Spectral Element Method

Support Modules

Technical Reference Manual

Joseph Schoonover

Contents

1 Purpose and Overview

Spectral Element Methods (SEMs) offer a flexible approach to solving partial differential equations in complicated geometry that allows for either algebraic or spectral convergence. Nodal SEMs are popular because they yield localized dense operations for computing the action of derivative operators. These dense computations can be performed in parallel with a trivial amount of modifications to existing serial code. Though the number of operations increases with the use of spectral element methods, there is a compensating gain in solution accuracy. With an algorithm that is appropriately parallelized, the numerical modeler can achieve efficient performance while producing highly accurate solutions.

The software described herein is a collection of support modules that enable the development of sophisticated NSEM solvers for partial differential equations. These include modules for

1. Memory and large code management (e.g. error exception handlers).
2. Miscellaneous, multipurpose subroutines (e.g. creating new file units, linked lists, hashtables)
3. Performing Lagrange interpolation, and computing operations on interpolants.
4. Solution storage handlers for Continous and Discontinuous Galerkin Spectral element methods.
5. Defining and manipulating a spectral element mesh.

The goal of this documentation is to supply you with an intuitive knowledge of software organization and supply you with the necessary theory to make practical use of the provided

modules to solve more complicated problems. This chapter contains information which helps motivate the software structure and may provide you with a more intuitive feel for its use. We start with a description of how domain decomposition is handled in a spectral element method since this will ultimately determine the software structure. Following the more descriptive discussion is a more detailed look at the spectral element discretizations that yield the various first order ($\nabla\cdot$, ∇, $\nabla\times$) and second order derivative operators. The descriptive overview and detailed discretization will give you a sense of the routines that are necessary for constructing a SE solver.

1.1 Descriptive Domain Decomposition

In a SE discretization, a global domain that is representative of the physical space that you are modelling is broken into a collection of *logical* segments (1-D), quadrilaterals (2-D), or hexahedrons (3-D) that are called *elements*. Logical quadrilaterals are geometric objects that can be mapped to a reference, or computational, quadrilateral. Logical hexahedrons are defined similarly. The elements can be generated via a structured or unstructured mesh generation process, so long as sufficient connectivity information is given to aid in data sharing amongst neighboring elements. Within each element is a structured *quadrature mesh* that is set by the basis function that is being employed. A sample spectral element mesh is shown in Fig. **??**.

All SEMs then consist of local operations within each element that can be done independent of other elements and a set of nearest neighbor data exchanges. Because of this decomposition, SEMs lend themselves easily to shared memory parallelization (threading with OpenMP), domain decomposed parallelization (message passing interface, e.g. Open-MPI), or some combination of the two.

1.2 Spectral Element Methods

First, we discuss a taxonomy of SEMs and relate them to general Spectral Methods and Pseudo-Spectral methods in order to provide context. In general, spectral methods refer to

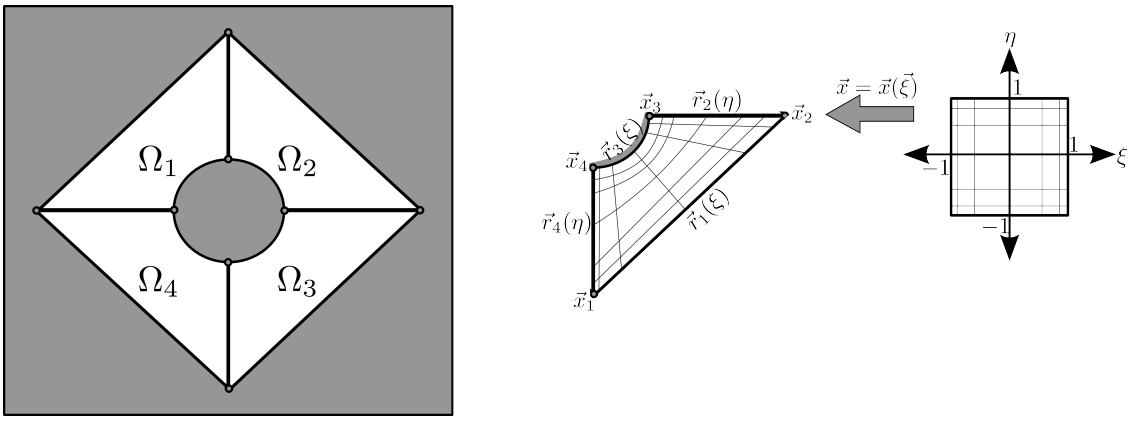

Figure 1.1: A simple schematic of a spectral element mesh. This schematic shows four spectral elements, with isoparametric (curved) sides.

the numerical algorithms which estimate solutions to PDEs and have error convergence rates that depend only on the smoothness of the solution. This error convergence is obtained by approximating the solution by a linear combination of basis functions. The choice in basis function is usually motivated by the application. For example, periodic systems can easily make use of Fourier basis which automatically satisfy periodic boundary conditions. Other problems in which the prescription of arbitrary boundary conditions is necessary make use of Jacobi polynomials. Historically, Chebyshev and Legendre polynomials have been used. In addition to the choice in basis function, the PDE can be approximated in the strong or the weak form. So-called "pseudo-spectral" methods approximate the strong form while *Galerkin* methods approximate the weak form.

Galerkin methods have typically been used for problems with complicated geometry where an unstructured mesh is necessary; these are *multi-domain* methods. In these problems, the physical domain is decomposed into many "spectral elements". Over each element, the solution is approximated by a suitable polynomial basis. The global solution is constructed by piecewise polynomial interpolation. Galerkin methods which require continuity between spectral elements are called *Continuous Galerkin Spectral Element Methods* (CGSEMs). Those which do not enforce continuity are *Discontinuous Galerkin Spectral Element Methods* (DGSEMs). Historically, CGSEMs have been applied to elliptic and parabolic PDEs while DGSEMs have been applied to hyperbolic problems.

2 SEM Discretizations

The main focus of the software is on Legendre-Galerkin Spectral Element Methods. The software can be broken into two distinct components:

1. SEM algorithm core
2. Geometry

Both of these components rely on basic interpolation routines which are used to evaluate certain variables (e.g., the geometric mapping or the solution) over each spectral element.

A generic discretization is outlined for hyperbolic conservation laws to motivate the necessary modular components of the software for DGSEMs. Similarly, a generic elliptic or parabolic PDE is discretized to motivate the modular components of the software for CGSEMs. This illustrates how the two distinct components can be used together to solve PDEs in complicated geometry. Additionally, this exposition will outline the necessary attributes of data-structures that will be used in an object-oriented approach.

2.1 DGSEM

A conservation law can be written in the form

$$\vec{s}_t + \nabla \cdot \vec{f} + \vec{q} = 0, \tag{2.1}$$

where $\vec{s}$ is a vector of *prognostic* variables, $\vec{f}$ is a vector of *conservative flux vectors* and $\vec{q}$ is a vector of *non-conservative source terms*.

The DGSEM discretizes (2.1) in its weak form. To obtain the weak form, (2.1) is weighted with a test function and integrated over the physical domain, call it Ω. To approximate the

integrals in the weak form, the domain is first divided into non-overlapping elements (Ω^κ) . Integration is performed over each element and the solution and test function are permitted to be piecewise discontinuous across elements. These assumptions lead to the statement

$$\int_{\Omega^\kappa} (\vec{s}_t + \vec{q})\phi \, d\Omega^\kappa - \int_{\Omega^\kappa} \vec{f} \cdot \nabla\phi \, d\Omega^\kappa + \oint_{\partial\Omega^\kappa} \phi \vec{f} \cdot \hat{n} \, dA^\kappa = 0, \quad \forall \phi \in \mathbb{C}_0(\Omega_\kappa), \quad \kappa = 1, 2, ..., K \tag{2.2}$$

where $\mathbb{C}_0(\Omega^\kappa)$ is the space of functions that are continuous over Ω^κ.

Equation (2.2) assumes that the integration over each element is independent; this *compactness* results from allowing piecewise discontinuous solutions. The third term in (2.2) is the integral of the conservative flux over the element boundary. This is the only term that involves communication with other elements. For hyperbolic problems, this communication is only between *neighboring elements* which share a common node (1-D), edge (2-D), or face (3-D).

The formulation presented in (2.2) only requires that we know the geometry of each element and the connectivity of a collection of elements. This allows for the use of either structured or unstructured mesh frameworks. Additionally, the elements which comprise the mesh can have curvilinear geometry. Define the mapping from physical space $\vec{x}$ to computational space $\vec{\xi}$ using

$$\vec{x} = \vec{x}(\vec{\xi}). \tag{2.3}$$

Section 2.3 provides the details on the metric terms that are introduced along with the form of the divergence, gradient, and curl under such a mapping. For simplicity, the computational domain is formed from tensor products of intervals $[-1, 1]$ in each coordinate direction. This criteria restricts the elements to be logically segments (1-D), quadrilaterals (2-D), or hexahedrons (3-D). Under the mapping (2.3), the weak form (2.2) becomes

$$\int_{\Omega^\xi} (\tilde{s}_t^\kappa + \tilde{q}^\kappa)\phi \, d\Omega^\xi - \int_{\Omega^\xi} \tilde{f}^\kappa \cdot \nabla_{\vec{\xi}}\phi \, d\Omega^\xi + \oint_{\partial\Omega^\xi} \phi \tilde{f}^{\kappa,*} \cdot \hat{n}^\xi \, dA^\xi = 0, \quad \forall \phi \in \mathbb{C}^0(\Omega^\xi) \tag{2.4}$$

where $\tilde{s} = J^\kappa \vec{s}$, $\tilde{q} = J^\kappa \vec{q}$, $\tilde{f} = (J^\kappa \vec{a}^{\kappa,i} \cdot \vec{f})\hat{a}^{\kappa,i}$, J^κ is the Jacobian of the transformation over the κ^{th} element, and $\vec{a}^{\kappa,i}$ are the contravariant basis vectors associated with the transformation of element κ (the $\hat{a}$ denotes a unit vector).

Given a conservative flux, a non-conservative source, internal element metrics, and global element connectivity, the discrete algorithm solves (2.4) by approximating the integrands

with interpolants and the integrals by discrete quadratures. First, the prognostic solution, source term, conservative flux, and mapping are approximated by Lagrange interpolants of degree N.

$$\vec{s} \approx I^N(\vec{s}) = \sum_{i,j,k=0}^{N} \vec{S}_{i,j,k} l_i(\xi^1) l_j(\xi^2) l_k(\xi^3) \tag{2.5a}$$

$$\vec{q} \approx I^N(\vec{q}) = \sum_{i,j,k=0}^{N} \vec{Q}_{i,j,k} l_i(\xi^1) l_j(\xi^2) l_k(\xi^3) \tag{2.5b}$$

$$\vec{f} \approx I^N(\vec{f}) = \sum_{i,j,k=0}^{N} \vec{F}_{i,j,k} l_i(\xi^1) l_j(\xi^2) l_k(\xi^3) \tag{2.5c}$$

$$\vec{x} \approx I^N(\vec{x}) = \sum_{i,j,k=0}^{N} \vec{X}_{i,j,k} l_i(\xi^1) l_j(\xi^2) l_k(\xi^3) \tag{2.5d}$$

An additional simplification is to approximate the product of interpolants by an interpolant of degree N which incurs an additional aliasing error.

$$\tilde{s} \approx I^N(I^N(J) I^N(\vec{s})) = \tilde{S} = \sum_{i,j,k=0}^{N} (J\vec{S}_{i,j,k}) l_i(\xi^1) l_j(\xi^2) l_k(\xi^3) \tag{2.6a}$$

$$\tilde{q} \approx I^N(I^N(J) I^N(\vec{q})) = \tilde{Q} = \sum_{i,j,k=0}^{N} (J\vec{Q})_{i,j,k} l_i(\xi^1) l_j(\xi^2) l_k(\xi^3) \tag{2.6b}$$

$$\tilde{f}_{(i,n)} \approx I^N(I^N(Ja^i_{(n)}) I^N(\vec{f})) = \tilde{F}_{(i,n)} = \sum_{i,j,k=0}^{N} (Ja^i_{(n)} \vec{F}_{(n)})_{i,j,k} l_i(\xi^1) l_j(\xi^2) l_k(\xi^3) \tag{2.6c}$$

Last, the space of test functions ($\mathbb{C}_0$) is approximated by the $\mathbb{P}^N$, the space of polynomials of degree N. Thus, the test function ϕ is replaced by each of the Lagrange interpolating polynomials,

$$\phi_{m,n,p} = l_m(\xi^1) l_n(\xi^2) l_p(\xi^3) \tag{2.7}$$

With this, equation (2.4) becomes

$$\int_{\Omega^\xi} (\tilde{S}^\kappa_t + \tilde{Q}^\kappa) \phi_{m,n,p} \, d\Omega^\xi - \int_{\Omega^\xi} \tilde{F}^\kappa \cdot \nabla_{\vec{\xi}} \phi_{m,n,p} \, d\Omega^\xi + \oint_{\partial\Omega^\xi} \phi_{m,n,p} \tilde{F}^{\kappa,*} \cdot \hat{n}^\xi \, dA^\xi = 0, \quad m,n,p = 0,1,...,N \tag{2.8}$$

The final step is to replace the integrals in (2.8) with discrete quadrature. For this we use the Legendre-Gauss quadrature, which yields exact integration for each term in (2.8). Additionally, the interpolation nodes are specified as the Legendre-Gauss nodes, which simplifies the integration. The algorithm is now presented in one, two, and three dimesnions.

Algorithm in One-Dimensions

In one-dimension, the global physical domain is partitioned into a set of line segments. The first term in (2.8) becomes

$$\int_{\Omega^\xi} \tilde{S}_t^\kappa \phi_m \, d\Omega^\xi = \sum_{\alpha=0}^{N} \left[\sum_{i=0}^{N} \left((\tilde{S}_i^\kappa)_t l_i(\xi_\alpha) \right) l_m(\xi_\alpha) \omega_\alpha \right] \tag{2.9}$$

where the ξ_α are the Legendre-Gauss nodes and the ω_m are the Legendre-Gauss weights. Since the interpolation nodes are equivalent to the quadrature nodes, (2.9) is greatly simplified by the use of the Kronecker-delta property of Lagrange interpolating polynomials,

$$l_i(\xi_m) = \delta_{i,m}. \tag{2.10}$$

This yields (2.9) as

$$\int_{\Omega^\xi} \tilde{S}_t^\kappa \phi_\alpha \, d\Omega^\xi = (\tilde{S}_\alpha^\kappa)_t \omega_\alpha. \tag{2.11}$$

Similarly,

$$\int_{\Omega^\xi} \tilde{Q}^\kappa \phi_\alpha \, d\Omega^\xi = \tilde{Q}_\alpha^\kappa \omega_\alpha. \tag{2.12}$$

The third term in (2.8) becomes

$$\int_{\Omega^\xi} \tilde{F}^\kappa \cdot \nabla_{\vec{\xi}} \phi_m \, d\Omega^\xi = \sum_{i=0}^{N} \tilde{F}_i^\kappa l'_m(\xi_i) \omega_i. \tag{2.13}$$

Lastly, the boundary flux becomes

$$\oint_{\partial\Omega^\xi} \phi_\alpha \tilde{F}^{\kappa,*} \cdot \hat{n}^\xi \, dA^\xi = \tilde{F}^{\kappa,*}(1) l_\alpha(1) - \tilde{F}^{\kappa,*}(-1) l_\alpha(-1). \tag{2.14}$$

Thus, the spatially discretized system can be written

$$\left(J_m \vec{S}_m \right)_t = - \left[\sum_{i=0}^{N} \hat{D}_{m,i} \tilde{F}_i + \left(\frac{l_m(1)}{w_m} \tilde{F}^*(1) - \frac{l_m(-1)}{w_m} \tilde{F}^*(-1) \right) \right] + J_m \vec{Q}_m; \quad m = 0, 1, ..., N, \tag{2.15}$$

where

$$\hat{D}_{m,i} = - \frac{l'_m(\xi_i) w_i}{w_m^{\xi^1}} \tag{2.16}$$

is the DG-Derivative matrix.

Up to this point, the treatment of the boundary flux has been relatively vague. In practice, the boundary flux is calculated using an approximate Riemman solver. This uses the solution state from the neighboring elements to approximate the flux across the shared boundary. The specification of the Riemman solver depends on the PDE system. Because of this, a discussion of the boundary flux is delayed until specific PDE systems are discussed.

Algorithm in Two-Dimensions

$$\begin{aligned}\left(J_{m,n}\vec{S}_{m,n}\right)_t = &-\left[\sum_{i=0}^{N}\hat{D}_{m,i}^{(\xi^1)}\tilde{F}_{i,n}^{(\xi^1)} + \left(\frac{l_m(1)}{w_m^{(\xi^1)}}\tilde{F}^*(1,\xi_n^2) - \frac{l_m(-1)}{w_m^{(\xi^1)}}\tilde{F}^*(-1,\xi_n^2)\right)\cdot\hat{\xi^1}\right] \\ &-\left[\sum_{j=0}^{N}\hat{D}_{n,j}^{(\xi^2)}\tilde{F}_{m,j}^{(\xi^2)} + \left(\frac{l_n(1)}{w_n^{(\xi^2)}}\tilde{F}^*(\xi_m^1,1) - \frac{l_n(-1)}{w_n^{(\xi^2)}}\tilde{F}^*(\xi_m^1,-1)\right)\cdot\hat{\xi^2}\right] \\ &+ J_{m,n}\vec{Q}_{m,n}; \quad m,n = 0,1,...,N\end{aligned} \tag{2.17}$$

Computing the divergence of the conservative flux in this framework can be viewed as calculating a sequence of derivatives in each computational direction. Two steps are required to compute the derivative in each direction. The first is an internal matrix-vector multiply, and the second is computing the weighted Riemann fluxes at the element boundaries. The latter is the only step which requires element-to-element communication.

Algorithm in Three-Dimensions

$$\begin{aligned}\left(J_{m,n,p}\vec{S}_{m,n,p}\right)_t = &-\left[\sum_{i=0}^{N}\hat{D}_{m,i}^{(\xi^1)}\tilde{F}_{i,n,p}^{(\xi^1)} + \left(\frac{l_m(1)}{w_m^{(\xi^1)}}\tilde{F}^*(1,\xi_n^2,\xi_p^3) - \frac{l_m(-1)}{w_m^{(\xi^1)}}\tilde{F}^*(-1,\xi_n^2,xi_p^3)\right)\cdot\hat{\xi^1}\right] \\ &-\left[\sum_{j=0}^{N}\hat{D}_{n,j}^{(\xi^2)}\tilde{F}_{m,j,p}^{(\xi^2)} + \left(\frac{l_n(1)}{w_n^{(\xi^2)}}\tilde{F}^*(\xi_m^1,1,\xi_p^3) - \frac{l_n(-1)}{w_n^{(\xi^2)}}\tilde{F}^*(\xi_m^1,-1,\xi_p^3)\right)\cdot\hat{\xi^2}\right] \\ &-\left[\sum_{k=0}^{N}\hat{D}_{m,n,k}^{(\xi^3)}\tilde{F}_{m,n,k}^{(\xi^2)} + \left(\frac{l_p(1)}{w_p^{(\xi^3)}}\tilde{F}^*(\xi_m^1,\xi_n^2,1) - \frac{l_p(-1)}{w_p^{(\xi^3)}}\tilde{F}^*(\xi_m^1,-\xi_n^2,-1)\right)\cdot\hat{\xi^3}\right] \\ &+ J_{m,n,p}\vec{Q}_{m,n,p}; \quad m,n,p = 0,1,...,N\end{aligned} \tag{2.18}$$

2.2 CGSEM

2.3 Geometry

2.4 Code Organization

Two classes of SEMs have been presented. Both require the use of Lagrange interpolating polynomials and Legendre Gauss or Gauss-Lobatto quadrature. This necessitates the development of low level support modules which provide a data structure and routines for interpolation and differentation. Auxiliary routines are provided separately to provide the Legendre Gauss or Gauss-Lobatto nodes and weights. To calculate the solution tendency, the solution at the quadrature nodes and at the boundaries is needed. Additionally, the quadrature weights and (some form of) the derivative matrix are needed. This motivates the development of two data structures. The first stores the "book-keeping" information such as the interpolant, derivative matrix, and the quadrature weights. The second stores the solution, flux, and tendency (for the DGSEM). Lastly, it was seen that a data structure is needed for handling the SE mesh information such as the element, edge, and node connectivity and internal geometry of each element. Together, these modules can be used to develop "high-end" solvers PDE systems. Under each class of modules, a single module is developed for one, two, and three dimensions.

3 Modules

3.1 Interpolation

3.2 Filters

Models are just that, they're models. We cannot currently hope to explicitly and accurately represent all of the spatial and temporal scales for a problem. Instead, models focus in on the spatial and temporal scales that are important and an informed decision is made to deal with the interactions with smaller scales. In the prototypical problem of shock formation (a la Burger's equation), small scales are forced into existence from the large scale. This nonlinear transfer of energy to small scales, called the "forward cascade", poses challenges for numerical models that can become unstable when too much energy is forced into the smallest resolvable scale. The comic shown in Fig. 3.1 shows what happens when you don't remove that energy from the grid scale (the red line). The blue line shows the same shock formation occurring, except that the numerical algorithm now includes a "filter" that turns on when too much energy appears in the grid-scale and violates the principles of the forward cascade (it is called "adaptive" because of its conditional action). The filter effectively removes the oscillations, so-called "grid-scale noise", and removes enough energy to maintain stable numerical integration. The filter used in generating the stable solution in Fig. 3.1 is a "Roll-off" filter, meaning that it damps marginally resolved modes much stronger than the well resolved modes. This Adaptive Roll-off Filter (ARF) is a parameterization for shock formation. Motivated by the need for maintaining numerical stability in a justifiable manner, a set of filters are provided with the SELF as one option towards meeting this goal. In this

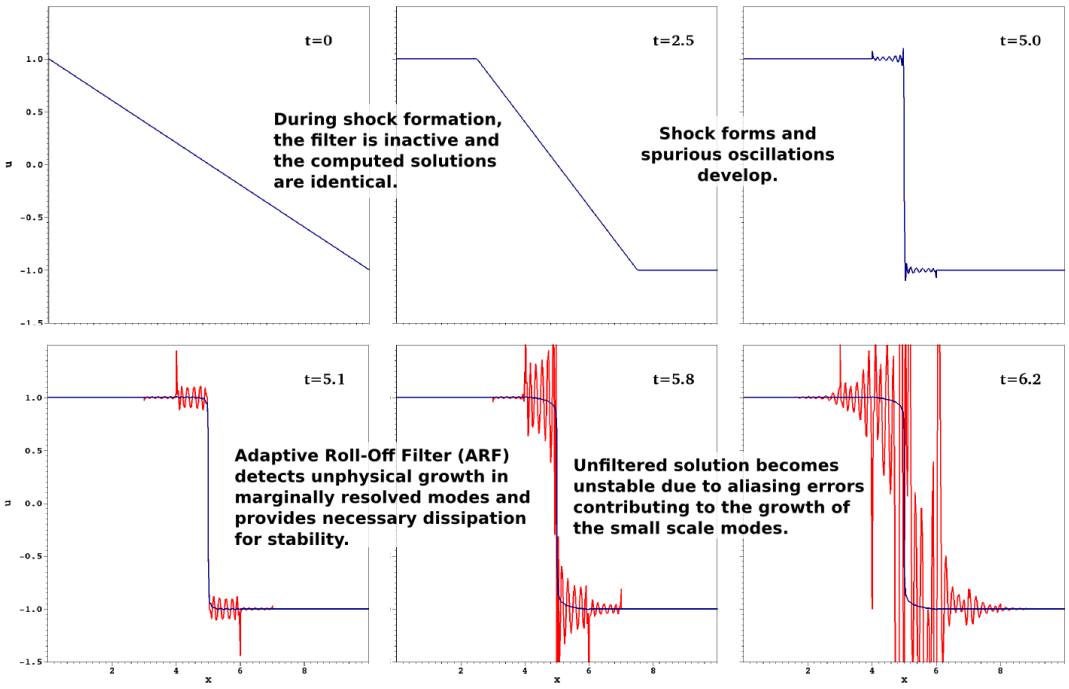

Figure 3.1:

section, the theory behind using filters is presented for 1-D. The advantage that polynomial filtering has over other parameterizations (e.g. Laplacian diffusion), is that the additional error incurred by such a filter is quantifiable.

In the "standard" nodal discretization of conservation laws presented in Eqns. (2.15) (2.17) and (2.18), additional aliasing errors can arise in approximating the flux by a polynomial of the same degree as the solution. This is always the case when the flux depends nonlinearly on the solution. The introduction of aliasing errors directly implies inexactness of the discrete integration and hence stability of the spatial discretization does not follow. However, the quadrature order can be increased in order to remove these aliasing errors which can then restore stability (provided the continuous problem is well posed). This approach is known as *over-integration*. In theory, over-integration would involve interpolation of the solution onto a higher order mesh, integration of the flux, and mapping back to the lower order mesh. In practice, this process can be done more efficiently by *polynomial de-aliasing* (Gassner and Beck, 2013; Flad et al., 2016). In the de-aliasing framework, over-integration

is achieved by applying a *modal-cutoff filter* to the solution variable that effectively projects the solution to a polynomial of low-enough degree so that the integration of the flux is exact.

As an example, suppose we are interested in solving Burger's equation in 1-D, as in Fig. 3.1,

$$u_t + \left(\frac{u^2}{2}\right)_x = 0 \tag{3.1a}$$

$$u(0,t) = 1 \tag{3.1b}$$

$$u(10,t) = -1 \tag{3.1c}$$

$$u(x,0) = -\frac{2x}{10} + 1 \tag{3.1d}$$

If u is approximated by a polynomial of degree N, then the integrand in the discretized system (2.15) for the flux integral is of degree $3N$. Gauss quadrature of order M is exact for integrands of degree $2M+1$, so

$$3N = 2M + 1 \tag{3.2}$$

guarantees exactness of the discrete integration and stability of the spatial discretization. In practice, the discretization of (3.2) would be performed at degree M, and at every iteration the solution would be projected onto a polynomial of degree N that satisfies the exactness criteria (3.2).

How to de-alias

Recall that the approximate solution can be expressed in *nodal* or *modal* form :

$$u \approx I_M[u] = \sum_{i=0}^{M} U_i l_i(\xi) = \sum_{j=0}^{M} \hat{U}_j L_j(\xi) \tag{3.3}$$

where the U_i are the nodal values of the interpolant, l_i are the Lagrange interpolating polynomials and $\hat{U}_j$ are the modal coefficients of the Legendre basis functions L_j. Other polynomial bases can be used, but here our attention is restricted to Legendre polynomials. Gauss quadrature of order M guarantees discrete orthogonality between the Legendre polynomials up to degree M. Because of this, weighting (3.3) with each Legendre polynomial and

performing discrete integration provides a formula for each of the modal coefficients,

$$\hat{U}_m = \sum_{k=0}^{M} U_k \frac{L_m(\xi_k)\sigma_k}{||L_m||_M^2}. \tag{3.4}$$

Eq. (3.4) can be conveniently be written in matrix form

$$\hat{U} = T\vec{U} \tag{3.5}$$

where $\hat{U}$ is a vector of the modal coefficients, $\vec{U}$ is a vector of the nodal values, and $T_{m,k} = \frac{L_m(\xi_k)\sigma_k}{||L_m||_M^2}$ is the nodal-to-modal transformation matrix. The inverse of T transforms from the modal space to the nodal space and is easily obtained by evaluating the Legendre polynomials at the interpolation nodes :

$$T_{m,k}^{-1} = L_k(\xi_m) \tag{3.6}$$

Filtering is done by transforming the nodal values to the modal coefficients, modifying the modal coefficients, and applying the inverse to map the "filtered" interpolant back to the nodal values.

1. Obtain the modal coefficients : $\hat{U} = T\vec{U}$
2. Apply a filter on the modal coefficents : $\tilde{U} = P\hat{U} = PT\vec{U}$
3. Map from the filtered modal coefficients to the nodal coefficients : $\vec{U}_f = (T^{-1}PT)\vec{U}$

When de-aliasing, the modal coefficients associated with Legendre polynomials higher than degree N are set to zero identically, so that the interpolant is effectively projected onto a lower order polynomial. The modal filter matrix that does this is a diagonal matrix with

$$P_{i,i} = \begin{cases} 1, & \text{if } i \leq N \\ 0, & \text{otherwise} \end{cases} \tag{3.7}$$

The matrix $T^{-1}PT$ with P given by (3.7) is called a *modal-cutoff filter*.

4 Software Validation

4.1 Lagrange Interpolation

Guassian Bump in 1-D

This example tests the functionality of the following modules :

1. Legendre.f90
2. Lagrange_1D_Class.f90

By interpolating through the Legendre-Gauss nodes, the error in the interpolating function should decay spectrally with increasing interpolating polynomial degree (Kopriva, 2009). The interpolant can be expressed as the sum of the actual function and an interpolation error,

$$I_N[f(s)] = f(s) + \epsilon(N), \tag{4.1}$$

where $I_N[f(s)]$ denotes the interpolation of $f(s)$ by a degree N. When the evaluation nodes are the Legendre Gauss nodes, $\epsilon(N)$ is a spectrally small error; it's rate of decay with N depends on the smoothness of f. Additionally, the derivative of the interpolant approximates the derivative of the function.

$$\frac{d}{ds}[I_N(f)] = \frac{df}{ds} + \delta(N) \tag{4.2}$$

where $\delta(N)$ is also a spectrally small error. For the interpolation of smooth functions, the error decays exponentially with N.s

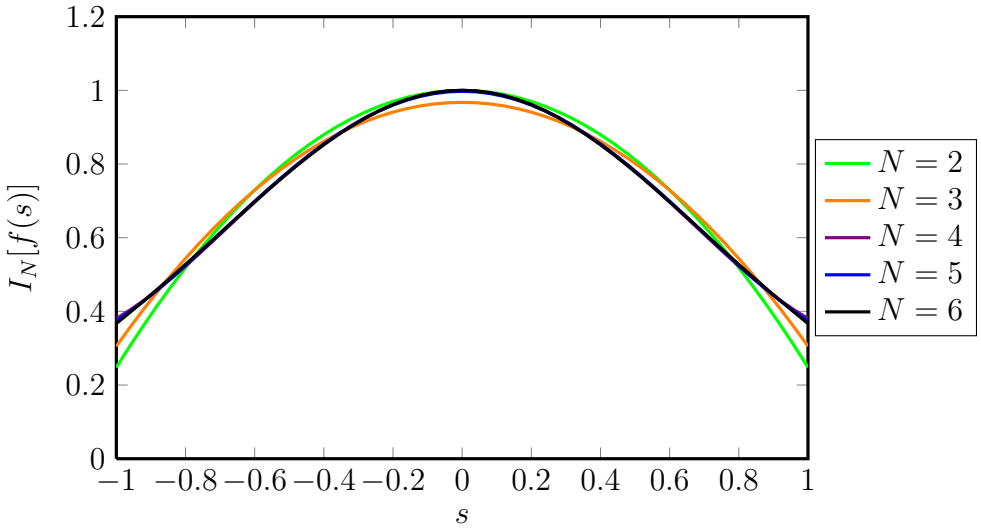

Figure 4.1: The interpolants are shown for polynomial degrees of $N = 2$ through $N = 6$. For presentational purposes, the interpolants are mapped to a uniform grid with 50 points.

A test program (∼`/src/interp/Testing/TestLagrangeInterpolation_1D.f90`) has been written to interpolate the function

$$f(s) = e^{-s^2} \tag{4.3}$$

and estimate the $\mathbb{L}_2$ error of the interpolation and differentiation. Differentiation is carried out by building the derivative matrix (`S/R CalculateDerivativeMatrix`) and performing a matrix vector multiply (`MATMUL`). The $\mathbb{L}_2$ error is estimated using Legendre-Gauss quadrature with 50 evaluation nodes, ie,

$$||f - I_N[f]||_{\mathbb{L}_2} = \sum_{k=0}^{K} \left(f(s_k) - I_N[f(s_k)]\right)^2 \sigma_k \tag{4.4}$$

where s_k and σ_k are the degree K Legendre Gauss quadrature nodes quadrature weights respectively.

Figures 4.1 and 4.2 show the interpolants and the interpolant derivatives respectively for polynomial degrees $N = 2$ through $N = 7$. Qualtitatively, these plots illustrate the convergence of the interpolants to the exact function and derivative. Figure 4.3 shows the $\mathbb{L}_2$ error norm in approximating the function and the derivative for polynomial degrees 2 through 17. Both errors exhibit exponential convergence, in agreement with the theoretical error bounds. Notice that the error in approximating the function is always smaller than in

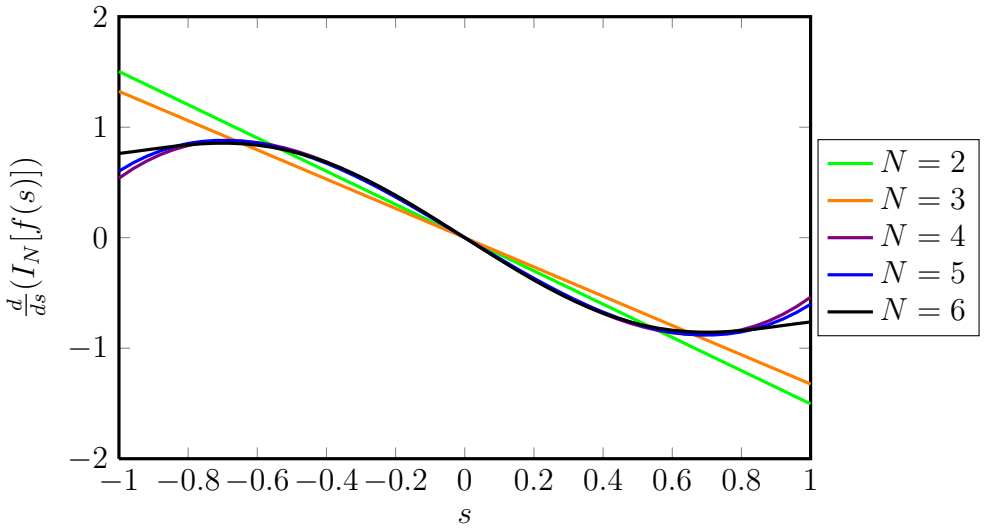

Figure 4.2: The derivatives of the interpolants are shown for polynomial degrees of $N = 2$ through $N = 6$. For presentational purposes, the derivatives are mapped to a uniform grid with 50 points using the `CoarseToFine` subroutine.

approximating its derivatine (for each polynomial degree). This indicates that approximating the derivative by taking the derivative of the interpolant incurs an additional error; the derivative of the interpolant is not always equal to the interpolant of the derivative.

Given that the function being interpolated is complete (we can take as many derivatives as we like), the theoretical error decay is exponential. This is in agreement with the error trend observed numerically, providing evidence that the interpolation and differentiation routines at the Legendre Gauss nodes are working.

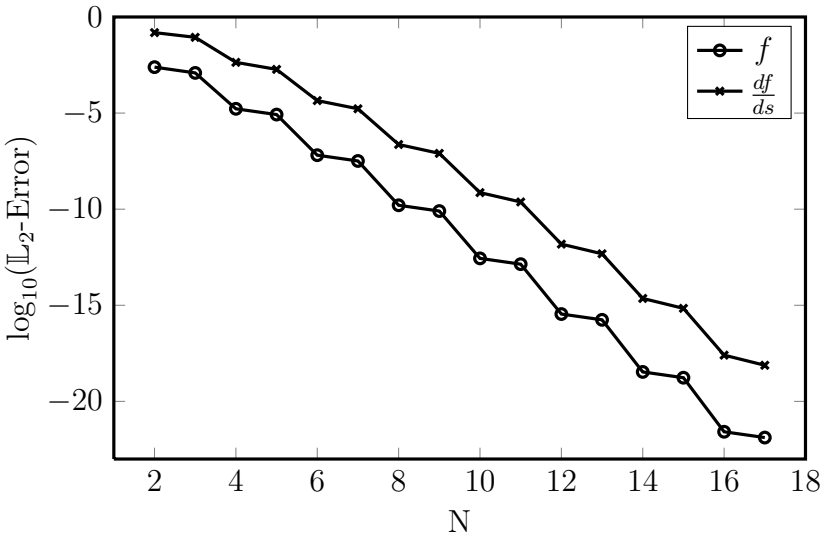

Figure 4.3: A semi-log plot of the $\mathbb{L}_2$ Error norm in approximating a gaussian function and its derivative are shown as a function of the approximating polynomial degree. The error decay is shown in this case to be exponential.

Bibliography

D. Flad, A. Beck, and C. Munz. Simulation of underresolved turbulent flows by adaptive filtering using the high order discontinous Galerkin spectral element method. 313:1–12, 2016.

G. Gassner and A. Beck. On the accuracy of high-order discretizations for underresolved turbulence simulations. *Theor. Comput. Fluid Dyn.*, 27:221–237, 2013.

D. Kopriva. *Implementing spectral methods for partial differential equations*, volume 33. Springer, 2009.