J Sci Comput
DOI 10.1007/s10915-009-9342-9

Quantification of Errors Introduced in the Numerical Approximation and Implementation of Smoothness-Increasing Accuracy Conserving (SIAC) Filtering of Discontinuous Galerkin (DG) Fields

Hanieh Mirzaee · Jennifer K. Ryan · Robert M. Kirby

Received: 15 June 2009 / Revised: 8 December 2009 / Accepted: 14 December 2009
© The Author(s) 2010. This article is published with open access at Springerlink.com

Abstract The discontinuous Galerkin (DG) method continues to maintain heightened levels of interest within the simulation community because of the discretization flexibility it provides. Although one of the fundamental properties of the DG methodology and arguably its most powerful property is the ability to combine high-order discretizations on an inter-element level while allowing discontinuities between elements, this flexibility generates a plethora of difficulties when one attempts to post-process DG fields for analysis and evaluation of scientific results. Smoothness-increasing accuracy-conserving (SIAC) filtering enhances the smoothness of the field by eliminating the discontinuity between elements in a way that is consistent with the DG methodology; in particular, high-order accuracy is preserved and in many cases increased. Fundamental to the post-processing approach is the convolution of a spline-based kernel against a DG field.

This paper presents a study of the impact of numerical quadrature approximations on the resulting convolution. We discuss both theoretical estimates as well as empirical results which demonstrate the efficacy of the post-processing approach when different levels and types of quadrature approximation are used. Finally, we provide some guidelines for effective use of SIAC filtering of DG fields when used as input to common post-processing and visualization techniques.

This paper is in honor of our mentor and collaborator Professor David Gottlieb, who taught us that implementations require approximations, approximations lead to "numerical crimes", and as with all crimes, numerical crimes come at a cost.

H. Mirzaee · R.M. Kirby (✉)
School of Computing, University of Utah, Salt Lake City, UT, USA
e-mail: kirby@sci.utah.edu

H. Mirzaee
e-mail: mirzaee@cs.utah.edu

J.K. Ryan
Delft Institute of Applied Mathematics, Delft University of Technology, Mekelweg 4, 2628 CD Delft, The Netherlands
e-mail: J.K.Ryan@tudelft.nl

Published online: 08 January 2010

Keywords High-order methods · Discontinuous Galerkin · SIAC filtering · Accuracy enhancement

1 Introduction

The discontinuous Galerkin (DG) methods provide a high-order extension of the finite volume method in much the same way as high-order or spectral/*hp* elements [15, 21] extended standard finite elements. The DG methodology allows for a dual path to convergence through both elemental *h* and polynomial *p* refinement, making it highly desirable for computational problems which require resolution fidelity. In the overview of the development of the discontinuous Galerkin method, Cockburn et al. [4] trace the developments of DG and provide a succinct discussion of the merits of this extension to finite volumes.

The primary mathematical advantage of DG is that unlike classic continuous Galerkin FEM, that seeks approximations which are piecewise continuous, the DG methodology only requires functions which are L_2 integrable. Much like FEM, DG uses the variational form; however, instead of constraining the solution to being continuous across element interfaces, the DG method merely requires weak constraints on the fluxes between elements. This feature provides a discretization flexibility that is difficult to match with conventional continuous Galerkin methods.

Lack of inter-element continuity, however, is often contrary to the smoothness assumptions upon which many post-processing algorithms such as those used in visualization are based. A class of post-processing techniques were introduced in [7, 18] as a means of gaining increased accuracy from DG solutions through the exploitation of the superior convergence rates of DG in the negative norm; these filters had as a secondary consequence that they increased the smoothness of the output solution. Building upon these concepts, in [20, 22] smoothness-increasing accuracy-conserving (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 smoothness *without destroying* (i.e. by maintaining) the order of accuracy of the original input DG solution.

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 paper 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 [7, 18] and the engineering literature [20, 22] employed consistent integration with Gaussian quadrature to guarantee that the numerical errors within the convolution operator could be driven below machine precision. In this paper we seek to quantify the impact of inexact quadrature on the filtering process and to assess whether it greatly impacts its use as intermediary stage between simulation and visualization in the scientific pipeline.

In this paper we examine a collection of common scenarios that might arise when one seeks to implement the aforementioned post-processing algorithms in the engineering context. We focus on one-dimensional and two-dimensional quadrilateral implementations over periodic domains as for these cases we can derive theoretical estimates as to the impact of the "numerical crimes" committed through the different quadrature strategies. 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

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 challenging geometric problem of find all the places in which regularity is decreased and generating a super-mesh based on this 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 visualization tool. It is this specific crime that we wish to address.

The paper is organized as follows. In Sect. 2 we discuss the discontinuous Galerkin method for systems of hyperbolic equations. In Sect. 3 we present the mathematical background on the smoothness-increasing accuracy-conserving filters being examined in this paper. In Sect. 4 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 Sect. 5 we present analysis which provides theoretical estimates which bound the numerical crimes committed when using the three aforementioned approaches. In Sect. 6 we present an empirical study which corroborates the error estimates we have derived. In Sect. 7 we summarize our results and provide guidelines based upon our study concerning under which circumstances one technique should be used versus another.

2 The Discontinuous Galerkin Method

In this paper, we focus our attention on simulation results that arise as solutions of the linear hyperbolic equation

$$\mathbf{u}_t + \nabla \cdot (a(\mathbf{x},t)\mathbf{u}) = 0, \tag{1}$$

where $\mathbf{x} \in \Omega$ and $t \in \mathbb{R}$. The DG formulation for this equation has been well studied in the series of papers [2, 3, 5, 9–12]. Here we present a brief introduction.

We use the weak form of (1) to derive our discontinuous Galerkin approximation. That is, we multiply by a test function $\mathbf{v}$ to obtain

$$\frac{d}{dt}\int_\Omega \mathbf{u}(\mathbf{x},t)\mathbf{v}\,d\mathbf{x} + \int_\Gamma (a(\mathbf{x},t)\mathbf{u})\cdot \hat{n}\mathbf{v}\,d\Gamma - \int_\Omega (a(\mathbf{x},t)\mathbf{u})\cdot \nabla\mathbf{v}\,d\mathbf{x} = 0, \tag{2}$$

where $\hat{n}$ denotes the unit outward normal to the boundary and Γ the boundary of our domain, Ω.

We can now define our discontinuous Galerkin approximation to (1) using (2). Begin by defining a suitable tessellation of the domain Ω, $\mathcal{T}(\Omega) = \tilde{\Omega}$. We note that the current form of the post-processor requires using a rectangular domain in two-dimensions. Secondly, we define an approximation space, $\mathbf{V}_h$, consisting of piecewise polynomials of degree less than or equal to k. Our discontinuous Galerkin approximation will then be of order $k+1$. Using the variational formulation and taking our test function $\mathbf{v}_h$ from our approximation space we obtain

$$\frac{d}{dt}\int_K \mathbf{u}(\mathbf{x},t)\mathbf{v}_h\,d\mathbf{x} + \sum_{e\in\partial K}\int_e (a(\mathbf{x},t)\mathbf{u})\cdot\hat{n}_{e,K}\mathbf{v}_h\,d\Gamma - \int_K (a(\mathbf{x},t)\mathbf{u})\cdot\nabla\mathbf{v}_h\,d\mathbf{x} = 0, \tag{3}$$

where $\hat{n}_{e,K}$ denotes the outward unit normal to edge e. We then obtain the numerical scheme

$$\begin{aligned}\frac{d}{dt}\int_K \mathbf{u}_h(\mathbf{x},t)\mathbf{v}_h\,d\mathbf{x} &+ \sum_{e\in\partial K}\int_e \mathbf{h}(\mathbf{u}_h(\mathbf{x}^-,t),\mathbf{u}_h(\mathbf{x}^+,t))\mathbf{v}_h\,d\Gamma\\ &- \int_K (a(\mathbf{x},t)\mathbf{u}_h)\cdot\nabla\mathbf{v}_h\,d\mathbf{x} = 0\end{aligned} \tag{4}$$

for all test functions $\mathbf{v}_h \in \mathbf{V}_h$ where $\mathbf{h}(\mathbf{u}(\mathbf{x}^-,t),\mathbf{u}(\mathbf{x}^+,t))$ represents the numerical flux and u_h is the DG approximation of degree k.

3 The Post-Processor

We concentrate on quantifying and improving the computational cost of the convolution for the Smoothness-Increasing Accuracy-Conserving (SIAC) filter that were first introduced as a class of post-processors for the discontinuous Galerkin method in [6, 7]. The filtering technique was extended to a broader set of applications, including as a streamline visualization filter, in [8, 14, 17, 18, 20, 22]. As a visualization filter, it is unique in that it takes into account the information about the discontinuous Galerkin approximation. For an approximation of degree k, we convolve it against a filter containing a linear combination of B-splines of degree k. This works to smooth the oscillations in the error and improve the order of the filtered solution to $2k+1$. Here we only present a brief introduction to this post-processing technique. For a more detailed discussion of the post-processor see [1, 7, 14, 18].

The post-processor itself is simply the discontinuous Galerkin solution convolved against a linear combination of B-splines. That is, in one-dimension,

$$u^\star(x) = \frac{1}{h}\int_{-\infty}^{\infty} K^{2(k+1),k+1}\left(\frac{y-x}{h}\right)u_h(y)\,dy, \tag{5}$$

where $u^\star$ is the post-processed solution, h is the uniform element size, u_h the DG solution of degree k and

$$K^{2(k+1),k+1}(x) = \sum_{\gamma=-k}^{k} c_\gamma^{2(k+1),k+1}\psi^{(k+1)}(x-\gamma). \tag{6}$$

The B-splines used in the post-processor are well studied and can be computed using the recurrence relation

$$\psi^{(1)} = \chi_{[-1/2,1/2]}, \tag{7}$$

$$\psi^{(k+1)} = \frac{1}{k}\left(\left(x+\frac{k+1}{2}\right)\psi^{(k)}\left(x+\frac{1}{2}\right) + \left(\frac{k+1}{2}-x\right)\psi^{(k)}\left(x-\frac{1}{2}\right)\right), \quad k\ge 1, \tag{8}$$

see [19]. The coefficients of the kernel, $c_\gamma^{2(k+1),k+1}$, can be found by using the property that the kernel must not destroy the accuracy of the approximation. More specifically, they reproduce polynomials of degree $2k$ by convolution. This leads to a system of equations

$$\Psi^{(k+1)}\mathbf{c}^{2(k+1),k+1} = \mathbf{X} \tag{9}$$

where the (i,γ)-th element of $\Psi^{(k+1)}$ is given by

$$\int_{\mathbb{R}} \psi^{(k+1)}(x-y-\gamma)y^i\,dy, \quad i=0,\dots,2k,\ \gamma=-k,\dots,k,$$

and $\mathbf{X}_i = x^i$. When using a symmetric B-spline kernel, it is only necessary to solve this system once and store the resulting coefficient vector, $\mathbf{c}^{2(k+1),k+1}$. We note that the coefficients are symmetric in this case, that is $c_{-j}^{2(k+1),k+1} = c_j^{2(k+1),k+1}$, $j=1,\dots,k$. We note that the two-dimensional case is tensor product of the one-dimensional case.

After solving for the coefficients, we then need to carry out the convolution

$$C_{i,\ell,k} = \frac{1}{h}\sum_{\gamma=-k}^{k} c_\gamma^{2(k+1),k+1}\int_{I_i}\psi^{(k+1)}\left(\frac{y-x}{h}-\gamma\right)\phi^{(\ell)}\left(\frac{y-x_i}{h}\right)dy, \quad \ell=0,\dots,k, \tag{10}$$

where I_i denotes the element and $\phi^{(\ell)}$ represents the ℓ^{th}-piecewise polynomial from our approximation space scaled to the interval $(-1/2,1/2)$. Using this notation, we then have that the post-processed solution is

$$u^\star = \sum_{i=j-k'}^{k'}\sum_{\ell=0}^{k} C_{i,\ell,k}u_i^{(\ell)} \tag{11}$$

where $k' = \lceil\frac{3k+1}{2}\rceil$ and $u_i^{(\ell)}$ are the polynomial modes on I_i. The outer sum is over the elements that fall within the support of the post-processor. We can see from this notation that if we choose to compute the post-processing coefficients ahead of time at specific points within an element, the post-processor simply becomes a series of small matrix-vector multiplications. However, if we choose to compute the convolution directly, it can become computationally expensive. The latter implementation, in which we take care of the convolution within the definition of $C_{i,\ell,k}$, is the one we concentrate on for the purposes of this paper. We will, in the next section, go into more detail as to how one implements the post-processor and the corresponding computational costs.

4 Implementation

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

4.1 Gaussian Quadrature Approaches

The post-processed solution, $u^\star(x)$, which is a piecewise polynomial of degree $2k+1$, can be evaluated exactly. As we mentioned in Sect. 3, $u^\star(x)$ is defined by (5), where $u_h(y) =$

$\sum_{\ell=0}^{k} u_i{}^{(\ell)}\phi_i^{(\ell)}(y)$, and $\phi_i^{(\ell)}$ are the basis functions of the projected function on cell $I_i = (x_i - \frac{h}{2}, x_i + \frac{h}{2})$. Therefore, for $x \in I_i$, we have

$$\begin{aligned} u^*(x) &= \frac{1}{h}\int_{-\infty}^{\infty} K^{2(k+1),k+1}\left(\frac{y-x}{h}\right)u_h(y)\,dy \\ &= \frac{1}{h}\sum_{j=-k'}^{k'}\int_{I_{i+j}} K^{2(k+1),k+1}\left(\frac{y-x}{h}\right)u_h(y)\,dy \\ &= \frac{1}{h}\sum_{j=-k'}^{k'}\int_{I_{i+j}} K^{2(k+1),k+1}\left(\frac{y-x}{h}\right)\left(\sum_{\ell=0}^{k} u_{i+j}^{(\ell)}\phi_{i+j}^{(\ell)}(y)\right)dy. \end{aligned} \tag{12}$$

We have used the compact support property of $K^{2(k+1),k+1}$ to reduce the area of the integral to a total support of $2k'+1$ elements where $k' = \lceil \frac{3k+1}{2} \rceil$.

As mentioned earlier in Sect. 3, 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*); 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 post-processed solution for $x \in I_i$, the algorithm is as follows:

For $j = -k', \dots, k'$:

- 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 post-processing 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 post-process an entire element (i.e. find the post-processed polynomial of degree $2k+1$ on a 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,

$$u^*(x) = \sum_{j=-k'}^{k'}\sum_{l=0}^{k} u_{i+j}^{(l)} C_{j,l,k}(x) \tag{13}$$

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.

In previous implementations, one computes the convolution of the basis functions of the DG solution and B-splines together for a given mesh once so that post-processing can be accomplished repetitiously for different DG solutions on the same mesh through mere matrix-vector multiplication. As the kernel is translation invariant when a uniform mesh is used, one needs only to compute a small set of matrices that can be used for post-processing all elements. As is presented in [14], it takes $O(N)$ operations to filter the entire domain, where N is the total number of elements.

However, the purpose of this paper 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 the finding of the consistent integration mesh. This is a geometric projection and intersection problem that, in general, is quite complex in two and three dimensions as it requires finding the intersection of mesh elements against knots of the B-spline kernel, and possibly the further refined tessellation of the subdomains into elements upon which integration can be performed (for instance, partitioning polyhedra into a collection of tetrahedra). 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 post-processing as the convolution kernel is merely a tensor-product of the one dimensional kernels. Secondly, in the case of a non-uniform mesh since the kernel is no longer translation invariant the post-processing coefficients need to be recomputed for each element as is mentioned in [14]. In order to avoid this re-computation, Curtis et al. proposed two strategies for post-processing over non-uniform 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.

4.2 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

$$u^*(x) = \frac{1}{h}\int_{-\infty}^{\infty} K^{2(k+1),k+1}\left(\frac{y-x}{h}\right)u_h(y)\,dy \tag{14}$$

using midpoint integration. For a complete overview of the derivation and implementation of the midpoint rule we refer the reader to [16]. In this case we evaluate the post-processed solution $u^*(x)$ using the midpoint rule to compute the integral in (14) 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 $[x_{\text{left}}, x_{\text{right}}]$.
- 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 $\Delta x = (x_{\text{right}} - x_{\text{left}})/2^{\text{level}}$ 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 $x_{\text{left}} = -\frac{3k+1}{2}$ and $x_{\text{right}} = \frac{3k+1}{2}$. This gives $\Delta x = \frac{3k+1}{2^{\text{level}}}$.
- Perform the midpoint integration,

$$\begin{aligned} u^*(x) &= \frac{1}{h}\int_{-\infty}^{\infty} K^{2(k+1),k+1}\left(\frac{y-x}{h}\right) u_h(y)\, dy \\ &= \frac{1}{h}\int_{x_{\text{left}}}^{x_{\text{right}}} K^{2(k+1),k+1}\left(\frac{y-x}{h}\right) u_h(y)\, dy \\ &= \frac{1}{h}\left(\Delta x \sum_{i=1}^{n} K_{i-1/2} u_{h,i-1/2}\right). \end{aligned} \tag{15}$$

Again, we follow a similar process in case of a 2D post-processor 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.

5 Quadrature Approximations of the Convolution Operator

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

5.1 Gaussian Quadrature on a Consistent Integration Mesh

5.1.1 Exact, Consistent Gaussian Quadrature

Gaussian Quadrature is well-known to integrate polynomials of degree $2m-1$ exactly by using m points, $x_1, x_0, \ldots, x_m$, and m weights $w_1, w_2, \ldots, w_m$ [13]. The formula for the quadrature is given by

$$\int_a^b f(x)\, dx = \sum_{j=1}^{m} f(y_j) w_j, \tag{16}$$

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

The main point of this discussion is to point out that in order to post-process one element that contains a discontinuous Galerkin approximation of degree k, we have a support size of $2k'+1$ elements for the post-processor, where $k' = \lceil \frac{3k+1}{2} \rceil$. There are two integral evaluations per element. Therefore we are performing $4k'+2$ Gaussian Quadratures of degree $2k+1$. Although Schumaker gives a simplified formula for integrating a B-spline against a polynomial [19], we should point out that we are convolving the B-splines against a piecewise polynomial.

To begin, let us look at the kernel performed using exact integration over a uniform mesh. In this case, to post-process element i, we have

$$u^\star(x) = \sum_{j=-k'}^{k'} \sum_{\ell=0}^{k} u_{i+j} \sum_{\gamma=-k}^{k} c_\gamma \frac{1}{h} \int_{I_{i+j}} \psi^{(k+1)}\left(\frac{y-x}{h} - \gamma\right) \phi^{(\ell)}\left(\frac{y-x_{i+j}}{h}\right) dy. \tag{17}$$

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

$$u^\star(x) = \sum_{j=-k'}^{k'} \sum_{\ell=0}^{k} u_{i+j} \sum_{\gamma=-k}^{k} c_\gamma \int_{-\xi_i+j-1/2}^{-\xi_i+j+1/2} \psi^{(k+1)}(\eta-\gamma)\phi^{(\ell)}(\xi_i+\eta-j)\, d\eta. \tag{18}$$

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

$$f(\eta,\xi_i) = \sum_{\ell=0}^{k} \psi^{(k+1)}(\eta-\gamma)\phi^{(\ell)}(\xi_i+\eta-j). \tag{19}$$

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 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 [13] 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

$$\int_a^b f(\eta,\xi_i)\, d\eta - \int_a^b p_k(\eta)\, d\eta = \int_a^b f[x_0,\ldots,x_{2k+1}] g_{2k+1}(x)\, dx \tag{20}$$

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

$$\int_a^b f(\eta,\xi_i)\,d\eta-\int_a^b p_k(\eta)\,d\eta=C(\xi_i,y_0,\dots,y_{k+1})f^{(2k+2)}(z,\xi_i),\quad z\in(a,b). \tag{21}$$

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

5.1.2 *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 less 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 (21), we have the following error estimate for one integral

$$\int_a^b f(\eta,\xi_i)\,d\eta-\int_a^b p_k(\eta)\,d\eta=C(\xi_i,y_0,\dots,y_{k+1})f^{(2k)}(z,\xi_i)=\mathsf{C}. \tag{22}$$

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.

5.2 Gaussian Quadrature on the DG Mesh

Next, we look at 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 may only use the error estimate given in (20) since the function on a given element is only $\mathcal{C}^{k-1}$. That is, the error is given by

$$\int_a^b f(\eta,\xi_i)\,d\eta-\int_a^b p_k(\eta)\,d\eta=\int_a^b f[x_0,\dots,x_{2k-1}]g_{2k+1}(x)\,dx. \tag{23}$$

5.3 Midpoint Quadrature on the DG Mesh

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

$$\int_a^b f(x)\,dx=(b-a)f(c) \tag{24}$$

where $c=(a+b)/2$ and f is as given in (19). It is well known that the midpoint error is given by

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

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

$$\sum_{j=0}^{m-1}\frac{1}{24}f''(z)\triangle x^3\le\mathsf{C}\,\triangle x^3 \tag{25}$$

where $\triangle x=\frac{3k+1}{2^m}$ since $x_{\text{left}}=-\frac{3k+1}{2}$, $x_{\text{right}}=\frac{3k+1}{2}$ and $\triangle x=(x_{\text{right}}-x_{\text{left}})/2^m$. 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.

6 Results

In this section we present the results for the various choices of 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 visualization purposes.

6.1 Consistent Integration with Inexact Gaussian Quadrature Approach

We consider the one dimensional projection of the function

$$u(x) = \sin(2\pi x), \quad x \in (0, 1) \tag{26}$$

onto a uniform mesh to mimic a DG solution. We should note that we always use exact integration to calculate the L^2 projections and we only focus on different integration strategies for the convolution operator used in the post-processor. We assume that we have a periodic interval in order to simplify the application of the post-processor. For this example, 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 Fig. 1. In this set of plots, we can see that the aliasing error is more sensitive at low orders and that the higher-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.

6.2 Input Mesh Based Gaussian Quadrature Approach

In this section we examine the results of not using a consistent integration mesh (i.e., ignoring knots of the B-spline kernel), but instead attempt to overcome the numerical crimes committed by integrating over the jump by increasing the number of quadrature points.

6.2.1 One-Dimensional DG

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

$$u(x) = \sin(2\pi x), \quad x \in (0, 1) \tag{27}$$

to a uniform mesh. We assume a periodic interval. Figures 2, 3 and 4 show the point-wise errors when post-processing on the DG mesh for $\mathbb{P}^2$, $\mathbb{P}^3$ and $\mathbb{P}^4$ polynomials, respectively. Considering $\mathbb{P}^k$ polynomials, we need $k+1$ Gauss points to exactly evaluate the inner products involved in post-processing of a DG solution on a consistent integration mesh. Using the same number of points when post-processing on the DG input mesh, we observe that the accuracy in terms of error is improved but that oscillations still exist (see Fig. 2 $Q = 3$, Fig. 3 $Q = 4$ and Fig. 4 $Q = 5$).

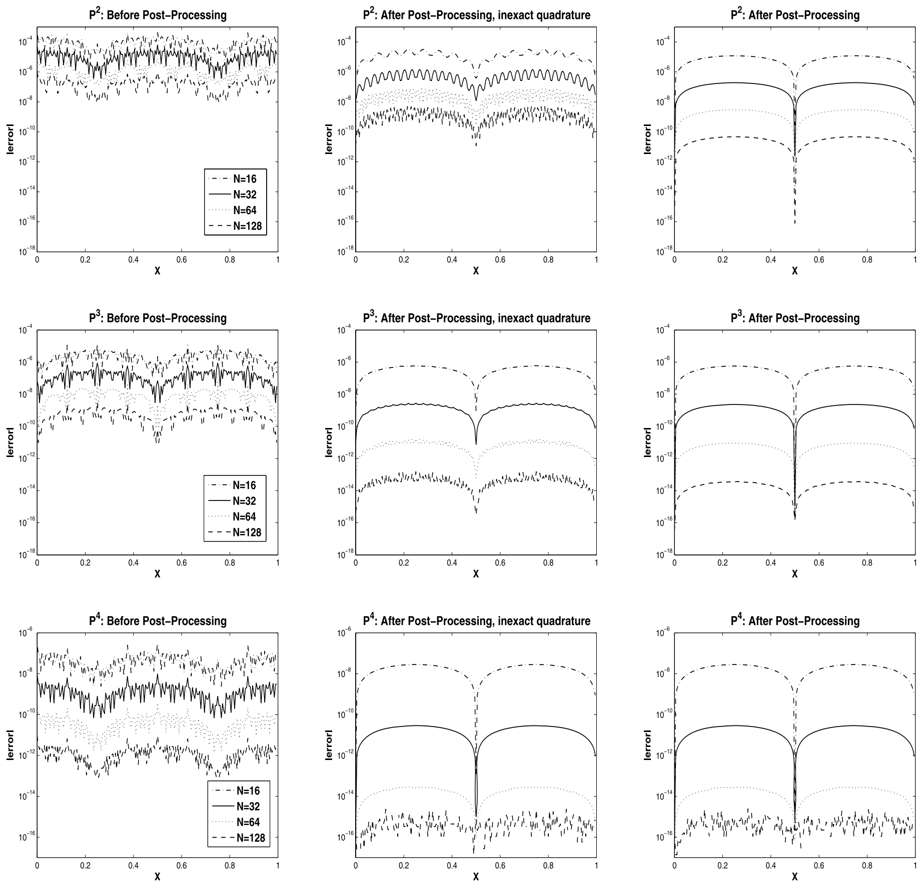

Fig. 1 Point-wise errors on a logarithmic scale before post-processing (*left*), after post-processing on the consistent integration mesh with inexact quadrature (*middle*) and after post-processing with exact quadrature (*right*)

6.2.2 *One-Dimensional DG—Non-Uniform Mesh*

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

$$u(x) = \sin(2\pi x), \quad x \in (0, 1) \tag{28}$$

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 post-processor implementation introduced in [14]. In the case of a non-uniform mesh, Table 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 non-uniform mesh, a quadrature the uses at least nine Gauss points seem to be required. For piecewise cubic, the errors using the lowest value of the quadrature are even worse (Table 2). And, unless exact quadrature is implemented, the errors are always worse when using ten elements.

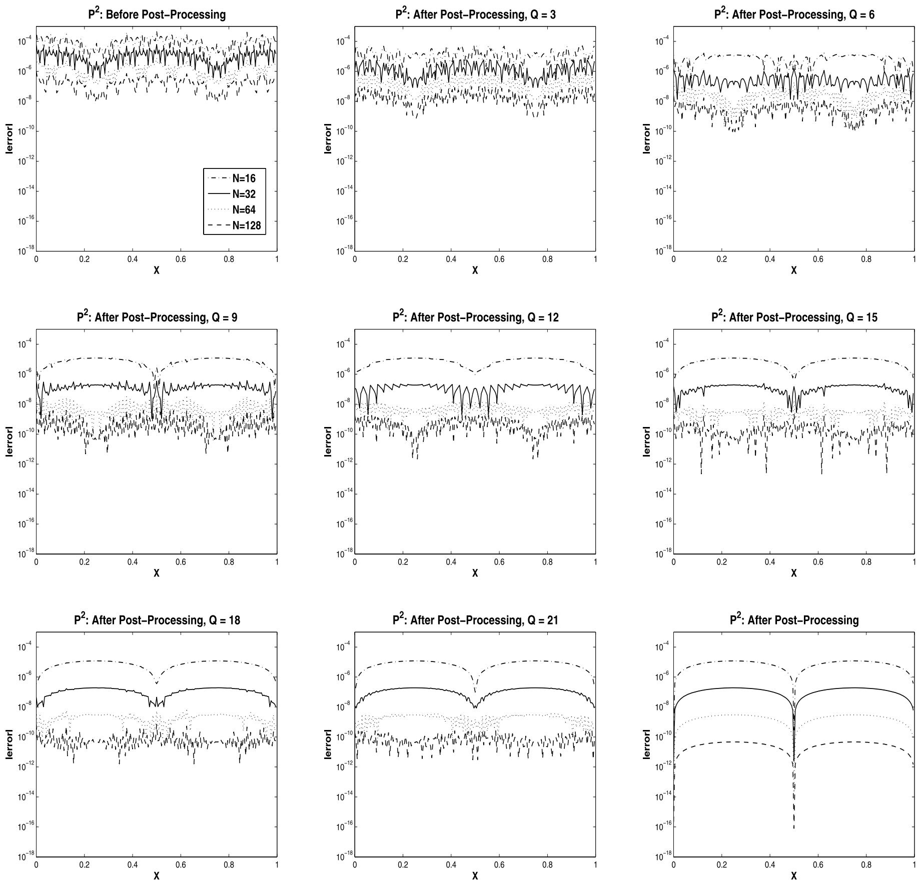

Fig. 2 Point-wise errors on a logarithmic scale when post-processing on the DG input mesh using $\mathbb{P}^2$ polynomials. *Top left*: errors before post-processing. *Bottom right*: errors after post-processing on the consistent integration mesh using exact quadrature. Q is the number of quadrature points

6.2.3 Two-Dimensional DG

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

$$u(x, y) = \sin(2\pi(x + y)), \quad x \in (0, 1), \ y \in (0, 1) \tag{29}$$

to a uniform mesh. Again we assume a periodic domain. Additionally, we see in Tables 3 and 4 that although the convergence rates with the post-processor are not always improved, the errors are always lower than for the input DG solution.

6.2.4 Two-Dimensional DG—Constant Coefficient Linear Advection Equation

For this example, we consider solutions of the equation,

$$u_t + u_x + u_y = 0, \quad (x, y) \in (0, 2\pi) \times (0, 2\pi), \quad T = 12.5 \tag{30}$$

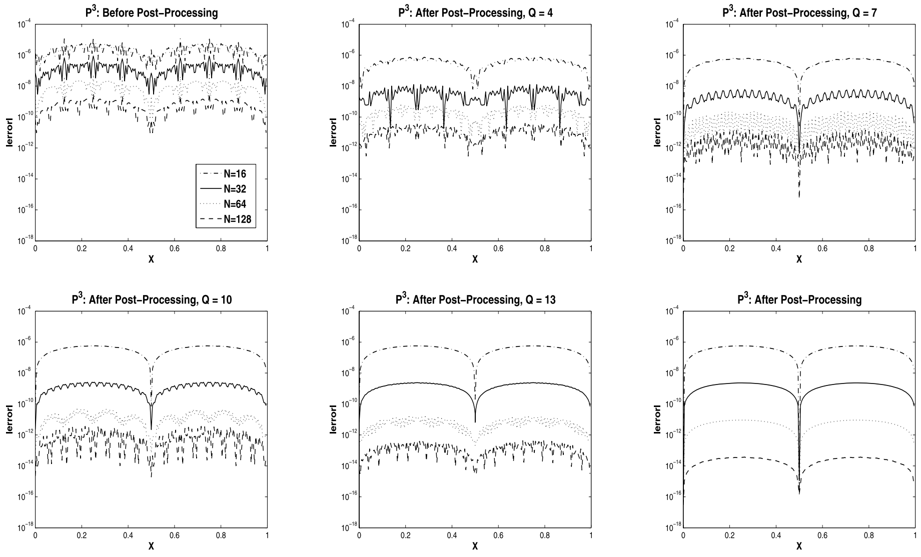

Fig. 3 Point-wise errors on a logarithmic scale using $\mathbb{P}^3$ polynomials

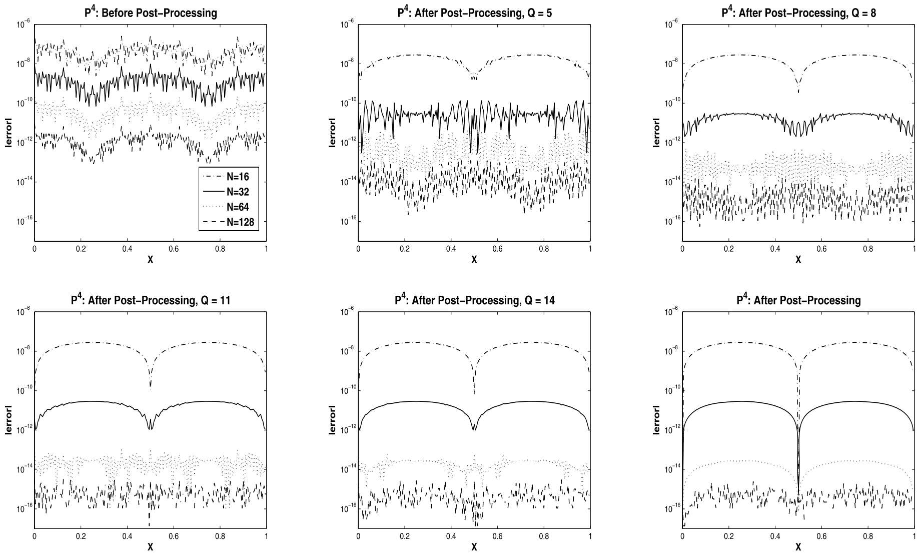

Fig. 4 Point-wise errors on a logarithmic scale using $\mathbb{P}^4$ polynomials

with initial condition $u(0, x, y) = \sin(x + y)$. Observe in Table 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 6 we also immediately improve both the errors and the convergence rate using only three Gauss points.

Table 1 Errors for one-dimensional DG using $\mathbb{P}^2$ polynomials for the non-uniform mesh. Before post-processing, after post-processing on the DG mesh where Q is the number of quadrature points and finally after post-processing 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 post-processing				After post-processing, $Q=3$			
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 post-processing, $Q=9$				After post-processing, $Q=12$			
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 post-processing, $Q=21$				After post-processing, $Q=35$			
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 post-processing, $Q=45$				After post-processing, 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

6.2.5 Two-Dimensional DG—Variable Coefficient Equation

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

$$u_t + (au)_x + (au)_y = f(x, y, t), \quad (x, y) \in (0, 2\pi) \times (0, 2\pi), \quad T = 12.5 \tag{31}$$

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 7 and 8 that the convergence rate is not always optimal, but that indeed, with a minimum number of Gauss points we improve the errors.

6.3 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.

6.3.1 One-Dimensional DG using midpoint quadrature

In this example we again consider the case of projecting

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

Table 2 Errors for one-dimensional DG using $\mathbb{P}^3$ polynomials for the non-uniform mesh. Before post-processing, after post-processing on the DG mesh where Q is the number of quadrature points and finally after post-processing 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 post-processing				After post-processing, $Q=4$			
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 post-processing, $Q=7$				After post-processing, $Q=16$			
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 post-processing, $Q=31$				After post-processing, 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

Table 3 Errors for two-dimensional DG using $\mathbb{P}^2$ polynomials. Before post-processing, after post-processing on the DG mesh where Q is the number of quadrature points and finally after post-processing 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 post-processing				After post-processing, $Q=3$			
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 post-processing, $Q=6$				After post-processing, $Q=9$			
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 post-processing, $Q=12$				After post-processing, 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 Errors for two-dimensional DG using $\mathbb{P}^3$ polynomials. Before post-processing, after post-processing on the DG mesh where Q is the number of quadrature points and finally after post-processing 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 post-processing				After post-processing, $Q=4$			
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 post-processing, $Q=7$				After post-processing, $Q=10$			
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 post-processing, $Q=13$				After post-processing, CI			
16^2	8.09E−07	–	1.15E−06	–	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 5 Errors for one-dimensional DG using $\mathbb{P}^1$ polynomials for the linear advection equation. Before post-processing, after post-processing on the DG mesh where Q is the number of quadrature points and finally after post-processing 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 post-processing				After post-processing, $Q=2$			
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 post-processing, $Q=5$				After post-processing, $Q=8$			
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 post-processing, $Q=11$				After post-processing, 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 6 Errors for two-dimensional DG using $\mathbb{P}^2$ polynomials for the linear advection equation. Before post-processing, after post-processing on the DG mesh where Q is the number of quadrature points and finally after post-processing 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 post-processing				After post-processing, $Q = 3$			
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 post-processing, $Q = 6$				After post-processing, 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

Table 7 Errors for two-dimensional DG using $\mathbb{P}^1$ polynomials for the variable coefficient advection equation. Before post-processing, after post-processing on the DG mesh where Q is the number of quadrature points and finally after post-processing 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 post-processing				After post-processing, $Q = 2$			
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 post-processing, $Q = 5$				After post-processing, $Q = 8$			
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 post-processing, $Q = 11$				After post-processing, 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

onto a piecewise polynomial space. The results are displayed in Fig. 5. The plots show that it takes many applications of the midpoint rule to get better errors for the post-processed solution than the initial projection, and that the midpoint rule is effective when we use

Table 8 Errors for two-dimensional DG using $\mathbb{P}^2$ polynomials for the variable coefficient advection equation. Before post-processing, after post-processing on the DG mesh where Q is the number of quadrature points and finally after post-processing 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 post-processing				After post-processing, $Q = 3$			
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 post-processing, $Q = 6$				After post-processing, $Q = 9$			
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 post-processing, $Q = 12$				After post-processing, $Q = 15$			
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 post-processing, $Q = 18$				After post-processing, 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

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 Fig. 6 we can see that the lines level off to the same as that for consistent integration error, which is the best possible scenario.

6.3.2 *Two-Dimensional DG using midpoint quadrature*

In this section we again consider the two dimensional projection problem given by (29). It is observed that in the case of 2D midpoint rule the convergence rate in the L^2-norm is linear, as is shown in the sample plot in Fig. 7. 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.

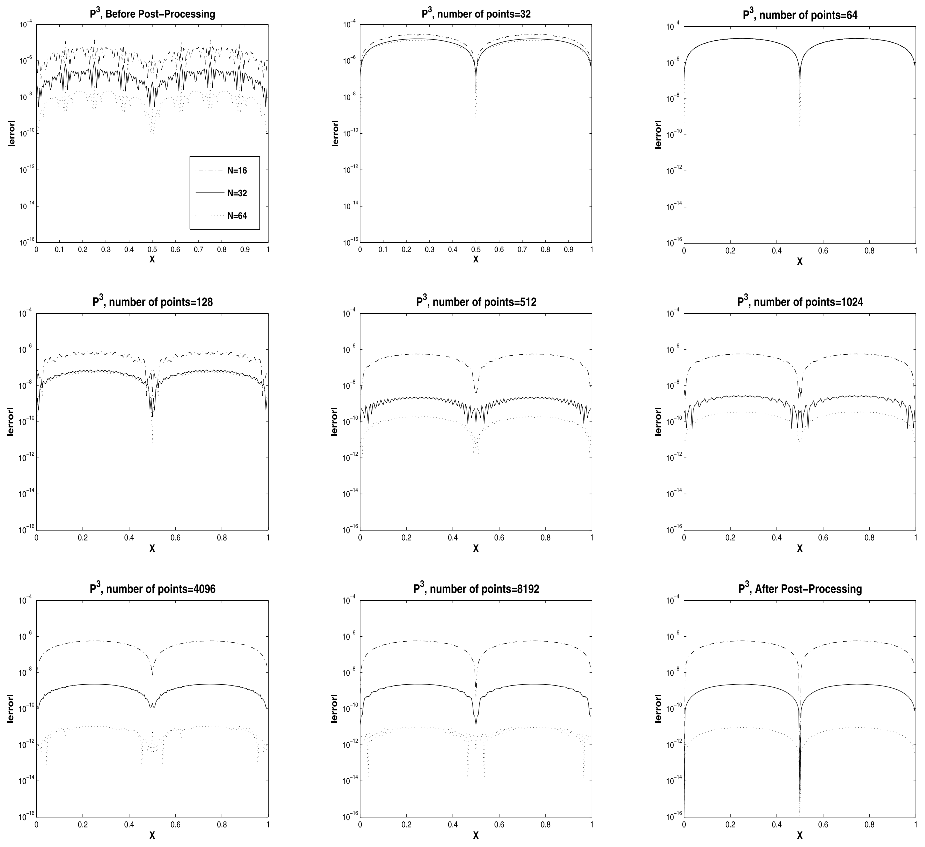

Fig. 5 Point-wise errors on a logarithmic scale when using midpoint integration for post-processing with different number of evaluation points. *Top left*: before post-processing. *Bottom right*: after post-processing on the consistent integration mesh

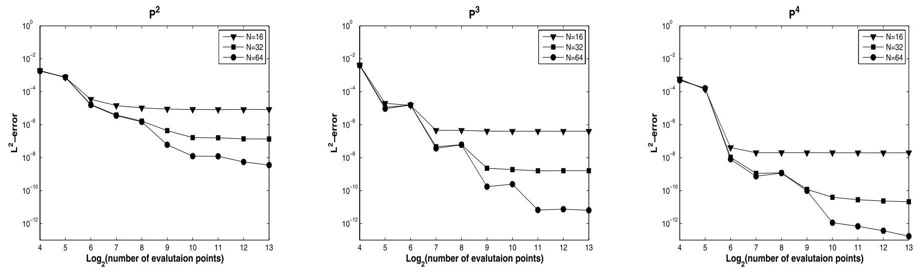

Fig. 6 Convergence of the L^2 errors when using midpoint integration for post-processing

6.4 Two-Dimensional Vector Field

As it is mentioned in [20, 22], smoothness-increasing, accuracy-conserving filtering can be applied to discontinuous Galerkin vector fields to enhance streamline integration. In this sec-

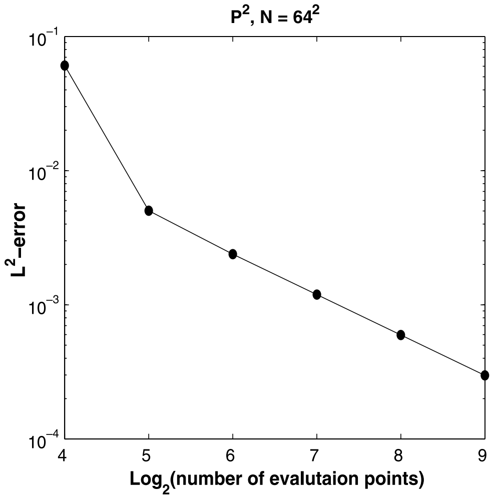

Fig. 7 Convergence of the L^2 errors when using midpoint integration for post-processing

tion 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

$$\begin{bmatrix} u(r,\theta) \\ v(r,\theta) \end{bmatrix} = \begin{bmatrix} \frac{1}{2}\cos(20\theta)\cos(\theta) - r\sin(\theta) \\ \frac{1}{2}\sin(20\theta)\sin(\theta) - r\cos(\theta) \end{bmatrix}. \tag{32}$$

This has streamlines which are oscillating closed circuits. In Fig. 8 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 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 post-processing benefit in terms of smoothness and accuracy to be of use in data processing and visualization.

7 Summary and Conclusions

This paper presents a study of the impact of numerical quadrature approximations used for evaluating the convolution operator in smoothness-increasing accuracy-conserving (SIAC) filters. We provide both theoretical estimates as well as empirical results which demonstrate the efficacy of the post-processing 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 solving the challenging geometric problem of find all the places in which regularity is decreased and generating a super-mesh based on this data, we considered what happens when only the original DG mesh is used as the underlying support mesh for integration. Under

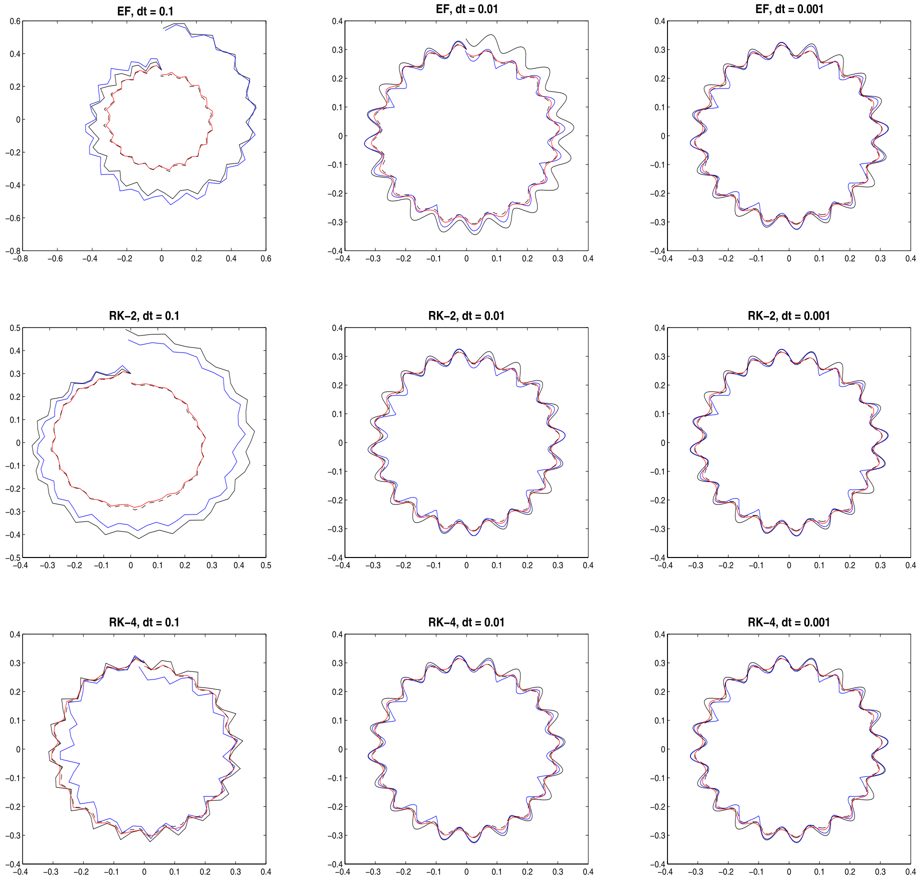

Fig. 8 (Color online) Streamline integration example based upon vector field mentioned in (32). *Solid black streamlines* denote "true" solution; *blue streamlines* were created based upon integration on an L_2 projected field; *red streamlines* were created based upon integration on a filtered field using consistent integration approach and *dashed black* streamlines were created based upon integration on a filtered field using the input mesh based approach

this scenario, we examined the use of Gaussian quadrature and of 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 draw from our results and discussions:

- The major uncontrollable cost in the post-processing algorithm is finding the consistent integration mesh when given an arbitrary tessellation. 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 post-processor 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 integrand contains 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 finding the consistent mesh is prohibitive, post-processing with input mesh based post-processing 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 the light of the our focus application area—visualization of DG solutions—input mesh based post-processing appears to provide a convenient means of obtaining smooth solutions with controllable accuracy.

We emphasis again that our 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 visualization tool. It is concerning this specific crime to which we have attempted to provide both theoretical and empirical insight.

Acknowledgements The first and third authors are sponsored in part by the Air Force Office of Scientific Research (AFOSR), Computational Mathematics Program (Program Manager: Dr. Fariba Fahroo), under grant number FA9550-08-1-0156. The second author is sponsored by the Air Force Office of Scientific Research, Air Force Material Command, USAF, under grant number FA8655-09-1-3055. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon.

Open Access This article is distributed under the terms of the Creative Commons Attribution Noncommercial License which permits any noncommercial use, distribution, and reproduction in any medium, provided the original author(s) and source are credited.

References

1. Bramble, J., Schatz, A.: Higher order local accuracy by averaging in the finite element method. Math. Comput. **31**, 94–111 (1977)
2. Cockburn, B.: Discontinuous Galerkin methods for convection-dominated problems. In: High-Order Methods for Computational Physics. Lecture Notes in Computational Science and Engineering, vol. 9. Springer, Berlin (1999)
3. Cockburn, B., Hou, S., Shu, C.-W.: The Runge-Kutta local projection discontinuous Galerkin finite element method for conservation laws IV: the multidimensional case. Math. Comput. **54**, 545–581 (1990)
4. Cockburn, B., Karniadakis, G., Shu, C.-W.: Discontinuous Galerkin Methods: Theory, Computation and Applications. Springer, Berlin (2000)
5. Cockburn, B., Lin, S.-Y., Shu, C.-W.: TVB Runge-Kutta local projection discontinuous Galerkin finite element method for conservation laws III: one dimensional systems. J. Comput. Phys. **84**, 90–113 (1989)
6. Cockburn, B., Luskin, M., Shu, C.-W., Süli, E.: Post-processing of Galerkin methods for hyperbolic problems. In: Proceedings of the International Symposium on Discontinuous Galerkin Methods. Springer, Berlin (1999)
7. Cockburn, B., Luskin, M., Shu, C.-W., Suli, E.: Enhanced accuracy by post-processing for finite element methods for hyperbolic equations. Math. Comput. **72**, 577–606 (2003)
8. Cockburn, B., Ryan, J.: Local derivative post-processing for discontinuous Galerkin methods. J. Comput. Phys. **28**, 8642–8664 (2009)
9. Cockburn, B., Shu, C.-W.: TVB Runge-Kutta local projection discontinuous Galerkin finite element method for conservation laws II: general framework. Math. Comput. **52**, 411–435 (1989)
10. Cockburn, B., Shu, C.-W.: The Runge-Kutta local projection P^1-discontinuous-Galerkin finite element method for scalar conservation laws. Math. Model. Numer. Anal. (M²AN) **25**, 337–361 (1991)
11. Cockburn, B., Shu, C.-W.: The Runge-Kutta discontinuous Galerkin method for conservation laws V: multidimensional systems. J. Comput. Phys. **141**, 199–224 (1998)

12. Cockburn, B., Shu, C.-W.: Runge-Kutta discontinuous Galerkin methods for convection-dominated problems. J. Sci. Comput. **16**, 173–261 (2001)
13. Conte, S., de Boor, C.: Elementary Numerical Analysis. McGraw-Hill, Tokyo (1972)
14. Curtis, S., Kirby, R., Ryan, J., Shu, C.-W.: Post-processing for the discontinuous Galerkin method over non-uniform meshes. SIAM J. Sci. Comput. **30**(1), 272–289 (2007)
15. Karniadakis, G., Sherwin, S.: Spectral/hp Element Methods for CFD, 2nd edn. Oxford University Press, London (2005)
16. Karniadakis, G.E., Kirby, R.M.: Parallel Scientific Computing in C++ and MPI. Cambridge University Press, New York (2003)
17. Ryan, J., Shu, C.-W.: On a one-sided post-processing technique for the discontinuous Galerkin methods. Methods Appl. Anal. **10**, 295–307 (2003)
18. Ryan, J., Shu, C.-W., Atkins, H.: Extension of a post-processing technique for the discontinuous Galerkin method for hyperbolic equations with application to an aeroacoustic problem. SIAM J. Sci. Comput. **26**, 821–843 (2005)
19. Schumaker, L.: Spline Functions: Basic Theory. Wiley, New York (1981)
20. Steffen, M., Curtis, S., Kirby, R., Ryan, J.: Investigation of smoothness enhancing accuracy-conserving filters for improving streamline integration through discontinuous fields. IEEE Trans. Vis. Comput. Graph. **14**, 680–692 (2007)
21. Szabó, B., Babuska, I.: Finite Element Analysis. Wiley, New York (1991)
22. Walfisch, D., Ryan, J.K., Kirby, R.M., Haimes, R.: One-sided smoothness-increasing accuracy-conserving filtering for enhanced streamline integration through discontinuous fields. J. Sci. Comput. **38**, 164–184 (2009)