Jack mesh case, as well as a Chevron triangular mesh, and demonstrate that it is indeed possible to obtain superconvergence of order $2k+1$ for these mesh types when the proper scaling of the filter, related to the translation invariant properties of the mesh, are employed. Furthermore we also introduce theoretical proof that these results can be extended to adaptive meshes that are constructed in a hierarchical manner – in particular, adaptive meshes whose elements are defined by hierarchical (integer) splittings of elements of size H, where H represents both the macro-element spacing used in the generation of the mesh and the minimum scaling used for the SIAC filter.

This chapter addresses these issues in the following manner: in Section 6.1, we present the theoretical extension of the SIAC filter to adaptive meshes; in Section 6.2, an emphasis on the difference between order improvement and error improvement is discussed through presenting various numerical examples. The result of these contributions has been published in [47].

6.1 Theoretical Kernel Scaling

In this section, a proof of the superconvergence of the DG solution through SIAC filtering for $h = \frac{1}{\ell}H$ where ℓ is a multi-integer is given. The main theorem is the following:

Theorem 6.1.1 *Let u_h be the DG solution to*

$$u_t + \sum_{n=1}^{d} A_i u_{x_i} + A_0 u = 0, \qquad \mathbf{x} \in \Omega \times [0,T],$$
$$u(\mathbf{x},0) = u_o(\mathbf{x}), \qquad \mathbf{x} \in \Omega,$$

where A_i, $i = 0,\dots,d$ are constant coefficients, $\Omega \subset \mathbb{R}^d$. The approximation is taken over a mesh whose elements are of size $h = \frac{1}{\ell}H$ in each coordinate direction where ℓ is a multi-integer (of dimension equal to the number of elements along one coordinate direction) and H represents the macro-element size of which any particular element is generated by *hierarchical integer partitioning of the macro-element. Given sufficient smoothness in the initial data,*

$$\|u - K_H \star u_h\|_\Omega \leq CH^{2k+1},$$

where K_H is the SIAC postprocessing kernel scaled by H.

Remark 1 *This theorem is more about geometry than the issues of superconvergence. Indeed, the difference between this estimate and the estimate in [19] has to do with the kernel scaling, H.*

Proof: The general estimate for a uniform mesh was proven in [19]. Here, we repeat the important points for this extension.

First, the error is split into two parts – the error from the design of the filter and the error from the approximation method used:

$$\|u - K_H \star u_H\|_\Omega \leq \underbrace{\|u - K_H \star u\|_\Omega}_{\text{Filtering exact solution}} + \underbrace{\|K_H \star (u - u_h)\|_\Omega}_{\text{Filtering of approximation}}. \tag{6.1}$$

The first part of this error estimate simply uses the fact that the kernel reproduces polynomials of degree less than or equal to degree $2k$ as well as a Taylor expansion. Details are given in [19]. Note that it is possible to make the estimate of the first term of arbitrarily high order. It is therefore the second term, the postprocessed approximation, that dominates the error estimate.

One key aspect of the second error estimate is a property of the B-Splines. That is,

$$D^\alpha(K_H \star (u - u_h)) = \widetilde{K}_H \star \partial_H^\alpha (u - u_h), \tag{6.2}$$

where K_H is the kernel using B-splines of order $k+1+\alpha$, $\widetilde{K}_H$ is the kernel using B-splines of order $k+1$, and the operator ∂_H^α denotes the differencing operator as defined in [11]. Another important factor in obtaining the bound on the approximation is the estimate by Bramble and Schatz [11] that bounds the L_2-error by the superconvergent error in the negative-order norm:

$$\|K_H \star (u - u_h)\|_\Omega \leq C \sum_{\alpha \leq |\ell|} \|D^\alpha(K_H \star (u - u_h))\|_{-\ell} \leq C \sum_{\alpha \leq |\ell|} \|\partial_H^\alpha (u - u_h)\|_{-\ell} \leq CH^{2k+1}. \tag{6.3}$$

This superconvergence in the negative-order norm was proven by Cockburn *et al.* in [19] for a *uniform* translation invariant mesh. This is a consequence of the B-Spline property that allows derivatives to be expressed as divided difference quotients. The divided difference quotient relationship as expressed in Equation (6.2) is only possible for an H-translation invariant mesh or those meshes in which the mesh element spacings are integer multiples of the characteristic length H. This occurs automatically for uniform quadrilateral or hexahedral meshes as well as structured triangular meshes. However, extension to adaptive meshes is possible by constructing the element spacing h as an integer partitioning of the fundamental characteristic size, $h = \frac{1}{\ell}H$, ℓ a multi-integer. When this is the case, one can observe that the translation operator as defined in [11] specified with respect to H can be related to translation with respect to the actual element size h, i.e., over the DG mesh, as follows:

$$T_H^m v(\mathbf{x}) = v(\mathbf{x} + mH) = v(\mathbf{x} + m\ell h) = T_h^{m\ell} v(\mathbf{x}). \tag{6.4}$$

The error estimate therefore follows as the divided difference operator in Equation (6.3), originally expressed in terms of H, can be expressed in terms of its integer multiples. The constant C in the right-most expression encapsulates the impact of the (integer multiple) adaptive spacing.

Remark 2 *Particular emphasis should be placed on the fact that this makes the SIAC filter applicable to adaptive meshes, provided the scaling is taken in the correct manner.*

6.2 Numerical Results

In this section, we discuss the importance of geometric mesh assumptions for obtaining improved errors versus improved order of accuracy. This is done by inspecting one equation for different mesh types. That is,

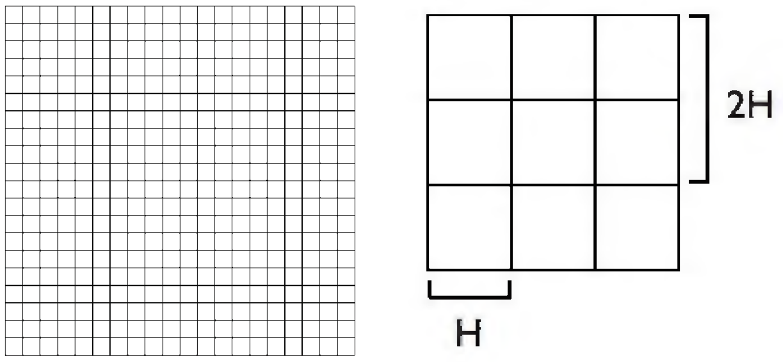
$$\begin{aligned} u_t + \nabla \cdot u &= 0, \qquad x \in [0, 2\pi]^2 \times [0, T] \\ u(\mathbf{x}, 0) &= \sin(x + y). \end{aligned}$$

An investigation of the filtered DG solution will be performed for meshes that include the uniform quadrilateral mesh, an adaptive mesh, a structured triangular mesh, a Union-Jack mesh, and a Chevron mesh. Additionally, for the first time, three-dimensional results over a hexahedral mesh are also given. Note that similar behavior has been observed for variable coefficient equations, as predicted by the theory (see Section 5.2). A particular emphasis will be placed on the distinction between reduced errors and higher order accuracy for a given scaling of the SIAC filter.

6.2.1 Uniform Quadrilateral Mesh

The first example presented is a study of the scaling for the SIAC filter for a uniform quadrilateral mesh, as shown in Figure 6.1.

The theory of [19] establishes that the scaling H used by the postprocessor should be the same as used to construct the mesh (i.e., the mesh is of uniform spacing H). However, according to Theorem 6.1.1, a scaling of any integer multiple, mH should also produce superconvergent accuracy. Indeed, in Table 6.1 the numerical results using different values of m for the kernel scaling are presented. It can be seen that as long as $m \geq 1$, accuracy of order $2k+1$ is obtained. Examining the errors closely, it becomes obvious that the errors are actually increasing as the kernel scaling becomes greater, even with this $2k+1$ convergence. A plot of absolute error versus different scalings is given in Figure 6.2. This plot demonstrates that the minimal error actually occurs with a SIAC filter scaling a bit less than the element spacing H and after this scaling, the errors begin increasing, although maintaining the $2k{+}1$ convergence rate. In Figure 6.3, contour plots of the errors for $N = 40$ for $\mathbb{P}^2$ and $\mathbb{P}^3$ polynomial approximations are presented for scalings of $0.5H$, H, and $2H$. The plots demonstrate that the errors get much smoother as we increase the scaling from $0.5H$ to H. The errors using $2H$ scaling are also smooth; however, the magnitude of the errors is larger.

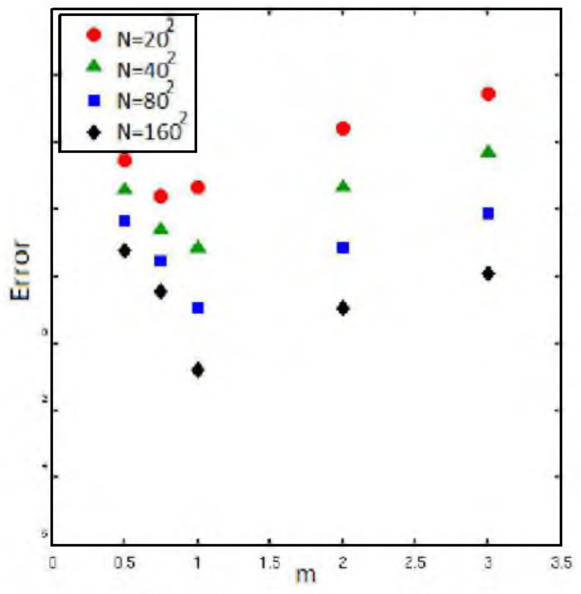


(a) Uniform quadrilateral mesh (b) Filter spacing

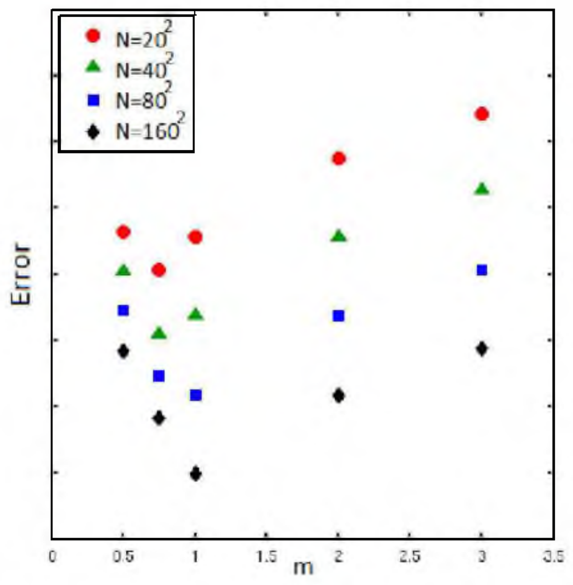
Figure 6.1. Example of uniform quadrilateral mesh (a) and diagram of filter spacing used (b).

Table 6.1. Table of L_2-errors for various scalings used in the SIAC filter for a uniform quadrilateral mesh. 'x' indicates the cases where the width of the kernel became larger than the computational field.

	$m = 0.5$		$m = 1$		$m = 2$		$m = 3$	
Mesh	L_2 error	Order	L_2 error	Order	L_2 error	Order	L_2 error	Order
	$\mathbb{P}^2$							
20^2	2.95E-05	—	4.48E-06	—	2.69E-04	—	x	—
40^2	3.75E-06	2.98	7.09E-08	5.98	4.45E-06	5.92	4.96E-05	—
80^2	4.7E-07	2.99	1.11E-09	5.99	7.06E-08	5.98	7.99E-07	5.95
160^2	5.88E-08	3	1.74E-11	6	1.11E-09	5.99	1.26E-08	5.99
	$\mathbb{P}^3$							
20^2	1.99E-07	—	1.38E-07	—	3.22E-05	—	x	—
40^2	1.27E-08	3.97	5.49E-10	7.97	1.38E-07	7.87	3.4E-06	—
80^2	7.97E-10	3.99	2.16E-12	7.99	5.49E-10	7.97	1.4E-08	7.93
160^2	4.99E-11	4	9.1E-15	7.89	2.16E-12	7.99	5.52E-11	7.98
	$\mathbb{P}^4$							
40^2	3.36E-11	—	4.41E-12	—	4.38E-09	—	x	—
80^2	1.06E-12	4.99	3.19E-15	10.4	4.41E-12	9.96	2.51E-10	—
160^2	3.37E-14	4.97	2.38E-15	0.421	3.79E-15	10.2	2.48E-13	9.98



(a) $\mathbb{P}^2$-polynomials



(b) $\mathbb{P}^3$-polynomials

Figure 6.2. Plots of error versus scaling (m) used in the SIAC filter for a uniform quadrilateral mesh for $\mathbb{P}^2$ (left) and $\mathbb{P}^3$ (right) polynomial approximations.

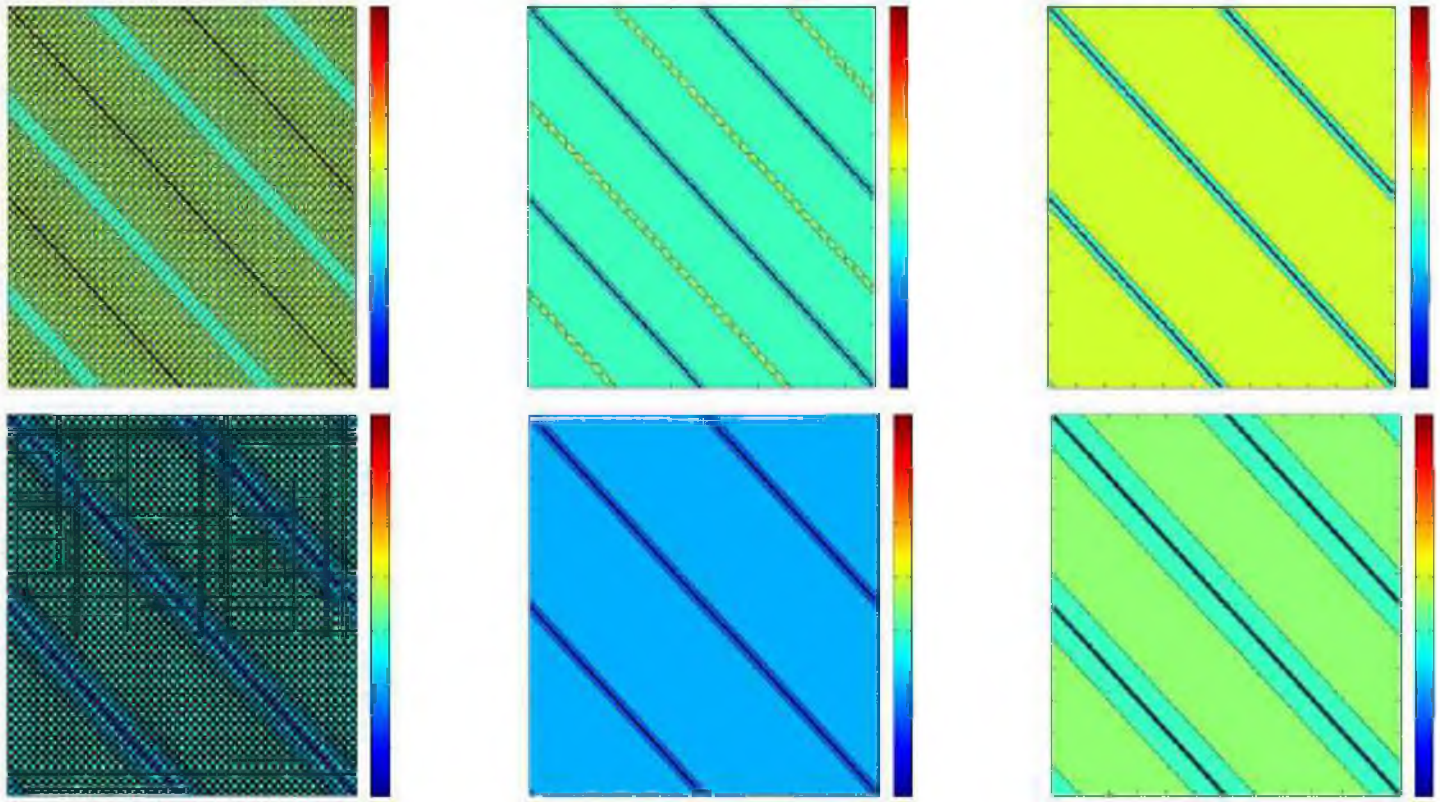
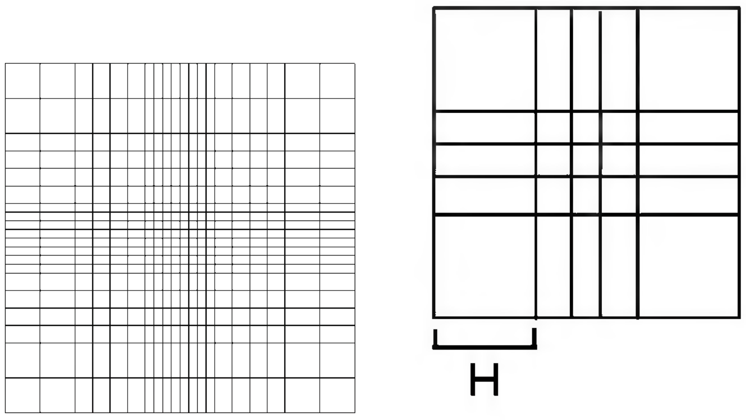


Figure 6.3. Contour plots using a scaling of $0.5H$ (left) and H (middle) and $2H$ (right) for a uniform quadrilateral mesh such as the one in Figure 6.1. Top row: $\mathbb{P}^2$, Bottom row: $\mathbb{P}^3$.

6.2.2 Quadrilateral Cross Mesh

In this example, we consider a variable-spacing quadrilateral mesh. The mesh was designed in the following manner: let $H = \frac{2}{N}$, where N is the total number of elements in one direction used in the approximation. We first divide the mesh into a collection of evenly-spaced quadrilateral macro-elements of size H. In order to generate the final mesh, we further split some of these quadrilateral elements (more towards the middle of the mesh) into two, four, or more quadrilateral subelements, i.e., each element of the new mesh is created by subdividing the macro-element of size H by some integer partition. This type of scaling gives an adaptive cross mesh, as shown in Figure 6.4. Note that although this mesh is not uniformly-spaced, the mesh construction proposed does meet a local hierarchical partitioning property which we have proven to be sufficient for observing superconvergence when applying the SIAC filter with a scaling of H.

In Table 6.2 ,the errors for mH where $m = 0.5, 1.0, 1.5, 2.0$ are given. Notice that one begins to see the correct superconvergent rate of order $2k + 1$ for a scaling of $m = 1.0$, as expected. The "x's" given in the table denote regions in which the chosen scaling of the kernel makes the kernel support wider than the mesh used in the approximation. In Figure 6.5, a plot of error versus m is given. Observe that the minimum error and the correct convergence rate occurs at H ($m = 1$), as predicted by the theory. In Figure 6.6, contour error plots are shown for different scalings of the SIAC filter.



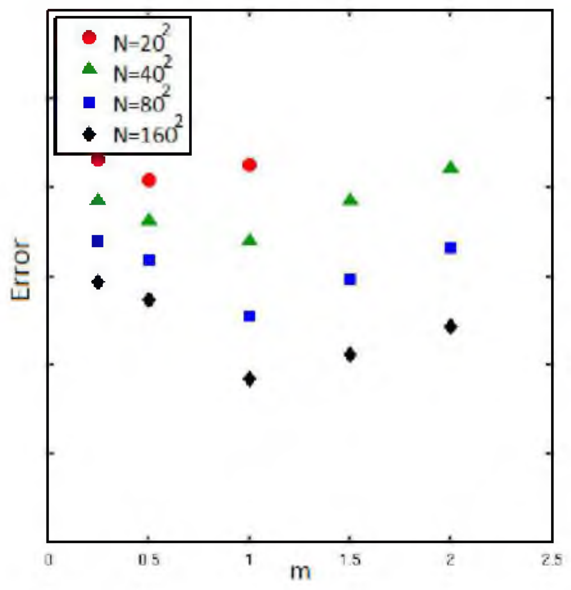
(a) Variable-spacing quadrilateral (cross) mesh

(b) Filter spacing

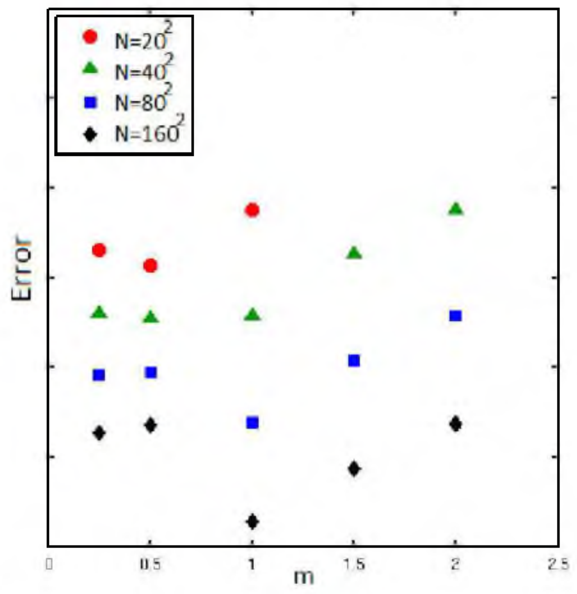
Figure 6.4. Example of a variable-spacing quadrilateral (cross) mesh (a) and diagram of filter spacing used (b).

Table 6.2. Table of L_2-errors for various scalings used in the SIAC filter for a variable-spacing cross mesh. 'x' indicates the cases where the width of the kernel became larger than the computational field.

	$m = 0.5$		$m = 1$		$m = 1.5$		$m = 2$	
Mesh	L_2 Error	Order	L_2 Error	Order	L_2 Error	Order	L_2 Error	Order
				$\mathbb{P}^2$				
20^2	1.5E-04	—	3.19E-04	—	x	—	x	—
40^2	1.83E-05	3.03	6.04E-06	5.72	5.11E-05	—	2.71E-04	—
80^2	2.34E-06	2.97	1.28E-07	5.56	8.52E-07	5.91	4.5E-06	5.91
160^2	2.95E-07	2.98	4.74E-09	4.75	1.77E-08	5.59	7.21E-08	5.96
				$\mathbb{P}^3$				
20^2	1.91E-06	—	3.23E-05	—	x	—	x	—
40^2	1.24E-07	3.95	1.38E-07	7.87	3.4E-06	—	3.22E-05	—
80^2	7.92E-09	3.96	5.72E-10	7.92	1.4E-08	7.93	1.38E-07	7.87
160^2	5E-10	3.99	3.75E-12	7.25	5.56E-11	7.97	5.5E-10	7.97
				$\mathbb{P}^4$				
40^2	6.61E-10	—	4.38E-09	—	2.4E-07	—	x	—
80^2	2.19E-11	4.91	7.87E-12	9.12	2.51E-10	9.9	4.38E-09	—
160^2	7.6E-13	4.85	3.71E-13	4.4	4.23E-13	9.21	4.38E-12	9.96



(a) $\mathbb{P}^2$-polynomials



(b) $\mathbb{P}^3$-polynomials

Figure 6.5. Plots of error versus various scalings (m) used in the SIAC filter for a variable-spacing cross mesh for $\mathbb{P}^2$ (left) and $\mathbb{P}^3$ (right) polynomial approximations.

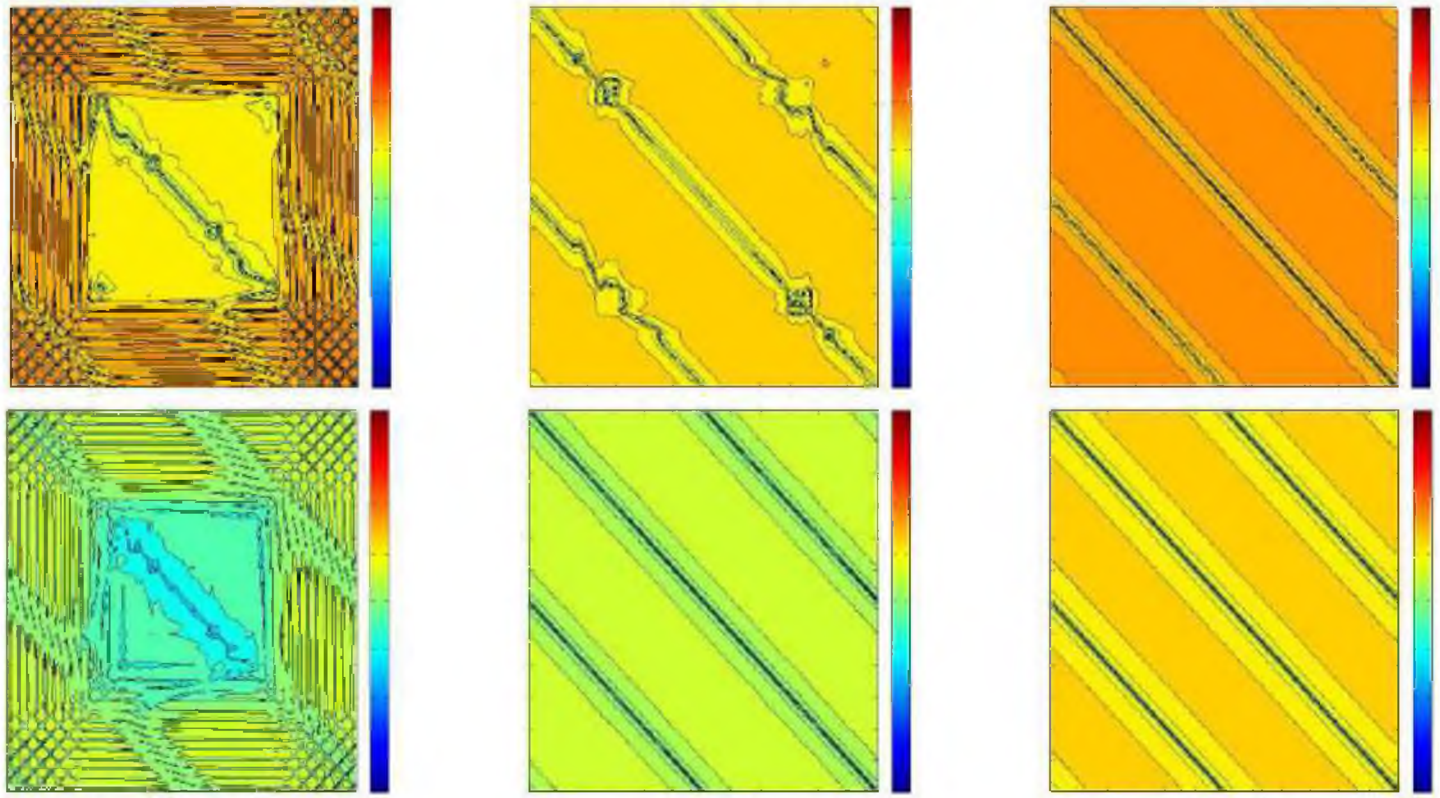
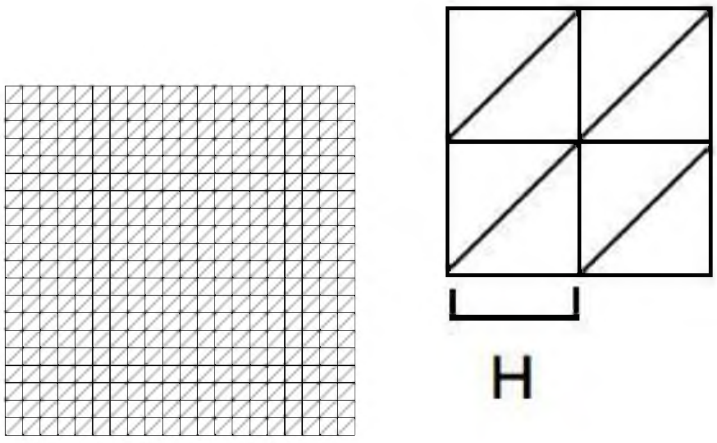


Figure 6.6. Contour plots using a scaling of $0.5H$ (left), H (middle), and $1.5H$ (right) for a variable-spacing cross quadrilateral mesh such as the one in Figure 6.4. Top row: $\mathbb{P}^2$, Bottom row: $\mathbb{P}^3$.

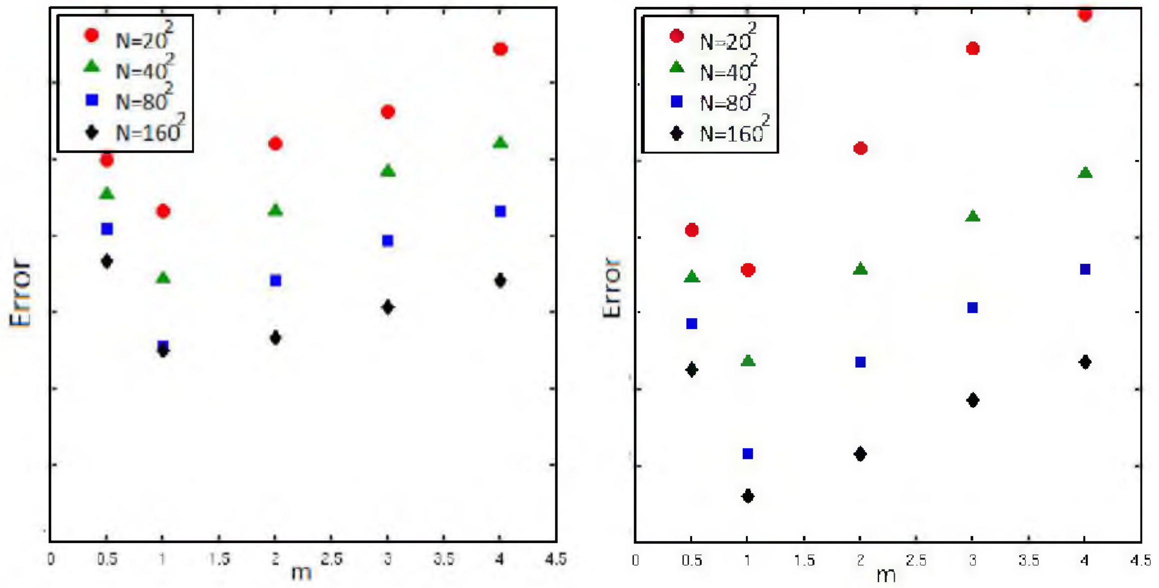
6.2.3 Structured Triangular Mesh

Once it has been established that superconvergence occurs for various scalings of quadrilateral meshes, it is then interesting to test whether these ideas extend to structured triangular meshes, as these meshes are also translation invariant. Below are the numerical results for three different types of structured triangular meshes: a uniform structured triangular mesh, a Union-Jack mesh, and a Chevron mesh. The uniform structured triangular mesh, as shown in Figure 6.7, is first examined to ensure the extension of the main ideas of Theorem 6.1.1 to this type of mesh. The DG errors together with the filtered errors were first presented in [46] and discussed in Chapter 5. In Figure 6.8, the L_2-errors are presented for various scalings. It can be seen that superconvergent accuracy of order $2k + 1$ occurs for scalings of mH, $m \geq 1$ although the errors increase with increasing m. In Figure 6.9, contour plots of the errors for scalings of $m = 0.5, 1, 2$ are also shown. A scaling of $0.5H$ and $2H$ produces worse errors than that of H, although scalings of H and $2H$ also produce smoothness in the errors. Furthermore, in Table 6.3, the ratio of kernel size to mesh size is given. It demonstrates that the ratio becomes larger for increasing polynomial order or increased m values, which means that the footprint of the postprocessor requires more elements in the computation and becomes less local. Table 6.4 provides the error values.



(a) Structured triangular mesh (b) Filter spacing

Figure 6.7. Example of structured triangular mesh (a) and diagram of filter spacing used (b).



(a) $\mathbb{P}^2$-polynomials

(b) $\mathbb{P}^3$-polynomials

Figure 6.8. Plots of error versus various scalings (m) used in the SIAC filter for a structured triangular mesh for $\mathbb{P}^2$ (left) and $\mathbb{P}^3$ (right) polynomial approximations.

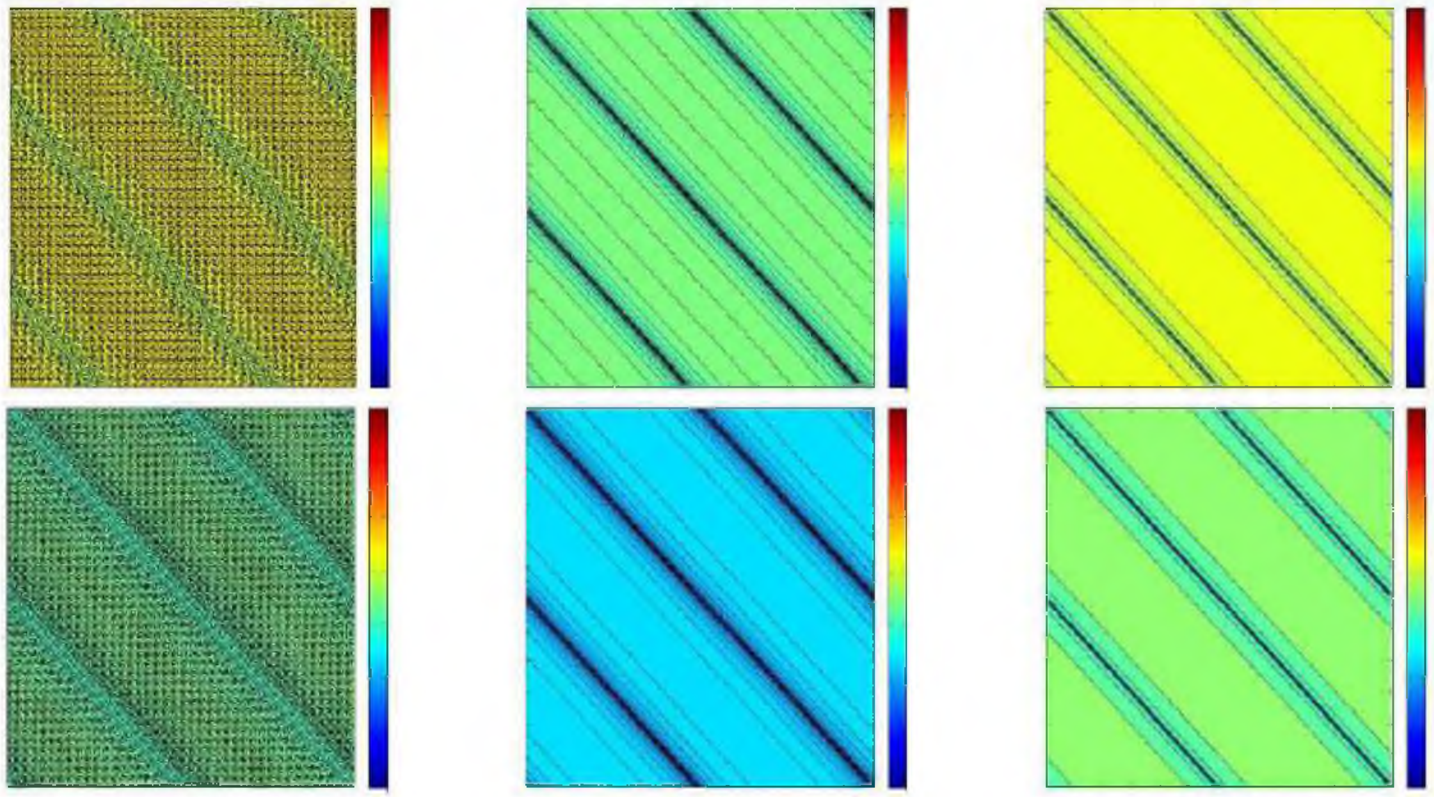


Figure 6.9. Contour plots using a scaling of $0.5H$ (left), H (middle), and $2H$ (right) for a structured triangular mesh such as the one in Figure 6.7. Top row: $\mathbb{P}^2$, Bottom row: $\mathbb{P}^3$.

Table 6.3. Kernel to mesh ratios for the structured triangular mesh cases (uniform, Jack, and Chevron). N^2 represents the number of quadrilateral elements.

H	$\mathbb{P}^2$	$\mathbb{P}^3$	$\mathbb{P}^4$
0.5	3.5/N	5/N	7.5/N
1	7/N	10/N	13/N
2	14/N	20/N	26/N
3	21/N	30/N	39/N
4	28/N	40/N	52/N

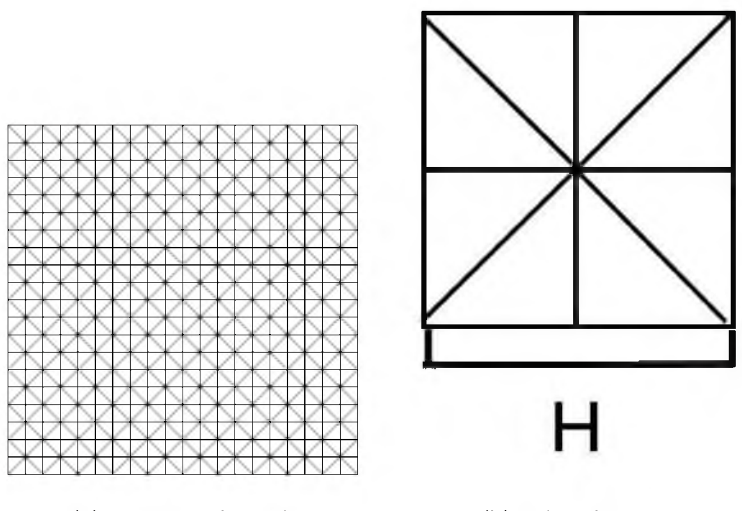
Table 6.4. Table of L_2-errors for various H scalings used in the SIAC filter for a structured triangle mesh.

–	$m = 0.5$		$m = 1$		$m = 2$		$m = 3$	
Mesh	L_2 Error	Order	L_2 Error	Order	L_2 Error	Order	L_2 Error	Order
	$\mathbb{P}^2$							
20^2	1.01E-04	—	4.65E-06	—	2.69E-04	—	x	—
40^2	1.25E-05	3.01	7.50E-08	5.95	4.46E-06	5.92	4.96E-05	—
80^2	1.57E-06	3.00	1.26E-09	5.89	7.06E-08	5.98	7.99E-07	5.96
160^2	2.28E-07	2.78	1.05E-09	0.259	2.14E-09	5.04	1.36E-08	5.87
	$\mathbb{P}^3$							
20^2	1.49E-06	—	1.38E-07	—	2.21E-04	—	x	—
40^2	9.15E-08	4.02	5.50E-10	7.97	1.38E-07	10.65	3.40E-06	—
80^2	5.70E-09	4.01	2.16E-12	7.99	5.49E-10	7.97	1.40E-08	7.93
160^2	3.52E-10	4.02	1.64E-13	3.72	2.2E-12	7.97	5.52E-11	7.98
	$\mathbb{P}^4$							
40^2	7.26E-10	—	4.41E-12	—	4.38E-09	—	3.47E-07	—
80^2	2.24E-11	5.02	1.75E-14	7.98	4.41E-12	9.96	2.51E-10	10.43
160^2	5.81E-013	5.27	2.38E-013	-3.77	2.37E-013	4.22	3.53E-013	9.47

6.2.4 Union-Jack Mesh

In Section 5.3.4, error results for the Union-Jack mesh were presented with a scaling of $\frac{H}{2}$ equal to the uniform spacing of the base quadrilateral mesh. It was noted at the time that the correct convergence order was not obtained (and from the theory was not expected to be obtained), but error improvement was observed. However, as Babuška *et al.* noted in [5, 55, 8, 7], this mesh is translation invariant in H, as seen in Figure 6.10.

In Table 6.5, errors for scaling of mH, where $m = 0.5, 1, 1.5, 2$ are presented. It is clearly seen that the superconvergence is observed for scalings of mH where $m \geq 1$. It is interesting to note that the mesh is not translation invariant in $1.5H$ but we see the superconvergence of order $2k + 1$. Additionally, the errors begin to worsen after the scaling of H. This is also seen in Figure 6.11 ($m = 1$). Additionally, in Figure 6.12 the differences in the errors between $0.25H$ scaling, $0.5H$, and H are shown. We obtain a much smoother contour plot with the H scaling.

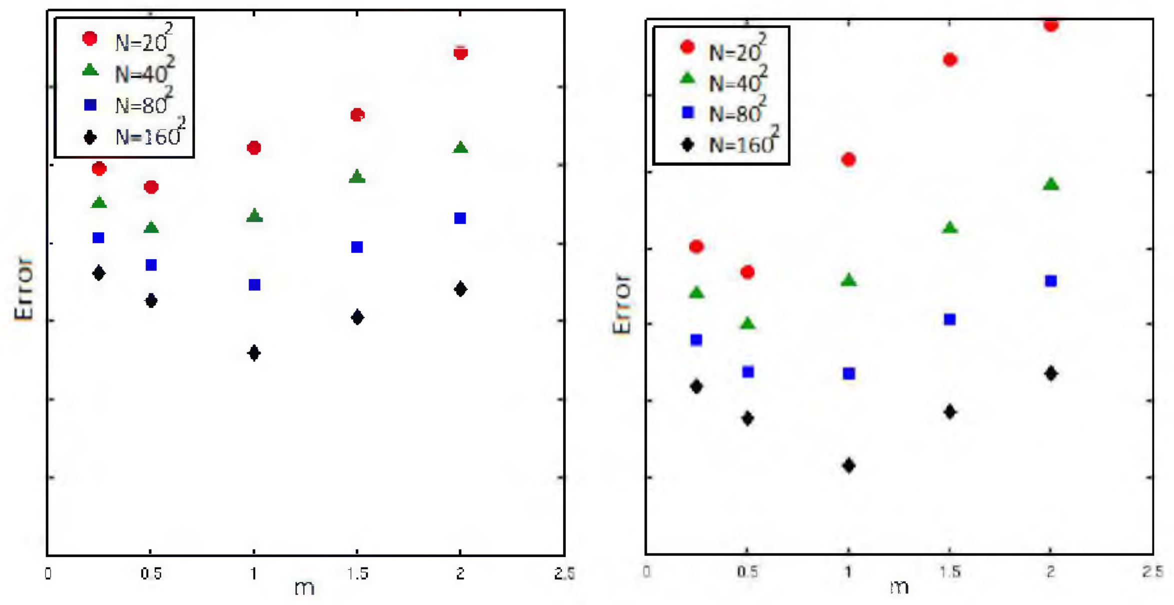


(a) Union-Jack mesh (b) Filter Spacing

Figure 6.10. Example of a Union-Jack mesh (a) and diagram of filter spacing used (b).

Table 6.5. Table of L_2-errors for various scalings used in the SIAC filter for a Union-Jack mesh.

–	$m = 0.5$		$m = 1$		$m = 1.5$		$m = 2$	
Mesh	L_2 Error	Order	L_2 Error	Order	L_2 Error	Order	L_2 Error	Order
	$\mathbb{P}^2$							
20^2	2.88E-05	—	2.87E-04	—	1.98E-03	—	7.92E-02	—
40^2	2.39E-06	3.59	5.01E-06	5.84	5.01E-05	5.31	2.70E-04	8.19
80^2	2.92E-07	3.03	8.81E-08	5.83	8.17E-07	5.94	4.47E-06	5.92
160^2	3.66E-08	3	1.65E-09	5.74	1.31E-08	5.96	7.11E-08	5.97
	$\mathbb{P}^3$							
20^2	2.39E-07	—	2.23E-04	—	8.64E-02	—	6.90E-02	—
40^2	9.97E-09	4.59	1.38E-07	10.66	3.40E-06	14.63	4.54E-05	13.89
80^2	5.89E-10	4.09	5.51E-10	7.97	1.39E-08	7.93	1.38E-07	8.37
160^2	3.65E-11	4.01	2.2E-12	7.97	5.52E-11	7.98	5.5E-10	7.97
	$\mathbb{P}^4$							
40^2	1.84E-09	—	4.77E-09	—	3.49E-07	—	2.72E-03	—
80^2	3.11E-12	9.21	5.34E-12	9.80	2.51E-10	10.44	4.38E-09	19.24
160^2	2.35E-13	3.73	2.34E-013	4.52	3.51E-13	9.48	4.42E-12	9.95



(a) $\mathbb{P}^2$-polynomials (b) $\mathbb{P}^3$-polynomials

Figure 6.11. Plots of error versus various scalings used in the SIAC filter for a Union-Jack mesh for $\mathbb{P}^2$ (left) and $\mathbb{P}^3$ (right) polynomial approximations.

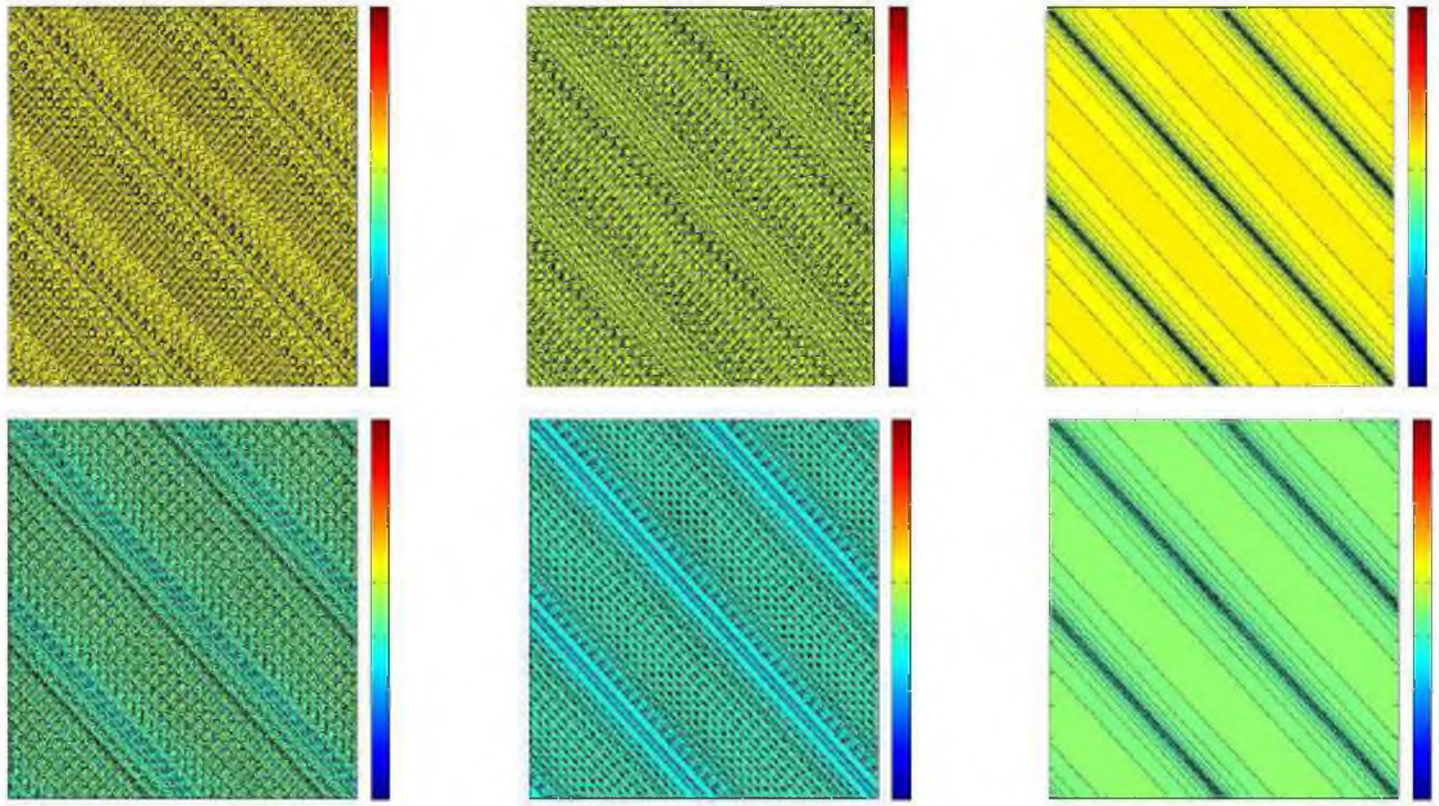


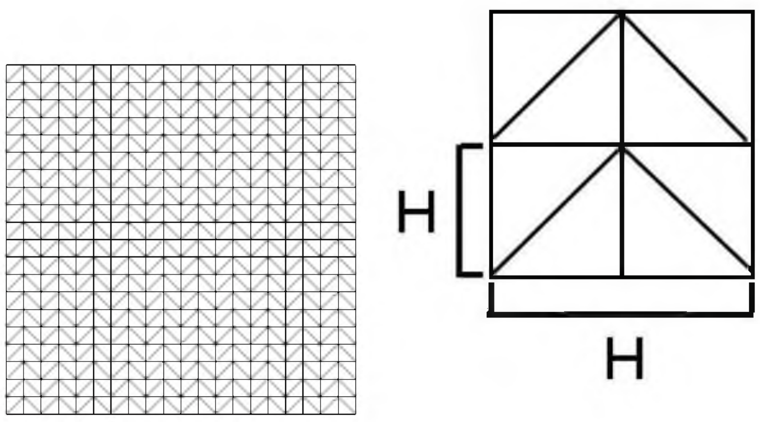
Figure 6.12. Contour plots using a scaling of $0.25H$ (left), $0.5H$ (middle), and H (right) for a Union-Jack mesh such as the one in Figure 6.10. Top row: $\mathbb{P}^2$, Bottom row: $\mathbb{P}^3$.

6.2.5 Chevron Mesh

In this example, the structured Chevron mesh presented in Figure 6.13 is examined. Note that the mesh is translation invariant for $H = 2h$ in the x_1-direction and $H = h$ in the x_2-direction where h denotes the spacing of the base quadrilateral mesh. For simplicity in the calculations, the kernel scaling for the x_1 and x_2 directions have been taken to be the same and equal to $H = 2h$. In Figure 6.14, the errors versus different scalings are presented similar to previous examples. Figure 6.15 depicts the contour plots. In Table 6.6, the errors are presented for various choices of m. The table shows mixed results in convergence order for a scaling of $m = 0.5$, but clear improvement to the theoretical order for $m = 1$ and larger.

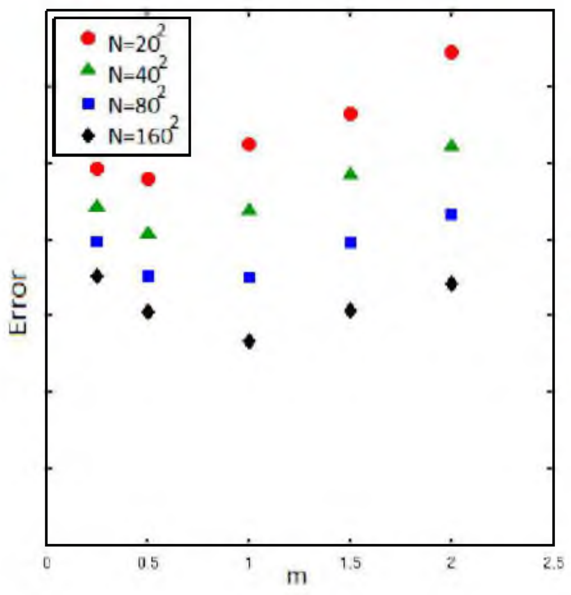
6.2.6 Hexahedral Mesh

The last example that we present is the first example of the extension of this SIAC filter to three-dimensions. The extension is for a uniform hexahedral mesh of spacing H. In Table 6.7, the errors for the discontinuous Galerkin solution of a three-dimensional DG projection problem are given along with the improved errors using the SIAC filter for various scalings mH. We can see that the added dimension does not reduce the order of convergence, and a superconvergent rate of $2k + 1$ is obtained. This is in agreement with the theory [19].

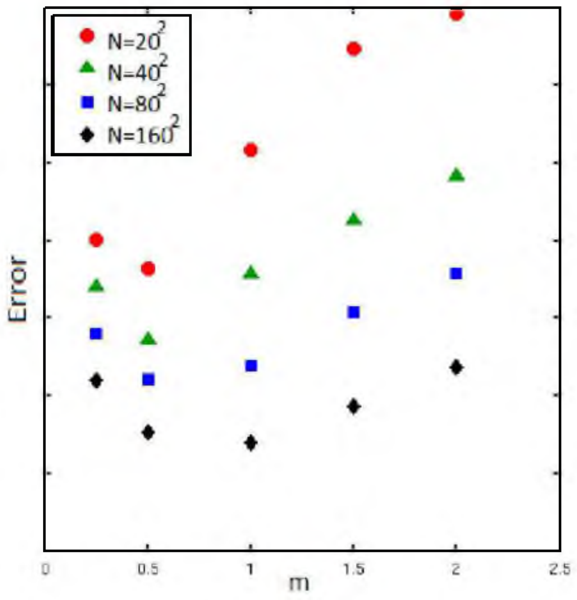


(a) Chevron mesh (b) Filter Spacing

Figure 6.13. Example of a Chevron mesh (a) and diagram of filter spacing used (b). H represents the minimum translation invariance of the mesh. This value is not necessarily the same for each direction, as it is shown in (b).



(a) $\mathbb{P}^2$-polynomials



(b) $\mathbb{P}^3$-polynomials

Figure 6.14. Plots of error versus various scalings used in the SIAC filter for a Chevron mesh for $\mathbb{P}^2$ (left) and $\mathbb{P}^3$ (right) polynomial approximations.

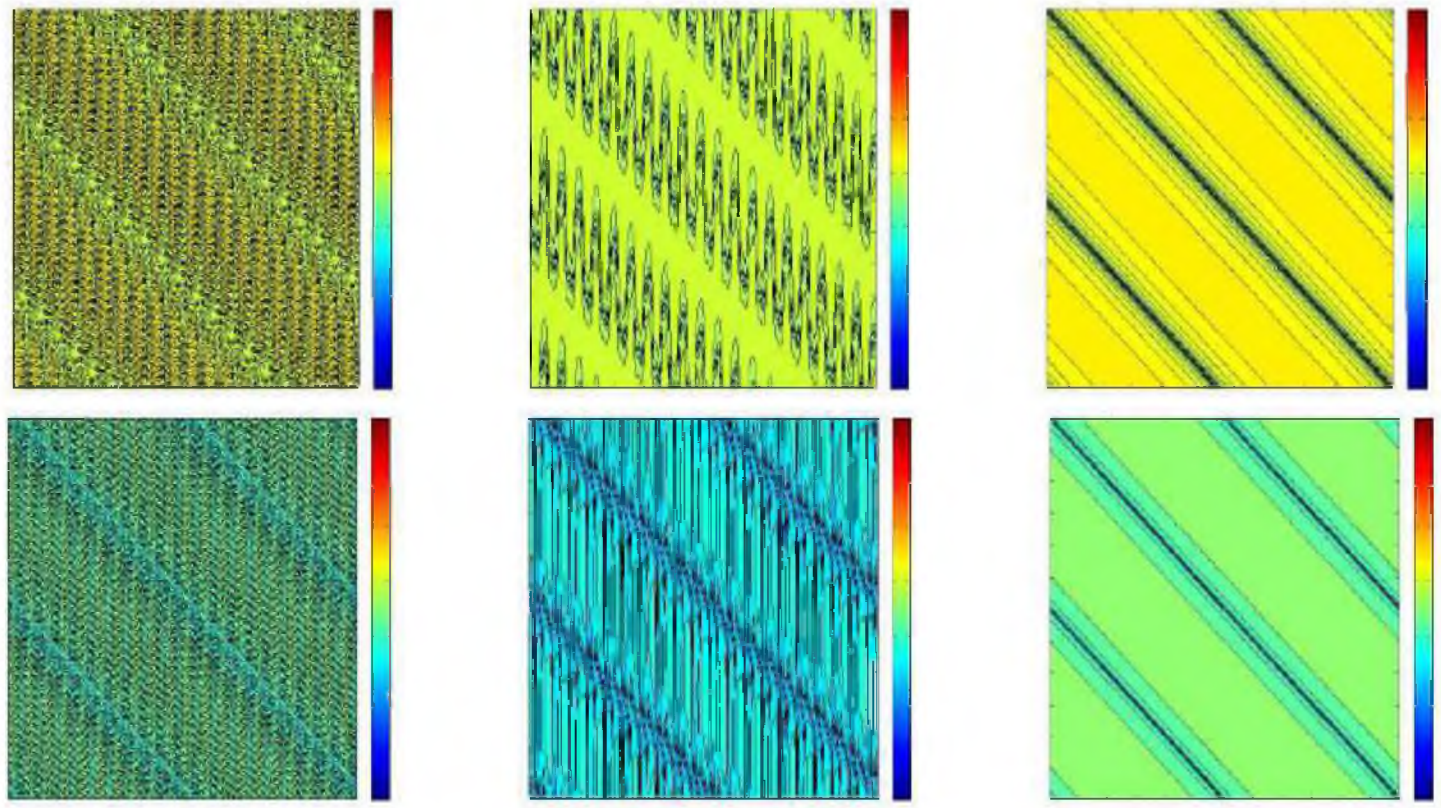


Figure 6.15. Contour plots using a scaling of $0.25H$ (left), $0.5H$ (middle), and H (right) for a Chevron mesh such as the one in Figure 6.13. Top row: $\mathbb{P}^2$, Bottom row: $\mathbb{P}^3$.

Table 6.6. Table of L_2-errors for various scalings used in the SIAC filter for a Chevron mesh.

–	$m = 0.5$		$m = 1$		$m = 1.5$		$m = 2$	
Mesh	L_2 Error	Order	L_2 Error	Order	L_2 Error	Order	L_2 Error	Order
	$\mathbb{P}^2$							
20^2	4.06E-05	—	3.05E-04	—	x	—	x	—
40^2	1.46E-06	4.79	5.57E-06	5.77	5.07E-05	—	2.71E-04	—
80^2	1.09E-07	3.74	1.06E-07	5.72	8.34E-07	5.93	4.49E-06	5.91
160^2	1.29E-08	3.09	2.2E-09	5.58	1.37E-08	5.93	7.17E-08	5.97
	$\mathbb{P}^3$							
20^2	1.96E-07	—	2.23E-04	—	x	—	x	—
40^2	2.75E-09	6.16	1.38E-07	10.66	3.40E-06	—	4.54E-05	—
80^2	2.82E-10	3.29	6.00E-10	7.85	1.40E-08	7.93	1.38E-07	8.37
160^2	1.17E-11	4.59	6.27E-12	6.58	5.56E-11	7.97	5.5E-10	7.97
	$\mathbb{P}^4$							
40^2	9.49E-11	—	4.38E-09	—	3.48E-07	—	x	—
80^2	6.01E-12	3.98	7.46E-12	9.20	2.51E-10	10.44	4.38E-09	—
160^2	4.67E-13	3.68	4.67E-13	4.0	5.38E-13	8.87	4.45E-12	9.94

Table 6.7. Table of L_2-errors for various scalings used in the SIAC filter for a uniform hexahedral mesh.

–	Original DG Error		$m = 0.5$		$m = 1$		$m = 1.5$	
Test Case	DG Error	Order	L_2 Error	Order	L_2 Error	Order	L_2 Error	Order
	$\mathbb{P}^2$							
20^3	1.82e-04	—	4.22E-05	—	6.71e-06	—	7.44e-05	—
40^3	2.28e-05	2.99	5.36e-06	2.97	1.06e-07	5.98	1.21e-06	5.94
	$\mathbb{P}^3$							
20^3	3.17e-06	—	1.57e-07	—	2.06e-07	—	5.09E-06	—
40^3	1.98e-07	3.99	1.00e-08	3.97	8.24e-10	7.97	2.13E-08	7.90

6.3 Summary and Conclusions

By implementing smoothness-increasing accuracy-conserving filtering, the errors for the DG solution can usually be improved from order $k+1$ to order $2k+1$ for linear hyperbolic equations. Additionally, due to the nature of the convolution kernel used in this SIAC filter, the smoothness of the solution is also improved from only having weak continuity to having continuity of $k-1$. However, care has to be taken with the mesh geometry and correct kernel scalings must be used. The emphasis of this chapter has been on the difference between error improvement versus order improvement in terms of geometry. In all our numerical examples, it has been demonstrated that it is possible to obtain superconvergence with the correct kernel scaling. However, if the scaling becomes too large, the errors worsen and can become worse than the original DG errors while maintaining superconvergence. We further note that when the mesh size is large enough, filtering with the true scaling parameter H yields the optimal results in terms of the magnitude of the error. Lastly, numerical results showing the effectiveness of the SIAC filter for a three-dimensional DG projection problem were presented. For this equation, superconvergence of order $2k + 1$ was obtained, showing that adding a dimension did not reduce the achieved convergence rate.

CHAPTER 7

APPLICATION OF SIAC FILTERING TO UNSTRUCTURED TRIANGULATIONS

As was mentioned previously, allowing discontinuity between element interfaces provides such flexibility with the discontinuous Galerkin method that is difficult to match with conventional continuous Galerkin methods. Although the DG methodology can be applied to arbitrary triangulations, the typical application of SIAC filters with mathematically proven properties, as shown in Chapters 5 and 6, has been to discontinuous Galerkin solutions obtained over translation invariant meshes. In an attempt to make the SIAC filter applicable to arbitrary tessellations, Curtis *et al.* [22] proposed a computational extension of this filtering technique to smoothly varying meshes as well as random meshes. They provided numerical results in one dimension, which confirmed the accuracy enhancement of $2k+1$, proved in [19], for smoothly varying meshes when a kernel scaling equal to the largest element size was used. For random meshes, there was no clear order improvement, which may have been due to an incorrect kernel scaling. To further expand the applicability of the SIAC filter, we previously demonstrated how to extend the postprocessing results, both theoretically and numerically, to structured triangular meshes.

As the assumption of any sort of regularity, including the translation invariance of the mesh, is a hindrance towards making the SIAC filter applicable to real-life simulations, in this chapter, we demonstrate for the first time the mathematical behavior and computational complexity of the extension of this filter to *unstructured* tessellations. We consider four examples: a simple Delaunay triangulation, a Delaunay triangulation with obvious change in element sizes, a Delaunay triangulation with splitting, and a stretched (anisotropic) triangulation. We show that it is indeed possible to obtain reduced errors and improved smoothness through a proper choice of kernel scaling. These results are promising as they pave the way towards a more generalized SIAC filtering technique that could be used for arbitrary unstructured tessellations.

We proceed in this chapter by providing the implementation details of the postprocessor for unstructured triangular meshes and we discuss the Sutherland-Hodgman clipping algorithm used to compute the mesh-kernel intersections. In Section 7.2, we give numerical results confirming the usefulness of our SIAC filter for the proposed triangulated meshes. We note that the result of these contributions is reported in a submitted manuscript [47].

7.1 Smoothness-Increasing Accuracy-Conserving Filters for Unstructured Triangular Meshes

In this section, we provide the implementation details of the postprocessor over unstructured triangular meshes. The implementation discussed in this section is used to produce the results given in Section 7.2.

In Chapters 3 and 5, we thoroughly discussed the extension of the SIAC filter to structured triangular meshes. Here, we simply take the existing implementation of the SIAC filter and apply the same ideas to unstructured triangular meshes.

The postprocessor takes as input an array of the polynomial modes used in the discontinuous Galerkin method and produces the values of the postprocessed solution at a set of specified evaluation points. We assume these evaluation points correspond with specific quadrature points which can be used at the end of the simulation for such things as error calculations. We examine how to calculate the postprocessed value at a single evaluation point. Postprocessing of the entire domain is obtained by repeating the same procedure for all the evaluation points. Let us consider the case of a discontinuous Galerkin solution produced over an unstructured triangular mesh, shown in Figure 7.1. We remind the reader that in two dimensions, the convolution kernel is the tensor product of one-dimensional kernels. Therefore, the postprocessed solution at $(x, y) \in T_i$, becomes

$$u^{\star}(x,y) = \frac{1}{h_{x_1} h_{x_2}} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} K\left(\frac{x_1 - x}{h_{x_1}}\right) K\left(\frac{x_2 - y}{h_{x_2}}\right) u_h(x_1, x_2) dx_1 dx_2, \tag{7.1}$$

where T_i is a triangular element, u_h is our approximate DG solution, and we have denoted

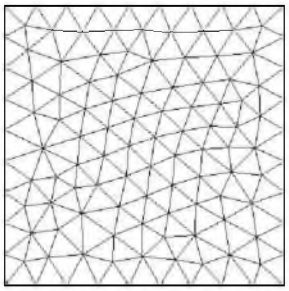


Figure 7.1. A sample unstructured triangular mesh.

the two-dimensional coordinate system as (x_1, x_2). The main difference in the implementation of Equation (7.1) for unstructured triangulations versus structured meshes is in the choices of h_{x_1} and h_{x_2} used to scale the kernel in the x_1 and x_2 directions, respectively. As we discussed in Chapter 6, $h_{x_1} = H_{x_1}$ and $h_{x_2} = H_{x_2}$, where H_{x_1} and H_{x_2} represent the minimum scaling for translation invariance of the mesh. For instance, for a uniform quadrilateral mesh, $h_{x_1} = h_{x_2} = H$ and is simply the uniform mesh spacing. We also provided mathematical proofs demonstrating that for an adaptive mesh whose elements are of size $h = \frac{1}{\ell}H$, ℓ a multi-integer, and H the size of the largest element, we can obtain the correct convergence order and smoothness enhancement in the postprocessed results by choosing H as the scaling parameter. However, for an unstructured triangulation, neither the translation invariant property nor any interelement relation, as in the adaptive mesh case holds. Therefore, we require another mechanism to find the proper scaling parameter. As it is not straightforward to speculate as to the width of the kernel support (and hence the corresponding neighboring information) necessary to generate accuracy conservation and smoothness enhancement, we will investigate how different choices of the scaling parameter lead to different postprocessing results. We start by considering a scaling equal to the largest side of all the triangular elements, which we refer to as H. We then continue by investigating the impact of a kernel scaling smaller or larger than H on postprocessing DG solutions. In particular, in Section 7.2, we present postprocessing results using $0.5H$, $0.75H$, H, $1.5H$, and $2H$ as kernel scaling values.

To calculate the integral involved in the postprocessed solution in Equation (7.1) exactly, we need to decompose the triangular elements that are covered by the kernel support into subelements that respect the kernel knots (which we also refer to as kernel breaks); the resulting integral is calculated as the summation of the integrals over each subelement. Figure 7.2 depicts a possible kernel-mesh intersection for a sample triangular element of the unstructured triangular mesh shown in Figure 7.1. As it is shown in Figure 7.2(b), we divide the triangular region into subregions over which there is no break in regularity. Furthermore, we choose to triangulate these subregions for ease of implementation. Choosing H as the kernel scaling value in each direction, we can therefore rewrite Equation (7.1) as

$$\begin{aligned} u^\star(x,y) =& \frac{1}{H^2}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} K\left(\frac{x_1 - x}{H}\right) K\left(\frac{x_2 - y}{H}\right) u_h(x_1, x_2)dx_1dx_2 \\ =& \frac{1}{H^2}\sum_{T_j \in Supp\{K\}} \int\int_{T_j} K\left(\frac{x_1 - x}{H}\right) K\left(\frac{x_2 - y}{H}\right) u_h(x_1, x_2)dx_1dx_2 \end{aligned} \tag{7.2}$$

where we have used the compact support property of the kernel to transform the infinite integral to finite local sums over elements. Each of the integrals over a triangle T_j then

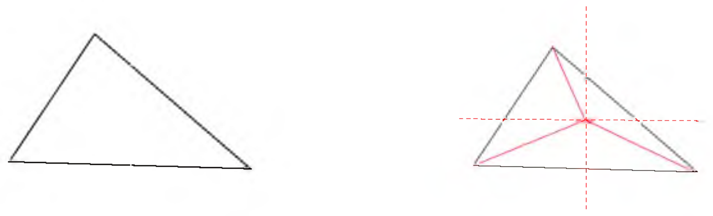

(a) Triangular element (b) Integration regions

Figure 7.2. Demonstration of integration regions resulted from the kernel-mesh intersection. Dashed lines represent the kernel breaks. Solid red lines represent a triangulation of the integration regions.

becomes

$$\begin{aligned}&\int\int_{T_j} K\left(\frac{x_1-x}{H}\right)K\left(\frac{x_2-y}{H}\right)u_h(x_1,x_2)dx_1dx_2\\ &=\sum_{n=0}^{N}\int\int_{\tau_n} K\left(\frac{x_1-x}{H}\right)K\left(\frac{x_2-y}{H}\right)u_h(x_1,x_2)dx_1dx_2 \qquad (7.3)\end{aligned}$$

where N is the total number of triangular subregions formed in the triangular element T_j as the result of kernel-mesh intersection.

We note here that to evaluate the integrals in Equation (8.6) exactly to machine precision, we first map via a Duffy transformation the triangular region τ_n to the standard triangular element defined as

$$T_{st} = \{(\xi_1,\xi_2)| -1 \leq \xi_1,\xi_2;\xi_1+\xi_2 \leq 0\}, \qquad (7.4)$$

and then we apply Guassian quadrature rules with enough quadrature points. For more information regarding the Gaussian quadrature and the various mappings involved in the integration, consult [46, 49, 40].

We further add that in order to find the footprint of the kernel on the DG mesh, we first lay a regular grid over our unstructured mesh. Each regular grid element contains the information of the triangles that intersect with it. In this way, we can easily find the extent of the kernel support on this regular grid and consequently compute the integration regions by solving a geometric intersection problem. For this, we apply the *Sutherland-Hodgman* clipping algorithm from computer graphics [69]. Next, we provide a brief overview of this algorithm.

7.1.1 The Sutherland-Hodgman Clipping Algorithm

The Sutherland-Hodgman clipping algorithm finds the polygon that is the intersection between an arbitrary polygon (the *subject polygon*) and a convex polygon (the *clip polygon*)

[69]. Figure 7.3 depicts a sample kernel-mesh overlap. We remind the reader that the convolution kernel used in the postprocessing algorithm is a linear combination of B-splines and therefore is a piecewise polynomial. Moreover, in two dimensions, it is the tensor product of one-dimensional kernels. Consequently, for implementation purposes, we will think of the footprint of the two-dimensional kernel as an array of squares, as depicted with red dashed lines in Figure 7.3 (left). Thereby, the problem of finding the integration regions becomes the problem of finding the intersection areas between each square of the kernel array (the clip polygon) and the triangular elements (the subject polygons) covered by the kernel support. Furthermore, it is clear that both the clip polygon and the subject polygons are convex.

To find the intersection area between a square of the kernel and a triangular element (Figure 7.3 (right)), we follow the Sutherland-Hodgman clipping algorithm and use a divide-and-conquer strategy. First, we clip the polygon (in our case, the triangular element) against the left clipping boundary (left side of the square in the kernel array). The resulting partially clipped polygon is then clipped against the top boundary, and then the process is repeated against the two remaining boundaries, as shown in Figure 7.4.

To clip against one boundary, the algorithm loops through all polygon vertices. At each step, it considers two of the vertices that we denote as *previous* and *current*. First, it determines whether these vertices are inside or outside the clipping boundary. This, of course, is a matter of comparing the horizontal or vertical position to the boundary's position. We then apply the following simple rules:

1. if the previous vertex and the current vertex are both inside the clipping boundary, output the current vertex,

2. if the previous vertex is inside the clipping boundary, and the current vertex is outside the clipping boundary, output the intersection point of the corresponding edge with the clipping boundary,

3. if the previous vertex and the current vertex are both outside the clipping boundary, then output nothing,

4. if the previous vertex is outside the clipping boundary and the current vertex is inside the clipping boundary, output the intersection point of the corresponding edge with the clipping boundary.

Following this procedure, we obtain a new polygon, clipped against one boundary, and ready to be clipped against the next boundary. Furthermore, we triangulate the resulting

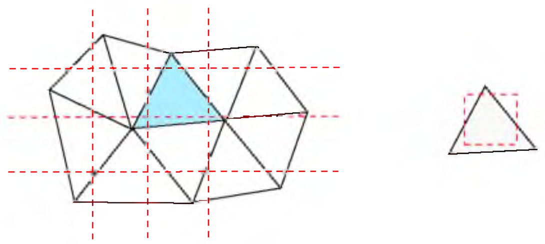


Figure 7.3. A sample kernel-mesh overlap (left). Dashed lines represent the two-dimensional kernel as an array of squares. In right, the intersection between a square of the kernel and a triangular element is shown.

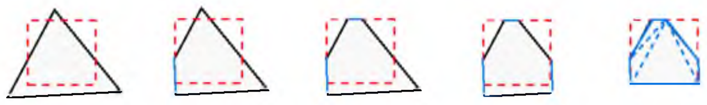


Figure 7.4. The Sutherland-Hodgman clipping. The final intersection area is triangulated for ease of implementation. Dashed red lines represent a square of the kernel, solid black lines represent the triangular DG element, solid blue lines represent the clipped area at each stage of the Sutherland-Hodgman algorithm and dashed blue lines represent the final triangulation of the integration region.

clipped area for ease of implementation of the quadrature rules (Figure 7.4 (right)). As the final triangulation is merely used for performing numerical quadrature, there is no rigorous requirement on the triangle element quality. We only require well-formed (*i.e.,* valid) triangles over which the approximate solution and the kernel can be evaluated and their product integrated.

7.2 Numerical Results

In this section, we provide postprocessing results that demonstrate the efficacy of the SIAC filter when applied to multiple unstructured triangulations. In all the examples, we consider the solutions of the constant coefficient linear advection equation given below:

$$u_t + u_x + u_y = 0, \qquad (x,y) \in (0,1) \times (0,1), \qquad T = 12.5 \tag{7.5}$$

with initial condition $u(0,x,y) = \sin(2\pi(x+y))$. We further note that to generate the various unstructured meshes, we used the Gmsh finite element mesh generator [29].

7.2.1 Simple Delaunay Triangulation

For this example, we consider a simple Delaunay triangulation of the domain given in Figure 7.1. As we discussed in Section 7.1, we consider the largest side of all the triangular elements and denote that with H. We then perform postprocessing using mH as the kernel scaling values, where $m = 0.5, 1.0, 1.5, 2.0$. Table 7.1 and Figure 7.5 provide the L_2-error results and plots for these scaling values. The $m = 2.0$ case is purposefully omitted from the table as this scaling is *not valid* for many of the meshes, *i.e.,* the kernel would become larger than the domain size. However, the $m = 2.0$ data are provided in the error plots when available. This omission has been done for all the tables presented herein. Generally speaking, as we increase the kernel width by using a larger m, the error decreases until it reaches a point where we obtain the minimal error value and it increases afterwards. For coarser mesh structures, it is often more beneficial to use a smaller kernel to avoid the not valid scenarios. We further note that the L_2-error values presented herein are always better than the initial DG errors by orders of magnitude.

In Figure 7.6, we present the point-wise error contour plots. As can be observed from these plots, the errors are highly oscillatory before the application of the postprocessor (left column). However, the postprocessor filters out the oscillations, and this effect is noticeably visible when using $\mathbb{P}^4$ polynomials. The magnitude of the error also decreases after postprocessing. Moreover, we get better results in terms of smoothness with a larger kernel; however, the error might increase in some cases.

Table 7.1. L_2-errors for various kernel scalings used in the SIAC filter for a simple Delaunay triangulation.

–	$m = 0.5$		$m = 0.75$		$m = 1.0$		$m = 1.5$	
Mesh	L_2 Error	Order	L_2 Error	Order	L_2 Error	Order	L_2 Error	Order
	$\mathbb{P}^2$							
878	1.14E-04	—	5.34E-05	—	6.54E-05	—	not valid	—
3588	1.40E-05	3.02	9.61E-06	2.47	6.20E-06	3.40	7.48E-06	—
14888	1.50E-06	3.22	1.02E-06	3.23	6.30E-07	3.30	2.30E-07	5.02
59264	2.07E-07	2.85	1.56E-07	2.70	1.09E-07	2.53	4.51E-08	2.35
	$\mathbb{P}^3$							
878	1.71E-06	—	8.09E-07	—	2.95E-06	—	not valid	—
3588	1.05E-07	4.02	3.09E-08	4.71	1.46E-08	7.65	2.43E-07	—
14888	8.55E-09	3.61	3.45E-09	3.16	1.20E-09	3.60	9.06E-10	8.06
59264	5.21E-10	4.03	2.48E-10	3.79	1.08E-10	3.47	1.87E-11	5.59
	$\mathbb{P}^4$							
878	2.37E-08	—	1.27E-08	—	not valid	—	not valid	—
3588	3.77E-09	2.65	3.69E-09	1.78	3.69E-09	—	not valid	—
14888	3.57E-11	6.72	7.39E-12	8.96	6.07E-12	9.24	1.01E-11	—
59264	2.44E-12	3.87	1.55E-12	2.25	1.03E-12	2.55	6.94E-13	3.86

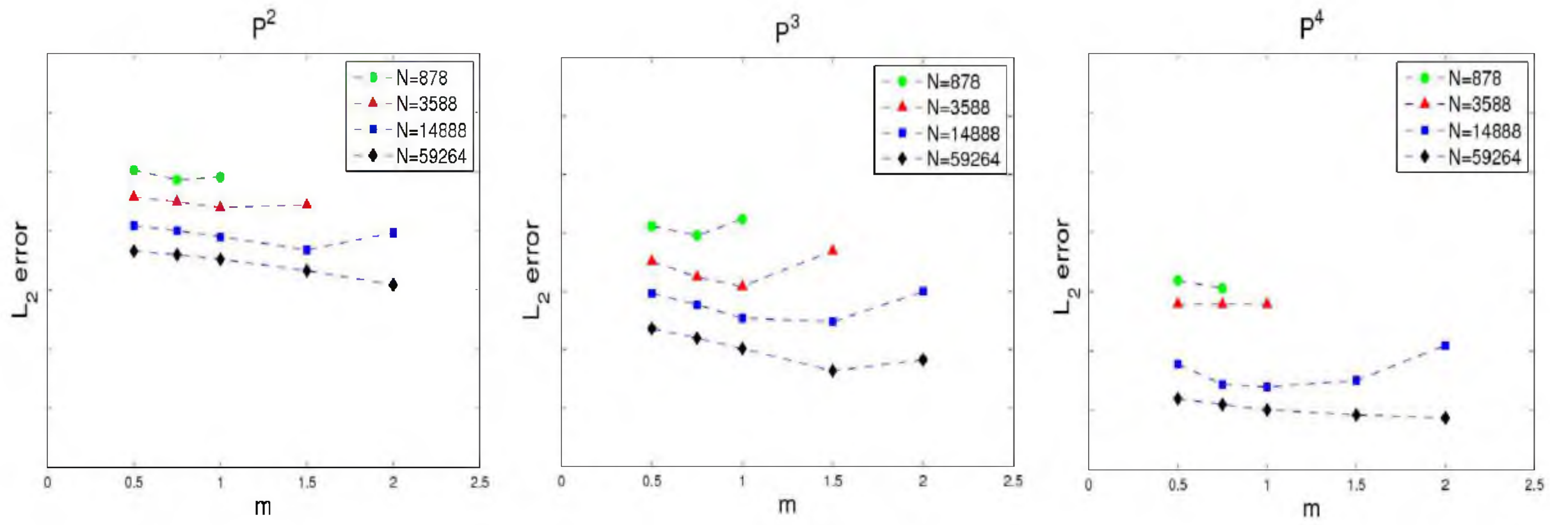


Figure 7.5. L_2-errors versus different kernel scalings when postprocessing over a simple Delaunay triangulation. Left: $\mathbb{P}^2$, middle: $\mathbb{P}^3$, and right: $\mathbb{P}^4$ polynomials. N represents the number of triangular elements in the mesh.

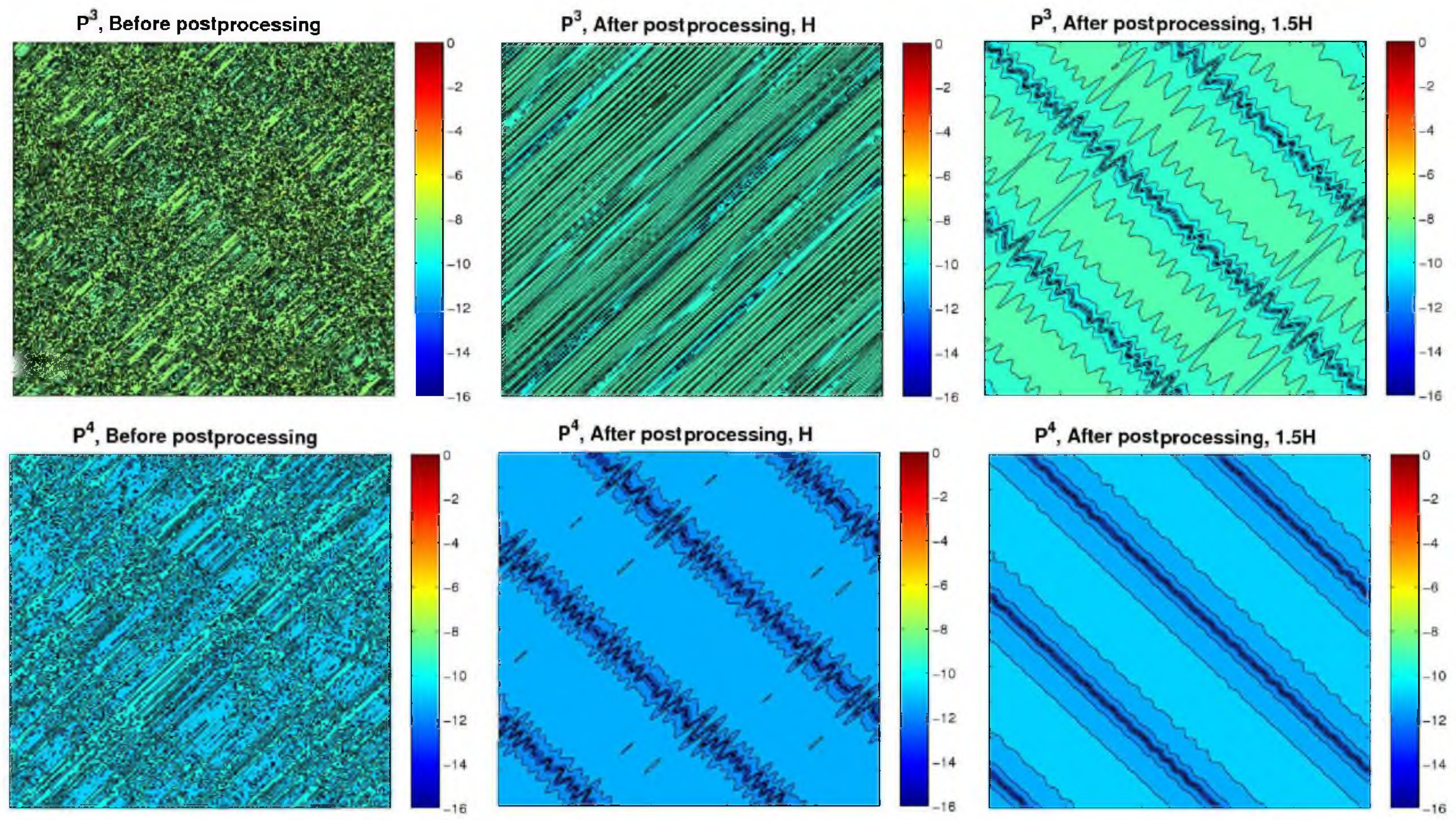
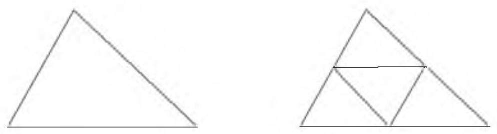


Figure 7.6. Point-wise error contour plots before and after postprocessing over a simple Delaunay triangulation with $N = 14888$ elements. Left column: before postprocessing; Middle column: H scaling; Right column: $1.5H$ scaling. Top row: $\mathbb{P}^3$ polynomials; Bottom row: $\mathbb{P}^4$ polynomials.

7.2.2 Delaunay Triangulation with Element Splitting

For this case, we took the unstructured mesh in Figure 7.1 and refined it with splitting (Figure 7.7). We suspected that this would give better postprocessing results due to the natural hierarchy of solution spaces generated. Table 7.2 and Figure 7.8 provide the L_2-error values with respect to different sizes of kernel scalings. Again, it is noted that generally, there is an optimal kernel scaling value for which we obtain the minimum L_2-error. In addition, Figure 7.9 presents the point-wise error contour plots. Comparing to the contour plots in Figure 7.6, these provide much smoother error values.



(a) Triangular Element (b) Splitting

Figure 7.7. Refining a sample triangular element by splitting.

Table 7.2. L_2-errors for various kernel scalings used in the SIAC filter for a triangulation with element splitting.

–	$m = 0.5$		$m = 0.75$		$m = 1.0$		$m = 1.5$	
Mesh	L_2 Error	Order	L_2 Error	Order	L_2 Error	Order	L_2 Error	Order
	$\mathbb{P}^2$							
776	1.10E-04	—	7.08E-05	—	1.25E-04	—	not valid	—
3104	1.22E-05	3.17	7.84E-06	3.17	6.45E-06	4.27	not valid	—
12416	1.46E-06	3.06	8.24E-07	3.25	5.02E-07	3.68	1.98E-06	—
49664	1.80E-07	3.15	1.09E-07	2.20	5.97E-08	3.07	8.11E-08	4.60
	$\mathbb{P}^3$							
776	1.47E-06	—	9.88E-07	—	8.52E-06	—	not valid	—
3104	1.23E-07	3.57	2.71E-08	5.18	1.30E-07	6.03	not valid	—
12416	1.17E-08	3.39	3.28E-09	6.02	1.99E-09	6.02	4.58E-08	—
49664	1.05E-09	3.47	2.34E-10	3.80	5.85E-11	5.08	6.20E-10	6.20
	$\mathbb{P}^4$							
776	2.68E-08	—	4.48E-08	—	not valid	—	not valid	—
3104	4.84E-10	5.79	2.52E-10	7.47	4.07E-09	—	not valid	—
12416	2.76E-11	4.13	6.66E-12	5.24	2.05E-11	7.63	not valid	—
49664	1.49E-12	4.21	7.81E-13	3.09	7.12E-13	4.85	5.17E-12	—

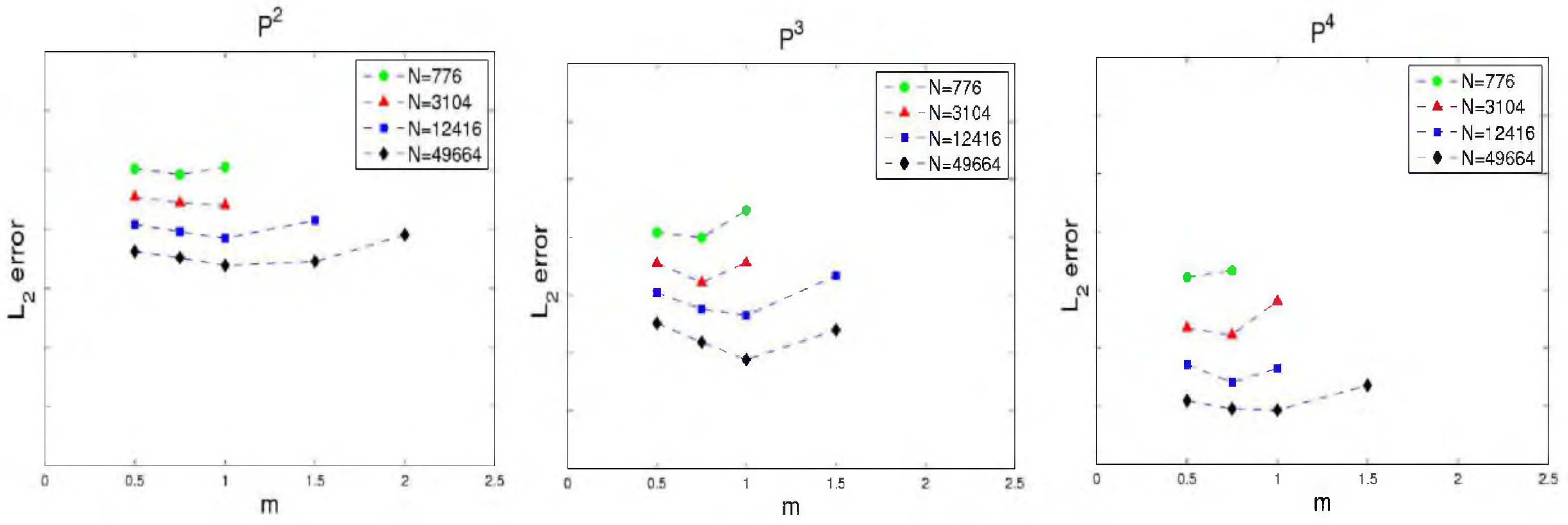


Figure 7.8. L_2-errors versus different kernel scalings when postprocessing over a triangulation with element splitting. Left: $\mathbb{P}^2$, middle: $\mathbb{P}^3$, and right: $\mathbb{P}^4$ polynomials. N represents the number of triangular elements in the mesh.

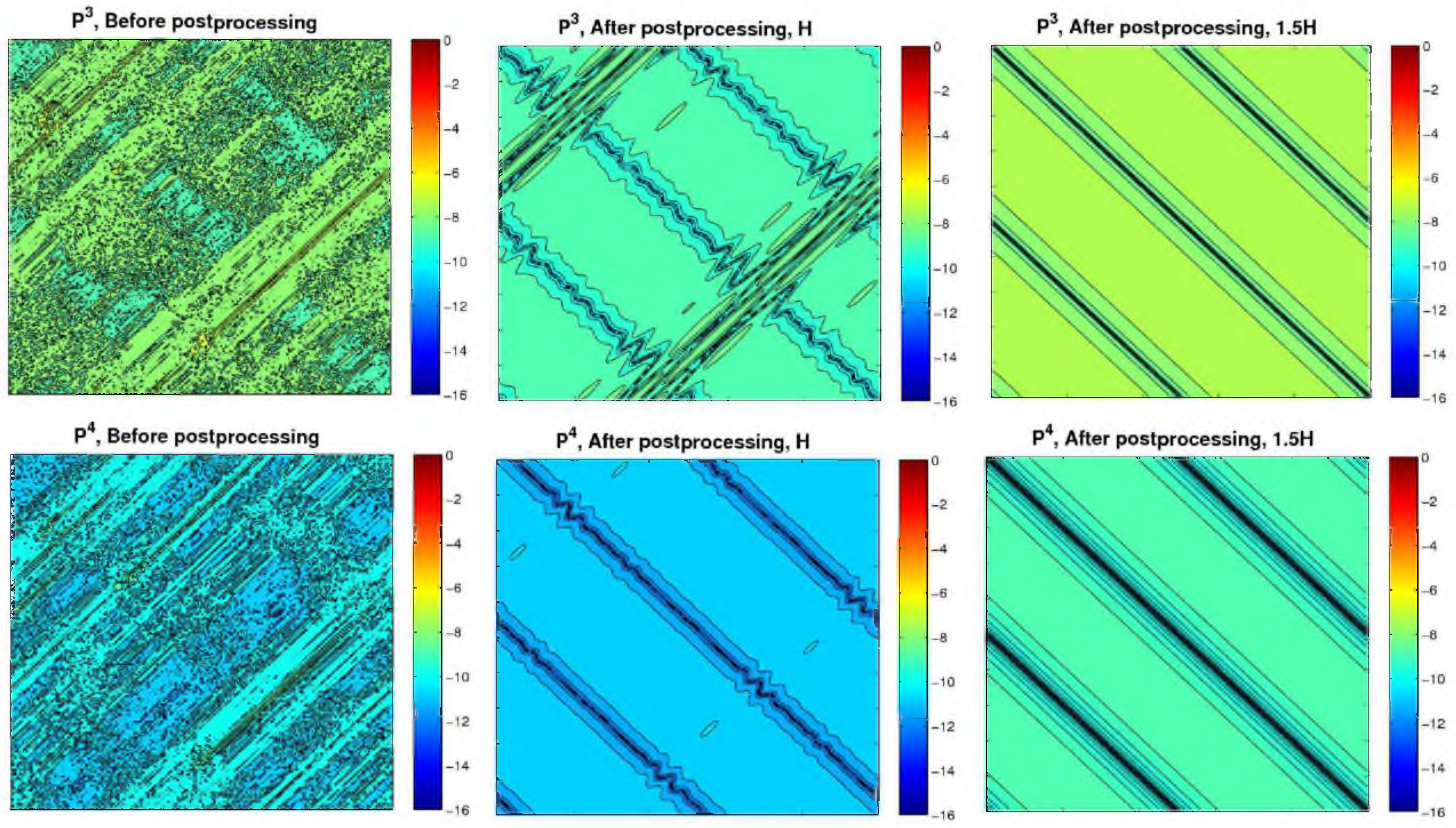


Figure 7.9. Point-wise error contour plots before and after postprocessing over a triangulation with element splitting with $N = 12416$ elements. Left column: before postprocessing; Middle column: H scaling; Right column: $1.5H$ scaling. Top row: $\mathbb{P}^3$ polynomials; Bottom row: $\mathbb{P}^4$ polynomials.

7.2.3 Delaunay Triangulation with Variable-Sized Elements

In this example, we examine two variants of the Delaunay triangulation of the domain, shown in Figure 7.10, where there is an obvious spatial transition in element size in the interior of the domain. This change was made in the middle of the mesh in order to maintain the periodic boundary conditions and simplify the application of the postprocessor.

Table 7.3 and Figure 7.11 present the L_2-errors for postprocessing over the Mesh Example 1 in Figure 7.10 using different kernel scaling values. Moreover, Figure 7.12 provides the point-wise error contour plots for this mesh. We observe similar postprocessing behavior to the previous mesh examples. Postprocessing results for the Mesh Example 2 are provided in Table 7.4 and Figures 7.13 and 7.14.

7.2.4 Delaunay Triangulation with Stretched Elements

Here, we consider a sample Delaunay mesh with stretched elements in the x-direction (Figure 7.15). This is the so-called *anisotropic* unstructured mesh and is often the type of mesh we see in practice when simulating flows which have strong preferential direction. Table 7.5 and Figures 7.16 and 7.17 provide the postprocessing results. Again, through the application of the SIAC filter, we are able to smooth out the oscillations in the error and obtain lower error values.

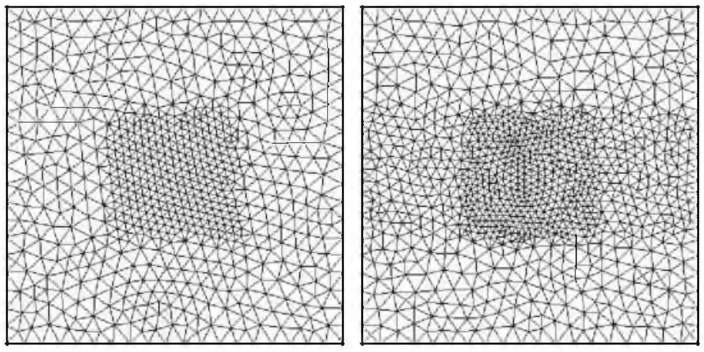

(a) Mesh Example 1 (b) Mesh Example 2

Figure 7.10. Examples of variable-sized unstructured Delaunay triangulation.

Table 7.3. L_2-errors for various kernel scalings used in the SIAC filter for the Mesh Example 1.

–	$m = 0.5$		$m = 0.75$		$m = 1.0$		$m = 1.5$	
Mesh	L_2 Error	Order	L_2 Error	Order	L_2 Error	Order	L_2 Error	Order
	$\mathbb{P}^2$							
1350	6.74E-05	—	4.68E-05	—	5.30E-05	—	not valid	—
5662	1.30E-05	2.37	7.59E-06	2.62	3.75E-06	3.82	6.78E-06	—
22960	1.66E-06	2.96	1.19E-06	2.67	7.43E-07	2.33	2.91E-07	4.54
90682	1.95E-07	3.08	1.44E-07	3.04	9.53E-08	2.96	3.37E-08	3.11
	$\mathbb{P}^3$							
1350	9.99E-07	—	3.90E-07	—	2.06E-06	—	not valid	—
5662	9.84E-08	3.34	2.32E-08	4.07	1.14E-08	7.49	2.20E-07	—
22960	7.59E-09	3.69	2.70E-09	3.10	9.15E-10	3.63	8.53E-10	8.01
90682	5.04E-10	3.91	1.87E-10	3.85	5.93E-11	3.94	1.05E-11	6.34
	$\mathbb{P}^4$							
1350	1.33E-08	—	8.41E-09	—	not valid	—	not valid	—
5662	6.22E-10	4.41	1.01E-10	6.37	1.41E-10	—	not valid	—
22960	2.56E-11	4.60	7.12E-12	3.82	6.08E-12	4.53	9.60E-12	—
90682	2.17E-12	3.56	1.38E-12	2.36	1.04E-12	2.55	6.83E-13	3.81

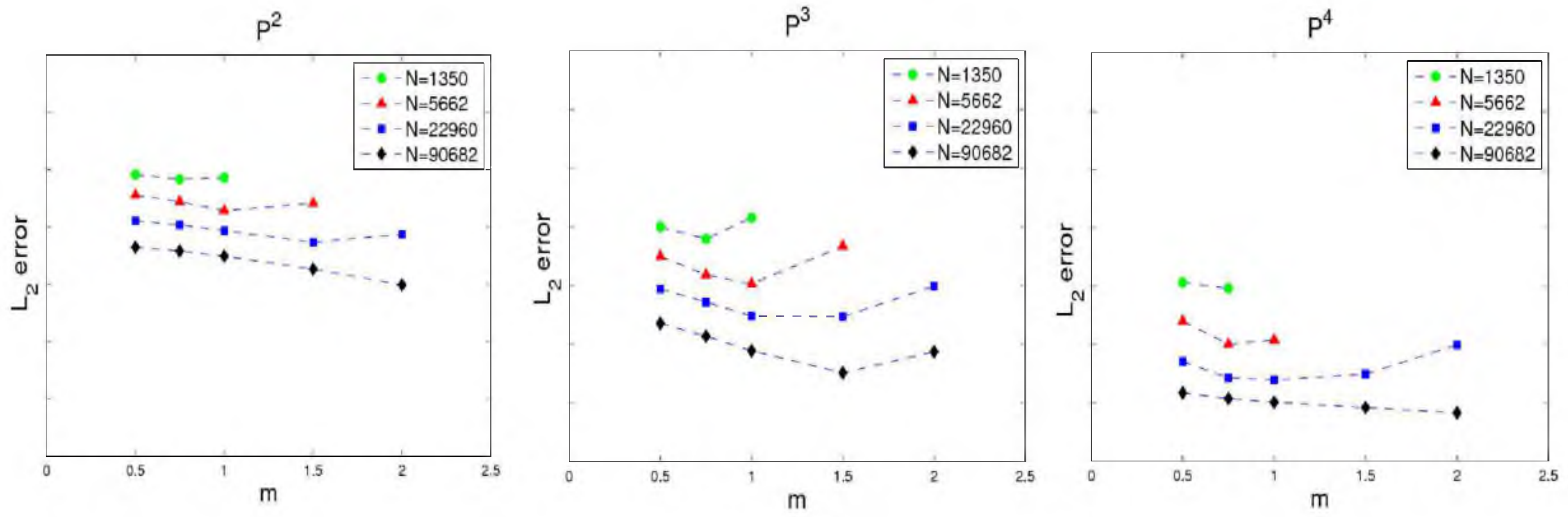


Figure 7.11. L_2-errors versus different kernel scalings when postprocessing over the Mesh Example 1. Left: $\mathbb{P}^2$, middle: $\mathbb{P}^3$, and right: $\mathbb{P}^4$ polynomials. N represents the number of triangular elements in the mesh.

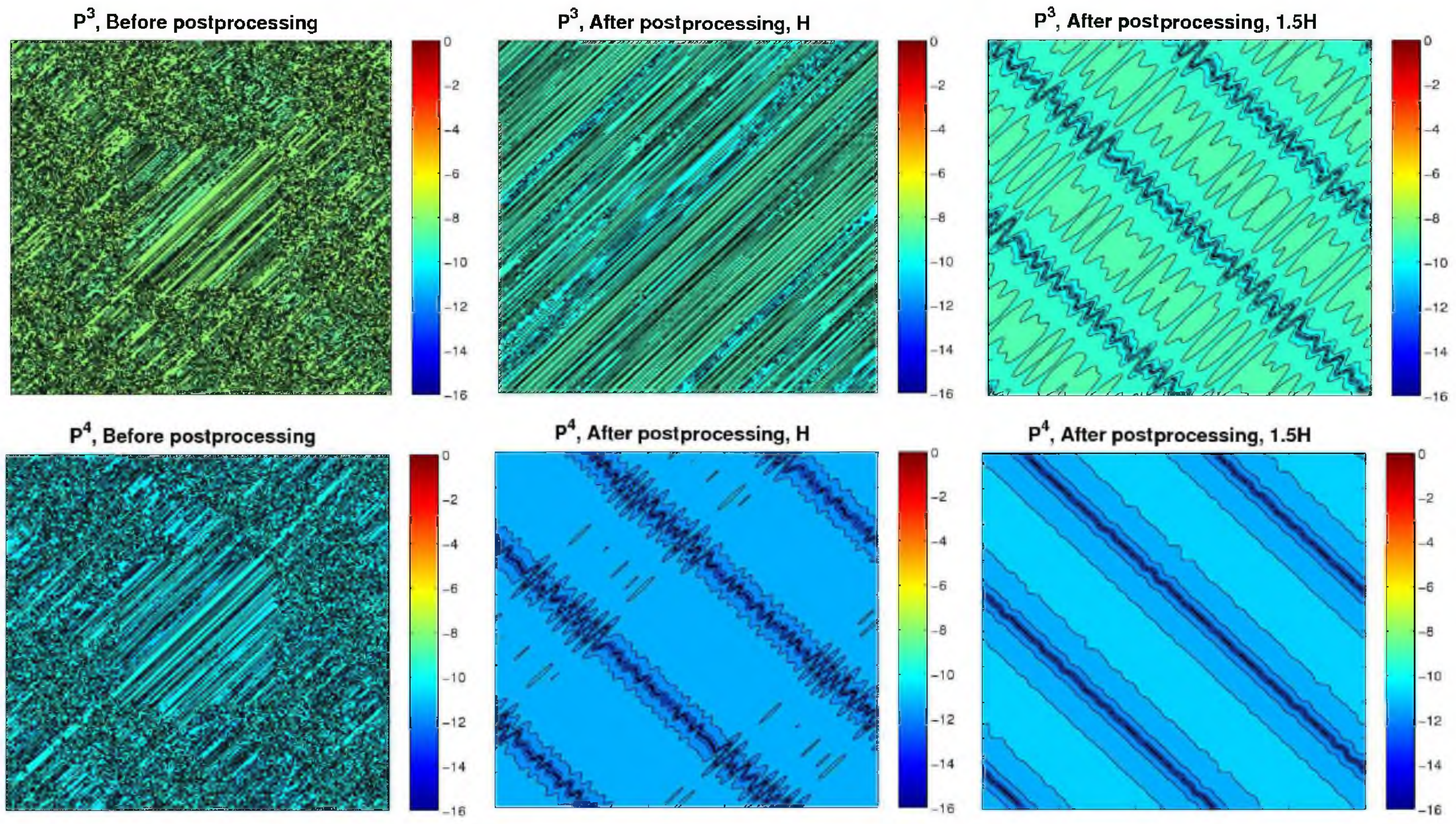


Figure 7.12. Point-wise error contour plots before and after postprocessing over the Mesh Example 1 with $N = 22960$ elements. Left column: before postprocessing; Middle column: H scaling; Right column: $1.5H$ scaling. Top row: $\mathbb{P}^3$ polynomials; Bottom row: $\mathbb{P}^4$ polynomials.

Table 7.4. L_2-errors for various kernel scalings used in the SIAC filter for the Mesh Example 2.

–	$m = 0.5$		$m = 0.75$		$m = 1.0$		$m = 1.5$	
Mesh	L_2 Error	Order	L_2 Error	Order	L_2 Error	Order	L_2 Error	Order
	$\mathbb{P}^2$							
1972	6.61E-05	—	4.40E-05	—	5.02E-05	—	not valid	—
8240	9.06E-06	2.86	4.69E-06	3.22	2.31E-06	4.44	6.52E-06	—
34562	1.18E-06	2.94	7.99E-07	2.55	4.68E-07	2.30	1.68E-07	5.27
138254	1.38E-07	3.09	9.60E-08	3.05	6.14E-08	2.93	2.32E-08	2.84
	$\mathbb{P}^3$							
1972	1.04E-06	—	3.97E-07	—	2.27E-06	—	not valid	—
8240	7.02E-08	3.88	2.01E-08	4.30	1.10E-08	7.68	2.17E-07	—
34562	6.35E-09	3.46	1.70E-09	3.56	4.01E-10	4.77	8.11E-10	8.06
138254	4.58E-10	3.79	1.65E-10	3.36	4.27E-11	3.23	7.08E-12	6.83
	$\mathbb{P}^4$							
1972	8.71E-09	—	9.29E-09	—	not valid	—	not valid	—
8240	3.75E-10	4.53	2.34E-10	5.31	2.69E-10	—	not valid	—
34562	2.06E-11	4.18	1.46E-11	4.00	1.45E-11	4.21	1.62E-11	—
138254	2.15E-12	3.26	1.27E-12	3.52	9.04E-13	4.00	6.28E-13	4.68

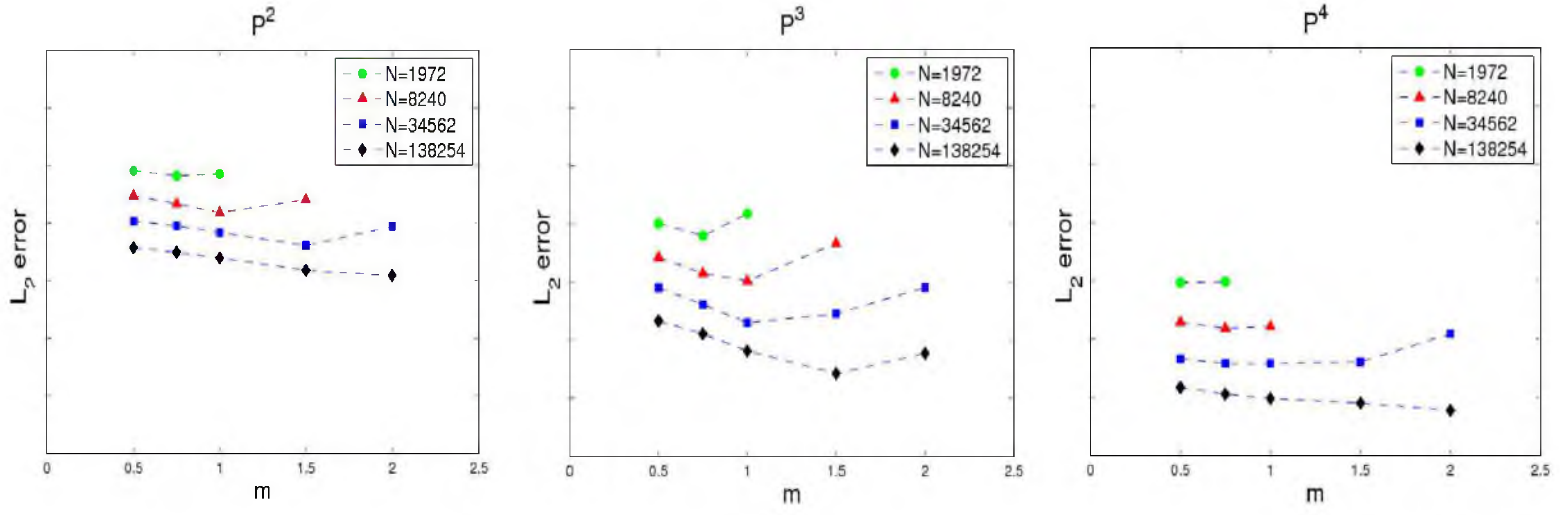


Figure 7.13. L_2-errors versus different kernel scalings when postprocessing over the Mesh Example 2. Left: $\mathbb{P}^2$, middle: $\mathbb{P}^3$, and right: $\mathbb{P}^4$ polynomials. N represents the number of triangular elements in the mesh.

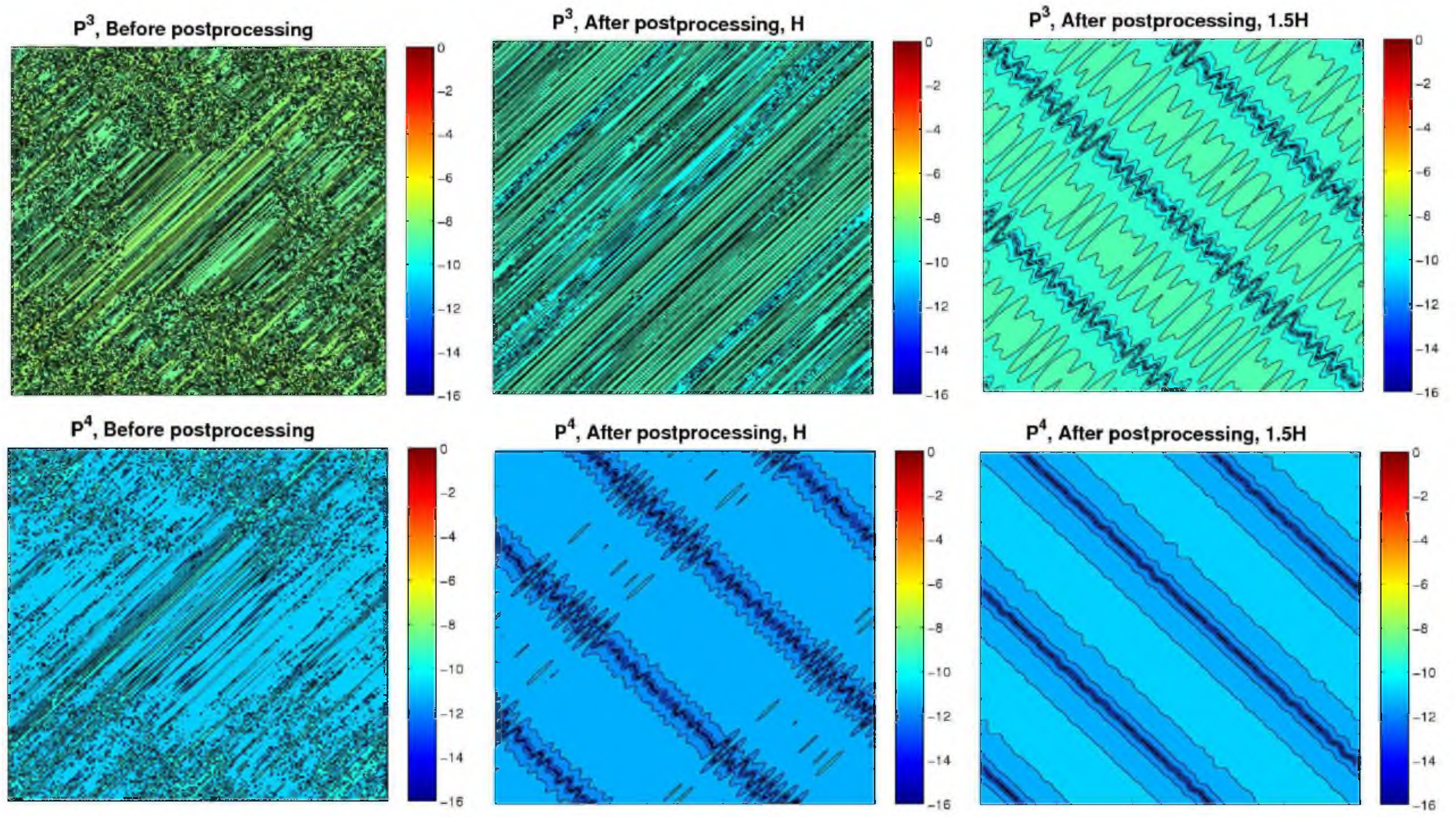


Figure 7.14. Point-wise error contour plots before and after postprocessing over the Mesh Example 2 with $N = 34562$ elements. Left column: before postprocessing; Middle column: H scaling; Right column: $1.5H$ scaling. Top row: $\mathbb{P}^3$ polynomials; Bottom row: $\mathbb{P}^4$ polynomials.

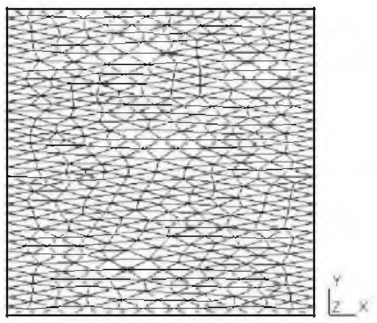


Figure 7.15. A sample unstructured triangular mesh with stretched elements in the x-direction.

Table 7.5. L_2-errors for various kernel scalings used in the SIAC filter for a stretched triangulation.

–	$m = 0.5$		$m = 0.75$		$m = 1.0$		$m = 1.5$	
Mesh	L_2 Error	Order	L_2 Error	Order	L_2 Error	Order	L_2 Error	Order
	$\mathbb{P}^2$							
796	3.38E-04	—	2.96E-04	—	not valid	—	not valid	—
3232	3.48E-05	3.27	1.86E-05	3.99	3.31E-05	—	not valid	—
12816	4.10E-06	3.08	2.51E-06	2.88	1.62E-06	4.35	5.54E-06	—
51430	6.79E-07	2.59	4.12E-07	2.60	2.13E-07	2.92	1.62E-07	5.09
	$\mathbb{P}^3$							
796	5.25E-06	—	not valid	—	not valid	—	not valid	—
3232	5.13E-07	3.35	2.04E-07	—	1.72E-06	—	not valid	—
12816	3.10E-08	4.04	6.46E-09	4.98	7.44E-09	7.85	1.79E-07	—
51430	2.23E-09	3.79	5.05e-10	3.67	1.80E-10	5.36	1.45E-09	6.94
	$\mathbb{P}^4$							
796	1.07E-07	—	not valid	—	not valid	—	not valid	—
3232	4.40E-09	4.60	7.07E-09	—	not valid	—	not valid	—
12816	7.38E-11	5.89	9.61E-12	9.52	1.08E-10	—	not valid	—
51430	2.60E-12	4.82	8.17E-13	3.55	7.00E-13	7.26	1.49E-11	—

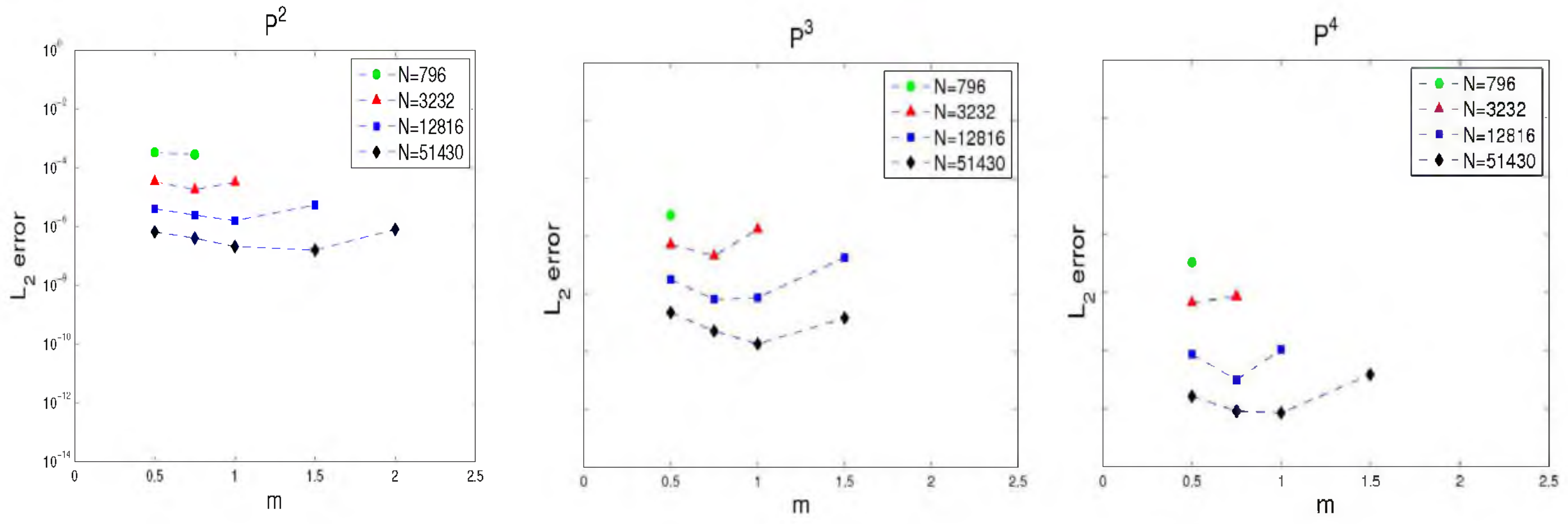


Figure 7.16. L_2-errors versus different kernel scalings when postprocessing over a stretched triangulation. Left: $\mathbb{P}^2$, middle: $\mathbb{P}^3$, and right: $\mathbb{P}^4$ polynomials. N represents the number of triangular elements in the mesh.

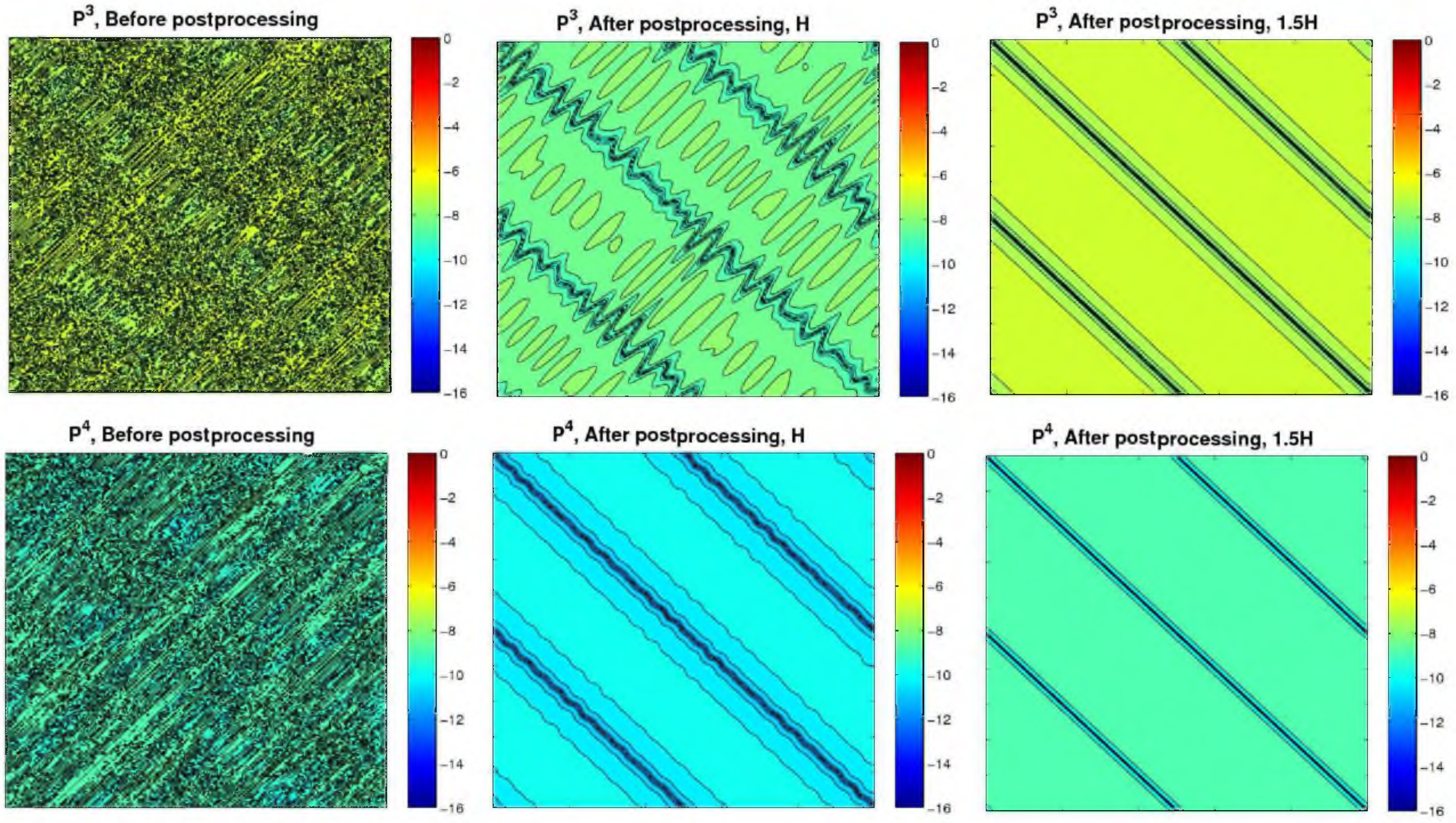


Figure 7.17. Point-wise error contour plots before and after postprocessing over a stretched triangulation with $N = 12816$ elements. Left column: before postprocessing; Middle column: H scaling; Right column: $1.5H$ scaling. Top row: $\mathbb{P}^3$ polynomials; Bottom row: $\mathbb{P}^4$ polynomials.

7.3 Conclusion

The smoothness-increasing accuracy-conserving filtering technique has traditionally been applied to translation invariant meshes. In some cases, random meshes in one dimension and structured smoothly varying meshes in two dimensions were also considered. However, as the assumption of any sort of regularity will restrict the application of this filtering technique to more complex simulations, we provided in this chapter the behavior and complexity of the computational extension of this filtering technique to totally unstructured tessellations. We note that this is an important step towards a more generalized SIAC filter. We considered various unstructured tessellations and demonstrated that it is indeed possible to get reduced errors and improved smoothness through a proper choice of kernel scaling. Lastly, GPU implementations of this SIAC filter were described. Using a single GPU, up to a 18$\times$ reduction in computational costs over the traditional CPU implementations can be obtained for these unstructured tessellations. We documented why care must be taken with the programming of the GPUs to obtain such a reduction when applied to SIAC postprocessing of DG solutions.

CHAPTER 8

APPLICATION OF THE SIAC FILTERING TO STRUCTURED TETRAHEDRAL MESHES

In this chapter, we attempt to address the potential usefulness of smoothness-increasing accuracy-conserving (SIAC) filters when applied to real-world simulations. As presented so far, the application of the SIAC filter never exceeded beyond two-space dimensions for time-dependent simulations. As tetrahedral meshes are often the type considered in more realistic scenarios, we contribute to the class of SIAC postprocessors by demonstrating the effectiveness of SIAC filtering when applied to structured tetrahedral meshes. These types of meshes are generated by tetrahedralizing uniform hexahedra and therefore, while maintaining the structured nature of a hexahedral mesh, they exhibit an unstructured tessellation within each hexahedral element. Moreover, we address the computationally intensive task of performing numerical integrations when one considers tetrahedral elements for SIAC filtering and provide guidelines on how to ameliorate these challenges through the use of more general cubature rules. We consider two examples of a hyperbolic equation and confirm the usefulness of SIAC filters in obtaining the superconvergence accuracy of $2k+1$ when applied to structured tetrahedral meshes. Additionally, as these filters improve the weak continuity in the DG method to $k-1$ continuity, we provide results that show how postprocessing is useful in extracting smooth isosurfaces of DG fields.

We proceed in this chapter by briefly discussing the theoretical foundations in SIAC filtering of DG solutions and how it applies to structured tetrahedral meshes. We then continue by presenting the detail of the implementation as well as important practical considerations. Section 8.2 provides numerical results confirming the usefulness of post-processing over structured tetrahedral meshes. The results of this contribution have been documented in a submitted manuscript [50].

8.1 Smoothness-Increasing Accuracy-Conserving Filter for Structured Tetrahedral Meshes

In this chapter, we consider accuracy enhancement of numerical solutions to three-dimensional linear hyperbolic equations of the form

$$\begin{aligned} u_t + \sum_{i=1}^{3} \frac{\partial}{\partial x_i}(A_i(\mathbf{x})u) &= 0, & \mathbf{x} \in \Omega \times [0,T], \\ u(\mathbf{x},0) &= u_o(\mathbf{x}), & \mathbf{x} \in \Omega \end{aligned} \tag{8.1}$$

where $\Omega \in \mathbb{R}^3$, and $A_i(\mathbf{x}),\ i = 1,2,3$ is bounded in the $L^\infty-$norm. We also assume smooth initial conditions are given along with periodic boundary conditions.

We continue by restating the main theorem in SIAC filtering of DG solutions, which was thoroughly discussed in Chapter 6:

Theorem 8.1.1 *Let u_h be the DG solution to the linear hyperbolic equation given in Equation* (8.1)*. The approximation is taken over a mesh whose elements are of size $h = \frac{1}{\ell}H$ in each coordinate direction where ℓ is a multi-integer (of dimension equal to the number of elements along one coordinate direction) and H represents the macro-element size of which any particular element is generated by hierarchical integer partitioning of the macro-element. Given sufficient smoothness in the initial data,*

$$\|u - K_H \star u_h\|_\Omega \leq \mathsf{C}H^{2k+1}, \tag{8.2}$$

where K_H is the postprocessing kernel in the SIAC filter scaled by H.

Comparing to the original theorem in [19], Theorem 8.1.1 encompasses a broader range of mesh structures. More detail along with several numerical examples can be found in Chapter 6.

As discussed in previous chapters, if within each macro-element of size H we assume equal partitioning, the resulting mesh will have a translation invariant structure. A translation invariant mesh is a type of mesh in which we can identify a repeating pattern [7]. Consequently, according to Theorem 8.1.1, we are able to obtain higher order accuracy in the L^2-norm when postprocessing translation invariant meshes. The structured tetrahedral mesh we consider in this paper will fall into this category.

To generate a translation invariant structured tetrahedral mesh, we first split the domain into uniform hexahedral elements and then subdivide each hexahedral element into tetrahedra. As it is mentioned in [26], there are eight different ways to divide a hexahedron into tetrahedra. These configurations are shown in Figure 8.1. As you notice from this figure, the

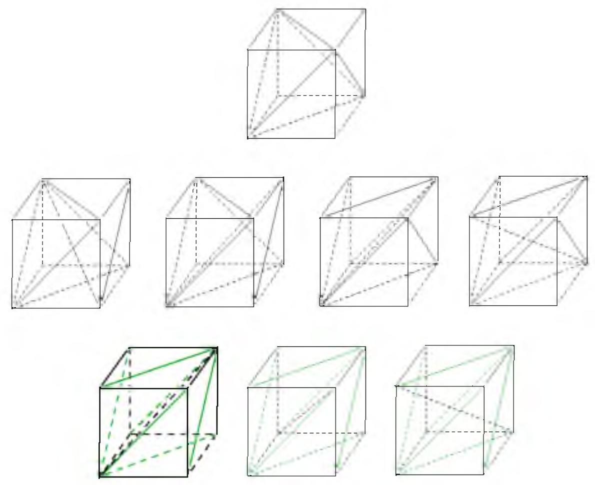


Figure 8.1. All possible subdivisions of a hexahedral element into five and six tetrahedra. When juxtaposing the hexahedral elements, it is necessary to flip the hexahedron in x-, y-, or z-direction with the first five configurations. No flipping is required with the configurations in the bottom row. We consider the lower left configuration as our structured tetrahedral mesh.

top configuration leads to 5 tetrahedra per element while the rest lead to six tetrahedra. Another point worth mentioning is that in the first five configurations (first and second row in Figure 8.1), we sometimes need to flip the hexahedral element when constructing the entire mesh so that the diagonal edges of adjacent hexahedra align. We are, however, not required to do that if we choose any of the last three configurations (bottom row in Figure 8.1). In either case, we are always able to identify a repeating pattern within these structured meshes. Here, we have chosen the lower left configuration in Figure 8.1 as our structured tetrahedral mesh. We continue by providing the detail of the implementation of the SIAC filter over structured tetrahedral meshes.

The convolution kernel in three dimensions is formed by the tensor product of one-dimensional kernels. That is

$$\hat{K}(x,y,z) = K(x) \times K(y) \times K(z). \tag{8.3}$$

Consequently the postprocessor in three dimensions will have the following form:

$$u^\star(x,y,z) = \\ \frac{1}{H_1H_2H_3}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} K\left(\frac{x_1-x}{H_1}\right)K\left(\frac{x_2-y}{H_2}\right)K\left(\frac{x_3-x}{H_1}\right)u_h(x_1,x_2,x_3)dx_1dx_2dx_3, \tag{8.4}$$

where u_h is the approximate DG solution of the numerical simulation and H_i, $i=1,2,3$ are the kernel scaling parameters in each direction.

We note that in Chapters 5 and 7, we thoroughly discussed the extension of the SIAC filter to structured and unstructured triangular meshes, respectively. Similarly, here we have taken the existing implementation of this filtering technique and applied it to structured tetrahedral meshes.

The convolution kernel in the SIAC filter along with the DG approximation u_h are piecewise polynomials. Therefore, to numerically evaluate the integral in Equation (8.4) exactly to machine precision, we need to subdivide the integration domain into regions of sufficient continuity. Previously in Chapter 7, we demonstrated that these integration regions can be found by solving a geometric intersection problem between a square and a triangle for triangular meshes. In three dimensions, the footprint of the kernel is contained in a cube that is further subdivided by the kernel knots into smaller cubes of $H_1 \times H_2 \times H_3$ dimensions. As a result, to find the regions of continuity as shown in Figure 8.2, we find the intersection region between a tetrahedral element and a cube. For this, we again apply the Sutherland-Hodgman clipping algorithm from computer graphics [69].

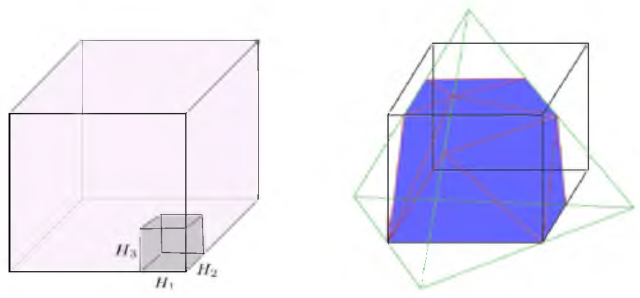
Following Theorem 8.1.1, the scaling parameters H_i, $i = 1, 2, 3$, which determine the extent of the kernel on the DG mesh, will be equal to the translation invariance of the mesh. This is necessary in order to observe the proper order of convergence in the L^2- norm after the application of the SIAC filter. It is therefore clear that H_i is equal to the uniform mesh spacing for the configurations in the bottom row of Figure 8.1 as the entire mesh could be constructed by exactly repeating the hexahedral element. However, for the other configurations, whenever we perform a flip in a direction i, the scaling H_i will be twice as large as the uniform mesh spacing.

To evaluate the postprocessed solution at a point denoted by (x, y, z), we center the kernel at that point. We then find the intersection regions and evaluate the resulted integrals. Therefore, the integral in Equation (8.4) now becomes

$$\begin{aligned} u^\star(x,y,z) =& \frac{1}{H^3}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \bar{K}(x_1)\bar{K}(x_2)\bar{K}(x_3)u_h(x_1,x_2,x_3)dx_1dx_2dx_3 \\ =& \frac{1}{H^3}\sum_{T_j\in Supp\{\hat{K}\}}\int_{T_j} \bar{K}(x_1)\bar{K}(x_2)\bar{K}(x_3)u_h(x_1,x_2,x_3)dx_1dx_2dx_3 \end{aligned} \tag{8.5}$$

where we have denoted $\bar{K}(x_i) = K(\frac{x_i - x}{H})$ for simplicity, and $Supp\{\hat{K}\}$ contains all the tetrahedral elements T_j that intersect with the kernel footprint.

We note that the final integration region resulted as the kernel-mesh intersection is itself a polyhedron. For ease of implementation, we further tetrahedralize this polyhedron



(a) Kernel footprint (b) Kernel-mesh intersection

Figure 8.2. Footprint of a three-dimensional kernel (a). Demonstration of an intersection between a tetrahedral element and a cube of the kernel footprint (b).

by triangulating its faces (as shown in Figure 8.2b) and connecting the resulting triangles to the centroid of the polyhedron. Consequently, the integral in Equation (8.5) becomes

$$\begin{aligned}&\int_{T_j} \bar{K}(x_1)\bar{K}(x_2)\bar{K}(x_3)u_h(x_1,x_2,x_3)dx_1dx_2dx_3 \\ =&\sum_{n=0}^{N}\int_{\tau_n} \bar{K}(x_1)\bar{K}(x_2)\bar{K}(x_3)u_h(x_1,x_2,x_3)dx_1dx_2dx_3 \end{aligned} \tag{8.6}$$

where N is the total number of tetrahedral subregions formed in the tetrahedral element T_j as the result of kernel-mesh intersection.

By numerically computing the integral in Equation (8.6) using a quadrature technique, we can now evaluate the postprocessed solution $u^*(x,y,z)$.

8.1.1 Practical Considerations

From the computational perspective, postprocessing over tetrahedral meshes is a very challenging task. Let us consider again the postprocessing formula given in Equation (8.4). Following our discussion in the previous section, in computing the postprocessed value at a single point (x,y,z), there exist three distinct steps:

1. Centering the kernel at (x,y,z) and identifying the support of the kernel over the DG mesh.

2. Solving a geometric intersection problem to obtain the integration regions.

3. Numerically evaluating the integrals by a means of quadrature rule.

In the case of structured tetrahedral meshes, it suffices to find the extent of the kernel on the basic hexahedral mesh which has a uniform structure. It is trivial that the footprint of the kernel over such a uniform mesh can be found in constant computational time. In the case of unstructured meshes, as it was performed in Chapter 7, we can always group the unstructured mesh elements within a regular grid. Consequently, we conclude that Step 1 has always a constant computational complexity.

To find the integration regions in Step 2, as mentioned in the previous section, we perform the Sutherland-Hodgman clipping algorithm. In this algorithm, we loop through the faces of one polyhedron and clip it against the second polyhedron. The computational complexity of this algorithm is $\mathcal{O}(n)$, where $n = f_1 \times f_2$ and is equal to 24 when finding the intersection between a cube and a tetrahedron. f_i, $i = 1, 2$ indicate the number of faces in each polyhedron. For structured tetrahedral meshes, the support of the kernel spans $3k+1$ hexahedral elements in each direction, k being the degree of the approximation. That is, for each evaluation point, we need to process $6 \times (3k+1)^3$ tetrahedra for the configuration we chose in this paper. As each tetrahedral element intersects with at most 8 cubes of the kernel, the cost of finding all the intersection regions will therefore be $8 \times 6 \times 24(3k+1)^3$ or $\mathcal{O}(k^3)$.

Given the amount of processing we need to perform in Step 2, we should maintain the computational cost of Step 3 as low as possible in order to have a tractable postprocessing algorithm. It appears that the main computational bottleneck in postprocessing over tetrahedral meshes lies in evaluation of the integrals. The key point to consider here is that the three-dimensional integral over a tetrahedral region τ_n as given in Equation (8.6) is in fact an expensive operator to evaluate due to several function evaluations. To understand why this is the case, consider quadrilateral and hexahedral meshes. For these mesh structures, the tensor product nature of the convolution kernel in higher dimensions would result in separation of the integrals and ultimately, the multidimensional integral could essentially be evaluated by computing one-dimensional integrals (refer to Chapter 3). This evaluation could even further be simplified if one also considers tensorial basis functions to represent the DG approximation in multidimensions [40]. Consequently, using tensor product Gaussian quadrature rules provides a convenient way to numerically evaluate the integrals involved in the convolution when dealing with tensor product mesh structures such as quadrilateral and hexahedral meshes. However, in the case of triangular or tetrahedral meshes, due to the dependency of the coordinate directions, the convolution integral is not separable (see Chapter 3 for triangles) and therefore, we can not reduce the cost of computing the 3D integral through evaluating 1D integrals. As a result, using the conventional tensor product quadrature rules will not be optimal in the sense of using the fewest function evaluations for a given approximation degree. A suitable alternative here is to use nontensor product formulas -generally known as *cubature* rules. The cubature rules are complicated to derive and are not known to very high orders. For our postprocessing experiments, we used the pregenerated points and weights by Zhang *et al.* in [76] and available for polynomials up to degree 14. We note that from equation (8.6), it is clear that the cubature rule we apply should be exact to integrate a polynomial integrand of degree $4k$. Tables 8.1 and 8.2 provide the number of points (see [76]) required to evaluate the convolution integral over each region of continuity using the aforementioned quadrature strategies. While there is no substantial difference in the number of quadrature points for the case of triangular elements, there is a noticeable difference in terms of computational efficiency when using cubature rules for tetrahedral meshes over tensor product quadrature. Note that we could not find the cubature points for a $k = 4$ DG approximation (polynomial integrand of degree 16). However, in practice, we were able to use even fewer cubature points, that are required to integrate a lower degree polynomial, to evaluate the convolution operator. In fact, using only 24 points for a P^2 approximation, 36 points for a P^3, and 46 points for a P^4 approximation seemed to be enough to provide the accuracy predicted by theory.

We further emphasize the use of the sum-factorization technique introduced in Section 3.4.1 for evaluating our DG approximation at the cubature points. The application of this

Table 8.1. Number of quadrature points required in each integration technique for triangular elements. k indicates the degree of the numerical approximation.

Triangles		
k	Tensor product	Cubature
P^2	16	12
P^3	25	19
P^4	49	33

Table 8.2. Number of quadrature points required in each integration technique for tetrahedral elements. k indicates the degree of the numerical approximation. We were not able to find the cubature points for the P^4 approximation.

Tetrahedra		
k	Tensor product	Cubature
P^2	125	46
P^3	343	140
P^4	729	-

technique, along with the cubature rules, led to a substantial decrease in the computational intensity of Step 3 of the postprocessing algorithm.

8.2 Numerical Results

In this section, we provide numerical results that demonstrate the effectiveness of SIAC filtering when applied to structured tetrahedral meshes. We consider a constant coefficient and a variable coefficient advection equation and show that it is indeed possible to gain the optimal convergence rate of $2k+1$ in the L^2- and L^∞-norms after the application of the SIAC filter. Moreover, to demonstrate the effectiveness of SIAC filtering in introducing smoothness back to our numerical approximation, we provide an example of isosurfaces of a DG field before and after the application of the postprocessor.

8.2.1 Constant Coefficient Advection Equation

For this example, we consider the following advection equation

$$u_t + u_x + u_y + u_z = 0, \qquad (x, y, z) \in (0,1) \times (0,1) \times (0,1), \qquad T = 12.5, \tag{8.7}$$

with initial condition $u(0, x, y, z) = \sin(2\pi(x + y + z))$. Table 8.3 provides the error results for three different mesh resolutions and polynomial degrees. From these results, it is again clear that SIAC filtering has been effective in raising the order of accuracy to $2k+1$ both in the L_2 and L_∞ norms.

8.2.2 Variable Coefficient Advection Equation

For this example, we consider solutions of the equation

$$u_t + (au)_x + (au)_y + (au)_z = f, \qquad (x, y, z) \in (0,1) \times (0,1)\,, \times (0,1) \qquad T = 12.5. \tag{8.8}$$

We implement a smooth coefficient $a(x, y, z) = 2 + \sin(2\pi(x + y + z))$, with an initial condition of $u(x, y, z, 0) = \sin(2\pi(x+y+z))$. Periodic boundary conditions are implemented in both directions and the forcing function, $f(x, y, z, t)$, is chosen so that the exact solution is $u(x, y, z, t) = \sin(2\pi(x + y + z - 2t))$. Table 8.4 demonstrates the error results before and after postprocessing. Similarly to the previous example, we see a clear improvement in the order of accuracy. Moreover, the magnitudes of the errors are lower after the application of the postprocessor.

Table 8.3. Errors before and after postprocessing the solutions of the constant coefficient advection equation over a structured tetrahedral mesh.

Mesh	L_2-error	Order	L_∞-error	Order	L_2-error	Order	L_∞-error	Order
–	Before postprocessing				After postprocessing			
$\mathbb{P}^2$								
6000	1.06E-03	—	6.78E-03	—	2.50E-04	—	7.50E-04	—
48000	1.28E-04	3.04	8.96E-04	2.91	5.52E-06	5.50	1.54E-05	5.60
384000	1.60E-05	3.00	1.13E-04	2.98	1.40E-07	5.30	3.82E-07	5.33
$\mathbb{P}^3$								
6000	1.21E-04	—	1.30E-03	—	3.72E-05	—	8.60E-05	—
48000	7.51E-06	4.01	8.41E-05	3.95	2.15E-07	7.43	4.43E-07	7.60
384000	4.76E-07	3.98	5.43E-06	3.99	1.07E-09	7.65	3.01E-09	7.20
$\mathbb{P}^4$								
6000	2.02E-05	—	1.70E-04	—	not valid	—	not valid	—
48000	6.53E-07	4.95	5.65E-06	4.91	2.02E-09	—	6.50E-09	—
384000	2.21e-08	4.88	1.89E-07	4.90	5.43E-12	8.56	1.79E-11	8.50

Table 8.4. Errors before and after postprocessing the solutions of the variable coefficient advection equation over a structured tetrahedral mesh.

Mesh	L_2-error	Order	L_∞-error	Order	L_2-error	Order	L_∞-error	Order
–	Before postprocessing				After postprocessing			
$\mathbb{P}^2$								
6000	1.78E-03	—	6.50E-03	—	3.02E-04	—	6.90E-04	—
48000	2.24E-04	2.99	8.83E-04	2.88	7.61E-06	5.31	1.58E-05	5.45
384000	2.82E-05	2.98	1.14E-04	2.95	1.93E-07	5.30	3.71E-07	5.41
$\mathbb{P}^3$								
6000	2.00E-04	—	1.10E-03	—	4.50E-05	—	9.10E-05	—
48000	1.32E-05	3.92	6.97E-05	3.98	3.06E-07	7.20	7.36E-07	6.95
384000	8.60E-07	3.94	4.51E-06	3.95	2.10E-09	7.18	5.79E-09	6.99
$\mathbb{P}^4$								
6000	2.91E-05	—	2.00E-04	—	not valid	—	not valid	—
48000	9.74E-07	4.90	6.79E-06	4.88	5.50E-09	—	8.43E-09	—
384000	3.15E-08	4.95	2.32E-07	4.87	1.68E-11	8.50	2.67E-11	8.30

8.2.3 Isosurfaces of a DG Field

Here, we again consider the advection equation given in Equation (8.7) but this time with $u(x, y, z) = \cos(2\pi x) + \cos(2\pi y) + \cos(2\pi z)$ as the initial condition. We consider an isosurface of the numerical approximation of this equation before and after the application of the SIAC filter. Figure 8.3 demonstrates an isosurface extracted from the analytical field for $isovalue = 0.2$. To extract an isosurface we follow the approach of the traditional Marching Cubes (MC) algorithm [44] with some modifications. For a given MC mesh (which is a hexahedral mesh), we loop through individual cubes and identify the cube that contains part of the isosurface for a given isovalue. In traditional MC, linear interpolation is used to find the surface/edge intersection along an edge of the cube. However, in our modified algorithm, we perform a higher order root-finding scheme. That is, we find the intersection of the higher order DG approximation with the edge of the cube by a means of root-finding. Therefore, along an edge of the cube that contains the isosurface, we find the intersection point by finding the roots of the following equation:

$$u_h(\mathbf{x}) - isovalue = 0, \tag{8.9}$$

where u_h is our numerical approximation as given in Equation (5.6). When generating isosurfaces using the postprocessed data, u_h is replaced by $u^\star$, the postprocessed value given in Equation (8.4). We add that by applying a root-finding mechanism, we are able to observe the discontinuities that exist in the solution data as long as the MC grid overlaps with the hexahedral mesh that was used to construct our structured tetrahedral mesh.

Figure 8.4 depicts a zoomed-in portion of the isosurface in Figure 8.3, but this time using the approximate DG solution u_h (Figure 8.4a) and the postprocessed solution $u^\star$ (Figure 8.4b) to find the point of intersection in Equation (8.9). As you notice, there are visible discontinuities in the isosurface constructed on the DG approximation whereas in the isosurface extracted using the postprocessed value, there is no discontinuity. In other words, through the application of the SIAC filter, we are indeed able to introduce smoothness back to our numerical solution.

8.3 Conclusion

From its early introduction by Bramble and Schatz in [11] to its later development for linear hyperbolic equations by Cockburn *et al.* in [18, 19], there has never been a demonstration of the effectiveness of the Smoothness-Increasing Accuracy-Conserving filter over three-dimensional mesh structures. In fact, the very first attempt of applying this filtering technique to meshes of nontrivial structures, mainly in one dimension, was in [22].

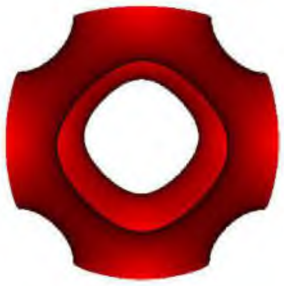
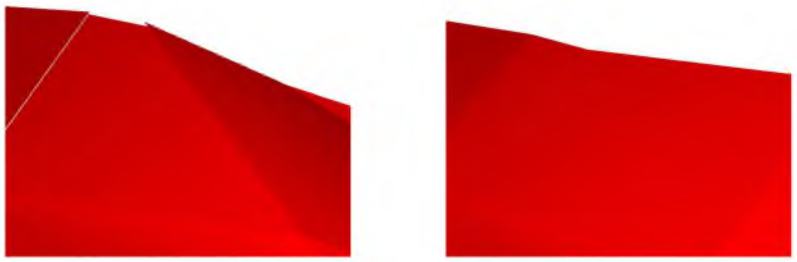


Figure 8.3. Isosurface constructed based on the analytical solution $u(x, y, z) = \cos(2\pi x) + \cos(2\pi y) + \cos(2\pi z)$ for *isovalue* $= 0.2$.



(a) Isosurface based on the DG solution.

(b) Isosurface based on the post-processed solution.

Figure 8.4. Comparison of isosurfaces before and after the application of the SIAC filter.

Later in a series of papers [46, 41, 47] (Chapters 5, 7, and 6), the extension to structured triangular meshes, general translation invariant meshes, as well as adaptive meshes and unstructured triangulations were provided, all in two-space dimensions. As our ultimate goal is the application of this filter to real-world simulations, we provided in this chapter, for the first time, computational results confirming the accuracy-conserving and smoothness-increasing capabilities of the SIAC filter over structured tetrahedral meshes. We considered two variants of a hyperbolic PDE and presented error results, which indicate that it is indeed possible to obtain the optimal $2k + 1$ order of accuracy through postprocessing. We further demonstrated how postprocessing is useful in extracting smooth isosurfaces of DG fields. We believe this is a significant contribution and a major step in extending the application of the SIAC filter beyond conventional 2D mesh structures.

CHAPTER 9

SUMMARY AND FUTURE WORK

Throughout this dissertation, we contributed to a class of postprocessors known as smoothness-increasing accuracy-conserving filters by providing mathematical foundations and numerical examples confirming the effectiveness of this filtering technique in a variety of circumstances. In particular, the following contributions were made:

- *A study of the numerical quadrature approximations used for evaluating the convolution operator in SIAC filters.* Theoretical estimates as well as empirical results that demonstrate the efficacy of the SIAC postprocessing approach when different levels and types of quadrature approximation are used were presented. This study was primarily for engineering circumstances when the trade-offs between time, resources, and accuracy are important. These contributions were documented in Chapter 4 and reported in the published peer-reviewed journal article: "Quantification of errors introduced in the numerical approximation and implementation of smoothness-increasing accuracy-conserving (SIAC) filtering of discontinuous Galerkin (DG) fields," H. Mirzaee, J. K. Ryan, and R. M. Kirby, Journal of Scientific Computing, Volume 45, Pages 447-470, 2010.

- *Application of the SIAC filters to structured triangular meshes.* The basic theoretical assumption in the previous implementations of the postprocessor limited the use to numerical solutions solved over a quadrilateral mesh. However, this assumption was restrictive, which in turn complicates the application of this postprocessing technique to general tessellations. We extended the current theoretical results to variable coefficient hyperbolic equations solved over structured triangular meshes and demonstrated the effectiveness of the application of this postprocessor to structured triangular meshes. These contributions were documented in Chapter 5 and reported in the published peer-reviewed journal article: "Smoothness-increasing accuracy-conserving (SIAC) postprocessing for discontinuous Galerkin solutions over structured triangular meshes," H. Mirzaee, L. Ji, J. K. Ryan, and R. M. Kirby, SIAM Journal of Numerical Analysis, Volume 49, Pages 1899-1920, 2011.

- *Improved errors versus higher order accuracy in applications of SIAC filters to DG solutions.* Smoothness-increasing accuracy-conserving (SIAC) filtering has demonstrated its effectiveness in raising the convergence rate for discontinuous Galerkin solutions from order $k + \frac{1}{2}$ to order $2k + 1$ for specific types of translation invariant meshes [19, 46]. Additionally, it improves the weak continuity in the discontinuous Galerkin method to $k - 1$ continuity. Typically, this improvement has a positive impact on the error quantity in the sense that it also reduces the absolute errors in the solution. However, not enough emphasis was placed on the difference between superconvergent accuracy and improved errors. This distinction is particularly important when it comes to interpreting the interplay between geometry and filtering as introduced through meshing. We presented a study of the impact of mesh scaling used in the SIAC filter and how it factors into the theoretical errors. These contributions were documented in Chapter 6 and reported in the peer-reviewed journal article: "Smoothness-increasing accuracy-conserving (SIAC) filtering for discontinuous Galerkin solutions: Improved errors versus higher-order accuracy," J. King, H. Mirzaee , J. K. Ryan and, R. M. Kirby, Journal of Scientific Computing, In press, 2012.

- *Application of the SIAC filters to unstructured triangular meshes.* Although the DG methodology can be applied to arbitrary triangulations, the typical application of SIAC filters has been to discontinuous Galerkin solutions obtained over translation invariant meshes such as structured quadrilaterals and triangles. As the assumption of any sort of regularity, including the translation invariance of the mesh, is a hindrance towards making the SIAC filter applicable to real-life simulations, we demonstrated for the first time the behavior and complexity of the computational extension of this filtering technique to fully *unstructured* tessellations. These results were promising as they paved the way towards a more generalized SIAC filtering technique. These contributions were documented in Chapter 7 and reported in the accepted journal article: "Smoothness-increasing accuracy-conserving (SIAC) filters for discontinuous Galerkin solutions over unstructured triangular meshes," H. Mirzaee, J. King, J. K. Ryan and, R. M. Kirby, SIAM Journal of Scientific Computing, accepted upon revision, 2012.

- *Application of the SIAC filters to structured tetrahedral meshes.* While there have been several attempts to demonstrate the usefulness of the SIAC filtering technique to nontrivial mesh structures, the application of the SIAC filter never exceeded beyond two-space dimensions. Thereby, we considered this contribution to be the very first attempt of its kind in demonstrating the potential usefulness of SIAC filtering when applied to real-world simulations. We consider two examples of a hyperbolic equation and demonstrate that it is indeed possible to obtain the superconvergence accuracy of $2k+1$ through the application of the SIAC filter. These contributions were documented in Chapter 8 and reported in the submitted journal article:"Smoothness-increasing accuracy-conserving (SIAC) filters for discontinuous Galerkin solutions: Application to structured tetrahedral meshes," H. Mirzaee, J. K. Ryan and, R. M. Kirby, SIAM Journal of Numerical Analysis, submitted, 2012.

The following contributions were made in order to provide the necessary steps and algorithms used to obtain the results in the above contributions:

- *Efficient implementation of SIAC filtering for DG solutions.* Quite often, a numerical practitioner is interested in explicit steps to make a numerical scheme applicable. We explicitly defined the steps to efficient computation of the postprocessor applied to different structured mesh tessellations. These contributions were documented in Chapter 3 and reported in the published peer-reviewed journal article: "Efficient implementation of smoothness-increasing accuracy-conserving (SIAC) filters for discontinuous Galerkin solutions," H. Mirzaee, J. K. Ryan, and R. M. Kirby, Journal of Scientific Computing, Volume 52, Pages 85-112, 2011.

The extension of the SIAC filter to general unstructured tessellations was discussed in Chapter 7 in two-space dimensions . There, we demonstrated how filtering can be performed effectively through proper choices of the kernel scaling parameter. Although the results in Chapter 7 were indicative of the usefulness of the SIAC filter in extracting the higher order accuracy from approximation data obtained over unstructured meshes, the underlying theoretical proofs strongly depend on the translation invariance of the mesh and therefore, the numerical behavior of the SIAC filter can not be proven for general unstructured meshes. Mathematical extension of the SIAC filter to unstructured tessellations is a very challenging task for which we will need to focus on improving the estimates for the divided differences of the error. This constitutes further research.

In Chapter 8, we briefly discussed how filtering the approximation data would naturally result in smooth isosurfaces of DG fields. We applied a modified version of the Marching

Cubes algorithm where we find the point of intersection via a higher order root-finding scheme rather than the usual linear interpolation. Using this approach of extracting isosurfaces, we further performed a preliminary investigation of the impact of postprocessing on the *Hausdorff distance* metric. Hausdorff distance is a very generic technique to define a distance between two nonempty sets [4]. Given two triangular representations of isosurfaces, one way to measure the error between them is by computing their Hausdorff distance. If we denote these two surfaces by S and S', the Hausdorff distance is defined by

$$d(S, S') = \max_{p \in S}(\min_{p' \in S'} \|p - p'\|_2), \tag{9.1}$$

that is, it is the maximum of the minimum distances between point samples of S and S'. Table 9.1 presents the Hausdorff distances and their means for a sample isosurface extracted from a DG projection of $\sin(x) \times \sin(y) \times \sin(z)$. We projected this function onto a $20 \times 20 \times 20$ hexahedral mesh and considered the isosurface for the $isovalue = 0.6$ and kept refining the Marching Cubes mesh. For each level of refinement, we computed the Hausdorff distance and its mean with respect to the exact isosurface. To generate the exact isosurface, we assumed a very fine mesh and the analytical representation of the function.

From Table 9.1, we observe that by refining the MC mesh, we reach a mesh resolution after which there is no convergence in the Hausdorff distance when using the DG approximation u_h to find the point of intersection. The convergence rate starts off the convergence rate of the MC algorithm and decreases to zero where the pointwise error in the approximation data begins to dominate. Comparing to the error results obtained using the filtered approximation, it takes more mesh refinements for the convergence rate to come to a halt. This could be related to the higher order estimates of the L^∞-norm after the application of the SIAC filter. On the other hand, the higher order accuracy in the Hausdorff distance using the filtered data could be realized by using a higher order polynomial approximation with the nonfiltered solution data u_h. A natural question that arise here is: which approach would be (could be made) computationally more efficient? This constitutes further research.

Lastly, going back to our motivating example in Chapter 1, we briefly discussed how post-processing discontinuous Galerkin approximations could result in more accurate streamline placements. In [67], Steffen *et al.* investigated how SIAC filtering of DG fields for streamline integration compares computationally to common adaptive error control approaches. The authors in [67] postprocessing over the entire DG field prior to streamline integration using the symmetric form of the kernel discussed throughout this dissertation. As the symmetric form of the kernel may not be applicable over the entire domain, Walfisch *et*

Table 9.1. Statistics calculated for isosurfaces extracted from the $\mathcal{P}^2$ DG-projection of the function $\sin(x) \times \sin(y) \times \sin(z)$ for $isovalue = 0.6$. The exact isosurface was generated on a $1000 \times 1000 \times 1000$ mesh. The polynomial modes were calculated on a $20 \times 20 \times 20$ DG mesh. MC stands for Marching Cubes.

	$\mathcal{P}^2$							
	DG approximation, u_h				Filtered approximation, $u_h^\star$			
MC mesh	Hausdorff	order	mean	order	Hausdorff	order	mean	order
20^3	5.44E-03	–	2.83E-03	–	5.43E-03	–	2.83E-03	–
40^3	1.34E-03	2.02	6.38E-04	2.14	1.32E-03	2.04	6.43E-04	2.13
80^3	3.79E-04	1.82	1.68E-04	1.92	3.91E-04	1.75	1.72E-04	1.90
160^3	1.17E-04	1.69	3.90E-05	2.10	1.08E-04	1.85	3.90E-05	2.14
320^3	5.70E-05	1.03	1.20E-05	1.70	2.70E-05	2.00	9.00E-06	2.11
640^3	5.70E-05	0.0	1.10E-05	0.12	8.00E-06	1.75	3.00E-06	1.58

al. [75] presented streamline integration using the one-sided form of the kernel given in [62]. Moreover, they considered filtering only along the streamline rather than over the entire computational field. However, the one-sided filter in [62] is not always robust in terms of producing lower error values near the boundaries. In addition, it does not acquire any information at the point where the postprocessed value needs to be computed. For streamline integrators, this could result in the advection of the streamline beyond the position where the velocity is zero and consequently leading to a wrong visualization of the vector field. In [73], Slingerland *et al.* proposed a position-dependent form of the kernel that alleviates the challenges posed by the original form of the one-sided kernel. However, the behaviour of this filter for streamline integration was never investigated. Additionally, uniform quadrilateral meshes were always considered. The extension of the SIAC filter for streamline integration over unstructured tessellation is the subject of ongoing research.

APPENDIX A

GAUSSIAN QUADRATURE RULES

In numerical analysis, a quadrature rule is an approximation of the definite integral of a function, usually stated as a weighted sum of function values at specified points within the domain of integration. Gaussian quadrature is a particularly accurate method for treating integrals where the integrand is smooth. In this technique, the integrand is represented as a Lagrange polynomial using the Q points ξ_i, which are to be specified, that is,

$$u(\xi) = \sum_{i=0}^{Q-1} u(\xi_i) h_i(\xi) + \epsilon(u), \tag{A.1}$$

where $\epsilon(u)$ is the approximation error. Therefore, to evaluate integrals as

$$\int_{-1}^{1} u(\xi) d\xi, \tag{A.2}$$

we obtain

$$\int_{-1}^{1} u(\xi) d\xi = \sum_{i=0}^{Q-1} w_i u(\xi_i) + R(u), \tag{A.3}$$

where

$$w_i = \int_{-1}^{1} h_i(\xi) d\xi \tag{A.4}$$

and

$$R(u) = \int_{-1}^{1} \epsilon(u) d\xi. \tag{A.5}$$

Since $u(\xi)$ is represented by a polynomial of order $Q-1$, we would expect the relation above to be exact if $u(\xi)$ is a polynomial of order at most $Q-1$. This would be true if, for example, we choose the points so that they are equispaced in the interval. There is, however, a better choice of zeros which permits exact integration of polynomials of higher order than $Q-1$. This remarkable fact was first recognised by Gauss and is at the heart of Gaussian quadrature.

There are three different types of Gauss quadrature known as Gauss, Gauss Radau, and Gauss-Lobatto. The difference between these three types comes from the different choices of quadrature zeros. Gauss quadrature uses zeros which have points that are interior to the interval, $-1 < \xi_i < 1$ for $i = 0, \cdots, Q-1$. In Gauss-Radau, the zeros include one of the end-points of the interval, usually $\xi = -1$, and in Gauss-Lobatto, the zeros include both end-points of the interval, that is $\xi = \pm 1$. We further note that the quadrature zeros are chosen so that $\xi_{i,P}^{\alpha,\beta}$ are the P zeros of the P^{th} order Jacobi polynomial $P_p^{\alpha,\beta}$. For further detail, consult [40].

APPENDIX B

CUBATURE RULES

Let T be a d-dimensional simplex; here $d = 2$ (triangle) or 3 (tetrahedron). A cubature rule $\mathcal{R}$ on T is defined as a set of point and weight pairs: $\mathcal{R} = (p_i, w_i)|i = 1, \cdots, n$, such that for any function $f(x)$ defined on a domain containing T and the points p_i, its integral on T can be approximated by:

$$\int_T f(x)dx \approx |T| \sum_{i=1}^{n} f(p_i)w_i, \tag{B.1}$$

where $n \in \mathbb{N}$ is the number of points, p_i are the quadrature points, w_i are the associated weights, $|T|$ denotes the area ($d = 2$) or volume ($d = 3$) of T.

When dealing with a simplex, it is often convenient to use barycentric coordinates. Let v_i, $i = 1, \cdots, d+1$, be the vertices of T. Then the barycentric coordinates $(\xi_1, \cdots, \xi_{d+1})$ of a point p with respect to T is determined by:

$$p = \sum_{i=1}^{d+1} \xi_i v_i \tag{B.2}$$

and $\sum_{i=1}^{d+1} \xi_i = 1$.

In finite element computations, numerical integration is widely used for computing integrals of functions or bilinear forms. For triangular meshes, numerical integrations on line segments, triangles, and tetrahedra are needed. In contrast to quadrilaterals or hexahedra on which quadrature formulas can be naturally derived from tensor products of one-dimensional Gauss quadrature rules, high-order nontensor product quadrature rules on triangles and tetrahedra are difficult to construct. In fact, many of the nontensor product rules published in finite element textbooks contain either negative weights or points outside of the integration domain, which are undesirable for numerical computations. For the purposes of this document, we followed the symmetric quadrature rules from [76] with all positive weights.

REFERENCES

[1] Ainsworth, M., and Oden, J. *A Posterori Error Estimation in Finite Element Analysis.* Wiley-Interscience, 2002.

[2] Appelö, D., and Petersson, N. A. A stable finite difference method for the elastic wave equation on complex geometries with free surfaces. *Communications in Computational Physics 5*, 1 (84-107), 2009.

[3] Ascher, U. M. *Numerical Methods for Evolutionary Differential Equations.* Society for Industrial and Applied Mathematics, PA, USA, 2008.

[4] Aspert, N., Santa-Cruz, D., and Ebrahimi, T. Mesh: Measuring errors between surfaces using the hausdorff distance. In *Proceedings of the IEEE International Conference on Multimedia and Expo* (2002), vol. I, pp. 705 – 708. `http://mesh.epfl.ch`.

[5] Babuška, I., Banerjee, U., and Osborn, J. E. Superconvergence in the generalized finite element method. *Numerische Mathematik 107* (2007), 353–395.

[6] Babuška, I., and Dorr, M. Error estimates for combined h- and p- version of the finite element method. *Numerical Mathematics* (1981), 37–257.

[7] Babuška, I., and Rodriguez, R. The problem of the selection of an *A-Posteriori* error indicator based on smoothing techniques. *International Journal of Numerical Methods in Engineering 36*, 4 (1993), 539–567.

[8] Babuška, I., Strouboulis, T., Upadhyay, C., and Gangaraj, S. Computer-based proof of the existence of superconvergent points in the finite element method; superconvergence of the derivatives in the finite element soultions of Laplace's, Poisson's, and the elasticity equations. *Numerical Methods for Partial Differential Equations 12* (1996), 347–392.

[9] Babuška, I., Szabo, B., and Katz, I. The p-version of the finite element method. *SIAM Journal of Numerical Analysis* (1981), 18–515.

[10] Bales, L. Some remarks on post-processing and negative norm estimates for approximations to nonsmooth solutions of hyperbolic equations. *Communications in Numerical Methods in Engineering 9* (1993), 701–710.

[11] Bramble, J. H., and Schatz, A. H. Higher order local accuracy by averaging in the finite element method. *Mathematics of Computation 31* (1977), 94–111.

[12] Cabral, B., and Leedom, L. Imaging vector fields using line integral convolution. *Computer Graphics 27* (1993), 263–272.

[13] Cockburn, B. Discontinuous Galerkin methods for convection-dominated problems. In *High-Order Methods for Computational Physics* (1999), vol. 9 of *Lect. Notes Comput. Sci. Eng.*, Springer, pp. 69–224.

[14] Cockburn, B. Devising discontinuous Galerkin methods. *Journal of Computational and Applied Mathematics*, 128 (2001), 187–204.

[15] Cockburn, B., Karniadakis, G., and Shu, C.-W. The development of the discontinuous Galerkin methods. In *discontinuous Galerkin methods: theory, computation and applications. Lecture notes on computational science and engineering*, 3-50, Ed., vol. 11. New York: Springer-Verlag, 2000.

[16] Cockburn, B., Karniadakis, G. E., and Shu, C.-W. *Discontinuous Galerkin Methods: Theory, Computation and Applications.* Springer-Verlag, Berlin, 2000.

[17] Cockburn, B., Luskin, M., Shu, C.-W., and Süli, E. Post-processing of Galerkin methods for hyperbolic problems. In *Proceedings of the International Symposium on Discontinuous Galerkin Methods* (1999), Springer, pp. 291–300.

[18] Cockburn, B., Luskin, M., Shu, C.-W., and Süli, E. Post-processing of Galerkin methods for hyperbolic problems. In *Proceedings of the International Symposium on Discontinuous Galerkin Methods* (1999), Springer, pp. 291–300.

[19] Cockburn, B., Luskin, M., Shu, C.-W., and Suli, E. Enhanced accuracy by post-processing for finite element methods for hyperbolic equations. *Mathematics of Computation 72* (2003), 577–606.

[20] Cockburn, B., and Ryan, J. K. Local derivative post-processing for discontinuous Galerkin methods. *Journal of Computational Physics 228* (2009), 8642–8664.

[21] Conte, S., and de Boor, C. *Elementary Numerical Analysis.* McGraw-Hill, Tokyo, 1972.

[22] Curtis, S., Kirby, R. M., Ryan, J. K., and Shu, C.-W. Post-processing for the discontinuous Galerkin method over non-uniform meshes. *SIAM Journal on Scientific Computing 30*, 1 (2007), 272–289.

[23] de Boor, C. *A practical guide to splines.* Springer-Verlag, New York, 2001.

[24] Demkowicz, L. *Computing with hp-adaptive finite elements: Volume 1, One and Two dimensional elliptic and maxwell problems.* Applied Mathematics and Nonlinear Science 7, Chapman and Hall/CRC, London, UK, 2007.

[25] Demkowicz, L. *Computing with hp-adaptive finite elements: Volume 2 Frontiers: Three dimensional elliptic and maxwell problems with applications.* Applied Mathematics and Nonlinear Science 11, Chapman and Hall/CRC, London, UK, 2007.

[26] Dompierre, J., Labbe, P., Vallet, M.-G., and Camarero, R. How to subdivide pyramids, prisms and hexahedra into tetrahedra. Tech. rep., Centre de Recherche en Calcul Appliqué, Montréal, Québec, 1999.

[27] Duffy, M. G. Quadrature over a pyramid or cube of integrands with a singularity at a vertex. *SIAM Journal of Numercial Analysis 19*, 6 (1982), 1260 – 1262.

[28] Entezari, A., Dyer, R., and Moller, T. Linear and cubic box splines for the body centered cubic lattice. In *VIS '04: Proceedings of the conference on visualization '04* (Wachington, DC, 2004), IEEE Computer Society, pp. 11–18.

[29] Geuzaine, C., and Remacle, J.-F. Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities. *International Journal of Numerical Methods for Engineering 79*, 11 (2009), 1309–1331.

[30] Gottlieb, D., and Tadmor, E. Recovering pointwise values of discontinuous data within spectral accuracy. In *Processings of US-Israel Workshop, Progress in Scientific Computing* (1985), vol. 6 of *Lect. Notes Comput. Sci. Eng.*, Birkhäuser Boston Inc., pp. 357–375.

[31] Hesthaven, J. S., and Warburton, T. *Nodal discontinuous Galerkin methods: Algorithm, Analysis and applicaitons.* Springer, New York, NY, 2008.

[32] Hou, H. S., and Andrews, H. C. Cubic splines for image interpolation and digital filtering. *IEEE Transactions on acoustics, speech, and signal processing 26* (1978), 508–517.

[33] Hughes, T. *The finite element method: Linear Static and Dynamic Finite Element Analysis.* Dover Publications, New York, NY, 2000.

[34] Ihm, I., Cha, D., and Kang, B. Controllable local monotonic cubic interpolation in fluid animations: natural phenomena and special effect. *Computer animation and virtual worlds 16*, 3-4 (2005), 365–375.

[35] I.J.Schoenberg. *Cardinal Spline Interpolation.* Society for Industrial Mathematics, Philadelphia, Pa, 1987. Conference board of the mathematical sciences regional conference series, No. 12.

[36] Ji, L., Xu, Y., and Ryan, J. K. Accuracy enhancement of the linear convection-diffusion equation in multiple dimensions. *Mathematics of Computation* (2012). Available online.

[37] Johnson, C., and Nävert, U. Analysis of some finite element methods for advection-diffusion problems. In *Mathematical and numerical Approaches to Asymptotic Problems in Analysis* (1981), vol. 47, North Holland Math. Stud., pp. 99–116.

[38] Jr., J. D. Superconvergence in the pressure in the simulation of miscible displacement. *SIAM Journal of Numerical Analysis 22* (1985), 962–969.

[39] Karniadakis, G. E., and Kirby, R. M. *Parallel Scientific Computing in C++ and MPI.* Cambridge University Press, New-York, NY, USA, 2003.

[40] Karniadakis, G. E., and Sherwin, S. J. *Spectral/hp element methods for CFD - 2^{nd} Edition.* Oxford University Press, UK, 2005.

[41] King, J., Mirzaee, H., Ryan, J. K., and Kirby, R. M. Smoothness-increasing accuracy-conserving (SIAC) filtering for discontinuous Galerkin solutions: Improved errors versus higher-order accuracy. *Journal of Scientific Computing* (2012). Available online.

[42] LeVeque, R. J. *Finite-Volume Methods for Hyperbolic Problems.* Cambrige University Press, Cambridge, UK, 2002.

[43] Li, B. Q. *Discontinous finite elements in fluid dynamics and heat transfer*. Springer, London, UK, 2006.

[44] Lorensen, W. E., and Cline, H. E. Marching cubes: A high resulotion 3d surface construction algorithm. In *SIGGRAPH '87 Proceedings of the 14th annual conference on Computer Graphics and interactive techniques* (1987), vol. 21, ACM New York, NY, USA, pp. 163–169.

[45] Majda, A., McDonough, J., and Osher, S. The Fourier method for nonsmooth inital data. *Math. Comp.*, 32 (1978), 1041–1081.

[46] Mirzaee, H., Ji, L., Ryan, J. K., and Kirby, R. M. Smoothness-increasing accuracy-conserving (SIAC) post-processing for discontinuous Galerkin solutions over structured triangular meshes. *SIAM Journal of Numerical Analysis 49* (2011), 1899–1920.

[47] Mirzaee, H., King, J., Ryan, J. K., and Kirby, R. M. Smoothness-increasing accuracy-conserving (siac) filters for discontinuous galerkin solutions over unstructured triangular meshes. accepted., 2012.

[48] Mirzaee, H., Ryan, J. K., and Kirby, R. M. Quantification of errors introduced in the numerical approximation and implementation of smoothness-increasing accuracy-conserving (SIAC) filtering of discontinuous Galerkin (DG) fields. *Journal of Scientific Computing 45* (2010), 447–470.

[49] Mirzaee, H., Ryan, J. K., and Kirby, R. M. Efficient implementation of smoothness-increasing accuracy-conserving (SIAC) filters for discontinuous Galerkin solutions. *Journal of Scientific Computing 52*, 1 (2011), 85–112.

[50] Mirzaee, H., Ryan, J. K., and Kirby, R. M. Smoothness-increasing accuracy-conserving (siac) filters for discontinuous galerkin solutions: Application to structured tetrahedral meshes. submitted to SIAM J. Numer. Anal.., 2012.

[51] Mitchell, D. P., and Netraveli, A. N. Reconstruction filters in computer-graphics. In *SIGGRAPH '88: Proceedings of the 15th annual conference on computer gaphics and interactive techniques* (New York, NY, 1988), ACM Press, pp. 221–228.

[52] Mock, M. S., and Lax, P. D. The computation of discontinuous solutions of linear hyperbolic equations. *Communications on Pure and Applied Mathematics 18* (1978), 423–430.

[53] Moller, T., Machiraju, R., Mueller, K., and Yagel, R. Evaluation and design of filters using a Taylor series expansion. *IEEE Transaction on Visualization and Computer Graphics 3*, 2 (1997), 184–199.

[54] Moller, T., Mueller, K., Kurzion, Y., and Yagel, R. Design of accurate and smooth filters for function and derivative reconstruction. *Proc. IEEE Symp. Volume Visualization (VVS '98)* (1998), 143–151.

[55] Narasimhan, R., and Babuška, I. Interior maximum norm estimates for finite element discretizations of the stokes equations. *Applicable Analysis 86*, 2 (2007), 251–260.

[56] Nurnberger, Slatexchumaker, L. L., and Zeifelder, F. Local Lagrange interpolation by bivariate C1 cubic splines. In *Mathematical methods for Curves and Surfaces: Oslo 2000* (Nashville, TN, 2001), Vanderbilt University, pp. 393–404.

[57] Parker, J. A., Kenyon, R. V., and Troxel, D. E. Comparison of interpolating methods for image resampling. *IEEE Transactions on Medical Imaging 2*, 1 (1983), 31–39.

[58] Patera, A. A spectral method for fluid dynamics: Laminar flow in a channel expansion. *Journal of Computational Physics* (1984), 54–468.

[59] Peiro, J., and Sherwin, S. Finite difference, finite element and finite volume methods for partial differential equaitons. *Handbook of Materials Modeling* (2005), 2415–2446.

[60] Reed, W., and Hill, T. Triangular mesh methods for the neutron transport equation. Tech. rep., Los Alamos Scientific Laboratory Report, Los Alamos, NM, 1973.

[61] Ryan, J., Shu, C.-W., and Atkins, H. Extension of a post-processing technique for the discontinuous Galerkin method for hyperbolic equations with application to an aeroacoustic problem. *SIAM Journal on Scientific Computing 26* (2005), 821–843.

[62] Ryan, J. K., and Shu, C.-W. On a one-sided post-processing technique for the discontinuous Galerkin methods. *Methods and Applications of Analysis. 10* (2003), 295–307.

[63] Sablonniere, P. Positive spline operators and orthogonal splines. *Journal of Approximaiton Theory 52* (1988), 28–42.

[64] Schumaker, L. *Spline Functions: Basic Theory.* John Wiley & Sons, New York, 1981.

[65] Stalling, D. *Fast texture-based algorithms for vector field visualization.* PhD thesis, Konrad-Zuse-Zentrum fur Informationstechnik, 1998.

[66] Stalling, D., and Hege, H.-C. Fast and resolution independent line integral convolution. *Proc. ACM SIGGRAPH '95* (1995), 249–256.

[67] Steffen, M., Curtis, S., Kirby, R., and Ryan, J. Investigation of smoothness-enhancing accuracy-conserving filters for improving streamline integration through discontinuous fields. *IEEE Transactions on Visualization and Computer Graphics 14*, 3 (2008), 680–692.

[68] Strange, G., and Fix, G. J. *Analysis of the finite element method.* Prentice Hall, Englewood Cliffs, NJ, 1973.

[69] Sutherland, I. E., and Hodgman, G. W. Reentrant polygon clipping. *Commun. ACM 17*, 1 (1974), 32–42.

[70] Thomée, V. High order local approximations to derivatives in the finite element method. *Mathematics of Computation 31* (1977), 652–660.

[71] Thomée, V. Negative norm estimates and superconvergence in Galerkin methods for parabolic problems. *Mathematics of Computation 31* (1980), 93–113.

[72] Tong, Y., Lombeyda, S., Hirani, A., and Desbrun, M. Discrete multiscale vector field decomposition. *ACM Transaction on Graphics 22*, 3 (2003), 445–452.

[73] van Slingerland, P., Ryan, J. K., and Vuik, C. Position-depnedent smoothness-increasing accuracy-conserving (SIAC) filtering for improving discontinuous Galerkin solutions. *SIAM Journal on Scientific Computing 33* (2011), 802–825.

[74] Wahlbin, L. Superconvergence in Galerkin finite element methods. vol. 1605 of *Lecture Notes in Mathematics*, Springer Verlag.

[75] Walfisch, D., Ryan, J. K., Kirby, R. M., and Haimes, R. One-sided smoothness-increasing accuracy-conserving filtering for enhanced streamline integration through discontinuous fields. *Journal of Scientific Computing 38*, 2 (2009), 164–184.

[76] Zhang, L., Cui, T., and Liu, H. A set of symmetric quadrature rules on triangles and tetrahedra. *Journal of Computational Mathematics 27*, 1 (2009), 89–96.

[77] Zienkiewicz, O., and Taylor, R. *Finite element method: Volume 1, The Basics.* Butterworth-Heinemann, Boston, MA, 2000.

[78] Zienkiewicz, O., and Taylor, R. *Finite element method: Volume 2, Solid Mechanics.* Butterworth-Heinemann, Boston, MA, 2000.

[79] Zienkiewicz, O., and Taylor, R. *Finite element method: Volume 3, Fluid Dynamics.* Butterworth-Heinemann, Boston, MA, 2000.