Application of the Level Set Method to Solid Rocket Motor Simulation

by

Kevin M. Albarado

A thesis submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Auburn, Alabama
August 4, 2012

Keywords: rocket propulsion, solid rocket, level set, discontinous Galerkin

Copyright 2012 by Kevin M. Albarado

Approved by

Roy Hartfield, Chair,Woltosz Professor of Aerospace Engineering
John Burkhalter, Professor Emeritus of Aerospace Engineering
Andrew Shelton, Assistant Professor of Aerospace Engineering
George Flowers, Professor of Mechanical Engineering, Dean of Graduate School

Abstract

The body of work encompassed in this thesis merges two advanced concepts for developing flow solutions with level sets to develop an accurate and efficient method for simulating solid rocket motor grain regression. The level-set method has been implemented using the discontinous Galerkin numerical method (DGM) to represent the burning surface area and chamber volume as a function of time. The combination of DGM with geometrically arbitrary solid motors presents a unique and novel environment for solid rocket motor analysis, giving the user more freedom for design and a more accurate result. This thesis provides a complete background and introduction to both solid rocket motor research completed and currently underway as well as an introduction to computational fluid dynamics and the level set method. Development of the LSM/DGM approach to the grain regression problem in two-dimensions is presented with complementary comparisons to analytical approaches. This work represents the first known implementation of the discontinous Galerkin method with a level set to solve the grain regression problem.

Acknowledgments

I would first like to thank my committee for the opportunity and encouragement to pursue this work. I would like thank Dr. Hartfield first and foremost for his knowledge and expertise in the rocket propulsion field. He provided the foundation from which this work developed. Dr. Shelton provided a unique perspective on how to approach this work, and also helped tremendously in implementation of the approach developed in this thesis. Without his input, the end product of this thesis would not have been realized. Finally, I thank Dr. Burkhalter for advice on life decisions and continuous support and encouragement. Without an open door policy from all three members of my committee, this work would not have been completed. I would also like to thank my family and friends for their continued support and encouragement as I make the transition from academic life to the “real world”.

Table of Contents

List of Figures

Chapter 1
Introduction

The roots of solid rocketry can be dated back to the 13th century, when the Chinese first used gunpowder as a solid fuel propellant in warfare. The Arabs, Indians, and Mongolese shortly followed the Chinese in developing some of the earliest rocket weaponry. The earliest rockets were simple bamboo rods filled with gunpowder and carried incendiary material and shrapnel. While history is filled with examples of rocketry use in warfare, solid rocket motor design would go relatively unchanged for hundreds of years. During the American Revolution, William Congreve developed rockets for use by the British military. The advancement made by Congreve in these rockets were the cone-shaped cavity in the end of the propellant. This cavity provided a higher burn area resulting in higher pressure and mass flow rate, in turn producing much higher thrust levels. This was a significant advancement leading engineers to begin experimenting with grain modification for performance enhancement. In the late 1800's, solid rockets saw improvement in propellants as Paul Vieille and Alfred Nobel began experimenting with explosive components such as nitroglycerine. In the early 1900's, most of the solid rocket motors produced were used in rockets for military use due to the long shelf life of the propellant. Improvements in solid rocket motors were restricted to mainly propellant development, led by John Parsons, Frank Malina, and Theodore von Karman until the 1960's. Parsons, Malina, and von Karman made significant advances in development of composite solid fuels that enhanced the stability of the motors along with advancing performance. During the 1960's, the idea of a Space Shuttle had developed, and the design of the two largest solid rockets in history began. Due to the unprecedented size of these motors, known as the Solid Rocket Boosters (SRB), numerous advancements in solid rocket science were required. During initial development of the SRBs, a simplified treatment of the grain regression problem was given to the analysis. The programs used in this analysis, while accurate, were somewhat tailored specifically to the exact problem at hand. These same tools built during the development of the Shuttle booster program are still in use today in industrial settings, such as Solid Performance Program [1]. SPP still remains the industry standard solid rocket motor analysis tool. A solid rocket motor is simple enough in design and implementation (which is what makes it advantageous over more complicated rocket motors) as it is only composed of four main components: propellant, igniter, case, and nozzle (see Figure 1.1).

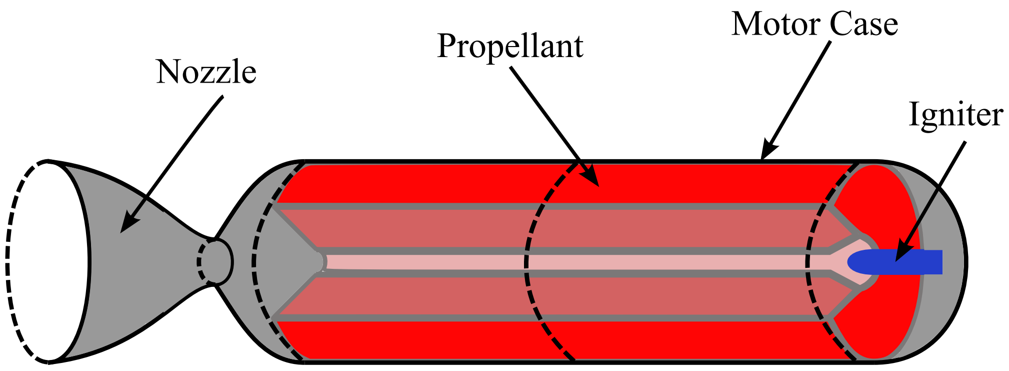


Figure 1.1: Generic SRM Schematic

Despite its simplicity, many physical phenomena occur during a solid motor firing. From ignition to burnout, engineers and scientists have made entire careers out of studying just a handful of the myriad of topics present in solid rocket motor research such as: performance estimation, ignition phenomena, grain regression simulation techniques, chamber aerodynamics, motor stability considerations, propellant solid mechanics, propellant manufacturing, and automated grain design amongst many more.

While this body of work focuses solely on grain regression simulation, a brief discussion of the remaining topics is necessary to fully understand the underlying physics in the unique environment of a solid rocket motor.

1.1 Performance Estimation

In order to analyze a solid rocket motor, some estimation of performance must be available. For a rocket motor, the performance parameters of interest are chamber pressure and thrust. A simple derivation for chamber pressure and thrust can be developed using lumped parameter analysis. The general thrust equation for any chemical propulsion system is a manipulation of the momentum equation assuming steady, uniform flow at the intake and exit planes of the motor nozzle:

$$T = \dot{m}\left(u_e - u_i\right) + A_e\left(p_e - p_a\right) \tag{1.1}$$

For a rocket motor, there is no incoming jet of momentum, so this equation simplifies to just

$$T = \dot{m}u_e + A_e\left(p_e - p_a\right) \tag{1.2}$$

Analyzing Equation 1.2, it should be noted that the unknowns in this equation are $\dot{m}$, u_e, and p_e, and as such are the variables to estimate with a rocket analysis program. These unknowns are simply functions of burn area and propellant properties. The continuity equation requires that the mass production rate of the propellant grain at an time must equal the mass egress rate of the nozzle. This is represented mathematically as

$$A_b r_b \rho_b = \frac{p_o A_t}{c^*} \tag{1.3}$$

Equation 1.3 assumes uniform, steady, and 1-D flow throughout both the combustion chamber and the nozzle. For 2- and 3-D flows, mass weighted averages apply. The burn rate, r_b, is empirically defined to be a function of propellant properties

$$r_b = a p_o^n \tag{1.4}$$

Substituting Equation 1.4 into 1.3 and rearranging yields an equation for the total chamber pressure.

$$p_o = \left(\frac{A_b}{A_t} a \rho_b c^*\right)^{\frac{1}{1-n}} \tag{1.5}$$

In this equation, it is assumed that the propellant burn rate constant, propellant density, characteristic velocity, and burn rate exponent (a, ρ_b, c^*, and n respectively) are knowns as well as the throat area, A_t. This implies that chamber pressure is function of burn area only for a given propellant choice. With total chamber pressure calculated, the mass flow rate is calculated as

$$\dot{m} = \frac{p_o A_t}{c^*} \tag{1.6}$$

Using isentropic nozzle performance equations, the exit pressure can be calculated by solving the Area Ratio-Mach number relation for Mach number, and using isentropic relations to find the pressure ratio.

$$\frac{A_e}{A_t} = \frac{1}{M_e}\left[\frac{2}{\gamma+1}\left(1 + \frac{\gamma-1}{2} M_e^2\right)\right]^{\frac{\gamma+1}{2(\gamma-1)}} \tag{1.7}$$

$$\frac{p_o}{p_e} = \left(1 + \frac{\gamma-1}{2} M_e^2\right)^{\frac{\gamma}{\gamma-1}} \tag{1.8}$$

Equations 1.2, 1.5, 1.6, 1.7, and 1.8 provide the necessary conditions to calculate thrust for a given burn area in a solid rocket motor. The primary work for this thesis is to determine accurately the burn area as a function of time for any generic solid motor. For a more in-depth derivation of the performance estimation see references [2, 3].

1.2 Ignition Phenomena

The ignition of a solid rocket motor has as much of an impact on overall performance as grain design and manufacturing. To start a solid rocket motor, an igniter introduces (usually at the head end) high temperature and high pressure gases or flames. These flames spread throughout the motor gradually heating up the propellant surface. When the propellant reaches the ignition temperature, the propellant burns, releasing large amounts of thermal energy into the chamber. Typically at this point a phenomenon known as pressure overpeak occurs (see Figure 1.2). The whole process takes place in only a few hundred milliseconds for the largest motors such as the Space Shuttle booster and is much shorter in smaller motors, but is significant in computing the total impulse for the motor.

This process has been studied extensively both experimentally and analytically. Reference [4] lists numerous characteristics of igniters including effects of temperature and pressure on ignition time as well as types of igniters and performance characteristics. Foster and Jenkins developed a computer model and performed validation experiments to determine whether a single port igniter or multiport igniter would be more effective at igniting the head end star on the Space Shuttle Redesigned Solid Rocket Motor [5]. Their work was a collaborative effort between Auburn University and NASA's Marshall Space Flight Center. Cho and Baek [6], Bai et al. [7], and Johnston [8] all developed numerical simulations of the internal aerodynamics during ignition in order to analyze ignition transients in an axisymmetric motor. However, Cho and Baek took this work a step further by including radiation effects to determine quantitatively what effect radiation has on ignition. They found that radiation plays an important role in the heat flux to the propellant surface, and without radiation a longer ignition delay is required.

1.3 Chamber Aerodynamics

Chamber aerodynamics can play a large roll in motor performance. While the local velocity gradients and temperature gradients are not large compared to conditions in the nozzle, they play a roll on local pressure as well as the local burn rate. If the burn rate gradient along the surface is large enough, some areas of the motor will burn more rapidly than others, leading to a divergence from predicted burn area. Aerodynamic predictions have been performed in both analytical approaches and full scale CFD.

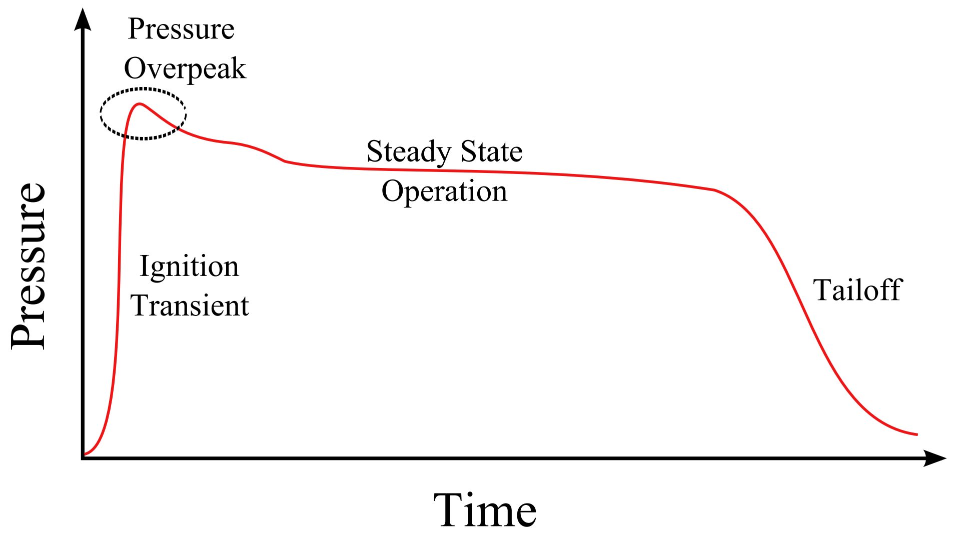


Figure 1.2: Typical pressure-time profile for a solid rocket motor

1.3.1 Analytical Approaches

Numerous assumptions can reasonably be made for solid rocket motor chamber aerodynamics. The set of assumptions made specifically can be referred to as the Taylor-Culick Profile. Taylor and Culick independently determined the conditions and assumptions for solid rocket motors in the 1950's and 60's [9, 10]. The assumptions are that the flow solution is incompressible, rotational, axisymmetric, and quasi-viscous. Majdalani et al. [11, 12, 13, 14, 15] has produced a series of publications detailing the analytical Taylor-Culick profile and its many applications. In [11], basic development of the Taylor-Culick profile was examined to determine the time-dependent velocity field for a flow-field with sidewall injection. Majdalani and Saad examined how an arbitrary headwall injection affected the flowfield solution (this particular effect would be observed in a hybrid rocket motor) [12]. Majdalani, Xu, Lin, and Liao used the Homotopy Analysis Method to examine the Taylor-Culick profile with regressing and injecting sidewalls [13]. This development proved to be vital in development of an analytical aerodynamic model that closely resembles a solid rocket motor.

1.3.2 Full Scale CFD

The use of Computational Fluid Dynamics (CFD) has been more sparse than its analytical counterpart, but has been instrumental in solving unsteady aerodynamic problems in solid rocket motors. Chedevergne, Casalis, and Majdalani [16] used Direct Numerical Simulation (DNS) to investigate the validity of the Taylor-Culick profile. Apte and Yang used Large Eddy Simulation (LES) to investigate the effects of turbulence in a rocket motor with respect to chemical combustion rates [17].

1.4 Motor Stability Considerations

Pressure fluctuations within solid motors can lead to undesired and sometimes catastrophic effects on the motors performance. This phenomena has been studied extensively as both a means to predict and a means to prevent. Vortex shedding has been shown to be a direct cause to these pressure fluctuations [18, 19]. The developers of SPP [1] have worked extensively to include modules in their program that would allow for motor stability analysis. Coats and Dunn [20] linked SSP (an offshoot analytical approach utilizing the Taylor-Culick profile) to SPP to automate stability predictions. This work was further developed by French to investigate tangential mode instabilities in grains with even and odd numbers of slots [21].

1.5 Propellant Solid Mechanics

The study of propellant solid mechanics involves examining the effect of stress and strain on the propellant block from both resting during pre-launch and accelerating during launch. Slump is the term used to describe the displacement of the interior propellant as opposed to the propellant bonded to the case wall from resting or accelerating during launch. This effect has been known to alter motor performance. Fiedler et al. investigated this effect directly via simulation by coupling a solid mechanics model to a solid motor grain regression model [22]. Renganathan, Rao and Jana investigated effects of slump on segmented motors under storage conditions using FEA and compared this analysis with experimental results [23]. Lajczok [24] performed an analysis on propellant properties during ignition. This study determined an effective propellant modulus to accurately predict propellant deformation under the immense pressure gradient prevalant shortly after ignition.

1.6 Automated Grain Design and Performance Control

Automated grain cross-section design and performance control has recently become a hot topic in the SRM field with the advances in computing technology. The "Achilles heel" of solid rocket motors has always been a lack of variability in the thrust-time profile. One primary reason for choosing a liquid rocket over a solid rocket motor is the added capability of a liquid to throttle. Nunn and Chafin developed a method for throttling a solid by way of a variable sized nozzle throat and gas injector in the combustion zone forward of the nozzle but aft of the propellant grain [25]. Of course, the issue of throttling a solid motor only becomes relevant when offdesign performance is required, and has been achieved in the past using pintles. If a motor is designed properly (i.e. designed for a specific mission), throttling can be avoided. Anderson first showed how this could be possible by linking a fully encompassing missile analysis tool (i.e. complete with aerodynamics, propulsion, mass properties, guidance and control, and 6-DOF) to a genetic algorithm [26]. This tool proved effective for designing entire missile interceptor systems with as much detail as was desired quickly and effectively. The idea for automated design using a genetic algorithm inspired Jenkins [27] and Albarado [28] to develop optimization programs to design uniform cross section solid motors for specific missions. Their work entailed development of a particle swarm/pattern search hybrid optimization scheme similar to that of Woltosz [29].

Chapter 2

Existing Internal Ballistics Models

Many of the common techniques for modeling internal ballistics (grain regression) are analytical in nature. As Barrere demonstrates[30], the grain cross section for a uniform solid rocket motor can be decomposed into a handful of parameters that geometrically define the motor. Using this predefined set of parameters, analytical equations can be developed to describe the internal ballistics of the motor. SPP expanded on this idea to produce the first method for analyzing motors that are nonuniform along the grain length (prominently the Shuttle boosters). This method was modified heavily in order to accurately model geometrically complex motors, and the true accuracy of the program was never fully realized. Only as recently as 2005 was a new method developed for performing the grain regression problem [31]. Willcox found that a concept known as a Signed Distance Function (SDF) could locate a surface within a domain, and determine where a surface will be at some prescribed burn distance later. Cavallini showed how an extension of the SDF called the Level Set Method (LSM) could be utilized to include erosive burning and internal aerodynamics with their programs Solid Propellant rocket motor Internal Ballistics (SPINBALL) and the Grain REGression model (GREG) [32, 33, 34]. To date, these two programs collectively represent one of the most accurate rocket motor analysis tools in literature and offers the computationally most efficient method for solving the grain regression problem. The present work aims to improve on their development of the grain regression problem by improving the propagation routine developed in GREG using a higher order integration scheme known as the Discontinous Galerkin Method.

2.1 Analytical Methods

Classically, solid rocket motor regression is performed via analytical techniques exploiting the geometric method for definition. The limiting factor for almost all solid rocket motor grain simulations exploiting this idea is their dependence on a specific method for defining the grain cross-section. There is no standard for parametrically defining rocket motors (such as wing span and chord length for aircraft wings), and thus these analytical techniques, while accurate, fail to be robust. Nevertheless, these analytical approaches will provide solutions for comparison to the LSM approach, and warrant further discussion. Barrere [30] is credited with developing the first systematic analytical techniques for star and wagon wheel configurations. The star grain, as defined by Barrere, is shown in the schematic below in Figure 2.1. A complete description of the Barrere star grain analytical method is given in Appendix A.

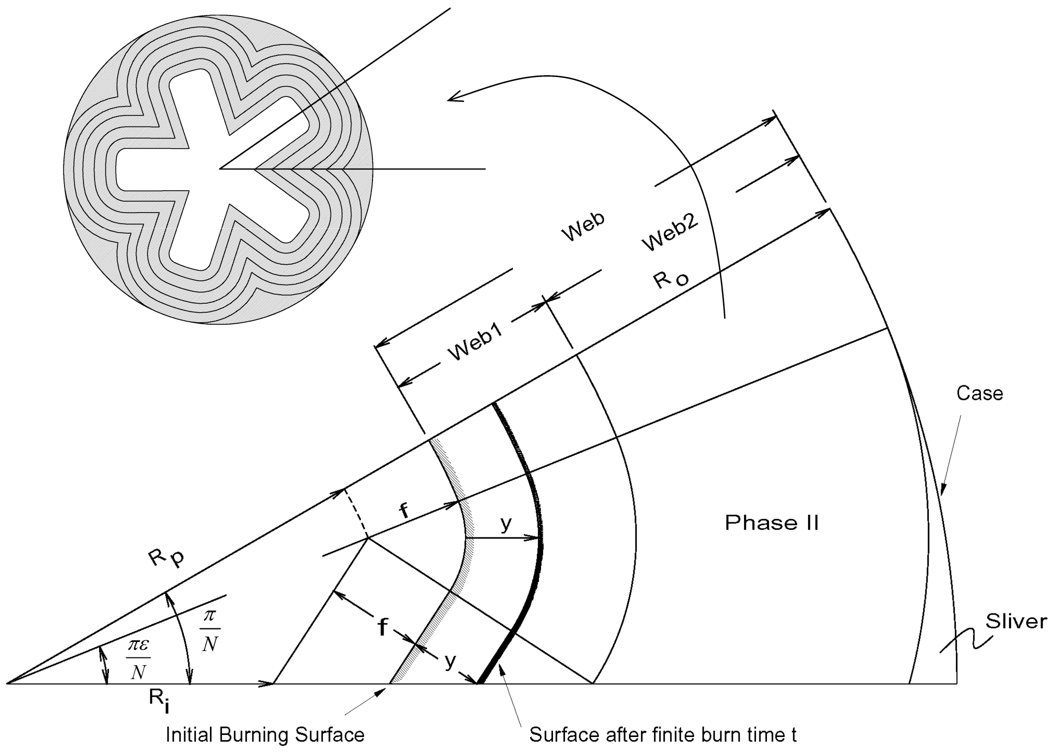


Figure 2.1: Star Grain Definition [27]

Using the same parameters, Hartfield et al. [35] review the analytical solution to the long spoke wagon wheel originally found in [30] and describe the analytical solution for the short spoke wagon wheel. Reference [36] gives an analytical description for how to solve grain regression in a slotted grain solid rocket motor. Sforzini [37, 38] and Ricciardi [39] developed expressions for performing grain regression in star grains and truncated star grains (slotted tube grains).

Coats and Dunn [1] expanded on the analytical methods approach and developed analytical equations for simple 3-D shapes that would be found in a typical solid rocket motor. These shapes are cone, cylinder, prism, sphere, and torus. From these shapes, designs such as finocyls, tapered stars, and forward stars can be designed and analyzed using SPP. This code has been expanded upon further to include the following advancements:

1. Motor stability and combustion stability predictions [20, 40, 41]
2. Advanced motor definition options including segmented motors, dual propellant motors, and case insulation [42]
3. Surface mesh generation [42]
4. Ignition transient calculations [42]
5. 3-D finite-volume gas-particle solver for internal aerodynamics [43]

2.2 Face-Offsetting Method

Another popular method for surface propagation is called the Face-Offsetting Method (FOM). This method, formalized by Jiao [44] is of Lagrangian formulation, meaning that the surface or curve of interest is explicitly propagated in a discrete manner. In order to perform this method, each panel or facet is propagated outward. Then the vertices connecting the faces are reconstructed using the method described by Jiao [44] with appropriate logic to handle the intersections of faces. To improve mesh quality, the vertices are then

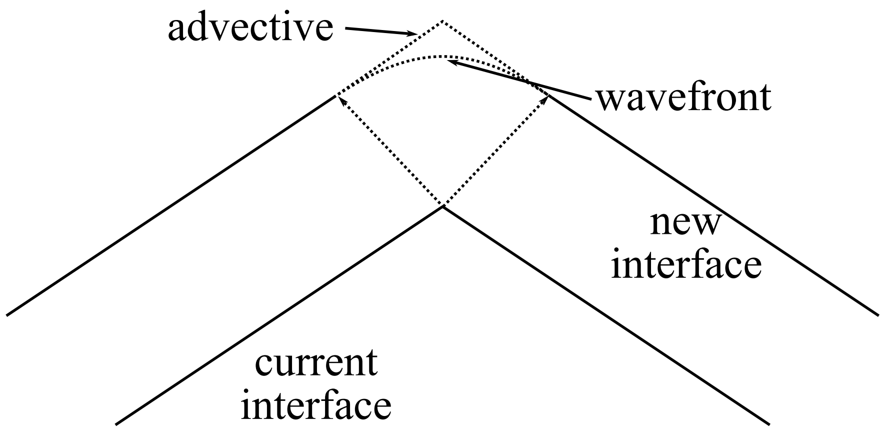
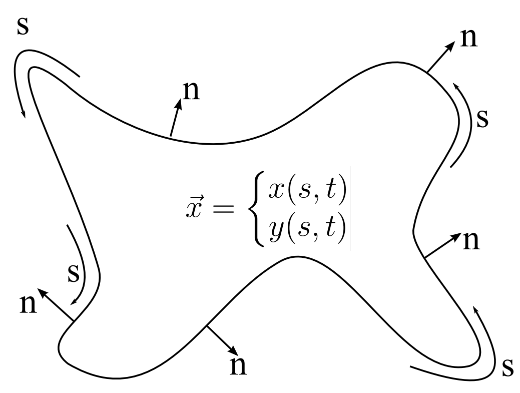


Figure 2.2: Face-Offsetting Method: Advective Reconstruction vs. Wavefront Reconstruction

redistributed along the new surface. The vertex reconstruction can be completed in one of two manners: advective and wavefront motion. An illustration of each is shown below in Figure 2.2. Advective reconstruction would be more suitable for concave surfaces while wavefront reconstruction would be more suitable for convex surfaces. FOM was developed further by researchers at the University of Illinois at Urbana-Champaign into a program called *Rocstar* [45, 46, 47, 48, 49]. *Rocstar* was developed as an analysis tool to perform everything from basic SRM performance analysis to off-design analysis that includes effects such as muliphase core flow, propellant solid mechanics (slumping) and erosive burning. The *Rocstar* software has been used to model propellant solid mechanics problems in the Space Shuttle boosters [46], turbulent flow effects in the RSRM [48], and propellant slumping in the Titan IV [49].

Chapter 3
Eulerian Approaches to Surface Tracking

In order to ensure that an SRM code can handle arbitrary motor geometries, i.e. the analysis method has no dependence on geometric definition, a discrete surface tracking method must be employed. There are two schools of thought to surface tracking that can be employed for the grain regression problem: Lagrangian and Eulerian. A Lagrangian formulation, like the face-offsetting method used in *Rocstar*, is fraught with numerical and stability issues. Consider the closed curve given in Figure 3.1a.



(a) Closed Curve

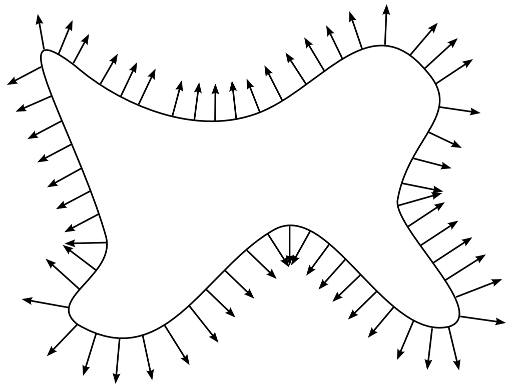

(b) Marker Particles

Figure 3.1: A Closed Propagating Curve

Using marker particles along the boundary (see Figure 3.1b), the Lagrangian formulation of surface propagation would dictate that each particle move normal to the curvature of the surface. For discretized treatments, this creates numerical issues in areas where particles begin to cluster together or pass by each other. This formulation also creates accuracy issues in areas where particles begin to spread out. Figure 3.2 demonstrates these numerical issues in convex and concave surfaces, showing interpretation of the particles versus what should physically occur. In order to alleviate these problems, subroutines must be developed

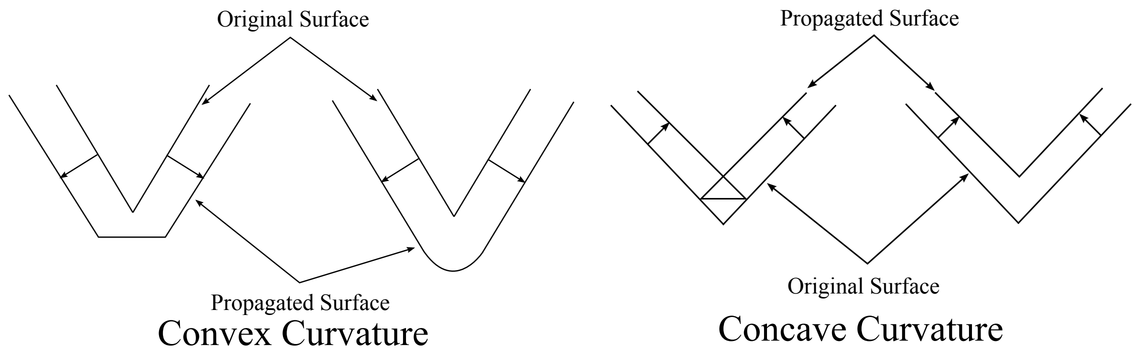
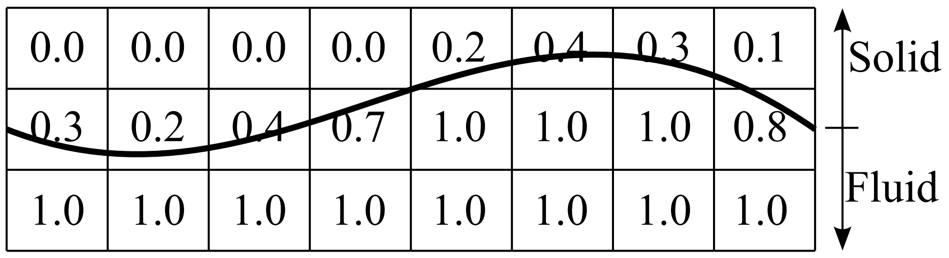


Figure 3.2: Convex and Concave curvature with Lagrangian propagation (left) and actual propagation (right)

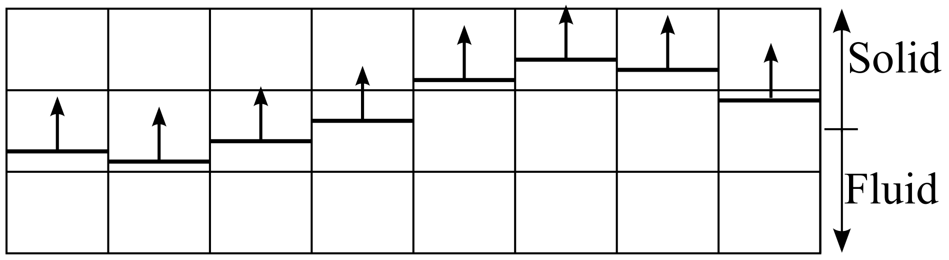
to determine which surfaces are convex and which are concave. Appropriate logic to round out convex curvature and create sharp points for concave curvature must also be employed. In cases where the front velocity is a function of the curvature, concave surfaces will form sharp edges, leading to singularities in the front velocity. When this occurs, a shock in the solution forms and smoothing must be applied in order to continue with the solution. The methodology used in [44, 45] has accounted for the issues associated with the marker-point method; however, more elegant solutions to this problem are available.

Another popular surface tracking method is "volume of fluid" (VOF) method which is an Eulerian formulation and uses a stationary mesh and tracks the motion of the interior region. Each cell is assigned a value based on the percentage of the element volume that is occupied by the surface or fluid of interest. This value is 0 in elements where the fluid is not present, between 0 and 1 where the fluid or surface cuts through a cell, and 1 where the entire cell is occupied (see Figure 3.3a). VOF handles the surface implicitly; however, because the only information stored for each cell is the percentage volume occupied, information regarding the curvature of the surface edge is not easily ascertainable. Figure 3.3b shows the interpretation by the VOF method of the curve given in Figure 3.3a. From the Figure, it is readily apparent how the initial curvature is lost in the VOF interpretation.

The Level Set Method (LSM) is an implicity method much like the VOF method to studying the evolution of boundary between two regions. The method was formalized by



(a) Numerical Representation



(b) Advection Front

Figure 3.3: Volume of Fluid Representation

Osher and Sethian in 1979 [50, 51]. The primary difference between LSM and VOF is the information stored within the domain. In VOF, the percentage of the cell occupied by a fluid or region of interest is stored. For a level set, the minimum distance to a closed surface within some domain is stored along with a sign indicating interior or exterior to the surface. By storing the distance do the surface rather than just the volume of the cell occupied by a region, more information regarding the evolution of the initial surface can be inferred. The level set method has been proven useful in numerous areas of engineering including graphics, computational fluid dynamics, material sciences, and other fields. Rather than explicitly tracking surface propagation, the signed distance function, simply referred to as the level set, is propagated which in turn implicitly propagates the surface. Propagation of the level set is handled using a transport equation of the form

$$\frac{\partial u}{\partial t} + \lambda \frac{\partial u}{\partial x} = 0 \tag{3.1}$$

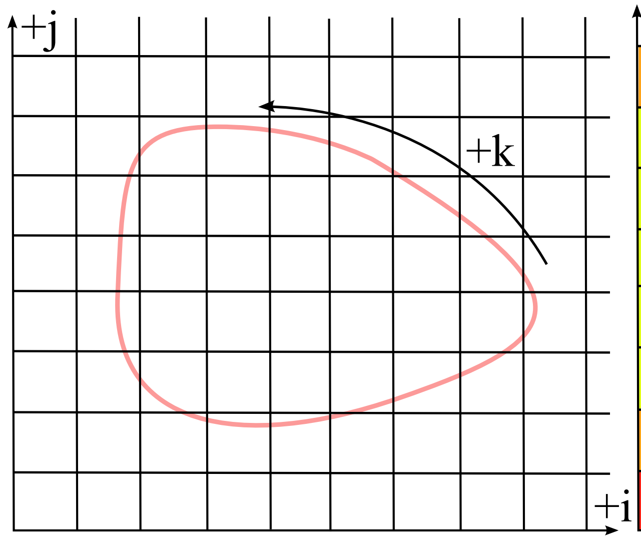
where λ is the characteristic wave speed for the posed problem. This approach offers a convenient and inexpensive alternative to the Lagrangian formulations which can become cumbersome with the logic and subroutines necessary to handle convex and concave curvature. The level set equation has some properties that can be taken advantage of to develop a stable scheme for integration. For one, the magnitude of the gradient of the signed distance function is equal to one everywhere in a general sense. In 2-D, if a uniform velocity field is assumed, the level set will simply convect in the direction of the z-axis with no shape changes. However, if a non-uniform velocity field is desired, there are relatively straightforward and inexpensive methods for implementing them (see [52, 53]).

The LSM approach propagates a function representing the distance to a curve in discretized space as opposed to propagating a discretized curve. It is this characteristic which gives the level set method an advantage over Lagrangian approaches. LSM handles convex and concave surface implicitly, eliminating the need for logic and subroutines to handle each case. The level set of a curve describes the surface as a function of distance from the original curve. At each time step, the zeroeth level set becomes the curve itself. To initialize the level set for a curve, begin with a discretized curve and overlay a discretized mesh on top of it. The level set is then initialized by finding the signed minimum distance (SDF) from each point in discretized space to the curve. Figure 3.4 represents a discretized curve and its complementary level set function at initialization.

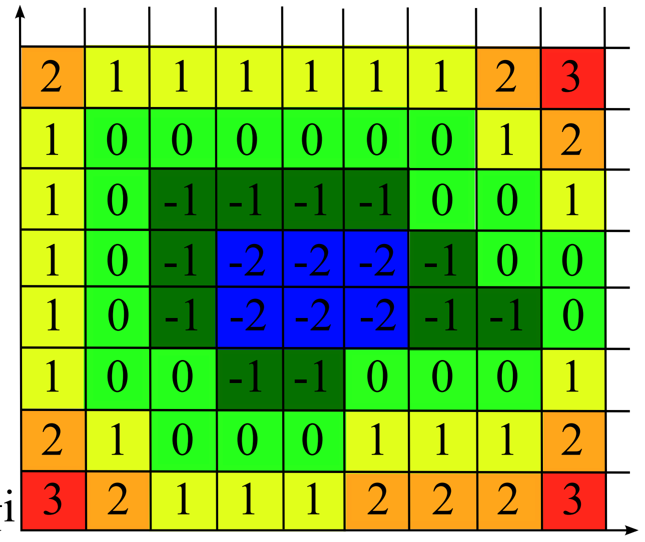
Determining the minimum distance from each grid point to the initial surface is simple enough, using the distance formula:

$$DF_{i,j} = \min\left[\sqrt{\left(x_{i,j} - \bar{x}_k\right)^2 + \left(y_{i,j} - \bar{y}_k\right)^2}\right] \quad \forall\, \bar{x}, \bar{y} \tag{3.2}$$

Of course, using Equation 3.2 will always yield a positive value, but for grid points located inside of the surface, the desired outcome is a negative value. In order to make this determination, the winding number of the point with respect to the curve is calculated [54]. The winding number is calculated by summing the angles subtended by each of edges as seen by the point of interest. The summation is always either 0 or $n\pi$. If the sum is $n\pi$ then



(a) Curve in Discretized Space



(b) Signed Minimum Distance

Figure 3.4: A Discretized Curve and its Complementary Level Set

the point lies within the curve, and if the sum is 0, the point lies outside the boundaries of the surface. A graphic representation of this is shown in Figure 3.5, where A has a winding number of 1, and B has a winding number of 0. This method for determining the signed minimum distance function is performed the same regardless of whether the problem is 2-D or 3-D in nature. The signed minimum distance is what allows LSM to handle convex and concave curves implicitly, in that the gradient of the level set will determine whether surfaces develop into sharp points (concave) or round valleys (convex).

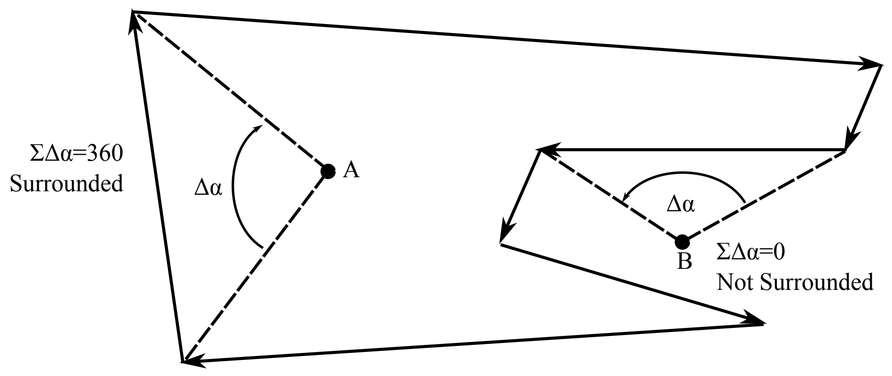


Figure 3.5: Winding Number Diagram

According to Osher and Sethian [50, 51], the Level Set equation can be written as

$$\begin{cases} \phi_t + r_b\left(\vec{x}, t\right)\left|\vec{\nabla}\phi\right| = 0 \\ \phi\left(\vec{x}, t = 0\right) = 0, \quad \forall \vec{x} \in \Gamma \end{cases} \tag{3.3}$$

Differentiating Equation 3.3 with respect to time yields

$$\frac{\partial \phi}{\partial t} + \sum \phi_{xi} \frac{\partial x_i}{\partial t} = 0 \tag{3.4}$$

$$\frac{\partial \phi}{\partial t} + \vec{V} \cdot \vec{\nabla}\phi = 0 \tag{3.5}$$

Here it is important to observe that the front propagation is related only to the normal component of the velocity field because the gradient, $\vec{\nabla}\phi$, at each point in $\vec{x}$ is defined normal to the level set. In fact, as Cavallini [32] shows, when writing the velocity vector in terms of its normal and tangential components, and substituting into Equation 3.5, Equation 3.6 can be arrived at, which is equivalent to the level set equation as described by Osher and Sethian above.

$$\frac{\partial \phi}{\partial t} + V_n\left|\vec{\nabla}\phi\right| = 0 \tag{3.6}$$

The level set equation, Equation 3.6 is a partial differential equation that falls into a class of equations known as Hamilton-Jacobi equations, where

$$\phi_t + V\left|\vec{\nabla}\phi\right| = 0 \quad \rightarrow \quad \phi_t + \mathcal{H}(\phi_x, \phi_y, \phi_z) = 0 \tag{3.7}$$

where $\mathcal{H}$ is the Hamiltonian defined for Equation 3.6 as

$$\mathcal{H}(x, y, z, \phi_x, \phi_y, \phi_z, t) = V\left|\vec{\nabla}\phi\right| \tag{3.8}$$

Chapter 4

First Order Grain Regression Program

4.1 Development and Implementation

The first order two dimensional burn regression program was developed as a test bed for the level set method. This program was developed using a similar setup to that found in references [32, 33]. From a mathematical sense, the approach taken to solid rocket motor simulation is the same approach taken in the studies of fluid dynamics except that the equations of motion are different.

Equation 3.3 must to be rewritten into a scheme that can be easily integrated. For the first order code, it was written of the form

$$\begin{aligned}\phi_{j,k}^{n+1} = \phi_{j,k}^{n} - \Delta t^n r_b|_{j,k}^{n} \Big[&\max\left(\max\left(D^{-x}\phi_{j,k}^{n},0\right)^2, \min\left(D^{+x}\phi_{j,k}^{n},0\right)^2\right) + \\ &\max\left(\max\left(D^{-y}\phi_{j,k}^{n},0\right)^2, \min\left(D^{+y}\phi_{j,k}^{n},0\right)^2\right)\Big]^{\frac{1}{2}}\end{aligned} \tag{4.1}$$

where

$$\begin{aligned} D^{-x}\phi_{j,k}^{n} &= \frac{\phi_{j,k}^{n} - \phi_{j-1,k}^{n}}{\Delta x} \quad ; \quad D^{+x}\phi_{j,k}^{n} = \frac{\phi_{j+1,k}^{n} - \phi_{j,k}^{n}}{\Delta x} \\ D^{-y}\phi_{j,k}^{n} &= \frac{\phi_{j,k}^{n} - \phi_{j,k-1}^{n}}{\Delta y} \quad ; \quad D^{+y}\phi_{j,k}^{n} = \frac{\phi_{j,k+1}^{n} - \phi_{j,k}^{n}}{\Delta y} \end{aligned}$$

This scheme is first order in space and time. To enhance the order for the time integration, back and error compensation was utilized as described by Dupont and Liu[55]. To perform back and forth error compensation, the initial level set is propagated forward in time one time step through normal operation. The result of this initial propagation is then propagated backwards in time, yielding a new level set at the same time step as the initial level set. The difference in the original level set and the new level set is then used to correct the original level set. The next time step is finally calculated using this corrected level set. Mathematically this appears as:

$$\hat{\phi}^{n+1} = f(\phi^n, +r_b^n) \tag{4.2}$$

$$\hat{\phi}^n = f\left(\hat{\phi}^{n+1}, -r_b^n\right) \tag{4.3}$$

$$\bar{\phi}^n = \phi^n + \frac{\phi^n - \hat{\phi}^n}{2} \tag{4.4}$$

$$\phi^{n+1} = f\left(\bar{\phi}, +r_b^n\right) \tag{4.5}$$

While this method works for propagating the propellant surface, what is of importance to rocket motor simulation is the surface area and chamber volume evaluation at each time step, as this determines chamber pressure and propellant weight. The volume and area can be calculated simply as:

$$V = \int_\Omega h\left(\phi\left(x\right)\right) dx \tag{4.6}$$

$$A = \int_\Omega \delta\left(\phi\left(x\right)\right) \left|\vec{\nabla}\phi\right| dx \tag{4.7}$$

where h is the Heaviside step function

$$h(\phi) = \begin{cases} 0 & \phi \leq 0 \\ 1 & \phi > 0 \end{cases} \tag{4.8}$$

and $\delta(\phi)$ is the Dirac delta

$$\delta(\phi) = \frac{dh(\phi)}{d\phi} \tag{4.9}$$

Because Equation 4.8 is a step function, Equation 4.9 cannot be evaluated easily in a numerical scheme. Furthermore, these evaluations, while sound mathematically, do not translate well to numerical schemes as theorized. To handle this problem, some smearing was applied to h and consequently δ as follows.

$$h(\phi) = \begin{cases} 0 & \phi < -\epsilon \\ \frac{1}{2} + \frac{\phi}{2\epsilon} + \frac{1}{2\pi}\sin\left(\frac{\pi\phi}{\epsilon}\right) & |\phi| \leq \epsilon \\ 1 & \phi > \epsilon \end{cases} \tag{4.10}$$

$$\delta(\phi) = \begin{cases} \frac{1}{2\epsilon} + \frac{1}{2\epsilon}\cos\left(\frac{\pi\phi}{\epsilon}\right) & |\phi| \leq \epsilon \\ 0 & \phi > \epsilon \end{cases} \tag{4.11}$$

In Equations 4.10 and 4.11, ϵ is a tuning parameter defined by some grid spacing characteristic length. Figure 4.1 demonstrates the smearing function for the Heaviside function and Dirac delta. Contributions for both surface area and chamber volume are made only by those cells contained within the boundary of the case.

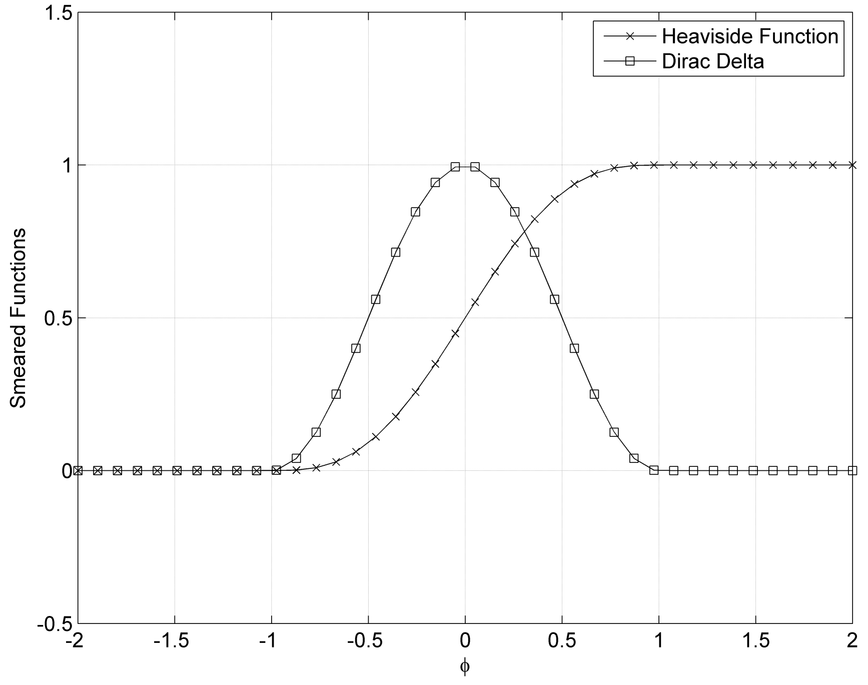


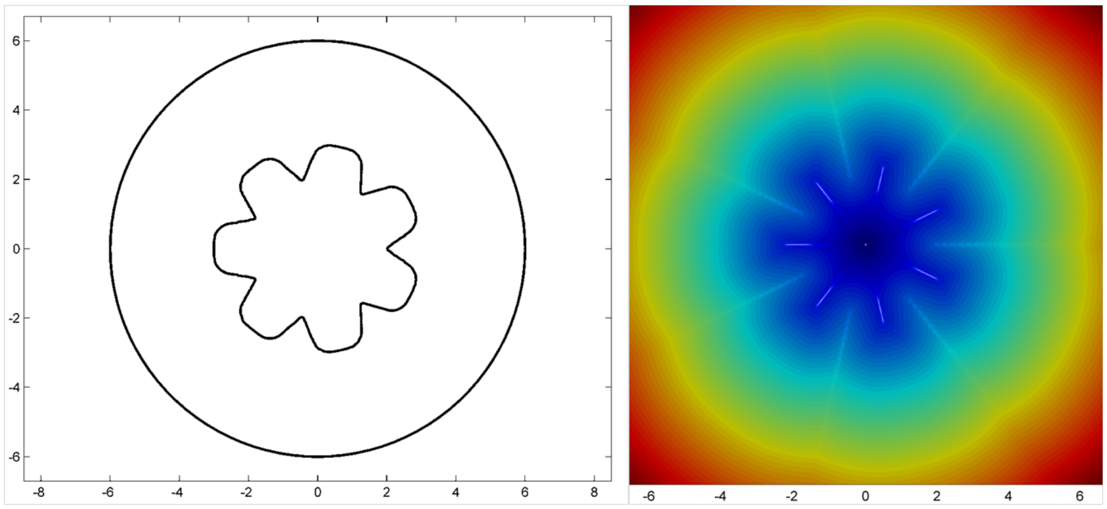
Figure 4.1: Smearing functions for arbitrary ϕ near the zeroeth level set

4.2 Results

The 2-D first-order accurate program (henceforth referred to as *AUBurnSim2D*) was developed to demonstrate the effectiveness of LSM at accurately capturing the burn regression physics. The following results serve as a justification for moving forward with the development of a higher order scheme.

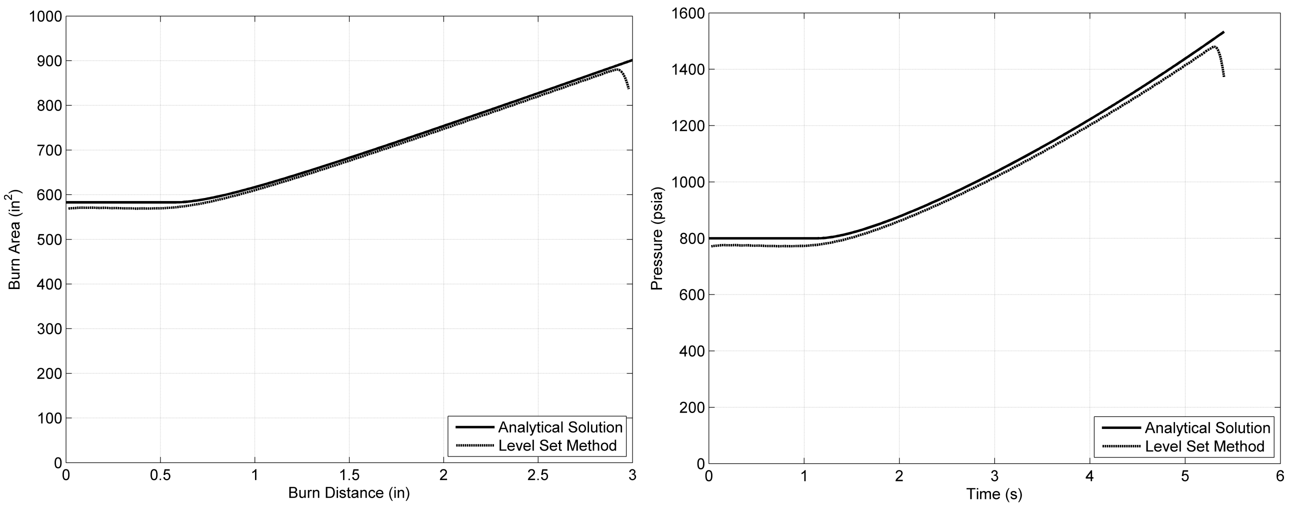
Neutral Burning Star

According to the Barrere model [30], it is possible to analytically design a motor which exhibits a neutral burning profile for phase I burning. Using this approach, a neutral burning motor was designed and the analytical result was comparied with the result from *AUBurnsim2D* for validation. Motor performance was simplified to lumped parameter analysis and local burn rates, erosive burning, and chamber aerodynamics were ignored in order to test geometric competence of the level set approach. The motor tested was a 7-pointed star, shown in Figure 4.2. Figure 4.2b was generated using a mesh size of 200 × 200.



(a) Motor Surface and Case (b) Signed Minimum Distance

Figure 4.2: A Neutral Burning Seven Pointed Star



(a) Burn Area vs Burn Distance (b) Pressure vs Time

Figure 4.3: Neutral Burning Star Results

The results of this simulation are shown in Figure 4.3. From the Figure, it is apparent that LSM performs well at predicting burn area (and therefore pressure) over time. However, there are a few key problems to note about Figure 4.3. While LSM was able to accurately trend the motor (neutral burn at the beginning followed by a progressive burn; burn time is correct), there was an underprediction in total pressure throughout burn of approximately 15 psia. Secondly, near burnout a dip in pressure is observed when using LSM, which disagrees with the analytical solution. Both of these effects can be attributed to the smearing required when calculating area and chamber volume. Shown in Figure 4.4 is a contour of the smeared Heaviside function for the 7-pointed star at ignition. From Equation 4.10, the function of the Heaviside equation is to step from a value of 0 to a value of 1 at the boundary, but in a smooth manner. This causes an apparent bandwidth at the boundary of approximately 2ϵ (dark region dividing the lighter sections in Figure 4.4).

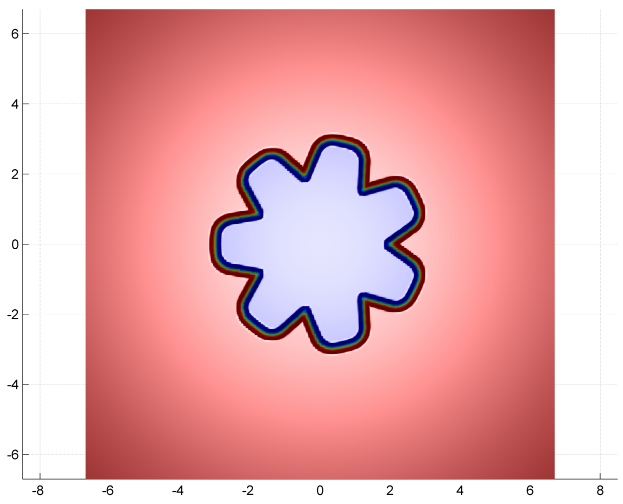


Figure 4.4: Smeared Heaviside function for 7-pointed star at ignition

A closer look at this bandwidth when interacting with the motor case reveals the source of the issue with the dip in pressure near burnout. Figure 4.5 is a schematical representation of what is occurring at the case wall. The actual motor surface is still completely within the boundary, and as such should be completely contributing to the surface area. However, because the surface must be represented numerically as this bandwidth, we see that part of the numerical surface is outside the case boundary. As such, areas of the numerical that should be contributing to the burn area have been neglected. The obvious fix to this issue is to slim down the bandwidth as much as possible with a finer mesh, which will in turn drive up memory useage and wall clock time.

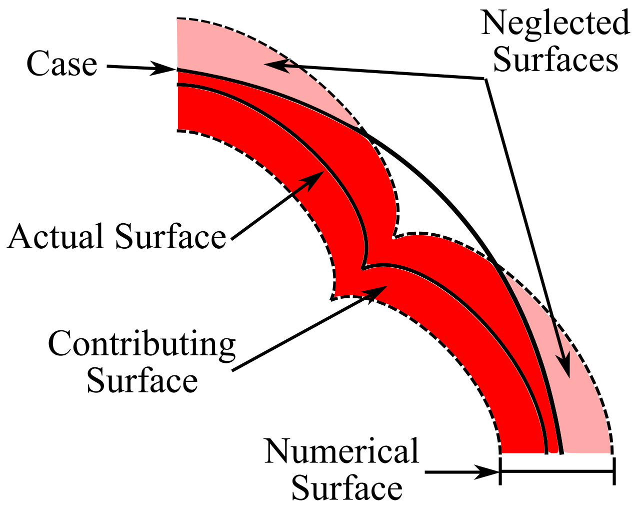
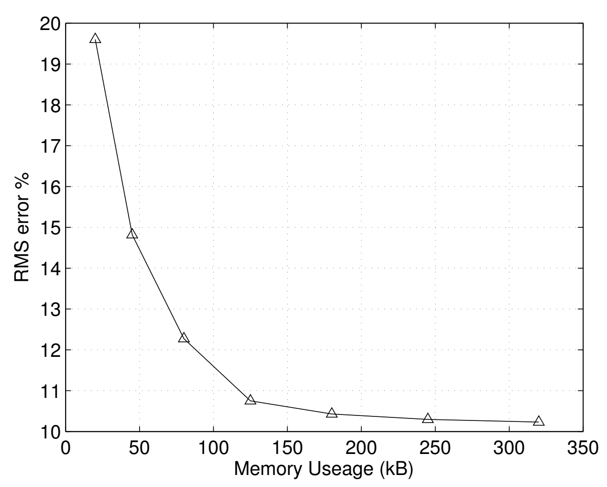
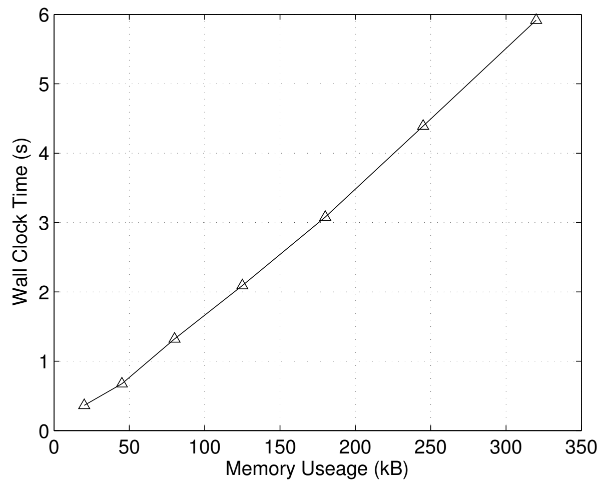


Figure 4.5: Motor Surface near the Case Boundary

A grid sensitivity study was performed using this neutral 7-pointed star to show how a finer mesh will in fact reduce error but will drive up memory usage and wall clock time. The mesh sizes studied varied between 50×50 and 200×200. The results of this grid study are shown below in Figure 4.6. The grid study showed that the RMS error decreased with finer and finer meshing. In fact, it should expected that a finer mesh will in fact produce a more accurate result. But also as was expected, the total runtime increased with a finer mesh size. From Figure 4.6, the lowest RMS error achievable is quickly realized by increasing mesh size, but the lowest RMS achievable was still around only 10%. It can be concluded from this study that attempts to resolve the grid further are not warranted as the gain in RMS error will be unattractively low with increasing memory useage and wall clock time.



(a) Memory Useage vs RMS Error



(b) Memory Useage vs Wall Clock

Figure 4.6: Grid Sensitivity Results

Offcenter Right Circular Perforated Grain

This second case using *AUBurnSim2D* is an offcenter right CP grain to demonstrate the robustness of LSM at handling more complex grains. The initial surface is simple enough, but the burning of the surface poses a problem for analytical methods when a small portion of the surface burns into the motor case before the rest of the surface. This case represents a common problem when manufacturing these motors: perfectly centering the bore in the grain. The case presented here represents an exaggerated version of this problem. Shown in Figure 4.7 is the time evolution of the offcenter CP grain. From the Figure, the surface initially burns into the motor case around at time of $t = 5.0$ seconds. A CP grain should have a progressive burn profile until the surface burns into motor case, at which point the burn becomes regressive. From Figure 4.8, the result is exactly that. At time $t = 5.0$ seconds, the pressure changes from progressive to regressive.

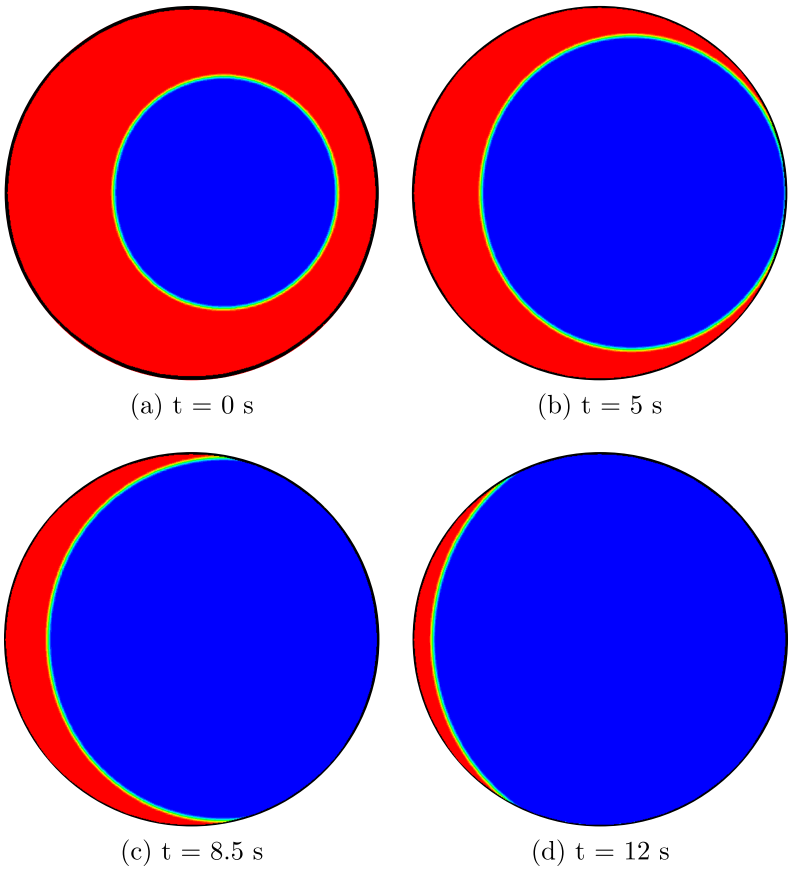

(a) t = 0 s (b) t = 5 s (c) t = 8.5 s (d) t = 12 s

Figure 4.7: Offcenter Circular Perforated Grain Regression

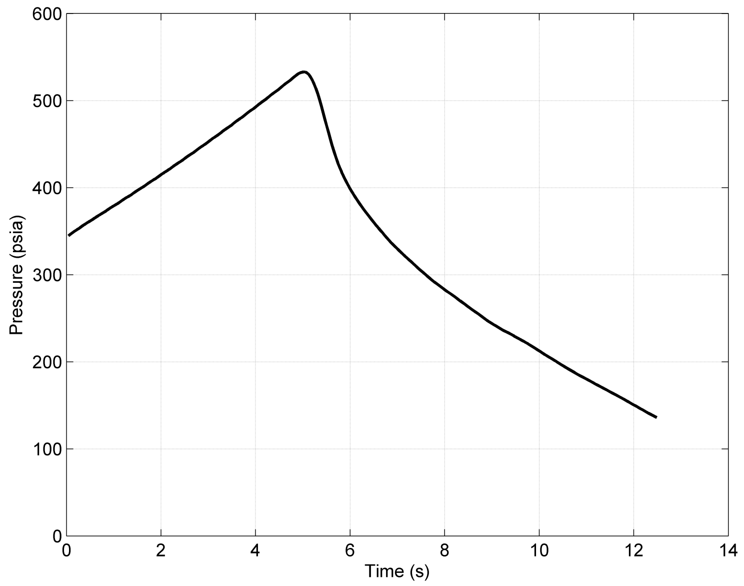


Figure 4.8: Pressure Time Profile for an Off-Center Cylindrical Perforated Grain

Chapter 5

Higher Order Grain Regression Program

5.1 Development and Implementation

The following sections detail the methods used to develop the higher order grain regression program. A discussion of the discontinuous Galerkin method is provided along with a detailed discussion of how to implement DG with Hamilton-Jacobi equation, as will be needed for the level set method. The next section discusses some of the modifications that were needed in order to stabilize the differential equation for integration. The final section of development and implementation discusses the method in which the motor case was incorporated into the level set area calculation.

5.1.1 Discontinuous Galerkin Method

The discontinuous Galerkin method (DGM) combines flavors of finite difference, finite element, and finite volume methods. Finite difference schemes approximate the solution locally with 1-D polynomials and satisfy the differential equation in a pointwise fashion. This makes finite differencing schemes simple to implement and still allow for high-order implementation. However, implementing high-order schemes cannot be implemented with geometric variance. Finite volume schemes also approximate the solution locally, using a cell average as opposed to polynomials. Finite volume schemes are robust and fast just as the finite differencing schemes due to local solution approximation. But, finite volume schemes cannot be modified to be higher order in general. Lastly, finite element methods define the solution in a non-local manner, satisfying the equation across the entire domain which make them suitable for high-order. However, they are implicit in time. The desired formulation would be to have local high-order approximation that can work with complex geometries.

Hesthaven [56] provides much of the technical background required to get started with DG. A comprehensive synopsis on development of DG can be found in Appendix B. Much of the discussion found in Appendix B emenates from a variety of sources, primarily [56] but also [57, 58, 59, 60]. Appendix C gives an example comparison of DG to a conventional finite volume method for solving the classical isentropic vortex problem. The basic transport equation

$$\frac{\partial u}{\partial t} + \frac{\partial f(u)}{\partial x} = 0 \tag{5.1}$$

in a DG sense becomes

$$\int_{D^k} b_i^k \frac{\partial u_h^k}{\partial t} dx - \int_{D^k} \frac{\partial b_i^k}{\partial k} f_h^k dx + \oint_{\partial D^k} b_i^k f^* \left(u_h^{k,-}, u_h^{k,+}; n_x^{k,-} \right) ds = 0 \tag{5.2}$$

In matrix vector form, this can be rewritten as

$$\mathcal{J}^k M \frac{d\widetilde{u}^k}{dt} - S^T \widetilde{f}^k + L_a \widetilde{f_a^*}^k + L_b \widetilde{f_b^*}^k = 0 \tag{5.3}$$

Since we want to determine the time derivative term for each cell, the final form of the algebraic equation becomes

$$\frac{d\widetilde{u}^k}{dt} = \mathcal{J}^k M S^T \widetilde{f}^k - \mathcal{J}^k M \left(L_a \widetilde{f_a^*}^k + L_b \widetilde{f_b^*}^k \right) \tag{5.4}$$

Equation 5.4 is useful when the exact flux and numerical flux is straightforward to calculate, such as the case with the Euler equations.

For states such as the level set equation, determining the gradient is more difficult and relies on the auxiliary variable approach like the first order LSM. However, from Equation 5.4, the tools needed to calculate the derivative of the level set equation are already available. Looking at Equation 5.5

$$\int b \frac{\partial u}{\partial t} dx + \int b \frac{\partial F}{\partial x} dx = 0 \tag{5.5}$$

In order to use with the level set, DG must be successfully implemented using a Hamiltonian. Cheng and Shu [61] discuss how to incorporate the basic Hamiltonian into DG schemes. Yan and Osher [62] describe how this Hamiltonian term can be replaced with a modified Hamiltonian term to incorporate artificial viscosity.

$$\int b\frac{du}{dt}dx + \int b\hat{H}(u_x)dx = 0 \tag{5.6}$$

where

$$\hat{H}(p_1,p_2,q_1,q_2) = H\left(\frac{p_1+p_2}{2},\frac{q_1+q_2}{2}\right) - \frac{1}{2}\alpha(p_1-p_2) - \frac{1}{2}\alpha(q_1-q_2) \tag{5.7}$$

This is just one of many ways to represent the modified Hamiltonian term. The Gudonov scheme in Equation 4.1 is also of Hamilton-Jacobi form where $\hat{H}$ takes on the form of

$$\begin{aligned}\hat{H}(p_1,p_2,q_1,q_2) = -\Delta t^n r_b|_{j,k}^n \Big[& \max\left(\max\left(p_1,0\right)^2, \min\left(p_2,0\right)^2\right) + \\ & \max\left(\max\left(q_1,0\right)^2, \min\left(q_2,0\right)^2\right)\Big]^{\frac{1}{2}}\end{aligned} \tag{5.8}$$

where p represents the x-derivative of ϕ using data from either the left or the right (1 and 2) and q represents the y-derivative of ϕ. Development of the derivatives in the x- and y-directions can be developed using the same operators used in Equation 5.4. In integral form, the derivatives in the x-direction would take the form

$$\left[\int_{D^k} b_i^k \frac{\partial u_h^k}{\partial x}\right]_{left} = -\int_{D^k} \frac{\partial b_i^k}{\partial x} u_h^k dx + \oint b_i^k f^*\left(u_h^{k,-}, u_h^{k+1,-}; n_x^{k,-}\right) ds \tag{5.9}$$

$$\left[\int_{D^k} b_i^k \frac{\partial u_h^k}{\partial x}\right]_{right} = -\int_{D^k} \frac{\partial b_i^k}{\partial x} u_h^k dx + \oint b_i^k f^*\left(u_h^{k+1,+}, u_h^{k,+}; n_x^{k,-}\right) ds \tag{5.10}$$

and in matrix vector form are expressed as

$$p_1 = -\mathcal{J}^k S^T u_i + \mathcal{J}^k L_{x_{right}} u_{i+1_{left}} - \mathcal{J}^k L_{x_{left}} u_{i_{left}} \tag{5.11}$$

$$p_2 = -\mathcal{J}^k S^T u_i + \mathcal{J}^k L_{x_{right}} u_{i_{right}} - \mathcal{J}^k L_{x_{left}} u_{i-1_{right}} \tag{5.12}$$

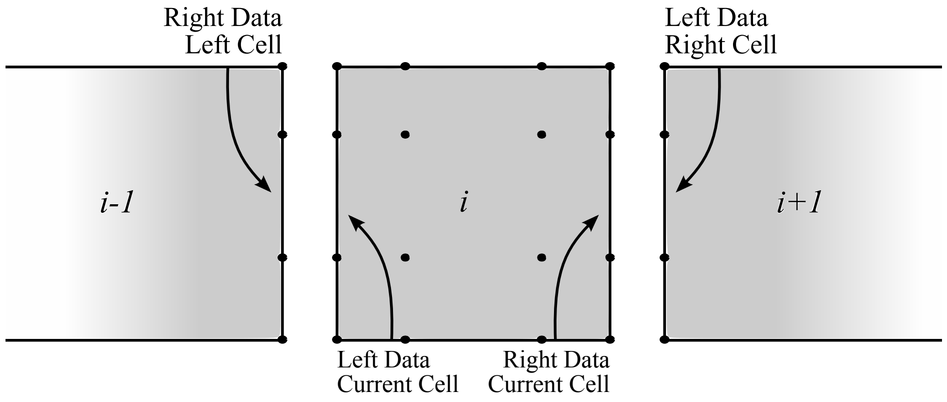
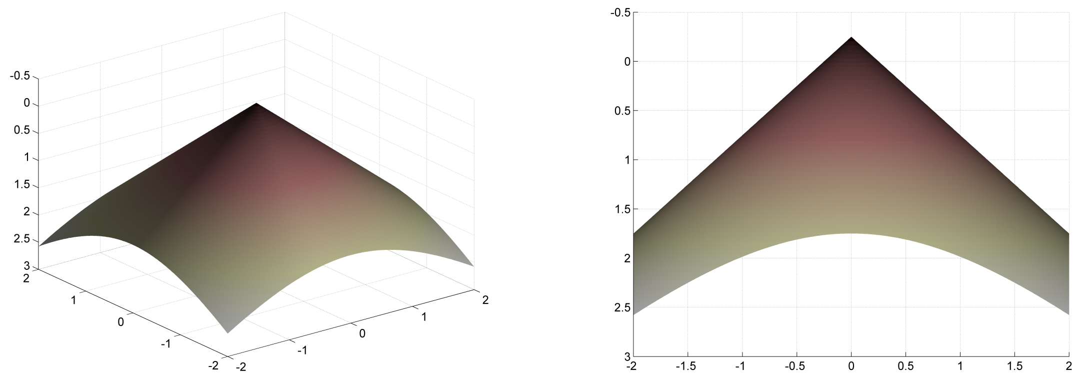


Figure 5.1: Stencil Used to Develop DG Derivatives

Using Figure 5.1 as a reference, Equation 5.11 can be interpreted as the derivative is equivalent to taking the derivative of the basis function for the cell *plus* lifted data from the left edge of the cell to the right *minus* lifted data from the left edge of the current cell. Similarly, Equation 5.12 can be interpreted as being equivalent to taking the derivative of the basis function for the cell *plus* lifted data from the right edge of the current cell *minus* lifted data from the right edge of the cell to the left. Derivatives in the y-direction are performed in exactly the same manner by replacing the stiffness matrices and lifting matrices appropriately and using data from upper and lower surfaces of cells above and below the current cell. The idea of using data from the left or data from the right is analogous to upwinding used for conservation laws.

5.1.2 Stabilizing the Differential Equation

To properly couple discontinuous Galerkin with the level set equation, additional measures must be taken to ensure that the numerical scheme is stable and accurate. Using a circle as an example, shown in Figure 5.2, sharp gradients occur in the level set when points in the domain are equidistant to multiple locations on the surface (such as the center of the circle). With any high order scheme, sharp gradients will cause numerical instabilities if not treated properly. In the case of the level set, a sharp gradient at the center of the circle will continue to grow sharper, corrupting the solution.



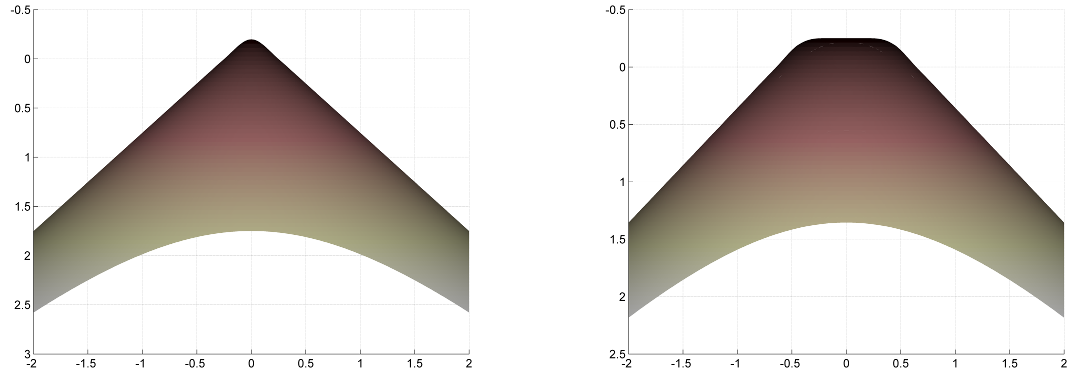
(a) Isometric View (b) Side View

Figure 5.2: Level set for a circle of radius = 0.25

Another issue that it is difficult to avoid is a case where the gradient is zero. In the example of the circle, this will happen if the center point, where the sharp gradient occurs, is located within a cell, rather than at the boundary of a cell. In that case, the sharp point will be smoothed out, effectively changing the state at that point from a sharp gradient to a gradient of zero. Using the basic time evolution scheme described in Chapter 5.1.1 for discontinuous Galerkin schemes, zero gradients (such as shown in the center of the circle shown in Figure 5.3, will develop incorrectly.

From the Figure, the zero gradient at the center effectively propagates outward toward the domain edges, steadily flattening the level set. Remember, the level set should simply convect in the z-direction for some constant wave speed. Thus, the result in Figure 5.3b does not make physical sense.

Osher and Fedkiw [52] provide a number of corrections for this, one being a method known as reinitialization. Reinitialization is a process performed periodically to "reset"



(a) Initial Surface

(b) Propagated with basic ODE

Figure 5.3: Level set propagation for a circle of radius = 0.25

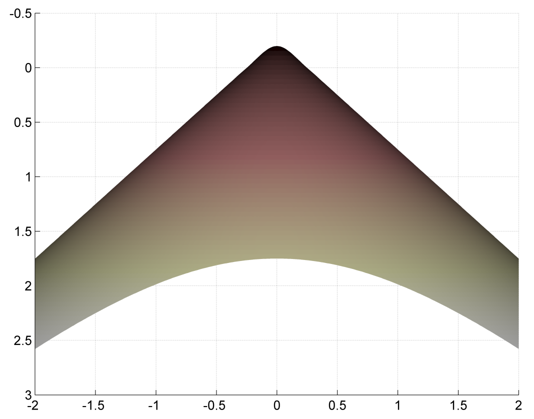
the gradient of the level set to its original shape (or close to it). The principal behind reinitialization is that only behavior at or near the zeroeth level set need to be conserved, and that behavior elsewhere can be reset without issue. The basic equation for reinitialization is the standard differential equation with constraint added, mathematically represented as:

$$\mathbf{h}(x) = \operatorname{sign}(\phi)\left(|\nabla\phi| - 1\right) \tag{5.13}$$

This constraint essentially takes advantage of the fact that the magnitude of the gradient of the level set should be equal to 1 everywhere. The downside to reinitialization is that it is costly due to the fact that it is an iterative process that is separate from the time evolution. However, it stands to reason that if the constraint were added to the original differential equation to be solved, smaller corrections to the level set would occur simultaneously with time evolution, and reduce the cost associated with reinitialization. The equation of motion thus becomes

$$\frac{\partial\phi}{\partial t} + |\nabla\phi| = \operatorname{sign}(\phi)\left(|\nabla\phi| - 1\right) \tag{5.14}$$

The same circle that was propagated without this correction in Figure 5.3 is given below in Figure 5.4. From the Figure, the level set was correctly propagated from state 5.4a to 5.4b.


(a) Initial Surface

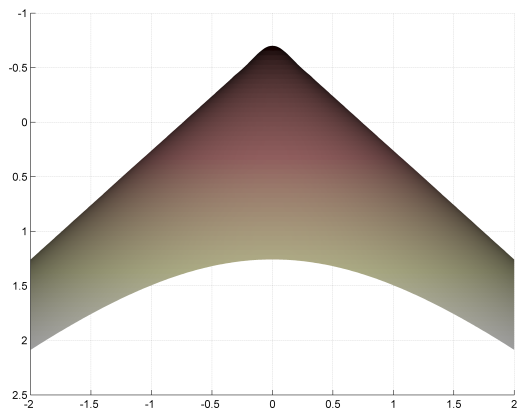
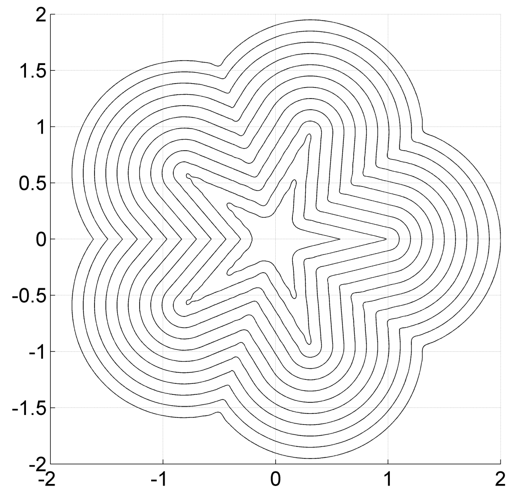

(b) Propagated Surface

Figure 5.4: Level set propagation for a circle of radius = 0.25 with source term

At this point, Equation 5.14 represents a robust equation for propagating simple, well behaved level sets, such as that for the circle. However, for more complex grains, such as a five pointed star shown below in Figure 5.5, more work must be done on the differential equation. Specifically, convex regions of the star (areas where the zeroeth contour will remain sharp) will cause numerical issues as time evolves.


(a) Top Down View

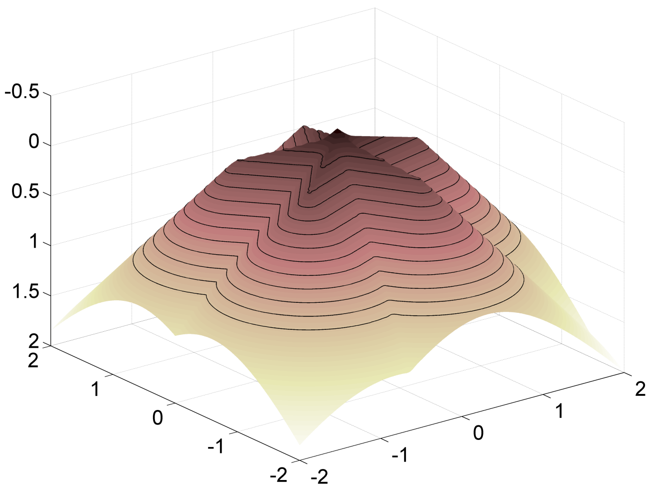

(b) Isometric View

Figure 5.5: Five Pointed Star Level Set

To demonstrate the numerical instability, the normalized residual was tracked during time evolution. It is worth noting here that the normalized residual at every time step should be 1. This is due to the special property of the level set which states that the gradient must be equal to one everywhere at all times. Since the 2-D level set is essentially being convected in the z-direction, the residual should be the same at every time step. The normalized residual for discontinuous Galerkin is calculated as

$$\mathcal{R}^n = \frac{\sqrt{\sum_i \sum_k rhs_{i,k}^2 w_k}}{\mathcal{R}^1} \tag{5.15}$$

By tracking this quantity over time, a sense of stability can be measured. To determine the proper time step, the Courant-Friedrichs-Lewy (CFL) condition was satisfied as follows

$$CFL = \lambda \Delta t \frac{d^2}{\Delta x} \tag{5.16}$$

where λ is the maximum wavespeed for the problem at hand. The traditional CFL condition essentially states that information at one point should not travel farther in a single time step than the adjacent cell. For DG, this definition is expanded to include the total degrees of freedom for a cell (d^2) [56]. The desired simulation time for this was set to a pseudo-time of 0.5, meaning that the level set values at simulation end should be exactly $\phi_0 + 0.5$. The residual for this simulation is shown below in Figure 5.6.

From the Figure, the simulation grew unstable very quickly, since the simulation only reached a psuedo-time of 0.03 out of 1.0. To relieve this effect, an explicit artificial viscosity term was added to the differential equation as discussed by [63] and [64], resulting in Equation 5.17.

$$\frac{\partial \phi}{\partial t} + |\nabla \phi| = \text{sign}(\phi)\left(|\nabla \phi| - 1\right) + \nu_x \nabla \cdot (\nabla \phi) \tag{5.17}$$

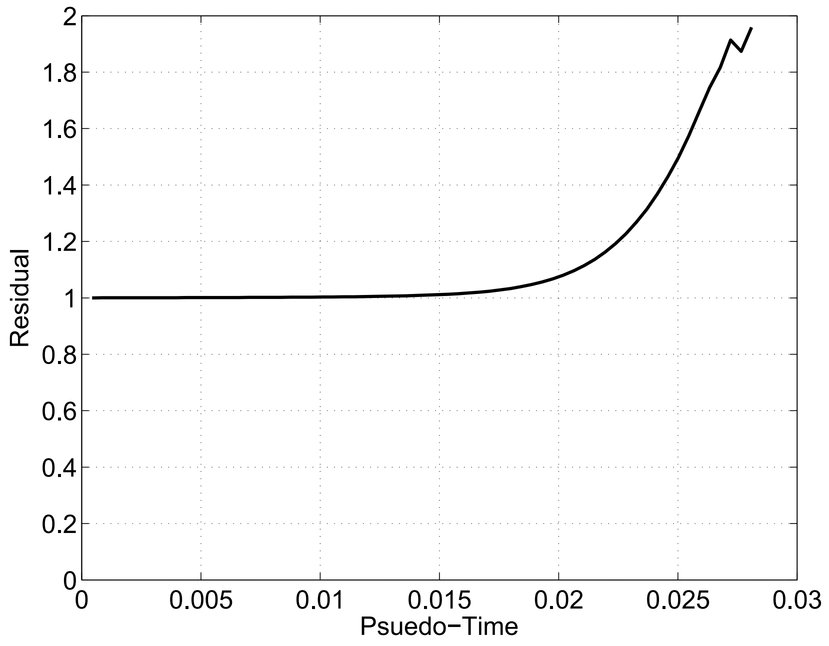


Figure 5.6: Normalized residual for five-pointed star using mesh of 10×10 with 9th order polynomial

In Equation 5.17, ν_x is the diffusion constant which is directly controlled by the user. Artificial viscosity, implicit or explicit, reduces the gradients in the state, effectively smoothing the state. Whether this is physical or not, it helps keep the solution stable. For the level set equation, it is desirable to keep the sharp ridges in the state as well maintained as possible; however sharp gradients tend to cause numerical instability. For this reason, the diffusion term was only used when $|\nabla\phi|$ was excessively large ($|\nabla\phi| >> 1$) or nearly 0. When either of these conditions were met, the tuning parameter was set to the desired value, otherwise the diffusion term was not added to Equation 5.17. In order to tune the dissipation constant and the following steps were taken.

- Set the CFL number to a sufficiently low number to drive the timestep down. A typical value is around 0.2.

- Begin with a low dissipation constant, and track the residual. ν set to 0 will give some insight into how much dissipation will be required.

- Slowly raise the dissipation constant until the residual remains approximately 1 or slightly less than 1.

- Drive the CFL number back up as high as possible while remaining stable

The results of this process for the five pointed star in Figure 5.5 are shown below in Figure 5.7.

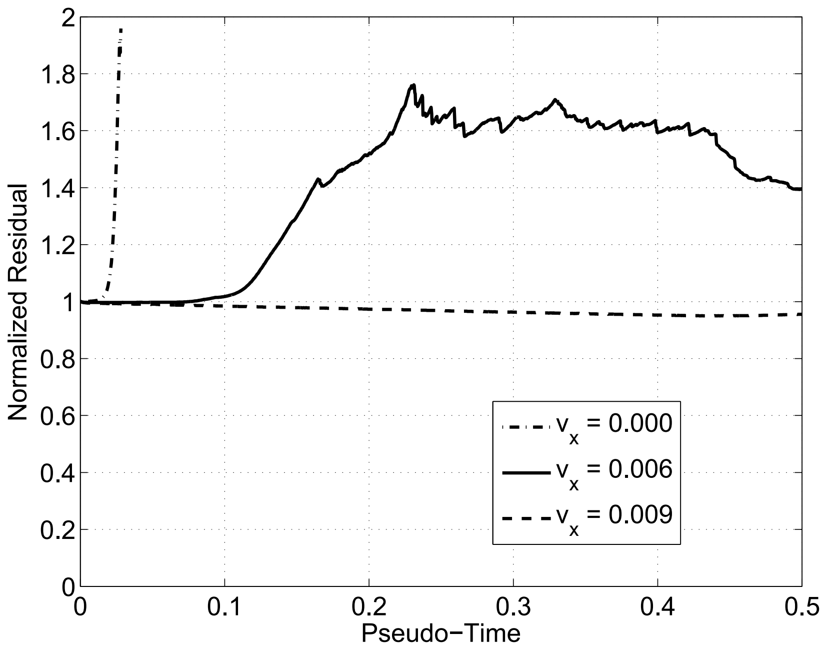
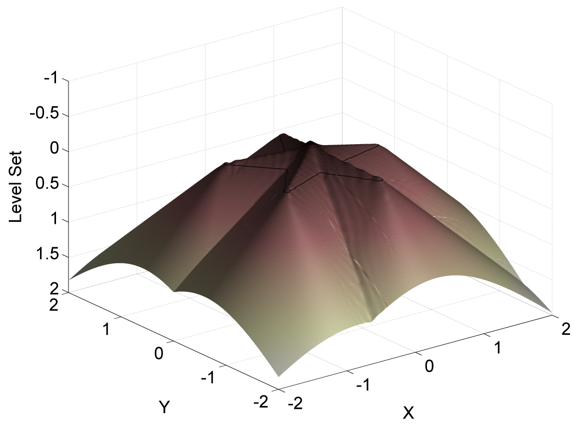


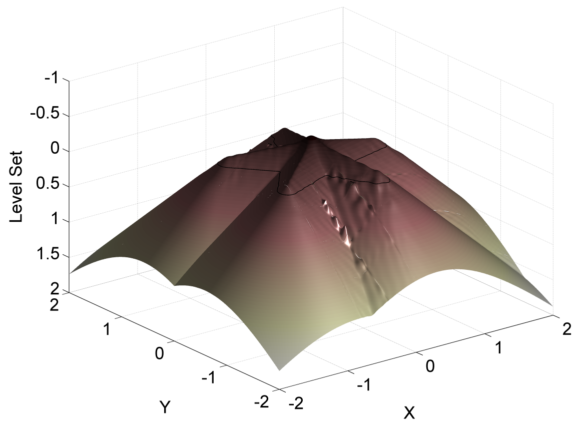
Figure 5.7: Normalized residuals for five-pointed star for varying dissipation constants

As a comparison demonstrating why it is necessary to keep the residual close to 1, the evolution of $\nu_x = 0.006$ was compared to $\nu_x = 0.009$ in Figures 5.8 and 5.9 respectively.

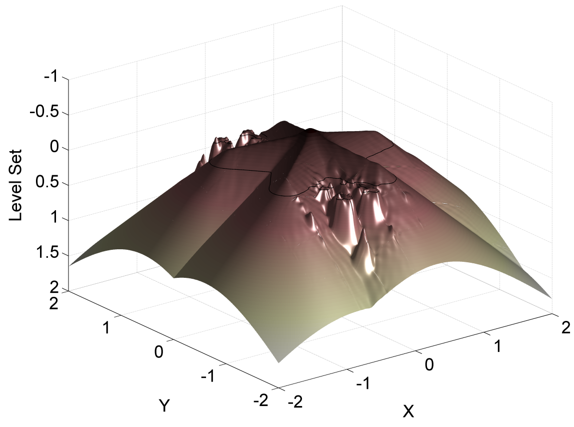
From Figure 5.8b, small wrinkles start to form which directly lead to the rise in the residual observed in Figure 5.7. This is where tracking the residual and fine tuning the dissipation constant becomes important. Although the simulation in Figure 5.8 continued to run without excessive rises in the residual, the simulation had already been corrupted, leading to a poor solution. From Figure 5.9,the level set simply convects in the z-direction, maintaining its original shape. While a few wrinkles do develop, there is enough dissipation to keep them from growing and compromising the solution. Figure 5.10 shows the zeroeth level set for each case at an approximate time of 0.2 after start. Clearly from Figure 5.10a, the zeroeth level set, the area that is most important in the domain, was corrupted without



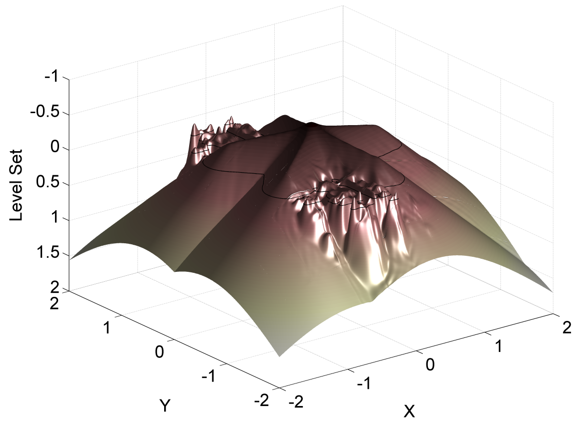
(a) t=0.022



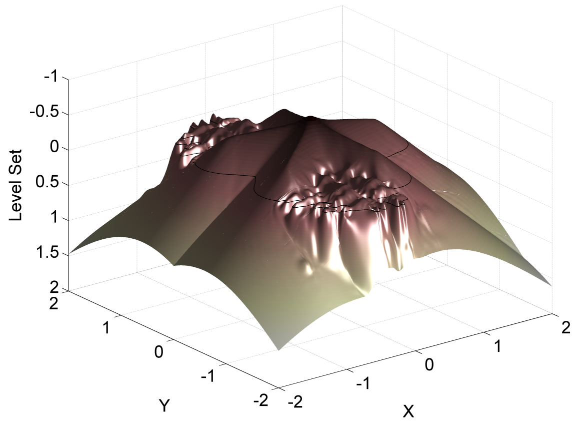
(b) t=0.110



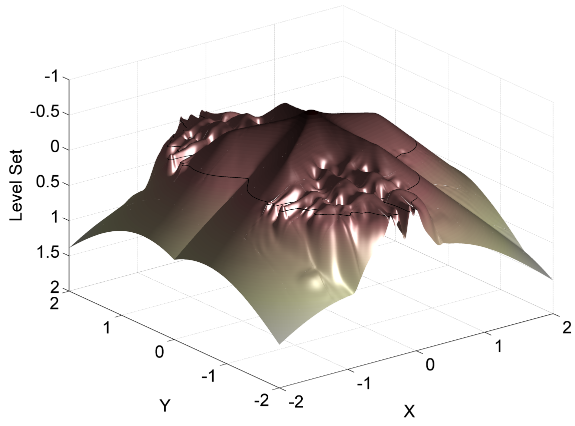
(c) t=0.200



(d) t=0.285

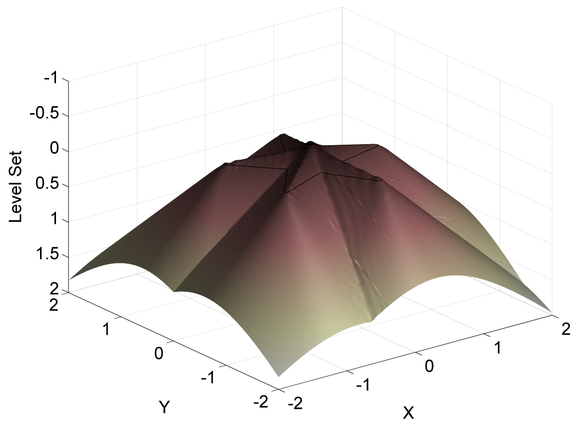


(e) t=0.373

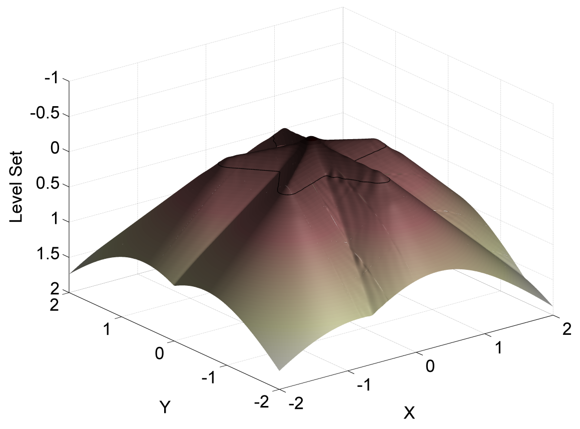


(f) t=0.460

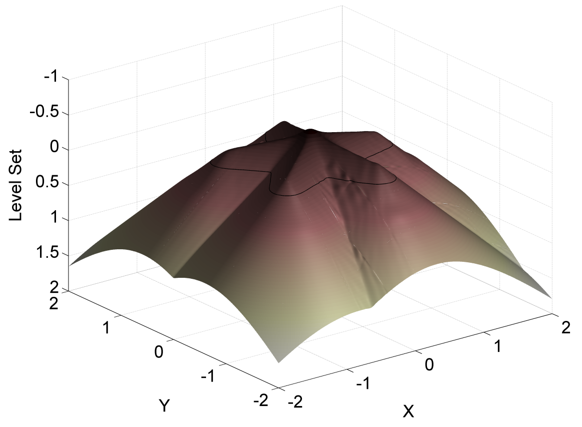
Figure 5.8: Level set propagation for a five pointed star with dissipation constant set to 0.006



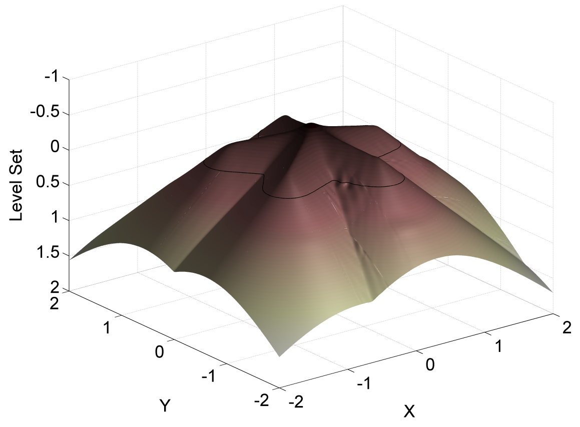
(a) t=0.022



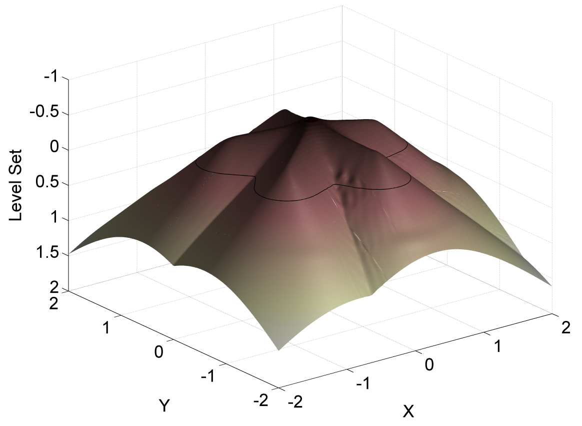
(b) t=0.110



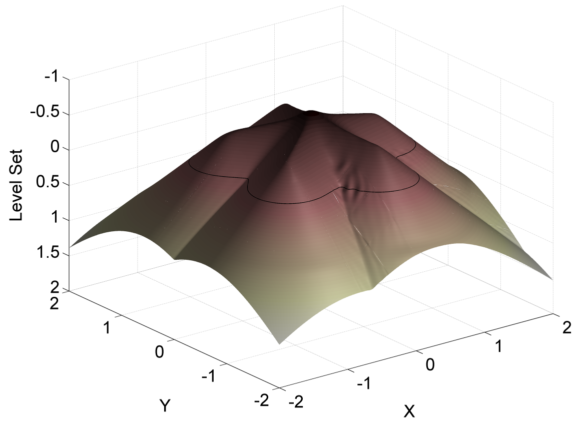
(c) t=0.200



(d) t=0.285



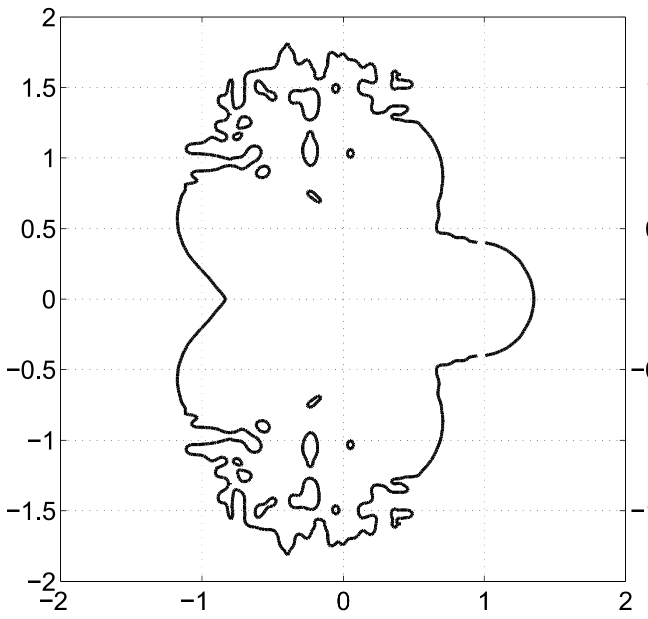
(e) t=0.373



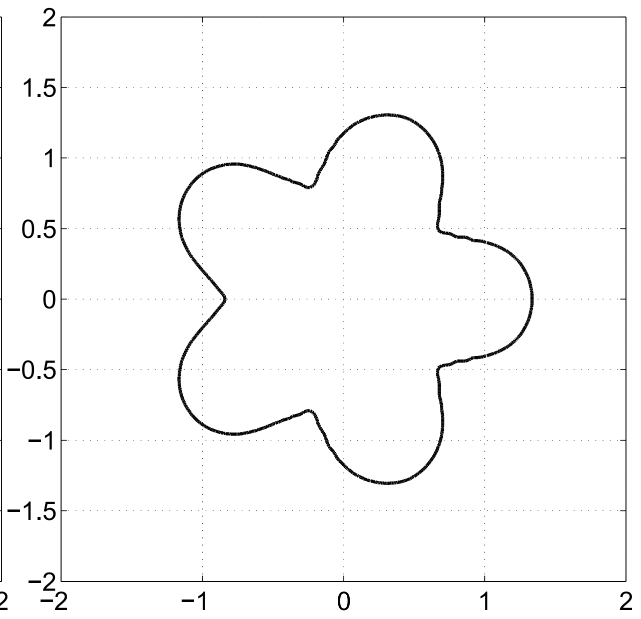
(f) t=0.460

Figure 5.9: Level set propagation for a five pointed star with dissipation constant set to 0.009

sufficient artificial dissipation. When sufficient artificial dissipation is provided, the solution remains well behaved, and the zeroeth level set evolves correctly.



(a) $\nu = 0.006$



(b) $\nu = 0.009$

Figure 5.10: Level set propagation for a five pointed star at $t = 0.2$

5.1.3 Motor Case Implementation

While it is important to evolve the level set correctly over time, it is equally important to accurately determine burn area (in 2-D, perimeter) at each time step for a rocket motor simulation program. There are issues associated with determining area accurately, particularly when it comes to determining area as the surface interferes with the motor case wall as was discussed in Section 4.2. Revisiting an explicitly defined case, shown in Figure 5.11, the bandwidth associated with smoothing the Dirac delta function, used to determine where the surface is, causes an unfavorable drop in calculated area near the boundary if grid spacing is relatively large.

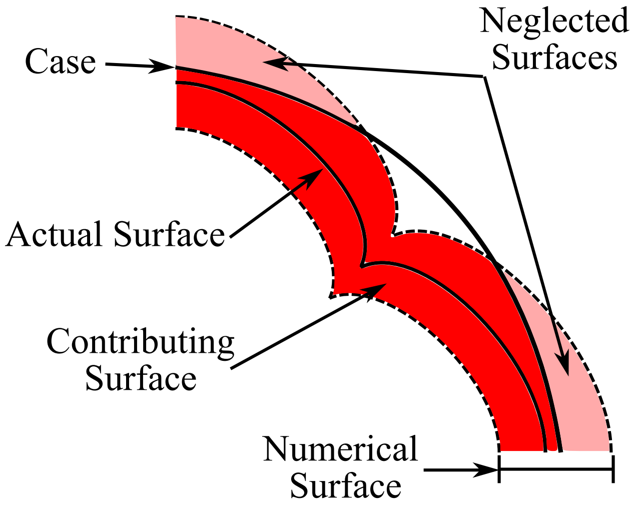


Figure 5.11: Motor Surface near the Case Boundary

This effect could be further exaggerated in DG grid where nodes are based on Lobatto quadrature as shown in Figure B.3 of Appendix B, and can be sparse in some areas and more densely populate other areas of a cell. Recall the equation for burn area:

$$A = \int_{\Omega} \delta\left(\phi\left(x\right)\right)\left|\vec{\nabla}\phi\right| dx \tag{5.18}$$

This equation can be modified to account for an arbitrarily defined case simply by defining a a step function defining locations inside and outside of the case. This new step function is incorporated into the burn area calculation as follows.

$$A = \int_{\Omega} h_{case}\left(\phi_{case}\right) \delta\left(\phi\left(x\right)\right) \left|\vec{\nabla}\phi\right| dx \tag{5.19}$$

In this instance, h_{case} is defined as

$$h_{case} = \begin{cases} 1 & \phi_{case} > 0 \\ \frac{1}{2}\left(1 + \cos\left(\frac{\pi\phi}{\varepsilon}\right)\right) & -\varepsilon < \phi_{case} < 0 \\ 0 & \phi_{case} < 0 \end{cases} \tag{5.20}$$

where a negative signed distance function defines the region exterior to the case, and the positive signed distance function defines the region interior to the case. By implicitly defining the case boundary, two advantages are gained: 1) the smeared surface is allowed to exceed the explicit boundary and continue to contribute to the area calculation, and 2) arbitrary case geometries can be implemented. An example of a 2-D cylindrical case and square case are given in Figure 5.12.

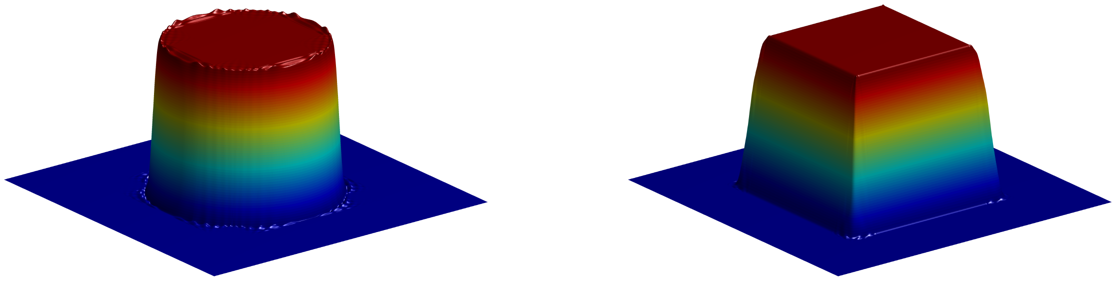
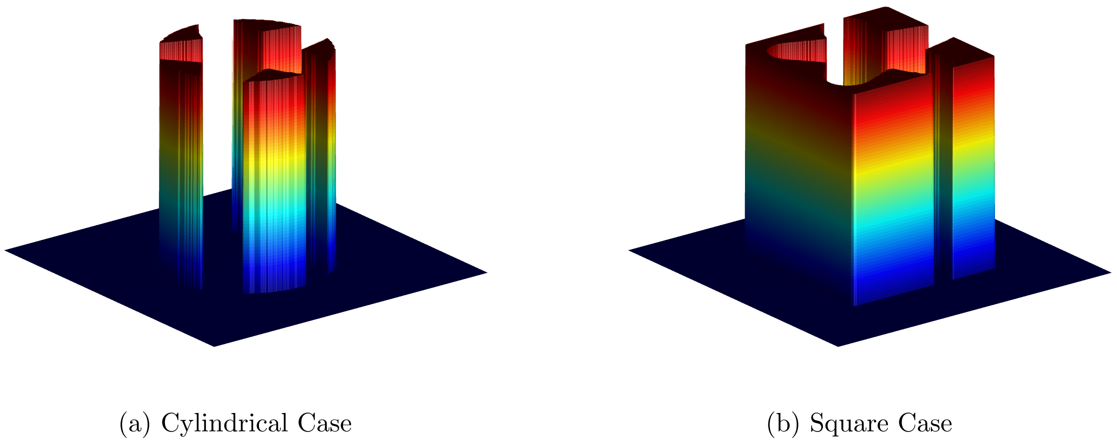

(a) Cylindrical Case (b) Square Case

Figure 5.12: Heaviside function for motor case

To demonstrate, the same five-pointed star given above was simulated with a circular case of radius equal 1 as well as with a square case with side length of 2. Figure 5.13 demonstrates the star as it burns into the two different case walls. In Figure 5.13, the raised surface represents the remaining propellant in the case.


(a) Cylindrical Case

(b) Square Case

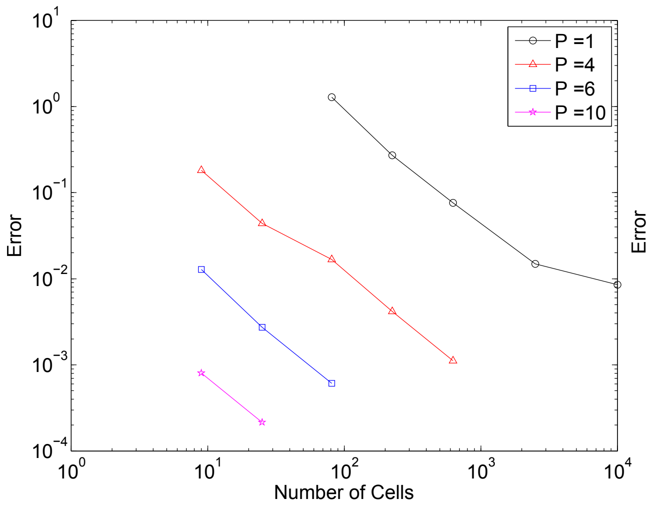
Figure 5.13: Five pointed star burning into the case wall

5.2 Results

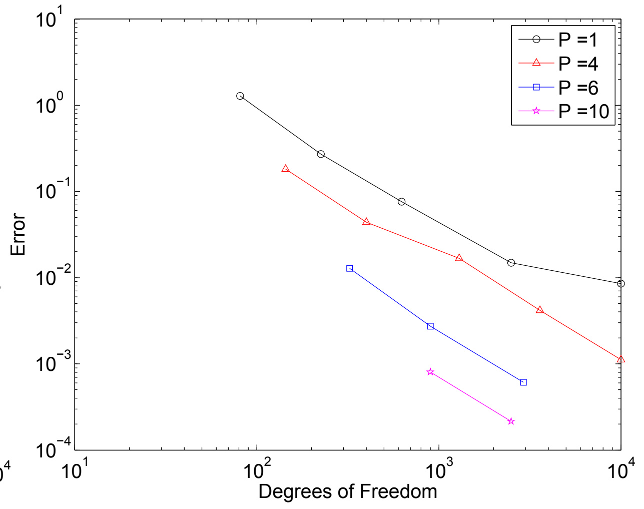
The discussion in Chapter 3 and Section 5.1.1 has led to the development of a robust and efficient approach to solving the grain regression problem. Two example cases were tested with this modified DG approach in order to confirm both robustness and accuracy. The first case is a circular grain cross section in a circular case, and was tested with varying number of cells using varying degree polynomials. A comparison was made between these higher order results and the first order program developed in Chapter 4. This comparison was made for a circular grain cross section of radius 0.7 with a outer case diameter of 2. The number of cells were varied from 9 to 625 in higher order schemes and from 81 to 10,000 in the first order scheme. A true comparison of the schemes looks not at the number of cells but rather the degrees of freedom per cell. Since first order cells contain only one degree of freedom, the number of cells is equivalent to the degrees of freedom. However, for higher order cells, the degrees of freedom can be calculated using Equation 5.21, and the total work required scales with degrees of freedom.

$$DOF = (d+1)^2 \cdot N_{cells} \tag{5.21}$$

The higher order schemes varied from degree 3 polynomial up to degree 9 polynomial, giving total degrees of freedom ranging from 144 up to 10,000, which is more on par for the degrees of freedom for the first order scheme. The results for the simulation are shown below in Figure 5.14. It should be noted that the error was a root-mean-square based error term between the simulation and the analytical solution for the entire level set at time t. In other words, Figure 5.14 gives a measure of a particular schemes ability to propagate the level set correctly. This comparison will only serve as a validation for using DG over conventional first order schemes, and does not speak to the accuracy of the scheme's to correctly determine area as a function of time.



(a) Error vs Number of Cells



(b) Error vs Degrees of Freedom

Figure 5.14: Comparison of polynomial order with error

From Figures 5.14a and 5.14b, it is clearly cheaper to use higher order integration schemes over conventional first order methods from the standpoint of number of cells and total degrees of freedom. For higher order polynomials, the reqiured number of cells is lower to achieve the same error at lower orders. The same trend is observed for error versus degrees of freedom. The second judgement for accuracy of the scheme is ability to determine area as a function of time. Figure 5.15 gives various burn area profiles for different configurations of number of cells and polynomial orders.

From Figure 5.15, two of the configuration have no issues propagating the level set; however, some issues arise when calculating area as the surface approaches the case boundary. In every case, a tail-off begins to form near the boundary around a distance of 0.3 due to case radius of 1 and inner radius of 0.7. Where this tail-off begins depends on the number of cells as well as the polynomial order. However, combining both high order and high number cells is not necessarily cheaper anymore. In Figure 5.16, error was compiled by comparing the analytical solution for burn area to the higher order schemes. From the Figure, the total degrees of freedom apparently dictate the error in area. This is directly related to the smoothed Heaviside function bandwidth problem previously discussed. The bandwidth is

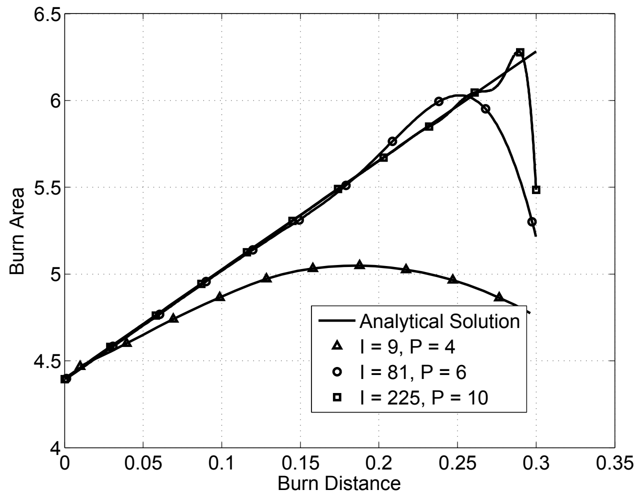
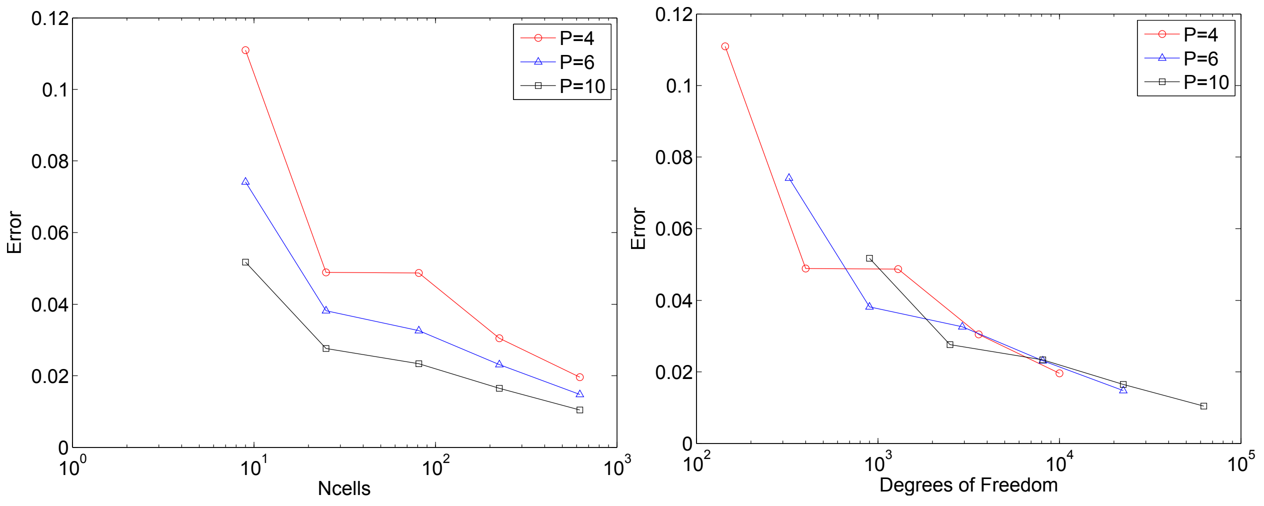


Figure 5.15: Burn area versus burn distance comparison between analytical method and DG schemes

proportional to characteristic length and inversely proportional to order polynomial. This is a situation where solution adaptivity could be useful. Modifying the mesh to use smaller cells or higher order polynomials near the case, and large cells/lower order polynomials away far from the case could alleviate this issue without penalty in computational cost.



(a) Error vs Number of Cells

(b) Error vs Degrees of Freedom

Figure 5.16: Comparison of polynomial order with error compiled using analytical area

The second test case is a more customary example of a star grain that would be found in typical rockets. It is a five pointed star with a maximum inner radius of 7 inches, a minimum inner radius of 5 inches, a case diameter of 20 inches, and a grain length of 24 inches. The fuel type was assumed to be PBAN/AP/AL. Shown in Figure 5.17 is the grain cross section (red), the motor case (blue), and the successive grain regression lines (black) effectively representing the level set function.

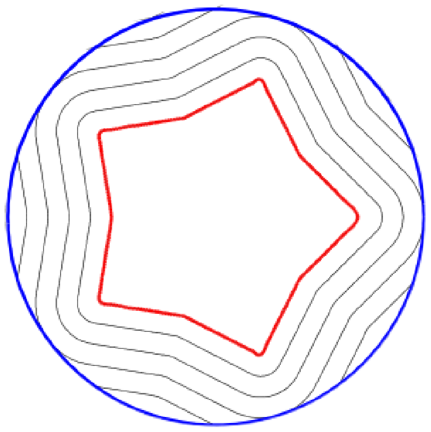
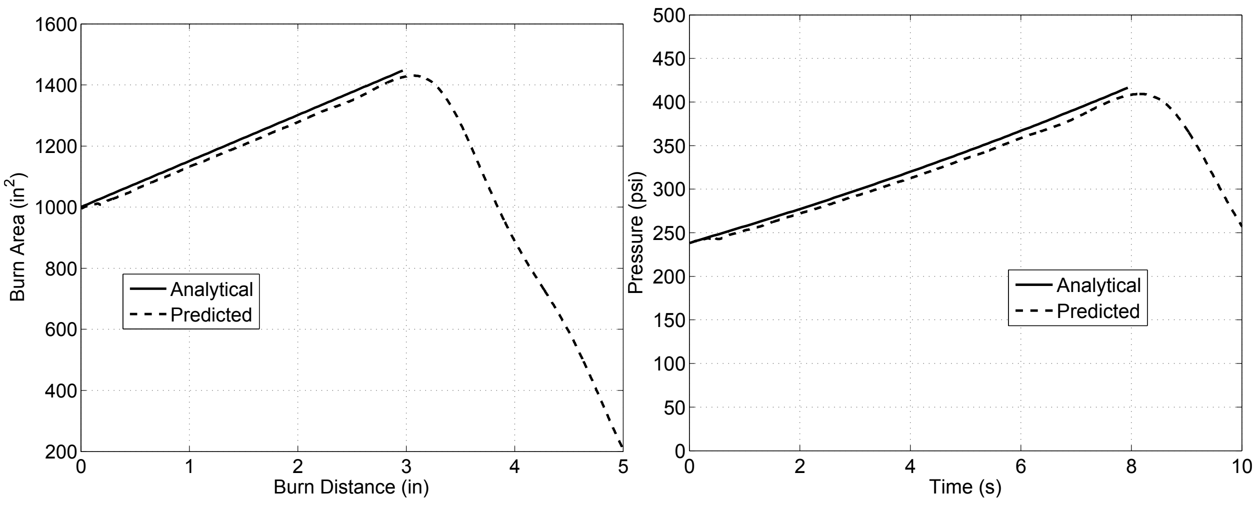


Figure 5.17: Five pointed star with grain cross with grain regression lines

This motor was tested using 81 total cells with a 9$^{\text{th}}$ order polynomial. Sufficient domain space was established to ensure that the zeroeth level set could not occupy areas both within the motor case and outside the global domain simultaneously. This helps to establish confidence that the calculation for area does not conflict with the domain boundary and corrupt the calculation for area. Figure 5.18 shows the burn area as a function of burn distance and the chamber pressure as a function of time. From Figure 5.18a, there is an underprediction in burn area of about 30 square inches throughout the burn distance. Likewise there is an underprediction in pressure versus time as well. This underprediction in area should be expected according to [52] due to the first order nature of the smeared heaviside and dirac delta functions. These two equations are first order accurate no matter the integration method. Therefore, there is a current limitation as to how accurate an area calculation can be using the level set method.

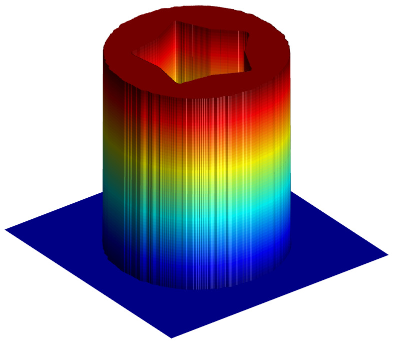


(a) Burn area versus burn distance (b) Pressure versus time

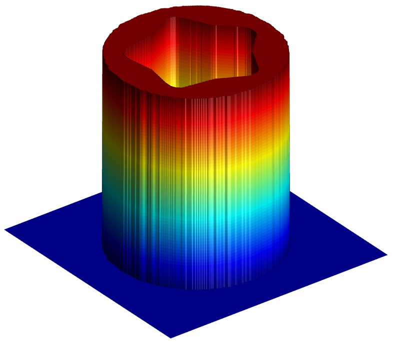
Figure 5.18: Comparison of five pointed star simulated with DG versus analytical solution

Despite the limitations on the area calculation, there are some positives that can be noted on these results. One key observation to make is the ability of this approach to correctly account for the circular case. For this test case, the analytical solution should (and does) end at a burn distance of 3 inches where phase 2 burning ends and phase 3 burning begins. In most analytical schemes, phase 3 burning is neglected. The DG scheme should also burn out or at least begin tailing off at the same burn distance of 3 inches. And indeed, from Figure 5.18a the tailoff begins to occur at 3 inches. Also, it would be expected that the final burnout would occur at a burn distance of 5 inches because the minimum inner radius of the grain started 5 inches from the motor case. Figure 5.19 shows the remaining propellant in the motor case at various burn distances. At a burn distance of 3 inches (Figure 5.19d), part of the propellant has reached the case wall and will no longer contribute to the burn area, as evidenced previously in Figure 5.18a.

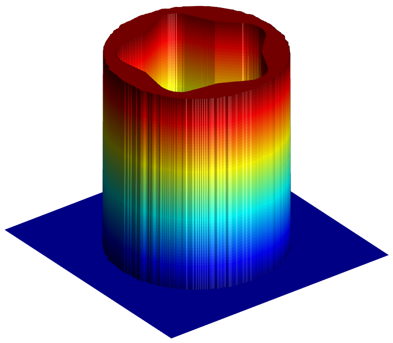
To demonstrate the generality of this SRM code, the same star grain was tested in a square case with a side length of 20 inches. The grain evolution for this test case can be



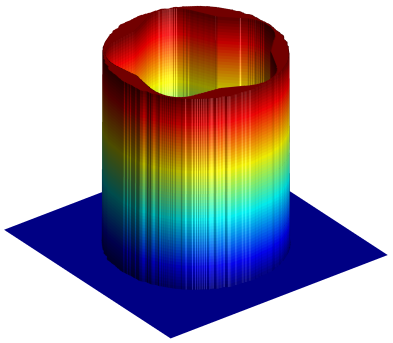
(a) Burn Distance = 0 in.



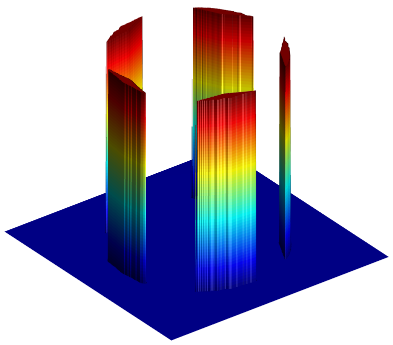
(b) Burn Distance = 1 in.



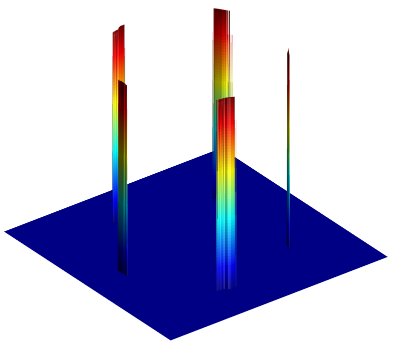
(c) Burn Distance = 2 in.



(d) Burn Distance = 3 in.

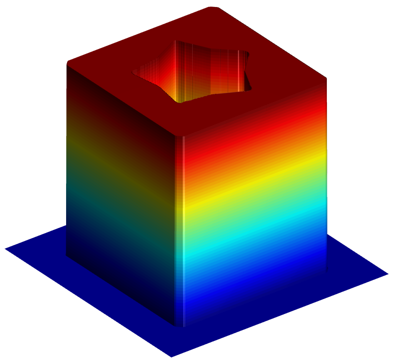


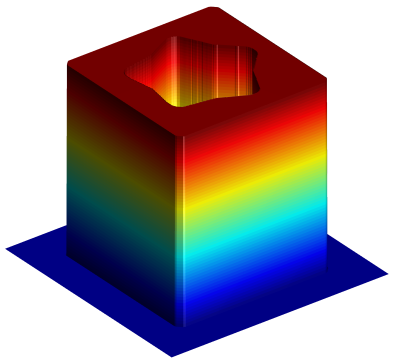
(e) Burn Distance = 4 in.

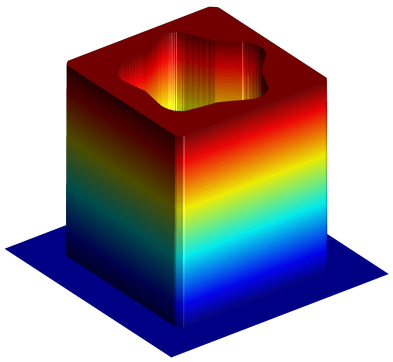


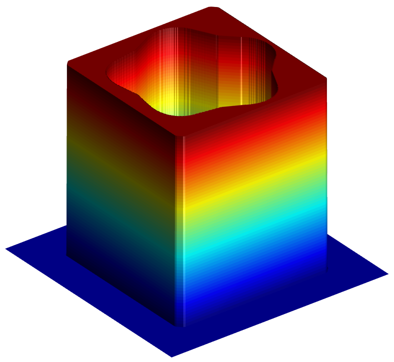
(f) Burn Distance = 5 in.

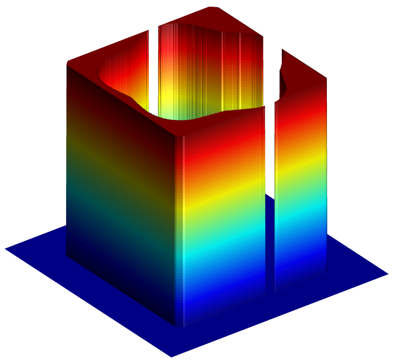
Figure 5.19: Propellant remaining for five pointed star at various burn distances for a circular motor case


(a) Burn Distance = 0 in.


(b) Burn Distance = 1 in.


(c) Burn Distance = 2 in.


(d) Burn Distance = 3 in.


(e) Burn Distance = 4 in.

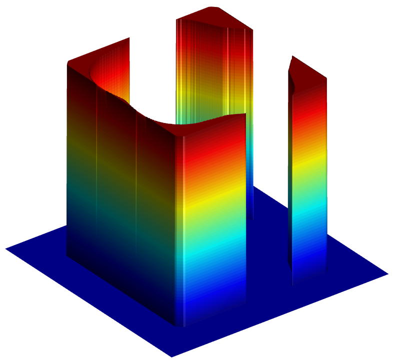
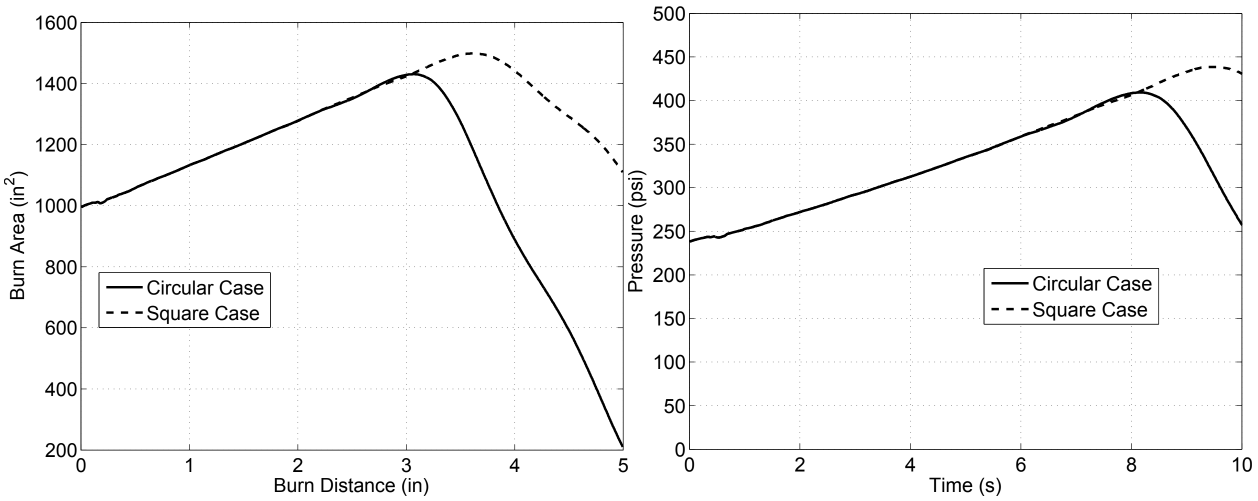

(f) Burn Distance = 5 in.

Figure 5.20: Propellant remaining for five pointed star at various burn distances for a square motor case

seen in Figure 5.20. From the Figure, it is apparent that geometric generality is conserved. For the circular case, at a burn distance of 3 inches, every star point reached the case wall. However, for a square case at the same burn distance, only one star point reached the case wall. In fact, after a burn distance of 5 inches, a relatively large amount of propellant still remains as was expected. Knowing the propellant regression, it should be expected that the burn area as a function of burn distance should be identical to that of the circular case for the first 3 inches. Shown in Figure 5.21 is a plot comparing the burn area and pressure profiles for the circular case to the profiles for the square case.



(a) Burn area versus burn distance

(b) Pressure versus time

Figure 5.21: Comparison of pressure profiles for star grain in a circular case and a square case

As was expected, the profiles are identical until the surface reaches the motor case. At the motor case, the burn area continues to increase until more star points reach the case wall thus causing the burn area to decrease. This result is useful from the standpoint that this approach can handle completely arbitrary propellant shapes, both interior or exterior.

Chapter 6

Capabilities and Further Development

6.1 Conclusions

This thesis developed a high fidelity motor simulation approach that is robust and arbitrary, as opposed to analytical techniques that require derivation of the solution for each type of grain cross-section. The methodology used combined previous work in the fields of surface tracking and propagation with higher order integration schemes to develop a fast and computationally efficient approach to grain surface evolution. Problems inherent to propagation of the signed distance function were identified, and successfully eliminated. The level set method had already proven to be an arbitrary approach to the grain regression problem, and the addition of the discontinuous Galerkin method to the level set approach developed into the robust and computationally efficient method discussed. The end product was a solid foundation for an approach to SRM simulation that has the potential to be the cheapest and most robust method available. The following remarks are concluded from the work performed in this thesis:

- The discontinuous Galerkin approach to surface evolution proved to be more efficient than traditional first order schemes that have previously been coupled with the level set method.
- The marriage between DG and LSM did require some modification of the level set differential equation. Inclusion of the slope correction source term and the explicit artificial source term did introduce complexity to the approach, but proved to be vital in properly evolving the level set over time. The source term should always be

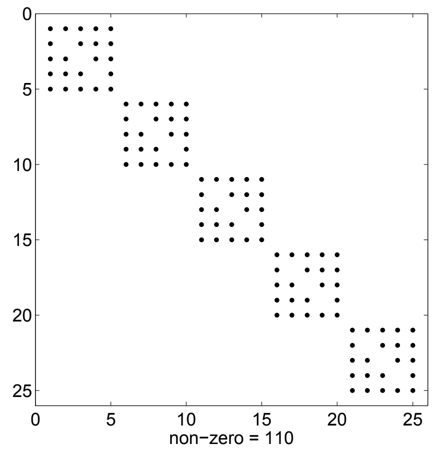
incorporated into finite volume type schemes. Explicit artificial viscosity is required for high order methods due to oscillations.

- While it was shown that surface tracking is more accurate and efficient with both decreasing number of cells and higher order polynomial, the limiting factor on error in area calculation proved to be total degrees of freedom.
- An approach to incorporating arbitrary motor case boundaries was developed and proved to be more robust and as accurate as previous work involving discretized methods.
- The method for calculating burn area is only first order accurate, and as such was limited in accuracy to total degrees of freedom and problem complexity. This issue is should be addressed regardless of scheme used for integration.

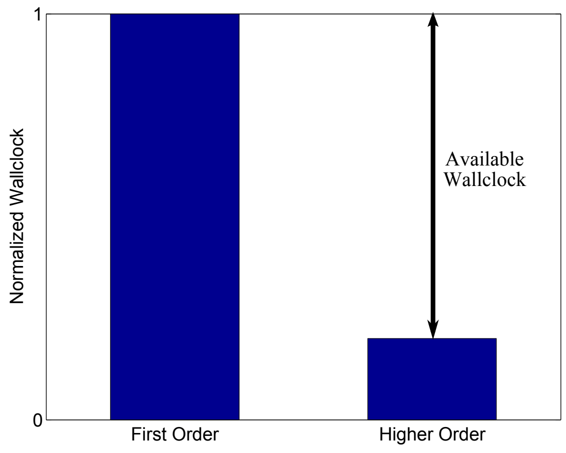
6.2 Recommendations

While a solid framework for an accurate solid rocket motor grain regression tool was developed, several key topics should be pursued for further improvement. A more accurate method for calculating area must be developed in order to move away from the first order limitation currently implemented. However, a higher order scheme for developing the area calculation will increase computational efficiency in both work and wallclock. It was not discussed in this work, but a decrease in wallclock time can be achieved for higher order schemes by taking advantage of sparse matrix multiplication. In its current implementation, any manipulation of the stiffness matrix (for example) requires N^2 calculations. But the stiffness matrix is sparse (see Figure 6.1), and would allow sparse multiplication of the large matrices that are used at higher order.

If sparse muliplication can be taken advantage of, the time required to complete a simulation can be further reduced. This savings in time can be utilized in one of two ways. The first utilization of time savings can trivially be taken at face value as time savings. The



(a) Non-zero entries for stiffness matrix of order 5



(b) Sample wallclock savings

Figure 6.1: Example utilization of sparse matrices

second utilization of time savings could be used to develop a more accurate calculation of area, include second order effects, or any other flavor of improvements to the simulation, while still remaining as cheap as first order schemes. Solution adaptivity could also be taken advantage of to provide more accuracy.

Moving forward, the current implementation of DG/LSM should be extended from two dimensions to three dimensions. Development of variable (non-uniform) wavespeed should be implemented to make inclusion of second order propellant burning effects such as multi-phase flow and erosive burning possible. It would also be desirable to test the three-dimensional code against real world motors as opposed to simpler analytical solutions. Some final improvements to the program would be to include the following: ignition and burnout transients, chamber aerodynamics, and propellant solid mechanics. A successful achievement of these recommendations would produce an end product believed to be the most robust and accurate simulation tool available.

Bibliography

[1] Dunn, S. and Coats, D., "3-D Grain Design and Ballistic Analysis Using the SPP97 Code," AIAA paper 1997-3340, 1997.

[2] Mattingly, J., *Elements of Propulsion: Gas Turbines and Rockets*, AIAA Education Series, 2006.

[3] Sutton, G. and Biblarz, O., *Rocket Propulsion Elements*, Wiley & Sons, 2001.

[4] "Solid Rocket Motor Igniters," NASA SP-8051, 1971.

[5] Foster, W. and Jenkins, R., "Analysis of Advanced Solid Rocket Motor Ignition Phenomena," NASA CR-199427, 1995.

[6] Cho, I. H. and Baek, S. W., "Numerical Simulation fo Axisymmetric Solid Rocket Motor Ignition Transient with Radiation Effect," *Journal of Propulsion and Power*, Vol. 16, No. 4, 1999, pp. 725–728.

[7] Bai, S., Han, S., and Pardue, B., "Two-Dimensional Axisymmetric Analysis of SRM Ignition Transient," AIAA paper 1993-2311, 1993.

[8] Johnston, W., "Solid Rocket Motor Internal Flow During Ignition," *Journal of Propulsion and Power*, Vol. 11, No. 3, 1995, pp. 489–496.

[9] Taylor, G., "Fluid Flow in Regions Bounded by Porous Surfaces," *Proceedings of the Royal Society, London, Series A*, Vol. 234, No. 1199, 1956, pp. 456–475.

[10] Culick, F., "Rotational Axisymmetric Mean Flow and Damping of Acoustic Waves in a Solid Propellant Rocket," *AIAA Journal*, Vol. 4, No. 8, 1966, pp. 1462–1464.

[11] Majdalani, J. and Moorhem, W. V., "Improved Time-Dependent Flowfield Solution for Solid Rocket Motors," *AIAA Journal*, Vol. 36, No. 2, 1998.

[12] Majdalani, J. and Saad, T., "Energy Steepened States of the Taylor-Culick Profile," AIAA paper 2007-5797, 2007.

[13] Majdalani, J., Xu, H., Lin, Z.-L., and Liao, S.-J., "Exact HAM Solutions for the Viscous Rotational Flowfield in Channels with Regressing and Injecting Sidewalls," AIAA paper 2010-7079, 2010.

[14] Maicke, B. A. and Majdalani, J., "On the Compressible Hart-McClure Mean Flow Motion in Simulated Rocket Motors," AIAA paper 2010-7077, 2010.

[15] Akiki, M. and Majdalani, J., "Quasi-Analytical Approximation of the Compressible Flow in a Planar Rocket Configuration," AIAA paper 2010-7080, 2010.

[16] Chedevergne, F., Casalis, G., and Majdalani, J., "DNS Investigation of the Taylor-Culick Flow Stability," AIAA paper 2007-5796, 2007.

[17] Apte, S. and Yang, V., "Unsteady Flow Evolution and Combustion Dynamics of Homogeneous Solid Propellant in a Rocket Motor," *Combustion and Flame, Elsevier Science Inc.*, 2002.

[18] Dotson, K., Koshigoe, S., and Pace, K., "Vortex Shedding in a Large Solid Rocket Motor Without Inhibitors at the Segment Interfaces," *Journal of Propulsion and Power*, Vol. 13, No. 2, 1997.

[19] Lupoglazoff, N. and Vuillot, F., "Parietal vortex shedding as a cause of instability for a long solid propellant motors – Numerical simulations and comparisons with firing tests," AIAA paper 1996-0761, 1996.

[20] Coats, D. and Dunn, S., "Improved Motor Stability Predictions for 3-D Grains Using the SPP Code," AIAA paper 1997-3251, 1997.

[21] French, J., "Tangential Mode Instability of SRMs with Even and Odd Numbers of Slots," AIAA paper 2002-3612, 2002.

[22] Fiedler, R., Jiao, X., Haselbacher, A., Geubelle, P., Guoy, D., and Brandyberry, M., "Simulations of Slumping Propellant and Flexing Inhibitors in Solid Rocket Motors," AIAA paper 2002-4341, 2002.

[23] Renganathan, K., Rao, B., and Jana, M., "Slump Estimation of Cylindrical Segment Grains of a Typical Rocket Motor under Vertical Storage Condition," *Trends in Applied Research*, Vol. 1, No. 1, 2006, pp. 97–104.

[24] Lajczok, M., "Effective Propellant Modulus Approach For Solid Rocket Motor Ignition Structural Analysis," *Computers and Structures*, Vol. 56, No. 1, 1995, pp. 101–104.

[25] Nunn, R. and Chafin, L. C., "Method and Means for Controlling the Thrust in a Solid Propellant Rocket Motor," 1971.

[26] Anderson, M., Burkhalter, J., and Jenkins, R., "Design of an Air to Air Interceptor Using Genetic Algorithms," AIAA paper 1999-408, 1999.

[27] Jenkins, R. and Hartfield, R., "Hybrid Particle Swarm-Pattern Search Optimizer for Aerospace Applications," AIAA paper 2010-7078, 2010.

[28] Albarado, K., Hartfield, R., Hurston, B., and Jenkins, R., "Solid Rocket Motor Performance Matching Using Pattern Search/Particle Swarm Optimization," AIAA paper 2011-5798, 2011.

[29] Woltosz, W., *The Application of Numerical Optimization Techniques to Solid-Propellant Rocket Motor Design*, Master's thesis, Auburn University, 1977.

[30] Barrere, M., Jaumotte, A., de Veubeke, B. F., and Vandenkerckhove, J., *Rocket Propulsion*, Elsevier Publishing Company, 1960.

[31] Willcox, M., Brewster, M. Q., Tang, K., Stewart, D. S., and Kuznetsov, I., "Solid Rocket Motor Internal Ballistics Simulation Using Three-Dimensional Grain Burnback," *Journal of Propulsion and Power*, Vol. 23, No. 3, 2007.

[32] Cavallini, E., *Modeling and Numerical Simulation of Solid Rocket Motors Internal Ballistics*, Ph.D. thesis, Sapienze Universita di Roma, Rome, Italy, 2009.

[33] Favini, B., Cavallini, E., and Giacinto, M. D., "An Ignition-to-Burn Out Analysis of SRM Internal Ballistic and Performances," AIAA paper 2008-5141, 2008.

[34] Cavallini, E., Favini, B., Giacinto, M. D., and Serraglia, F., "SRM Internal Ballistic Numerical Simulation by SPINBALL Model," AIAA paper 2009-5512, 2009.

[35] Hartfield, R., Jenkins, R., Burkhalter, J., and Foster, W., "A Review of Analytical Methods for Predicting Grain Regression in Tactical Solid Rocket Motors," *Journal of Spacecraft and Rockets*, Vol. 41, No. 4, 2004, pp. 689–693.

[36] Hartfield, R., Burkhalter, J., Jenkins, R., Anderson, M., and Witt, J., "Analytical Development of a Slotted Grain Solid Rocket Motor," AIAA paper 2002-4298, 2002.

[37] Sforzini, R., "An Automated Approach to Design of Solid Rockets Utilizing a Special Internal Ballistics Model," AIAA paper 80-1135, 1980.

[38] Sforzini, R., "Design and Performance Analysis of Solid-Propellant Rocket Motors Using a Simplified Computer Program," NASA CR-129025.

[39] Ricciardi, A., "Generalized Geometric Analysis of Right Circular Cylindrical Star Perforated and Tapered Grains," *AIAA Journal of Propulsion and Power*, Vol. 8, No. 1, 992, pp. 51–58.

[40] French, J., "Three Dimensional Combustion Stability Modeling for Solid Rocket Motors," AIAA paper 1998-3702, 1998.

[41] French, J. and Coats, D., "Automated 3-D Solid Rocket Combustion Stability Analysis," AIAA paper 1999-2797, 1999.

[42] Coats, D., Dunn, S., and Berker, D., "Improvements to the Solid Performance Program," AIAA paper 2003-4504, 2003.

[43] French, J. and Tullos, J., "Solid Rocket Motor Grid Generation and CFD for Internal Ballistcs Analysis," AIAA paper 2005-4162, 2005.

[44] Jiao, X., "Face offsetting: A unified approach for explicit moving interfaces," *Journal of Computational Physics*, 2006.

[45] Fiedler, R., *Rocstar 3 User's Guide*, University of Illinois at Urbana-Champaign, 2008.

[46] Dick, W., Fiedler, R., and Heath, M., “Building Rocstar: Simulation Science for Solid Propellant Rocket Motors,” AIAA paper 2006-4590, 2006.

[47] Jiao, X., Zheng, G., Lawlor, O., Alexander, P., Campbell, M., Heath, M., and Fiedler, R., “An Integration Framework for Simulations of Solid Rocket Motors,” AIAA paper 2005-3991.

[48] Fiedler, R., Wasistho, B., and Brandyberry, M., “Full 3-D Simulation of Turbulent Flow in the RSRM,” AIAA paper 2006-4587, 2006.

[49] Fiedler, R., Namazifard, A., Campbell, M., Xu, F., Aravas, N., , and Sofronis, P., “Detailed Simulations of Propellant Slumping in the Titan IV SRMU PQM-1,” AIAA paper 2006-4592, 2006.

[50] Osher, S. and Sethian, J., “Fronts Propagating with Curvature Dependent Speed: Algorithms Based on Hamilton-Jacobi Formulations,” *Journal of Computational Physics*, Vol. 79, No. 2, 1988, pp. 12– 49.

[51] Sethian, J., “Curvature and the Evolution of Fronts,” *Communication of Mathematical Physics*, Vol. 101, No. 4, 1985.

[52] Osher, S. and Fedkiw, R., *Level Set Methods and Dynamic Implicit Surfaces*, Texts in Applied Mathematics, Springer, 2003.

[53] Adalsteinsson, D. and Sethian, J., “The Fast Construction of Extension Velocities in Level Set Methods,” *Journal of Computational Physics*, Vol. 148, pp. 2–22.

[54] Sutherland, I. E., Sproull, R. F., and Schumacker, R. A., “A Characterization of Ten Hidden-Surface Algorithms,” *ACM Comput. Surv.*, Vol. 6, No. 1, 1974, pp. 1–55.

[55] Dupont, T. and Liu, Y., “Back and forth error compensation and correction methods for removing errors induced by uneven gradients of the level set function,” *Journal of Computational Physics*, Vol. 190, 2003, pp. 311–324.

[56] Hesthaven, J. and Warburton, T., *Nodal Discontinuous Galerkin Methods*, Texts in Applied Mathematics, Springer, 2008.

[57] Cockburn, B. and Shu, C., “The Runge-Kutte discontinuous Galerkin method for convection-dominated problems,” *Journal of Scientific Computing*, Vol. 16, No. 3, 2001, pp. 173–261.

[58] Karniadakis, G. and Sherwin, S., *Spectral/hp Element Methods for CFD*, Oxford University Press, 1999.

[59] Gautschi, W., *Orthogonal Polynomials: Computation and Approximation*, Oxford University Press, 2004.

[60] Golub, G. and van Loan, C., *Matrix Computations*, John Hopkins University Press, 1996.

[61] Cheng, Y. and Shu, C.-W., "A discontinuous Galerkin finite element method for directly solving the Hamilton-Jacobi equations," *Journal of Computational Physics*, Vol. 223, 2007, pp. 398–415.

[62] Yan, J. and Osher, S., "A local discontinuous Galerkin method for directly solving Hamilton-Jacobi equations," *Journal of Computational Physics*, Vol. 230, 2011, pp. 232–244.

[63] Klöckner, A., Warburton, T., and Hesthaven, J., "Viscous Shock Capturing in a Time-Explicit Discontinuous Galerkin Method," *Mathematical Modeling of Natural Phenomenon*, Vol. 10, No. 10, 2010, pp. 1–42.

[64] Persson, P. and Peraire, J., "Sub-Cell Shock Capturing for Discontinuous Galerkin Methods," AIAA paper 2006-112, 2006.

Appendix A

Analytical method for a star grain

Beginning with Figure A.1, the star grain is defined with the following parameters:

- R_p – maximum inner radius with no fillet
- R_i – minimum inner radius
- N – number of star points
- f – fillet radius
- ε – fraction of the angle allowed for a single star point taken up by the structure of the star point (angular fraction)

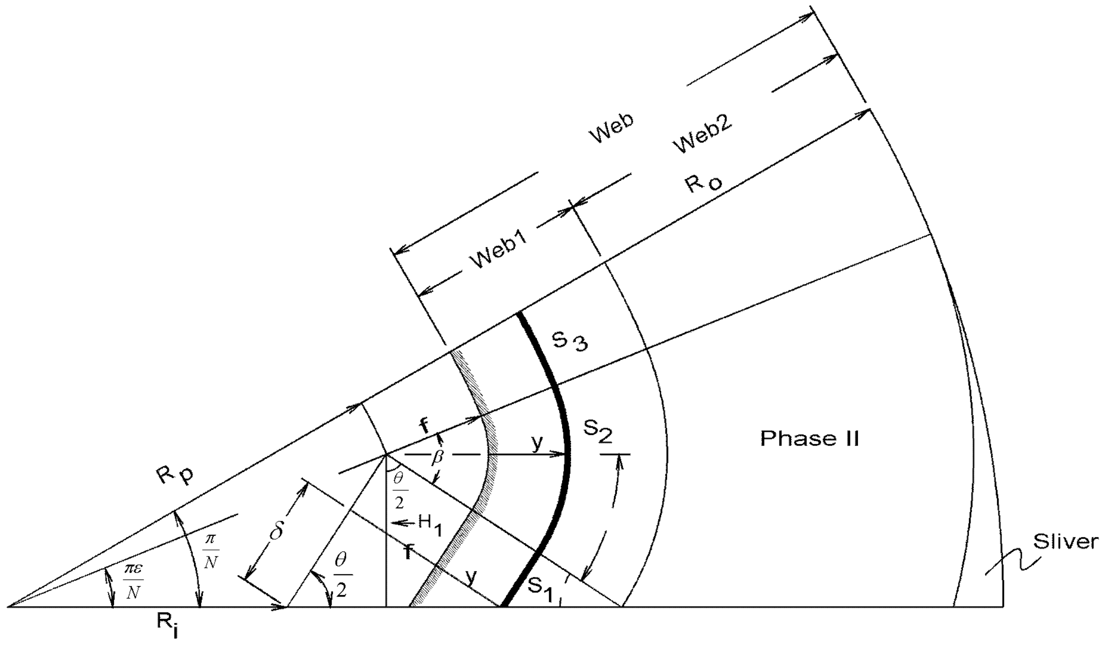


Figure A.1: Star Grain Definition

An analytical expression for the grain regression for Phase I burning can be determined from these 5 parameters (from the Figure, Phase I burning ends when the star point burns into the lower boundary. From Figure A.1, the height to the origin of the fillet radius is

$$H_1 = R_p \sin\left(\frac{\pi\varepsilon}{N}\right) \tag{A.1}$$

The star point half angle is defined as

$$\frac{\theta}{2} = \tan^{-1}\left(\frac{H_1 \tan\left(\frac{\pi\varepsilon}{N}\right)}{H_1 - R_i \tan\left(\frac{\pi\varepsilon}{N}\right)}\right) \tag{A.2}$$

The three burn perimeters (S_1, S_2, and S_3) can be calculated as follows.

$$S_1 = \frac{H_1}{\sin\left(\frac{\theta}{2}\right)} - (y+f)\cot\left(\frac{\theta}{2}\right) \tag{A.3}$$

$$S_2 = (y+f)\left(\frac{\pi}{2} - \frac{\theta}{2} + \frac{\pi\varepsilon}{N}\right) \tag{A.4}$$

$$S_3 = (R_p + y + f)\left(\frac{\pi}{N} - \frac{\pi\varepsilon}{N}\right) \tag{A.5}$$

The total burn perimeter is then calculated as

$$S = 2N\left(S_1 + S_2 + S_3\right) \tag{A.6}$$

The port area is calculated as

$$A_{p1} = \frac{1}{2}H_1\left[R_p \cos\left(\frac{\pi\varepsilon}{N}\right) + H_1 \tan\left(\frac{\theta}{2}\right)\right] - \frac{1}{2}S_1^2 \tan\left(\frac{\theta}{2}\right) \tag{A.7}$$

$$A_{p2} = \frac{1}{2}(y+f)^2\left(\frac{\pi}{2} - \frac{\theta}{2} + \frac{\pi\varepsilon}{N}\right) \tag{A.8}$$

$$A_{p3} = \frac{1}{2}\left(R_p + y + f\right)^2\left(\frac{\pi}{N} - \frac{\pi\varepsilon}{N}\right) \tag{A.9}$$

$$A_p = 2N\left(A_{p1} + A_{p2} + A_{p3}\right) \tag{A.10}$$

As previously mentioned, this set of equations defines phase I burn. Phase II burning begins with the following condition.

$$y + f = \frac{H_1}{\cos\left(\frac{\theta}{2}\right)} \tag{A.11}$$

Phase II burn perimeter can be calculated with the following set of equations.

$$\beta = \left(\frac{\pi}{2} - \frac{\theta}{2} + \frac{\pi\varepsilon}{N}\right) \tag{A.12}$$

$$\gamma = \tan^{-1}\left(\frac{\sqrt{(y+f)^2 - H_1^2}}{H_1}\right) - \frac{\theta}{2} \tag{A.13}$$

$$S_2 = (Y+f)(\beta - \gamma) \tag{A.14}$$

$$S_3 = (R_p + y + f)\left(\frac{\pi}{N} - \frac{\pi\varepsilon}{N}\right) \tag{A.15}$$

$$S = 2N\,(S_2 + S_3) \tag{A.16}$$

It should be noted that the first burn perimeter, S_1, is not present in phase II because the star point has burned to the boundary, making S_1 equal to zero. Phase II port area is calculated as follows.

$$A_{p1} = \frac{1}{2}H_1\left[R_p\cos\left(\frac{\pi\varepsilon}{N}\right) + \sqrt{(y+f)^2 - H_1^2}\right] \tag{A.17}$$

$$A_{p2} = \frac{1}{2}(y+f)^2(\beta - \gamma) \tag{A.18}$$

$$A_{p3} = \frac{1}{2}\,(R_p + y + f)^2\left(\frac{\pi}{N} - \frac{\pi\varepsilon}{N}\right) \tag{A.19}$$

$$A = 2N\,(A_{p1} + A_{p2} + A_{p3}) \tag{A.20}$$

The end of phase II burning occurs when the burn distance is equal to the web thickness which can be calculated as

$$Web = R_o - (R_p + f) \tag{A.21}$$

Appendix B

Derivation for Discontinuous Galerkin Approach

Consider the linear scalar transport equation

$$\frac{\partial u}{\partial t} + \frac{\partial f(x)}{\partial x} = 0, \quad x \in [L, R] = \Omega \tag{B.1}$$

$$f(u) = au \tag{B.2}$$

Initial conditions are defined as

$$u(x, 0) = u_0(x) \tag{B.3}$$

and boundary conditions are given as an inflow boundary

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

To continue, the domain, Ω, is approximated in K (1-D) non-overlapping elements, demonstrated as a stencil in Figure B.1, and the global state is approximated in a piecewise continuous.

$$x \in D^k \;\; :: \;\; u(x,t) \simeq u_h(x,t) = \bigoplus_{k=1}^{K} u_h^k(x,t) \tag{B.4}$$

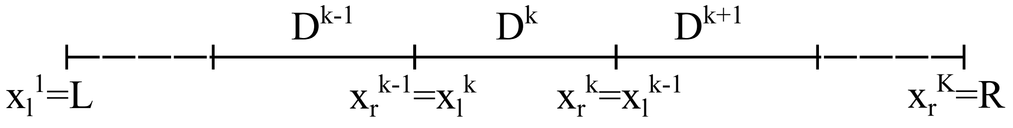


Figure B.1: Stencil for 1-D example

From Equation B.4, the local state is defined using a higher-order local basis function:

$$u_h^k(x,t) = \sum_{j=0}^{N} \widetilde{u}_j^k(t) b_j^k(x) \tag{B.5}$$

Here, each $\widetilde{u}_j^k$ is the modal coefficient for the expansion, and b_j^k is the basis function a function space $\mathbb{P}_N$. The local residual for the approximate solution is calculated as

$$R_h^k(x,t) = \frac{\partial u_h^k}{\partial t} + \frac{\partial f_h^k}{\partial x} \tag{B.6}$$

The residual is required to vanish in a Galerkin sense, so the residual becomes

$$\int_{D^k} b_i^k R_h^k dx = \int_{D^k} b_i^k \frac{\partial u_k^k}{\partial t} dx + \int_{D^k} b_i^k \frac{\partial f_h^k}{\partial x} dx = 0, \quad i = 0, ..., N \tag{B.7}$$

Thus the solutions on each element are not required to be continuous at element interfaces (hence the name *discontinuous* Galerkin). Using the divergence theorem to integrate Equation B.7 results in

$$\int_{D^k} b_i^k \frac{\partial u_h^k}{\partial t} dx + \int_{D^k} \frac{\partial}{\partial x} \left(b_i^k f_h^k \right) dx - \int_{D^k} \frac{\partial b_i^k}{\partial x} f_i^k dx = 0 \tag{B.8}$$

$$\int_{D^k} b_i^k \frac{\partial u_h^k}{\partial t} dx + \oint_{\partial D^k} b_i^k f_h^k n_x^k ds - \int_{D^k} \frac{\partial b_i^k}{\partial x} f_i^k dx = 0 \tag{B.9}$$

Because the state was approximated to be piecewise continuous, special care must be given to the second term in Equation B.9. As is performed in finite element methods, a numerical flux function will replace the flux at the interface between adjacent cells.

$$x \in \partial D^k :: \;\; f_h^k n_x^k \rightarrow f^* \left(u_h^{k,-}, u_h^{k,+}; n_x^{k,-} \right) \tag{B.10}$$

Defining the interface flux in this manner allows for the following:

- Boundary conditions set using $u_h^{k,+}$
- Numerical flux egressing from the *right* edge is the negative of the flux egressing from the *left* edge of adjacent cell $\rightarrow f^* \left(u_h^{k,-}, u_h^{k,+}; n_x^{k,-} \right) = -f^* \left(u_h^{k,+}, u_h^{k,-}; n_x^{k,+} \right)$
- Numerical flux of two identical states is simply the flux $\rightarrow f^* \left(u, u; n_x \right) \equiv f(u) n_x$

This leaves the final equation as

$$\int_{D^k} b_i^k \frac{\partial u_h^k}{\partial t} dx + \oint_{\partial D^k} b_i^k f^* \left(u_h^{k,-}, u_h^{k,+}; n_x^{k,-} \right) ds - \int_{D^k} \frac{\partial b_i^k}{\partial x} f_i^k dx = 0 \tag{B.11}$$

A large part of DG is determining exactly what to use as the bases function, $b_i^k(x)$. These will typically be some sort of polynomial: linear connections, cubics, Taylor series expansion, etc. For a more comprehensive listing of polynomials and the theory behind them, see [59].

Consider a single cell k in D^k. Using a standard interval of $x \rightarrow \xi \in (-1, 1)$ yields access to a class of functions known as recursion polynomials, where polynomial of degree N relies on the solution to polynomial degree $(N-1)$. The state space for this standard interval is of the form

$$u_h(\xi) \in \mathbb{P}_N \tag{B.12}$$

The polynomial function space is restricted to the given domain and vanishes outside of this domain. We now need to define the polynomial formula to fit our space. Expanding the state, we arrive at two possible choices: modal vs. nodal.

$$u_h(\xi) = \sum_{m=0}^{N} \widetilde{u}_m \pi_m(\xi) = \sum_{n=0}^{N} \widehat{u}_n l_n(\xi) \tag{B.13}$$

In Equation B.13, the first representation is modal ($b_i \to \pi_i$) and the second representation is nodal ($b_i \to l_i$). Each representation is mathematically equivalent, but there are advantages and disadvantages to both. Graphically, these two representations look entirely different (see Figure B.2 for an example of $\mathbb{P}_5$).

From Figure B.2a, there are a total of six characteristic curves (a constant line plus one for each of the 5 polynomials) that combine in a modal sense to form the state for the cell. Figure B.2b gives the same $\mathbb{P}_5$ represented with 6 nodes (circles). In the nodal sense, when the basis is queried at a node, a curve is generated yielding a value of 1 at that node, and 0 at every other node.

Re-examining the modal representation,

$$u_h(\xi) = \sum_{m=0}^{N} \widetilde{u}_m \pi_m(\xi) \tag{B.14}$$

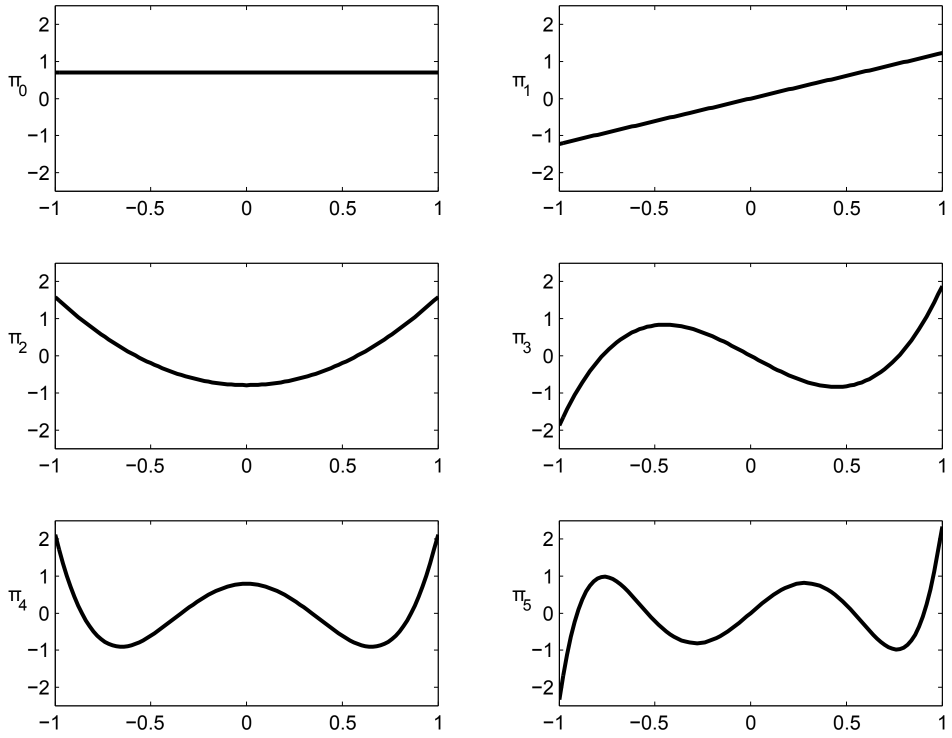
it is apparent that we can determine the modal coefficients using an L^2 projection.

$$\begin{aligned}
\int_{-1}^{1} \pi_i(\xi) u(\xi) d\xi &= \int_{-1}^{1} \pi_i(\xi) u_h(\xi) d\xi \\
&= \int_{-1}^{1} \pi(\xi) \left(\sum_{m=0}^{N} \widetilde{u}_m \pi_m(\xi) \right) d\xi \\
&= \sum_{m=0}^{N} \left(\int_{-1}^{1} \pi_i(\xi) \pi_m(\xi) d\xi \right) \widetilde{u}_m \\
&= \sum_{m=0}^{N} M_{im} \widetilde{u}_m
\end{aligned} \tag{B.15}$$

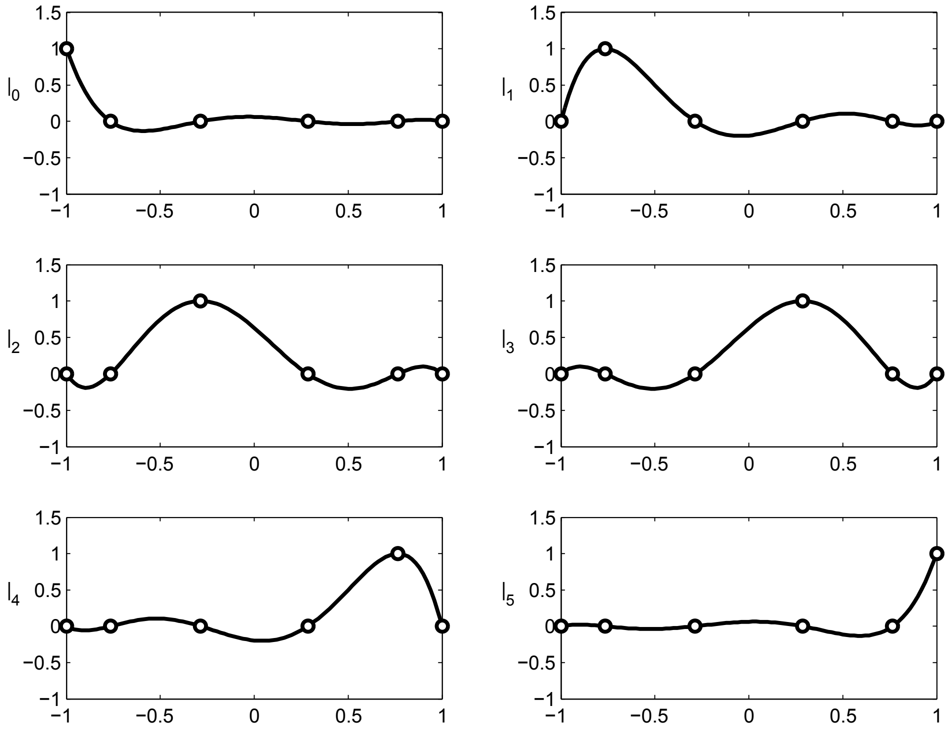
where M_{im} is the mass matrix. Because a solution will require inversion of the mass matrix, an ill-conditioned matrix will lead to numerical inaccuracy or possibly even instability. Therefore, it is important to intelligently choose the basis function to represent the state. Simply choosing any polynomial basis, say monomials, will lead to poorly conditioned matrix, leading to losses in accuracy with higher-order polynomials. Using instead something like an orthonormal basis (three-term recursion relation) will lead to a more well-conditioned matrix.

The three-term recurrence relation is expressed as

$$\sqrt{\beta_{m+1}} \pi_{m+1}(\xi) = (\xi - \alpha_m) \pi_m(\xi) - \sqrt{\beta_{m-1}} \pi_{m-1}(\xi) \tag{B.16}$$



(a) Modal Basis



(b) Nodal Basis

Figure B.2: Modal versus Nodal Graphical Representation

where $m = 0, ..., N-1$, and

$$\pi_{-1} \doteq 0, \quad \pi_0 \doteq \frac{1}{\sqrt{\beta_0}}$$

In these recurrence relations, α and β provide special subclasses of recurrence polynomials such as Chebychev and Legendre. For Legendre polynomials these coefficients are

$$\alpha_m = 0, \quad \beta_m = \begin{cases} 2 & \text{if } m = 0 \\ 1/(4 - m_{-2}) & \text{if } m \geq 1 \end{cases} \tag{B.17}$$

Looking at the nodal representation of Equation B.13,

$$u_h(\xi) = \sum_{n=0}^{N} \widehat{u}_n l_n(\xi) \tag{B.18}$$

we can use Lagrange interpolating polynomials through a given set of nodal locations, ξ_i.

$$l_n(\xi) = \prod_{\substack{i=0 \\ i \neq n}}^{N} \frac{\xi - \xi_i}{\xi_n - \xi_i} \tag{B.19}$$

Evaluating the Legendre polynomials at the same nodal locations that we set for these Lagrange polynomials yields the Vandermonde matrix, which, as will be shown, forms the foundation for the operators required for use in DG. The use of Lagrange polynomials and Legendre polynomials offer some powerful properties. Lagrange interpolating polynomials are non-heirarchical, meaning that each polynomial is of degree N. This leads to the property demonstrated in Figure B.2b. On the other hand, Legendre polynomials are heirarchical, where each polynomial is of 1 degree higher than the previous polynomial degree, shown in Figure B.2a. Lagrange polynomials are convenient for boundary data extraction while Legendre polynomials provide an accurate way to perform integration.

$$V_{ij} \doteq \pi_j(\xi_i) \tag{B.20}$$

The Vandermonde matrix also offers a convenient way to switch back and forth between modal and nodal representation. However, one important observation to make from Equation B.21 is that the Vandermonde matrix must well conditioned enough to be invertible without penalty. Because we have chosen to use orthonormal basis, the conditioning of the Vandermonde matrix is entirely dependent on nodal locations. This leads into a discussion on quadrature.

$$\vec{u}_h \doteq u_h(\vec{\xi}) = V\widetilde{u} = \widehat{u} \tag{B.21}$$

Quadrature is a weighted sum of function values at predetermined points within a standard domain. Numerous quadrature rules exist for approximating the exact integral of a function. In general, this approximation would look like

$$\int_{-1}^{1} f(\xi)d\xi \simeq \sum_{j=1}^{Np} \omega_j f(\xi_j) \tag{B.22}$$

where ω is the quadrature weights, and ξ is the quadrature nodal locations. For Gaussian quadrature, nodal locations will fall in $f \in \mathbb{P}_{2N_p-1}$ with $-1 < \xi_{1\to N_p} < 1$. This quadrature rule, while well conditioned, exlcudes the endpoints of the standard interval, so we choose instead to use Gauss-Lobatto quadrature, nodal locations will fall in $f \in \mathbb{P}_{2N_p-3}$ with $\xi_1 = -1$ and $\xi_{N_p+1} = 1$. Mathematically, this is expressed as Equation B.23. A sample of Gauss-Lobatto quadrature cells are shown in Figure B.3 for varying degree polynomials.

$$\int_{-1}^{1} f(\xi)d\xi = \omega_0 f(\xi_0) + \sum_{i=1}^{N_p} \omega_i f(\xi_i) + \omega_{N_p+1} f(\xi_{N_p+1}) \tag{B.23}$$

Looking back now at the original problem to solve,

$$\frac{\partial u}{\partial t} + \frac{\partial f(u)}{\partial x} = 0 \tag{B.24}$$

and rewriting in descretized form, we find

$$\int_{D^k} b_i^k \frac{\partial u_h^k}{\partial t} dx - \int_{D^k} \frac{\partial b_i^k}{\partial k} f_h^k dx + \oint_{\partial D^k} b_i^k f^* \left(u_h^{k,-}, u_h^{k,+}; n_x^{k,-} \right) ds = 0 \tag{B.25}$$

In this form, it is apparent that discontinuous Galerkin is a local weighted residual statement for each element based on the standard element. In order for this form to be useful, a transformation from the standard element to the physical element is required. This can be achieved using the following set of equations.

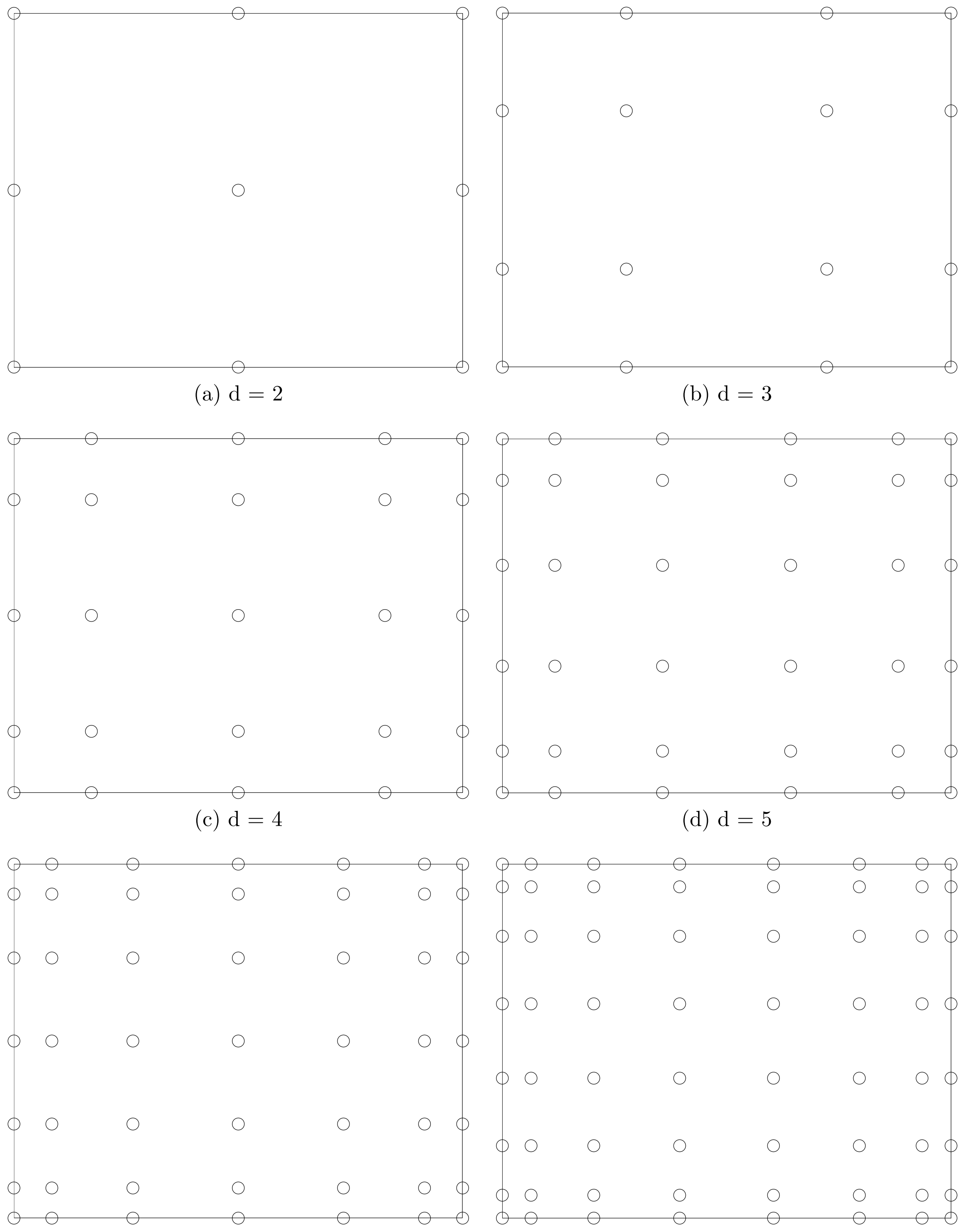
$$x \in D^k :: \;\; x = \mathcal{M}^k(\xi) \tag{B.26}$$

$$\mathcal{M}^k(\xi) = x_a^k + \frac{1+\xi}{2} \left(x_b^k - x_a^k \right) \tag{B.27}$$

$$x \in D^k :: \;\; dx = \mathcal{J}^k d\xi, \quad \mathcal{J}^k = \frac{x_b^k - x_a^k}{2} \tag{B.28}$$

where $\mathcal{J}^k$ is the Jacobian of the transformation. Using the coordinate transformation, the state decomposition can be rewritten as

$$u_h^k(x) = u_h \left(\mathcal{M}^k(\xi) \right) = \sum_{j=0}^{N} \widetilde{u}_j^k(t) b_j(\xi) \tag{B.29}$$



(a) d = 2 (b) d = 3 (c) d = 4 (d) d = 5 (e) d = 6 (f) d = 7

Figure B.3: Lobatto quadrature for varying degrees of freedom

It is also assumed that the flux can be decomposed in the same manner. Now, re-examining Equation B.25, the time derivative term can be rewritten as

$$\begin{aligned}
\int_{D^k} b_i^k \frac{\partial u_n^k}{\partial t} dx &= \int_{D^k} b_i^k(x) \frac{\partial}{\partial t} \left(\sum_{j=0}^{N} \widetilde{u}_j^k(t) b_j^k(x) \right) dx \\
&= \sum_{j=0}^{N} \left(\int_{D^k} b_i^k(x) b_j^k(x) dx \right) \frac{d\widetilde{u}_j^k}{dt} \\
&= \sum_{j=0}^{N} \left(\int_{-1}^{1} b_i(\xi) b_j(\xi) \mathcal{J}^k d\xi \right) \frac{d\widetilde{u}_j^k}{dt} \\
&= \mathcal{J}^k \sum_{j=0}^{N} M_{ij} \frac{d\widetilde{u}_j^k}{dt} \qquad \text{(B.30)}
\end{aligned}$$

The mass matrix, M_{ij}, is an operator that can determined before time evolution begins rather than at each iteration. Following a similar process as the time derivative, the spatial derivative term can be rewritten as

$$\int_{D^k} \frac{db_i^k}{dx} = \sum_{j=0}^{N} \left(S^T\right)_{ij} \widetilde{f}_j^k \qquad \text{(B.31)}$$

where S^T is the stiffness matrix and is defined as

$$S_{ij} \doteq \int_{-1}^{1} b_i(\xi) \frac{db_j(\xi)}{d\xi} d\xi \qquad \text{(B.32)}$$

The numerical flux term in Equation B.25 must be handled significantly different. Because the setup thus far has allowed for discontinuities at the boundaries of cells, some numerical flux function must be used in order to interpret jumps at the boundaries. Writing this flux as an expansion

$$f^*(s) = f^* \left(u_h^{k,-}, u_h^{k,+}; n_x^{k,-} \right)_s = \sum_{j=0}^{N} \widetilde{f^*_s}_j^k b_j(s) \qquad \text{(B.33)}$$

It can be shown that using this expansion, the cell boundary term can be written as

$$\oint_{\partial D^k} b_i^k f^* \left(u_h^{k,-}, u_h^{k,+}; n_x^{k,-} \right) ds = \sum_{m=0}^{N} \left(L_a\right)_{im} \widetilde{f^*_a}_m^k \sum_{n=0}^{N} \left(L_b\right)_{im} \widetilde{f^*_b}_n^k \qquad \text{(B.34)}$$

where L_a and L_b are lifting matrices. The lifting matrices are called so because they lift lower dimensional boundary information to higher dimensional element data. So in matrix vector form, Equation B.25 can be written as

$$\mathcal{J}^k M \frac{d\widetilde{u}^k}{dt} - S^T \widetilde{f}^k + L_a \widetilde{f_a^*}^k + L_b \widetilde{f_b^*}^k = 0 \qquad \text{(B.35)}$$

The matrix vector form (element operator form) given above is the useful form of DG. For more information regarding formulation of the operator matrices and time evolution using DG, refer to [57, 56, 58, 59, 60].

Appendix C

Comparison of First-Order Finite Element to Discontinuous Galerkin: A Classic Fluid Dynamics Example

The two-dimensional isentropic vortex problem is often posed in comparing numerical methods because it is well behaved and smooth and the exact solution is known. The isentropic vortex is created by introducing perturbations to the velocity and temperature in a uniform flow field. The exact solution to the convection of a vortex is such that the vortex strength will be maintained but will move with a bulk velocity. The given freestream conditions for this problem are as follows:

$$u_\infty = M_\infty \cos\alpha \tag{C.1}$$
$$v_\infty = M_\infty \sin\alpha \tag{C.2}$$
$$p_\infty = \frac{1}{\gamma} \tag{C.3}$$
$$T_\infty = \frac{1}{\gamma} \tag{C.4}$$

For the problem given here, the freestream Mach number was set to 0.5 and the specific heat ratio was set to 1.4. This results in a freestream density of 1 and an acoustic speed of 1. The perturbations to develop the initial vortex are proportional (in a way) to the distance from the vortex origin (set at the domain origin for simplicity). Perturbations are then calculated as follows:

$$\delta u = \delta y f \tag{C.5}$$
$$\delta v = +\delta x f \tag{C.6}$$
$$\delta T = -\frac{\gamma - 1}{2\gamma} f^2 \tag{C.7}$$

where

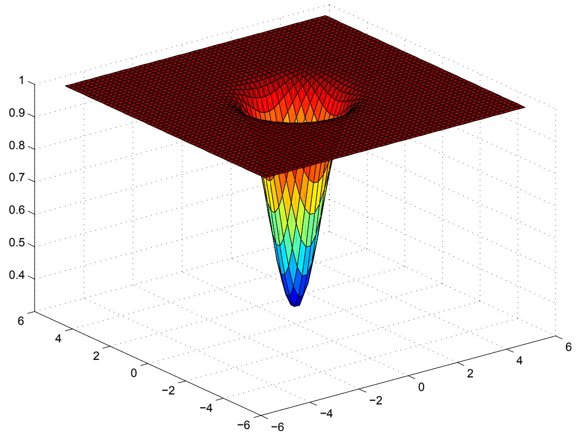
$$(\delta x, \delta y) = (x - x_o, y - y_o) \tag{C.8}$$

$$f = \frac{b}{2\pi} \exp\left[a(1 - r^2)\right] \tag{C.9}$$

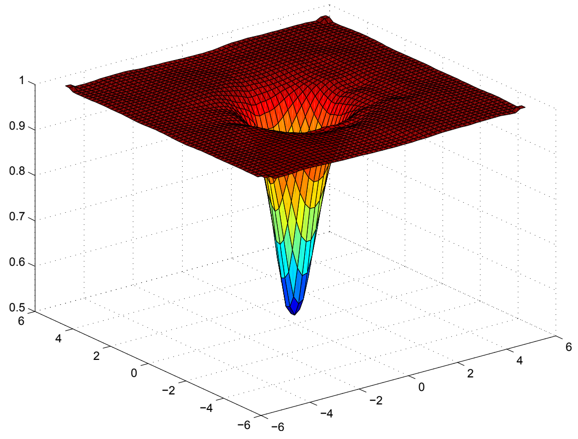
$$r^2 = \delta x^2 + \delta y^2 \tag{C.10}$$

The vortex strength parameters were set as $a = 0.5$, $b = 5$. The computational mesh set on the domain $(x, y) \in (-5, 5) \times (-5, 5)$. Using finite element method, the mesh size was set to 50×50, while for DG, the mesh size requirement was only 4×4.

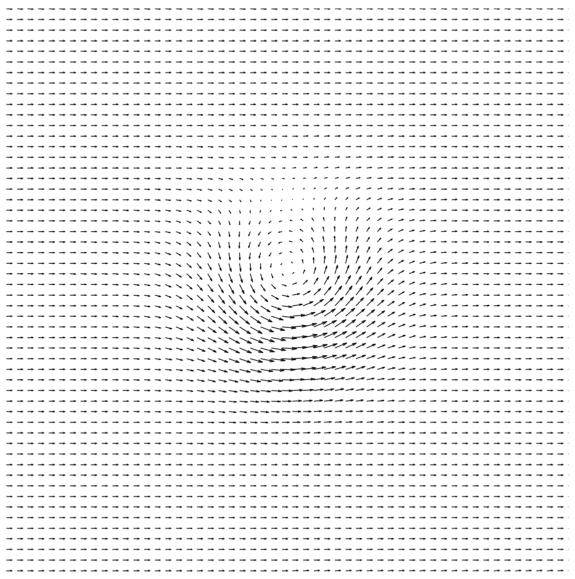
The vortex was set to convect, and the boundary conditions were set as periodic so that as the vortex leaves the domain on say the left boundary, it re-enters the domain on the right boundary. With the domain and Mach number set as specified, and a time span of 20 seconds, the final location of the center of the vortex should be exactly the initial location. Results for the finite element method are shown in Figure C.1.



(a) Initial Density



(b) Final Density


(c) Initial Velocity Field

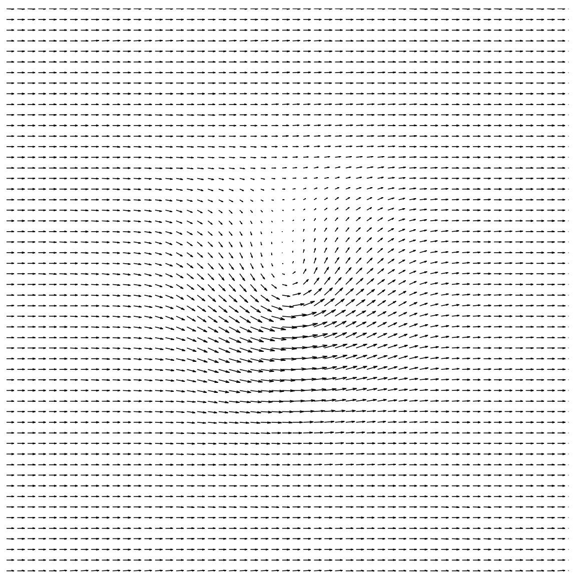
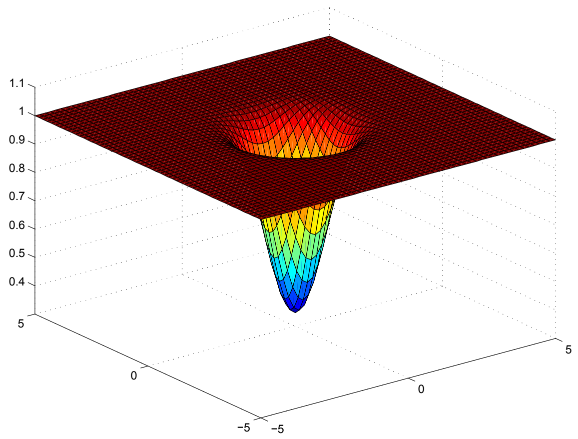
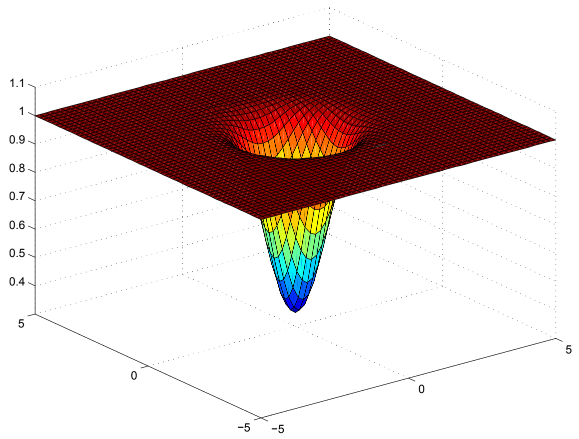

(d) Final Velocity Field

Figure C.1: Euler Solution to Isentropic Vortex using Finite Element

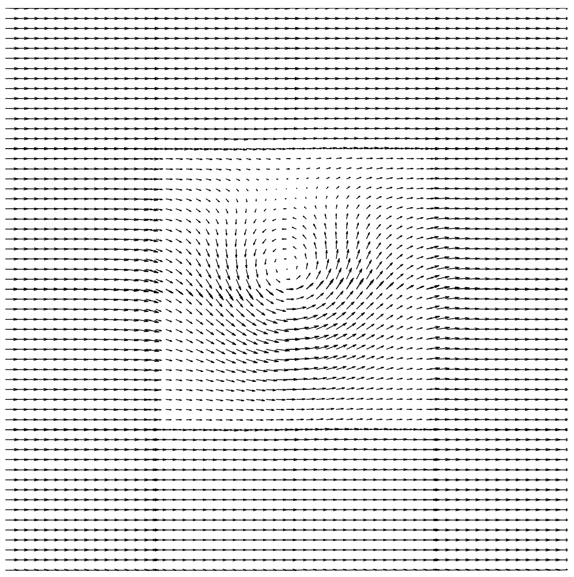
From Figures C.1a and C.1b, it is apparent that the first order scheme for time evolution was unable to fully resolve the vortex after 20 seconds. Looking at the scale on the z-axis, the vortex has dissipated (non-physical) and there appear to be slight wrinkles in the solution for the density. From the velocity fields (Figures C.1c and C.1d), it appears as though some of the velocity has also dissipated. Looking ahead at the results for DG, it is apparent that the features of the flow were maintained accurately with little to no dissipation (Figure C.2). It should be noted that in both Figures C.2c and C.2d, the solution appears different from those in Figure C.1. The artifacts present are a function of plotting only. The important note to take away from the results is the similarities between the initial and final states.



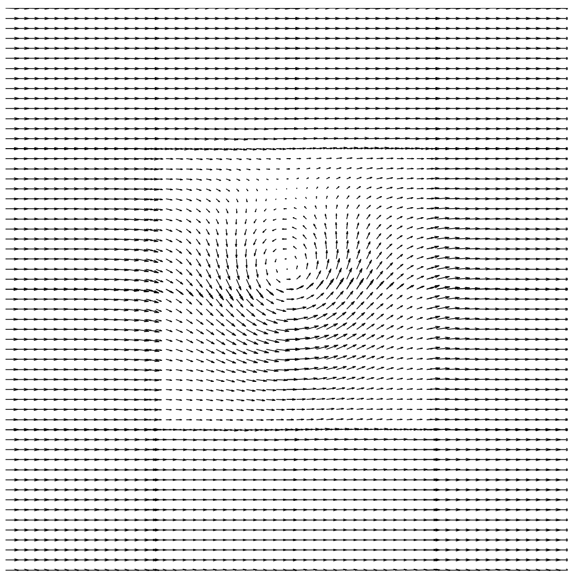
(a) Initial Density



(b) Final Density



(c) Initial Velocity Field



(d) Final Velocity Field

Figure C.2: Euler Solution to Isentropic Vortex using Discontinuous Galerkin

Appendix D

DG/LSM Fortran Source

Main Evolution Source

```
program main

INTEGER ::    NI,NJ,d,Np,Np2,steps,kk,i,j,k,itmax,kchek,islow,output,n_stage
INTEGER,ALLOCATABLE,DIMENSION(:)::East,West,North,South
INTEGER,ALLOCATABLE,DIMENSION(:)::Ekx,Wkx,Nky,Sky
INTEGER,ALLOCATABLE,DIMENSION(:,:)::Kx,KY
REAL ::  epsilon,staring,ending,start_time,end_time,pi,burnstop,res_sum,res1,lambda
REAL ::  CFL,dx,dy,ex,ey,Hmax,Hmin,Jx,Jy,Jk,dstep,y,xmax,xmin,ymax,ymin,lref,gamma,alpha(3,3)
REAL,ALLOCATABLE,DIMENSION(:)::ww,www,p1,p2,q1,q2,delta,umode,Hmode,ypos,area,rhs,res,cmode
REAL,ALLOCATABLE,DIMENSION(:,:)::Sx,Sy,V,Bnm
REAL,ALLOCATABLE,DIMENSION(:,:,:)::Lx,Ly,phx,phy,Ham,Hea,caseheavi
REAL,ALLOCATABLE,DIMENSION(:,:,:)::u,uold
CHARACTER*20  ::  DIRNAME,fname


if (islow.eq.1) call system('cleanse') !clean output folder

!read in input settings
open(unit=11,file='inputsettings.txt')
read(11,*)NI,NJ
read(11,*)d
read(11,*)epsilon
read(11,*)CFL
read(11,*)xmin,ymin
read(11,*)xmax,ymax
read(11,*)ex,ey
read(11,*)Hmax,Hmin
read(11,*)lref,gamma
read(11,*)burnstop,itmax
read(11,*)islow,output
close(11)

!determine necessary domain settings
dx   = (xmax-xmin)/NI
dy   = (ymax-ymin)/NJ
lref=lref*min(dx,dy)/d**2.0
Np   = d+1;
Np2  = 4*Np-2;

Jx   = 0.5*dy;
Jy   = 0.5*dx;
Jk   = .25*dx*dy;
lambda=1.0+gamma
dstep = CFL/(lambda*d**2./min(dx,dy))
epsilon=epsilon*min(dx,dy)/(d+1)

steps = int(min(dx*NI,dy*NJ)/dstep)
pi=acos(-1.0);


!*****************************************
!       ALLOCATE MEMORY FOR MATRICES
```

```
ALLOCATE(ypos(steps));ypos=0;
ALLOCATE(area(steps));area=0;
ALLOCATE(res(steps));res=0;
ALLOCATE(Sx(Np**2,Np**2));Sx=0;
ALLOCATE(Sy(Np**2,Np**2));Sy=0;
ALLOCATE(Kx(Np,2));Kx=0;
ALLOCATE(Ky(Np,2));Ky=0;
ALLOCATE(Lx(Np**2,Np,2));Lx=0;
ALLOCATE(Ly(Np**2,Np,2));Ly=0;
ALLOCATE(Bnm(Np2**2,Np**2));Bnm=0;
ALLOCATE(ww(Np2**2));ww=0;
ALLOCATE(www(Np**2));www=0;
ALLOCATE(u(Np**2,NI,NJ));u=0;
ALLOCATE(uold(Np**2,NI,NJ));uold=u;
ALLOCATE(caseheavi(Np**2,NI,NJ));caseheavi=0;
ALLOCATE(umode(Np2**2));umode=0;
ALLOCATE(Hmode(Np2**2));Hmode=0;
ALLOCATE(cmode(Np2**2));cmode=0;
ALLOCATE(delta(Np2**2));delta=0;
ALLOCATE(East(NI));East=0;
ALLOCATE(West(NI));West=0;
ALLOCATE(North(NJ));North=0;
ALLOCATE(South(NJ));South=0;
ALLOCATE(Ekx(NI));Ekx=1;
ALLOCATE(Wkx(NI));Wkx=2;
ALLOCATE(Nky(NJ));Nky=1;
ALLOCATE(Sky(NJ));Sky=2;
ALLOCATE(p1(Np**2));p1=0;
ALLOCATE(p2(Np**2));p2=0;
ALLOCATE(q1(Np**2));q1=0;
ALLOCATE(q2(Np**2));q2=0;
ALLOCATE(phx(Np**2,NI,NJ));phx=0;
ALLOCATE(phy(Np**2,NI,NJ));phy=0;
ALLOCATE(Ham(Np**2,NI,NJ));Ham=0;
ALLOCATE(Hea(Np**2,NI,NJ));Hea=0;
ALLOCATE(rhs(Np**2));rhs=0;

!*******************************************

if (Np.lt.10) write(DIRNAME,91) 'DG_',Np
if (Np.ge.10) write(DIRNAME,92) 'DG_',Np
call chdir(DIRNAME)


!read in stiffness matrix
open(unit=22,file='Stiffness.dat')
do i=1,Np**2
   read(22,*)(Sx(i,j),j=1,Np**2)
enddo
Sx=Jx/Jk*Sx

do i=1,Np**2
   read(22,*)(Sy(i,j),j=1,Np**2)
enddo
close(22)
Sy=Jy/Jk*Sy


!read in Lifting Matrices
open(unit=23,file='LiftingX.dat')
do k=1,2
   do i=1,Np**2
      read(23,*)(Lx(i,j,k),j=1,Np)
   enddo
enddo
close(23)
Lx=Jx/Jk*Lx
```

```
open(unit=24,file='LiftingY.dat')
do k=1,2
   do i=1,Np**2
      read(24,*)(Ly(i,j,k),j=1,Np)
   enddo
enddo
close(24)
Ly=Jy/Jk*Ly



!read in tall vandermonde matrix
open(unit=25,file='TallVandermonde.dat')
do i=1,Np2**2
   read(25,*)(Bnm(i,j),j=1,Np**2)
enddo
close(25)


!read in weights for tall vandermonde matrix
open(unit=26,file='TallWeights.dat')
do i=1,Np2**2
   read(26,*)ww(i)
enddo
close(26)

!read in weights for regular lobatto points
open(unit=26,file='LobWeights.dat')
do i=1,Np**2
   read(26,*)www(i)
enddo
close(26)



!read in initial state matrix
call chdir('..')
open(unit=27,file='initial_state.dat')
do i=1,NI
   do j=1,NJ
      read(27,*)(u(k,i,j),k=1,Np**2)
   enddo
enddo
close(27)

open(unit=28,file='case_heavi.dat')
do i=1,NI
   do j=1,NJ
      read(28,*)(caseheavi(k,i,j),k=1,Np**2)
   enddo
enddo
close(28)


! Setup north, south, east, and west boundaries and nodes
do i=1,Np
   Kx(i,1)=1+(i-1)*Np;
   Kx(i,2)=(i-1)*Np+Np;
   Ky(i,1)=i;
   Ky(i,2)=Np**2-(Np-i)
enddo
do i=1,NI
   East(i)=i+1
   West(i)=i-1
enddo
do j=1,NJ
   North(j)=j+1
```

```
    South(j)=j-1
enddo
East(NI)=NI;
West(1)=1
North(NJ)=NJ
South(1)=1
Ekx(NI)=2;
Wkx(1)=1;
Nky(NJ)=2;
Sky(1)=1;
uold=u;

!set up alpha for staging RK-TVD
n_stage=3;
alpha(1,:)=(/ 1.0  , 0.75  , 1.0/3.0 /)
alpha(2,:)=(/ 0.0  , 0.25  , 2.0/3.0 /)
alpha(3,:)=(/ 1.0  , 0.25  , 2.0/3.0 /)
call cpu_time(start_time) !wallclock
y=0;
kk=1;
kchek=10;
!begin time evolution
do while (dstep*kk.le.burnstop)
    res_sum=0;
    !begin stage integration
    do ijk=1,n_stage

! perform single time evolution here
    do j=1,NJ
        do i=1,NI
            ! first derivatives
            p1=-matmul(Sx,u(:,i,j))+matmul(Lx(:,:,2),u(Kx(:,Ekx(i)),East(i),j))&
                 &                    -matmul(Lx(:,:,1),u(Kx(:,1),i,j));
            p2=-matmul(Sx,u(:,i,j))+matmul(Lx(:,:,2),u(Kx(:,2),i,j))&
                 &                    -matmul(Lx(:,:,1),u(Kx(:,Wkx(i)),West(i),j));

            q1=-matmul(Sy,u(:,i,j))+matmul(Ly(:,:,2),u(Ky(:,Nky(j)),i,North(j)))&
                 &                    -matmul(Ly(:,:,1),u(Ky(:,1),i,j));
            q2=-matmul(Sy,u(:,i,j))+matmul(Ly(:,:,2),u(Ky(:,2),i,j))&
                 &                    -matmul(Ly(:,:,1),u(Ky(:,Sky(j)),i,South(j)));

            p1=p1;p2=p2
            q1=q1;q2=q2


            phx(:,i,j)=0.5*(p1+p2);
            phy(:,i,j)=0.5*(q1+q2);


            !form Hamiltonian
            Ham(:,i,j)=(phx(:,i,j)**2.+phy(:,i,j)**2.)**0.5;
            Ham(:,i,j)=Ham(:,i,j)+gamma*u(:,i,j)/sqrt(u(:,i,j)**2.+lref**2.)*(Ham(:,i,j)-1.);
            Ham(:,i,j)=Ham(:,i,j)-0.5*(p1-p2)-0.5*(q1-q2);
           !calculate area
            if (ijk.eq.1) then
                umode=matmul(Bnm,u(:,i,j));
                Hmode=matmul(Bnm,Ham(:,i,j));
                cmode=matmul(Bnm,caseheavi(:,i,j));



                delta=0;
                do k=1,Np2**2
                    if (abs(umode(k)).le.epsilon) delta(k)=0.5/epsilon+0.5/epsilon&
&*cos(pi*umode(k)/epsilon);
                enddo
                area(kk)=area(kk)+Jk*sum(cmode*Hmode*delta*ww)
            endif
        enddo
    enddo

    do j=1,NJ
        do i=1,NI
            ! second derivatives
            rhs=-Ham(:,i,j)
            if (any(abs(phx(:,i,j)).gt.Hmax))then
                p1=matmul(Lx(:,:,2),0.5*(phx(Kx(:,Ekx(i)),East(i),j)+phx(Kx(:,2),i,j)))&
                & -matmul(Lx(:,:,1),0.5*(phx(Kx(:,1),i,j)+phx(Kx(:,Wkx(i)),West(i),j)));
                p1=(p1-matmul(Sx,phx(:,i,j)));
                rhs=rhs+ex*p1;
            elseif (any(abs(phx(:,i,j)).lt.Hmin))then
                p1=matmul(Lx(:,:,2),0.5*(phx(Kx(:,Ekx(i)),East(i),j)+phx(Kx(:,2),i,j)))&
                & -matmul(Lx(:,:,1),0.5*(phx(Kx(:,1),i,j)+phx(Kx(:,Wkx(i)),West(i),j)));
                p1=(p1-matmul(Sx,phx(:,i,j)));
                rhs=rhs+ex*p1;
            endif

            if (any(abs(phy(:,i,j)).gt.Hmax))then
                q1=matmul(Ly(:,:,2),0.5*(phy(Ky(:,Nky(j)),i,North(j))+phy(Ky(:,2),i,j)))&
                & -matmul(Ly(:,:,1),0.5*(phy(Ky(:,1),i,j)+phy(Ky(:,Sky(j)),i,South(j))));
                q1=(q1-matmul(Sy,phy(:,i,j)));
                rhs=rhs+ey*q1
            elseif (any(abs(phy(:,i,j)).lt.Hmin))then
                q1=matmul(Ly(:,:,2),0.5*(phy(Ky(:,Nky(j)),i,North(j))+phy(Ky(:,2),i,j)))&
                & -matmul(Ly(:,:,1),0.5*(phy(Ky(:,1),i,j)+phy(Ky(:,Sky(j)),i,South(j))));
                q1=(q1-matmul(Sy,phy(:,i,j)));
                rhs=rhs+ey*q1
            endif
            !calculate residual
            if (ijk.eq.3) res_sum=sqrt(res_sum**2+sum(rhs*www*rhs));
            u(:,i,j)=alpha(1,ijk)*uold(:,i,j)+alpha(2,ijk)*u(:,i,j)+alpha(3,ijk)*dstep*rhs

        enddo
    enddo



    enddo
    call cpu_time(end_time)

! end single time evolution, perform checks, write output if necessary
if (islow.eq.1) then
    if (mod(kk,output).eq.0)then
        call chdir('output')
        write(fname,94)'grain_output',kk,'.dat'
94  format(a12,i4,a4)
        open(unit=41,file=fname)
        write(41,*)NI,NJ,d
        write(41,*)xmin,xmax,ymin,ymax
        do j=1,NJ
            do i=1,NI
                write(41,*)(uold(k,i,j),k=1,Np**2)

            enddo
        enddo
        close(41)
        call chdir('..')
    endif
endif
if (kk.eq.1)then
    res1=res_sum
endif
res(kk)=res_sum/res1;
ypos(kk)=kk*dstep;
if (kk.eq.kchek)then
    write(*,*)kk,ypos(kk),res(kk)
```

```
   kchek=kchek+50;
endif
if (res(kk).gt.2)goto 51
kk=kk+1;
uold=u;

enddo
51 write(*,*)'Number of Steps: ',kk-1
write(*,*)'Burn Distance  : ',ypos(kk-1)
write(*,*)'Burn Area      : ',area(kk-1)
write(*,*)'Residual       : ',res(kk-1)

open(unit=81,file='residual.dat')
write(81,*)'Variables = "t","residual","area"'
do i=1,kk-1
   write(81,*)ypos(i),res(i),area(i)
enddo
close(81)



write(*,93)'Run Time = ',end_time-start_time,' seconds'


!write out final state matrix


open(unit=31,file='final_state.dat')
do i=1,NI
   do j=1,NJ
      write(31,*)(u(k,i,j),k=1,Np**2)
   enddo
enddo
close(31)
open(unit=31,file='postprocess_specs.dat')
write(31,*)NI,NJ,d,xmin,xmax,ymin,ymax
close(31)

91 FORMAT(a3,i1)
92 FORMAT(a3,i2)
93 FORMAT(a11,1x,f14.8,1x,a8)
99 FORMAT(e16.8,2x,e16.8,2x,e16.8)
END
```

Initialization Source

```
program shell
!shell program used for setting up and running level set propagation program

INTEGER :: NI,NJ,d,Np,Np2,i,j,k,Ns
REAL :: epsilon_case,xmin,xmax,ymin,ymax,dump,xs,ys,dx,dy,ro,ri,lambda
REAL :: beta,pi,fillet,eps_ang
REAL,ALLOCATABLE,DIMENSION(:)::qx,qy,px,py,xi,yi
REAL,ALLOCATABLE,DIMENSION(:,:)::xdomain,ydomain
REAL,ALLOCATABLE,DIMENSION(:,:,:)::phi,heavi,ci
CHARACTER*20 :: gtype,ctype

!phi is for the grain
!heavi is for the case

CHARACTER*20 :: DIRNAME

interface
   function setup_grain(xs,ys,dx,dy,listx,listy,Np,Ns,gtype,xc,yc,ci)&
```

```
         result(dist)
      real,dimension(Np**2)::listx,listy,dist
      real,dimension(Ns,2,4)::ci
      real :: xs,ys,dx,dy,xc(Ns*6+1),yc(Ns*6+1)
      integer :: Np,Ns
      character*20 :: gtype
   end function setup_grain
end interface
interface
   function setup_case(xs,ys,dx,dy,listx,listy,Np,ro,ctype,eps)&
         result(heav)
      real,dimension(Np**2) :: listx,listy,heav
      real :: xs,ys,dx,dy,x,y,ro,eps
      character*20 :: ctype
   end function setup_case
end interface

open(unit=11,file='inputsettings.txt')
read(11,*)NI,NJ
read(11,*)d
read(11,*)dump,epsilon_case
read(11,*)dump
read(11,*)xmin,ymin
read(11,*)xmax,ymax
close(11)

open(unit=12,file='design.txt')
read(12,*)ro
read(12,*)rp
read(12,*)ri
read(12,*)lambda
read(12,*)Ns
read(12,*)fillet
read(12,*)eps_ang
read(12,*)gtype
read(12,*)ctype

dx=(xmax-xmin)/NI
dy=(ymax-ymin)/NJ
Np=d+1
Np2=2*Np-2
epsilon_case=epsilon_case*min(dx,dy)/(d+1.0)
pi=acos(-1.0)

!ALLOCATE MEMORY FOR STATE AND CASE MATRICES

ALLOCATE(xdomain(NI+1,NJ+1));xdomain=0;
ALLOCATE(ydomain(NI+1,NJ+1));ydomain=0;
ALLOCATE(phi(Np**2,NI,NJ));phi=0;
ALLOCATE(heavi(Np**2,NI,NJ));heavi=0;
ALLOCATE(qx(Np**2));qx=0;
ALLOCATE(qy(Np**2));qy=0;
ALLOCATE(xi(Ns*6+1));xi=0;
ALLOCATE(yi(Ns*6+1));yi=0;
ALLOCATE(ci(Ns,2,4));ci=0;
!ALLOCATE(px(Np2**2));px=0;
!ALLOCATE(py(Np2**2));py=0;

if (Np.lt.10) write(DIRNAME,91)'DG_',Np
if (Np.ge.10) write(DIRNAME,92)'DG_',Np
call chdir(DIRNAME)

open(unit=12,file='LobattoPoints.dat')
do i=1,Np**2
   read(12,*)qx(i),qy(i)
enddo
close(12)
```

```
!open(unit=13,file='GaussPoints.dat')
!do i=1,Np2**2
!    read(13,*)px(i),py(i)
!enddo
!close(13)

call chdir('..')

!set initial condition
do j=1,NJ+1
   do i=1,NI+1
      xdomain(i,j)=xmin+(i-1)*dx
      ydomain(i,j)=ymin+(j-1)*dy
   enddo
enddo

!setup initial discretized surface
if (gtype.eq.'barrere')then
   call barrere(rp,ri,fillet,eps_ang,Ns,xi,yi,ci)
elseif (gtype.eq.'sstar')then
   call hard_star(rp,ri,Ns,xi,yi)
elseif (gtype.eq.'hellfire') then
   xi(1)=ro;
   yi(1)=lambda;
endif

! setup initial state
open(unit=21,file='initial_state.dat')
open(unit=22,file='case_heavi.dat')
do i=1,NI
   do j=1,NJ
      xs=xdomain(i,j)
      ys=ydomain(i,j)
      phi(:,i,j)=setup_grain(xs,ys,dx,dy,qx,qy,Np,Ns,gtype,xi,yi,ci);
      heavi(:,i,j)=setup_case(xs,ys,dx,dy,qx,qy,Np,ro,ctype,epsilon_case);
      write(21,*)(phi(k,i,j),k=1,Np**2)
      write(22,*)(heavi(k,i,j),k=1,Np**2)
   enddo
enddo
close(21)
close(22)

91 FORMAT(a3,i1)
92 FORMAT(a3,i2)
END PROGRAM

subroutine barrere(rp,ri,f,eps,Ns,xi,yi,ci)
real :: rp,ri,f,eps,xi(6*Ns+1),yi(6*Ns+1),H1,theta2
integer :: Ns,i,j,k,ii,jj,kk
real :: x1(4,2),c1(2),c2(2),c3(2),c4(2)
real :: x(6),y(6),xr(6),yr(6),ci(Ns,2,4),ang,pi
pi=acos(-1.0)
H1=rp*sin(pi*eps/Ns)
theta2=atan(H1*tan(pi*eps/Ns)/(H1-ri*tan(pi*eps/Ns)))
x1(1,1)=(rp+f)*cos(pi/Ns);
x1(1,2)=-(rp+f)*sin(pi/Ns);
x1(2,1)=(rp+f)*cos(pi*eps/Ns);
x1(2,2)=-(rp+f)*sin(pi*eps/Ns);
x1(3,1)=rp*cos(pi*eps/Ns)+f*cos(-pi/2+theta2);
x1(3,2)=-(rp*sin(pi*eps/Ns)+f*sin(-pi/2+theta2));
x1(4,1)=ri+f*cos(theta2);
x1(4,2)=0;
c1=(/0,0/)
c2=(/rp*cos(pi*eps/Ns),-rp*sin(pi*eps/Ns)/)
c3=(/c2(1),-c2(2)/)
c4=(/c1(1),-c1(2)/)
k=1;
do j=1,4
```

```
   x(k)=x1(j,1);
   y(k)=x1(j,2);
   k=k+1
enddo
do j=3,2,-1
   x(k)=x1(j,1)
   y(k)=-x1(j,2)
   k=k+1
enddo
k=1
do ii=1,Ns
   ang = (ii -1.0)*2.0*pi/Ns
   xr=x*cos(ang)-y*sin(ang)
   yr=x*sin(ang)+y*cos(ang)
   ci(ii,1,1)=c1(1)*cos(ang)-c1(2)*sin(ang)
   ci(ii,2,1)=c1(1)*sin(ang)+c1(2)*cos(ang)
   ci(ii,1,2)=c2(1)*cos(ang)-c2(2)*sin(ang)
   ci(ii,2,2)=c2(1)*sin(ang)+c2(2)*cos(ang)
   ci(ii,1,3)=c3(1)*cos(ang)-c3(2)*sin(ang)
   ci(ii,2,3)=c3(1)*sin(ang)+c3(2)*cos(ang)
   ci(ii,1,4)=c4(1)*cos(ang)-c4(2)*sin(ang)
   ci(ii,2,4)=c4(1)*sin(ang)+c4(2)*cos(ang)
   do jj=1,6
      xi(k)=xr(jj)
      yi(k)=yr(jj)
      k=k+1;
   enddo
enddo
xi(k)=x(1)
yi(k)=y(1)
return
end subroutine

subroutine hard_star(rp,ri,Ns,xi,yi)
real :: rp,ri,xi(6*Ns+1),yi(6*Ns+1),beta,dbeta
integer :: Ns,i,j,k
pi=acos(-1.0)
xi(1)=rp;yi(1)=0
k=2;
beta=0;
dbeta=pi/Ns
do i=1,Ns
   beta=beta+dbeta

   xi(k)=ri*cos(beta);yi(k)=ri*sin(beta)
   k=k+1

   beta=beta+dbeta

   xi(k)=rp*cos(beta);yi(k)=rp*sin(beta)
   k=k+1

enddo
return
end subroutine
```

```
function setup_grain(xs,ys,dx,dy,listx,listy,Np,Ns,gtype,xc,yc,ci)&
     result(dist)

real,dimension(Np**2) :: listx,listy,dist
real,dimension(Ns,2,4) :: ci
real :: xs,ys,dx,dy,x,y,ro,ri,lambda,xc(Ns*6+1),yc(Ns*6+1)
integer :: Np,Ns
integer :: i,j,k
character*20 :: gtype
interface
   function straight_star(x,y,Ns,xc,yc)result(phi)
```

```
      real :: x,y,phi,xc(300),yc(300)
      integer :: Ns
   end function straight_star
end interface

do i=1,Np**2
   x=xs+0.5*(listx(i)+1.0)*dx
   y=ys+0.5*(listy(i)+1.0)*dy
   if (gtype.eq.'sstar') then
      dist(i)=straight_star(x,y,Ns,xc,yc)
   elseif (gtype.eq.'barrere') then
      dist(i)=dist_to_barrere(x,y,Ns,xc,yc,ci)
   elseif (gtype.eq.'hellfire') then
      dist(i)=dist_to_hellfire(x,y,xc(1),yc(1))
   endif
enddo


return
end function

function dist_to_hellfire(x,y,ro,lambda)result(phi)
integer i,j,k
real :: x,y,ro,lambda,r

r=sqrt(x**2.0+y**2.0)

if (r.gt.0.5*ro)then
   phi=r-0.5*ro-0.5*lambda
else
   phi=(1-r)-0.5*ro-0.5*lambda
endif

end function


function dist_to_barrere(x,y,Ns,xc,yc,ci)result(phi)
integer :: Ns,i,j,k
real :: x,y,phi,xc(Ns*6+1),yc(Ns*6+1),ci(Ns,2,4)
real :: p0(2),p1(2),p2(2),p(2),c(2),d(Ns*6),s(Ns*6),dmin,sig

p=(/x,y/)
k=1
do i=1,Ns
   p0=(/ci(i,1,1),ci(i,2,1)/)
   p1=(/xc(k),yc(k)/)
   p2=(/xc(k+1),yc(k+1)/)
   call dist_to_arc(p,p0,p1,p2,d(k),s(k))
   k=k+1

   p0=(/ci(i,1,2),ci(i,2,2)/)
   p1=(/xc(k),yc(k)/)
   p2=(/xc(k+1),yc(k+1)/)
   call dist_to_arc(p,p0,p1,p2,d(k),s(k))
   k=k+1

   p1=(/xc(k),yc(k)/)
   p2=(/xc(k+1),yc(k+1)/)
   call dist_to_seg(p,p1,p2,d(k),s(k))
   k=k+1

   p1=(/xc(k),yc(k)/)
   p2=(/xc(k+1),yc(k+1)/)
   call dist_to_seg(p,p1,p2,d(k),s(k))
   k=k+1

   p0=(/ci(i,1,3),ci(i,2,3)/)
   p1=(/xc(k),yc(k)/)
   p2=(/xc(k+1),yc(k+1)/)
   call dist_to_arc(p,p0,p1,p2,d(k),s(k))
   k=k+1

   p0=(/ci(i,1,4),ci(i,2,4)/)
   p1=(/xc(k),yc(k)/)
   p2=(/xc(k+1),yc(k+1)/)
   call dist_to_arc(p,p0,p1,p2,d(k),s(k))
   k=k+1
enddo

dmin=minval(d);
where(d /= dmin)
   s=0.0
end where
sig=sum(s)
phi=dmin*sgn(sig)


end function

function straight_star(x,y,Ns,xc,yc)result(phi)
integer :: Ns
real :: x,y,phi,ro,ri,lambda,xc(300),yc(300)
integer :: i,j,k
real :: x0(2),x1(2),x2(2),d(2*Ns),s(2*Ns),dmin
!interface
!   subroutine dist_to_seg(p,p0,p1,d,s)
!      real,intent(in)::p(2),p0(2),p1(2)
!      real,intent(out)::d,s
!  end subroutine
!end interface

x0=(/x,y/)
do i=1,2*Ns
   x1=(/xc(i), yc(i)/)
   x2=(/xc(i+1), yc(i+1)/)
   call dist_to_seg(x0,x1,x2,d(i),s(i))
enddo
dmin=minval(d);
where(d /= dmin)
   s=0.0
end where
sig=sum(s)
phi=dmin*sgn(sig)

return
end function

!-------------------------------------------------------------------------------

function setup_case(xs,ys,dx,dy,listx,listy,Np,ro,ctype,eps)&
     result(heav)

real,dimension(Np**2) :: listx,listy,heav
real :: xs,ys,dx,dy,x,y,ro,eps
integer :: Np
integer :: i,j,k
character*20 ::  ctype
interface
   function circle(x,y,ro,eps)result(heav)
     real :: x,y,heav,ro,eps
   end function circle
end interface
interface
   function square(x,y,ro,eps)result(heav)
     real :: x,y,heav,ro,eps
   end function square
end interface

do i=1,Np**2
   x=xs+0.5*(listx(i)+1.0)*dx
   y=ys+0.5*(listy(i)+1.0)*dy
   if (ctype.eq.'circle') then
      heav(i)=circle(x,y,ro,eps)
   elseif (ctype.eq.'square')then
      heav(i)=square(x,y,ro,eps)
   else
      write(*,*)'Check case type: ',ctype
   endif
enddo


return
end function

function circle(x,y,ro,eps)result(heav)

real :: x,y,heav,ro,phi,eps,pi
pi=acos(-1.0)
phi=-(sqrt(x**2.0+y**2.0)-ro)
if (phi.gt.0.0)then
   heav=1.0
elseif (phi.gt.-eps)then
   heav=0.5*(1+cos(phi*pi/eps))
else
   heav=0.0;
endif
return
end function

function square(x,y,ro,eps)result(heav)
real :: x,y,ro,eps,x1(5),x2(5),p(2),p1(2),p2(2)
integer :: i,j,k
real :: d(4),s(4),pi,sig,phi,dmin
pi=acos(-1.0)
x1=(/-1, 1, 1, -1, -1/)*ro
x2=(/-1, -1, 1, 1, -1/)*ro
p=(/x,y/)
do i=1,4
   p1=(/x1(i), x2(i)/)
   p2=(/x1(i+1), x2(i+1)/)
   call dist_to_seg(p,p1,p2,d(i),s(i))
enddo

dmin=minval(d);
where(d /= dmin)
   s=0.0
end where
sig=sum(s)
phi=-dmin*sgn(sig)


if (phi.gt.0.0)then
   heav=1.0
elseif (phi.gt.-eps)then
   heav=(1+cos(phi*pi/eps))
else
   heav=0.0;
endif



return
end function
```

```
!------------------------------------------------------------------------------
subroutine dist_to_arc(p,p0,p1,p2,d,s)
real,intent(in)::p(2),p0(2),p1(2),p2(2)
real,intent(out)::d,s
real,dimension(2)::u1,u2,v,w1,w2
real :: pi,d1,d2,s1,s2,q1,q2,q
pi=acos(-1.0)
u1=p1-p0
u2=p2-p0
v=p-p0

q1=angle_on_circle(u1);
q2=angle_on_circle(u2);
q=angle_on_circle(v);

if (q2.lt.q1) q2=2*pi+q2
if (q.lt.0.and.q.lt.q1) q=2.0*pi+q

if (q.gt.q1.and.q.lt.q2) then
   d = sqrt(v(1)**2.+v(2)**2.)-sqrt(u1(1)**2.+u1(2)**2.)
   s = sign(1.0,d)
   d = abs(d)
else
   w1 = (/-u1(2),u1(1)/)/sqrt(u1(1)**2.+u1(2)**2.)
   d1 = sqrt((p(1)-p1(1))**2.+(p(2)-p1(2))**2.)
   s1 = sign(1.0,cross2d(p-p1,w1))
   w2 = (/-u2(2),u2(1)/)/sqrt(u2(1)**2.+u2(2)**2.)
   d2 = sqrt((p(1)-p2(1))**2.+(p(2)-p2(2))**2.)
   s2 = sign(1.0,cross2d(p-p2,w2))
   if (d1.lt.d2) then
      d = d1
      s = s1
   else
      d=d2
      s=s2
   endif
endif


end subroutine

function angle_on_circle(v)result(theta)
real :: v(2),u(2),theta,q

u=(/1,0/)

q = dot_product(v,u)
if (q.ne.0) then
   q = q/(sqrt(v(1)**2.+v(2)**2.)*sqrt(u(1)**2.+u(2)**2.))
endif

theta = sgn(cross2d(u,v))*acos(q)

end function

subroutine dist_to_seg(p,p0,p1,d,s)
real,intent(in)::p(2),p0(2),p1(2)
real,intent(out)::d,s
real,dimension(2) :: u,v0,v1,pb,vb
real :: c1,c2,b
u=p1-p0
v0=p-p0
v1=p-p1

c1=dot_product(v0,u)
c2=dot_product(u,u)
```

```
if (c1.le.0.0)then
   d=sqrt(v0(1)**2.+v0(2)**2.)
   s=sign(1.0,cross2d(v0,u))
elseif (c2.le.c1) then
   d=sqrt(v1(1)**2.0+v1(2)**2.0)
   s=sign(1.0,cross2d(v1,u))
else
   b=c1/c2
   pb=p0+b*u
   vb=p-pb
   d=sqrt(vb(1)**2.0+vb(2)**2.0)
   s=sign(1.0,cross2d(vb,u))
endif


end subroutine

function sgn(a)result(b)
real :: a,b
b=sign(1.0,a)
if (b.ge.0)b=1.0

end function

function cross2d(u,v)result(c)
real,dimension(2):: u,v
real :: c

c=u(1)*v(2)-u(2)*v(1)

end function
```