Unstructured Nodal Discontinuous Galerkin Method for Convection-Diffusion Equations Applied to Neutral Fluids and Plasmas

Yang Song

Dissertation submitted to the Faculty of the Virginia Polytechnic Institute and State University in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Aerospace Engineering

Bhuvana Srinivasan, Chair
Colin S. Adams
Christopher J. Roy
Wayne A. Scales

24th February, 2020
Blacksburg, Virginia

Keywords: Nodal Discontinuous Galerkin, Magnetohydrodynamics, Two-Fluid Plasma Model, Plasma Instabilities, Convection-Diffusion, Reconstruction, Mass of Fluid

Copyright 2020, Yang Song
Unstructured Nodal Discontinuous Galerkin Method for Convection-Diffusion Equations Applied to Neutral Fluids and Plasmas

Yang Song

(ABSTRACT)

In recent years, the discontinuous Galerkin (DG) method has been successfully applied to solving hyperbolic conservation laws. Due to its compactness, high order accuracy, and versatility, the DG method has been extensively applied to convection-diffusion problems. In this dissertation, a numerical package, PHORCE, is introduced to solve a number of convection-diffusion problems in neutral fluids and plasmas. Unstructured grids are used in order to randomize grid errors, which is especially important for complex geometries. PHORCE is written in C++ and fully parallelized using the MPI library. Memory optimization has been considered in this work to achieve improved efficiency. DG algorithms for hyperbolic terms are well studied. However, an accurate and efficient diffusion solver still constitutes ongoing research, especially for a nodal representation of the discontinuous Galerkin (NDG) method. An affine reconstructed discontinuous Galerkin (aRDG) algorithm is developed in this work to solve the diffusive operator using an unstructured NDG method. Unlike other reconstructed/recovery algorithms, all computations can be performed on a reference domain, which promotes efficiency in computation and storage. In addition, to the best of the authors’ knowledge, this is the first practical guideline that has been proposed for applying the reconstruction algorithm on a nodal discontinuous Galerkin method. TVB type and WENO type limiters are also studied to deal with numerical oscillations in regions with strong physical gradients in state variables. A high-order positivity-preserving limiter is also extended in this work to prevent negative densities and pressure. A new interface tracking method, mass of fluid (MOF), along with its bound limiter has been proposed in this work to compute the mass fractions of different fluids over time. Hydrodynamic models, such as Euler and Navier-Stokes equations, and plasma models, such as ideal-magnetohydrodynamics (MHD) and two-fluid plasma equations, are studied and benchmarked with various applications using this DG framework. Numerical computations of Rayleigh-Taylor instability growth with experimentally relevant parameters are performed using hydrodynamic and MHD models on planar and radially converging domains. Discussions of the suppression mechanisms of Rayleigh-Taylor instabilities due to magnetic fields, viscosity, resistivity, and thermal conductivity are also included.

This work was partially supported by the US Department of Energy under grant number DE-SC0016515.

The author acknowledges Advanced Research Computing at Virginia Tech for providing computational resources and technical support that have contributed to the results reported within this work. URL: http://www.arc.vt.edu
High-energy density (HED) plasma science is an important area in studying astrophysical phenomena as well as laboratory phenomena such as those applicable to inertial confinement fusion (ICF). ICF plasmas undergo radial compression, with an aim of achieving fusion ignition, and are subject to a number of hydrodynamic instabilities that can significantly alter the implosion and prevent sufficient fusion reactions. An understanding of these instabilities and their mitigation mechanisms is important in allowing for a stable implosion in ICF experiments. This work aims to provide a high order accurate and robust numerical framework that can be used to study these instabilities through simulations.

The first half of this work aims to provide a detailed description of the numerical framework, PHORCE. PHORCE is a high order numerical package that can be used in solving convection-diffusion problems in neutral fluids and plasmas. Outstanding challenges exist in simulating high energy density (HED) hydrodynamics, where very large gradients exist in density, temperature, and transport coefficients (such as viscosity), and numerical instabilities arise from these regions if there is no intervention. These instabilities may lead to inaccurate results or cause simulations to fail, especially for high-order numerical methods. Substantial work has been done in PHORCE to improve its robustness in dealing with numerical instabilities. This includes the implementation and design of several high-order limiters. A novel algorithm is also proposed in this work to solve the diffusion term accurately and efficiently, which further enriches the physics that PHORCE can investigate.

The second half of this work involves rigorous benchmarks and experimentally relevant simulations of hydrodynamic instabilities. Both advection and diffusion solvers are well verified through convergence studies. Hydrodynamic and plasma models implemented are also validated against results in existing literature. Rayleigh-Taylor instability growth with experimentally relevant parameters are performed on both planar and radially converging domains. Although this work is motivated by physics in HED hydrodynamics, the emphasis is placed on numerical models that are generally applicable across a wide variety of fields and disciplines.
To my wife Pei, son Xinghe, and parents Guangsen and Huozhen
Acknowledgments

I have received tremendous help and support from many people during this both challenging and fulfilling five-year-long journey. This dissertation would not be possible without them and I would like to thank them all.

First of all, I would like to thank my advisor, Professor Bhuvana Srinivasan, for the continuous moral and financial support and the guidance, in both research and life, over the past five years. As her first Ph.D. student, who have stayed in her group for the longest time, I have witnessed how her diligence, easygoingness, and wisdom draws a rapid expansion of the group.

I would like to thank my committee members, Professor Colin Adams, Professor Christopher Roy, and Professor Wayne Scales, for their time and valuable suggestions. Last but not least, I would like to thank Professor Tim Warburton, whose discontinuous Galerkin course and textbook have helped me tremendously on my research.

Writing PHORCE from scratch was not an easy task. In fact, before reaching the satisfaction of each successful implementation, my work mostly involved tedious testing, uncomfortable debugging, and the confusion while deriving the algorithm. Fortunately, my fellow students and postdocs has been really helpful for both my research and sanity. Dr. Petr Cagas, who has been sitting next to me in our office for the past four years, has been involved in many meaningful technical discussions with me on discontinuous Galerkin method. Outside of research, we also talk about interesting cultures and exchange opinions on a lot of things. Robert Masti and Dr. Shunxian Cao have been great companions during our fishing trips, although we have argued a lot about who is the best angler among us. Dr. Jianxun Wang has helped me a lot since I joined Virginia Tech. Dr. Jinglong Wu and Dr. Di Zhang have been really good workout partners for me. Suresh Murugaiyan and I have had many deep discussions on CFD code design and I have learned a lot about finite volume method from him. I would also like to thank Dr. Ratan Bera, Chirag Rathod, Kolter Bradshaw, John Rodman, and Gaurav Sakhawala for reading through my dissertation.

I am truly grateful for the constant support and encouragement from my parents, Guangsen Song and Huozhen Chen. I talk with them once a week and their wisdom has often cleared my doubt and given me strength.
Of course, I want to thank my beautiful wife, Pei Qiu, and beloved son, Ethan (Xinghe) Song. Their company has made this journey much more enjoyable. Foremost I want to thank Pei, who has stood by me through all my travails, my absences, my fits of pique and impatience. Without her support, it would be impossible for me to make it this far.
# Contents

1 Introduction 1

1.1 Research Objectives ................................................. 8

2 Equation Systems 10

2.1 Eigenvalues and Eigenvectors ........................................ 11
2.2 Euler Equations ...................................................... 11
2.3 Navier-Stokes Equations .............................................. 13
2.4 Two-fluid Plasma Equations ........................................... 14
2.5 Magnetohydrodynamics (MHD) Equations ............................. 15
  2.5.1 Ideal-MHD Equations ............................................. 15
  2.5.2 Generalized Lagrange Multiplier(GLM)-MHD Equations .......... 19
2.6 Viscosity, Resistivity, and Heat Conduction ................................ 20
2.7 Interface Tracking Methods ........................................... 21
  2.7.1 Level Set Method ................................................. 22
  2.7.2 Volume of Fluid Method ......................................... 22
  2.7.3 Mass of Fluid Method ........................................... 22

3 Nodal Discontinuous Galerkin Method for Hyperbolic Conservation Laws 25

3.1 Governing Equation and Discretization ................................ 25
3.2 Nodal Discontinuous Galerkin Method ................................ 26
3.3 Initial Conditions .................................................... 30
3.4 Boundary Conditions .................................................. 32
3.4.1 Prescribed Boundary ........................................ 32
3.4.2 Slip Wall ..................................................... 33
3.4.3 Non-slip Wall ............................................... 33
3.4.4 Conducting Wall .............................................. 33
3.4.5 Periodic Boundary .......................................... 33

3.5 Time Stepping ................................................... 35

3.6 Discussion of Limiters ......................................... 36
3.6.1 Shock Sensor ............................................... 36
3.6.2 Slope Limiter ............................................... 38
3.6.3 WENO Limiter ............................................... 40
3.6.4 Positivity Preserving Limiter .............................. 45
3.6.5 MOF Bound Limiter ....................................... 46
3.6.6 Effect of Limiters on $\nabla \cdot B$ Errors .............. 49

4 Affine Reconstructed DG for Diffusion .......................... 51
4.1 Governing Equation and Discretization ...................... 51
4.1.1 Governing Equation ...................................... 51
4.1.2 Discretization ............................................. 52
4.2 NDG Elemental Operation for Diffusion ....................... 53
4.3 Affine Reconstructed Algorithm ................................ 54
4.3.1 Non-affine Mapping in Quadrilaterals .................. 54
4.3.2 Enclosed Parallelogram ................................... 55
4.3.3 Storage and Computational Efficiency .................. 58
4.3.4 Reordering Nodes in the Reference Domain $(r, s)$ ... 58
4.3.5 Reconstruction ........................................... 59

5 Code Verification and Benchmark ................................ 61
5.1 Order of Accuracy ............................................ 61
5.2 Hyperbolic Solver ............................................ 63
5.2.1 Euler Equations ..................................... 63
5.2.2 GLM-MHD Equations ................................. 69
5.3 Diffusion Solver ........................................... 70
5.3.1 Scalar Equations with Diffusion ....................... 71
5.3.2 Navier-Stokes Equations .............................. 79
5.3.3 Summary of aRDG Tests ............................... 81

6 Rayleigh-Taylor Instability ............................... 84
6.1 Theory .................................................... 84
   6.1.1 Magnetic Field ........................................ 85
   6.1.2 Viscosity ............................................. 85
   6.1.3 Thermal Conduction .................................. 87
   6.1.4 Resistivity ............................................ 87
6.2 Growth Rate Validation of Single-mode RTI .......... 87
6.3 Statistical Study of Multimode RTI .................. 88
6.4 Effect of Resistivity .................................... 91
6.5 Experimentally Relevant RTI ........................... 93
   6.5.1 Effects of Magnetic Field ............................ 95
   6.5.2 Effects of Viscosity .................................. 97
   6.5.3 Effects of Thermal Conduction ....................... 98
6.6 Radial Implosion ......................................... 102

7 Conclusion ................................................. 111

Bibliography .................................................. 123

Appendices .................................................... 124

Appendix A PHORCE User Guide .......................... 125
   A.1 Installation .............................................. 125
   A.2 Mesh Generation Using Gmsh .......................... 127
A.3 Input File ................................................................. 130
A.4 Set Up an Application Code ........................................ 134
  A.4.1 Initial Conditions ............................................... 134
  A.4.2 Boundary Conditions ........................................... 138
  A.4.3 Application Driver .............................................. 140
A.5 Run the Code .......................................................... 141
A.6 Post-processing and Visualization in ParaView ................. 141

Appendix B PHORCE Developer Guide: Equation System ............. 147

Appendix C PHORCE Developer Guide: Diffusion Terms ............... 153
  C.1 Implementation of calc_diffusive_flux_volume ............... 153
  C.2 Implementation of calc_diffusive_flux_surface ............... 155
List of Figures

1.1 The Crab Nebula ................................................................. 4
1.2 ICF demonstration ............................................................. 5
1.3 Cross-code comparison of 2D implosion ................................. 6
1.4 Anatomy of an FRC ............................................................ 6
1.5 A cross-section of the ion density profile for a 3D FRC ............ 7
1.6 Architecture of PHORCE ...................................................... 9
2.1 A diagram for planar Rayleigh-Taylor instability setup .......... 24
3.1 Example of Legendre-Gauss-Lobatto (LGL) nodes on a triangle for orders from 1 to 9 ................................................. 28
3.2 Affine transformation between physical element \( \Omega_k \) and reference element \( I \) . 29
3.3 Convergence tests of isentropic vortex with different initialization methods . 31
3.4 Illustration of boundary condition setups ................................ 32
3.5 Illustration of classical periodic boundary condition setups ........ 34
3.6 Illustration of azimuthal periodic boundary condition setups ........ 35
3.7 The stencil of \( \{ \Omega_{i \in \{0,1,2,3\}} \} \) .................................................. 39
3.8 Configurations of \( \Omega_i \) and \( \Omega_0 \) for example 1 ................ 42
3.9 Incorrect extrapolation from the solution in \( \Omega_i \) to \( \Omega_0 \) for example 1 ........ 43
3.10 Corrected extrapolation from the solution in \( \Omega_i \) to \( \Omega_0 \) for example 1 ........ 43
3.11 Configurations of \( \Omega_i \) and \( \Omega_0 \) for example 2 ................ 44
3.12 Incorrect extrapolation from the solution in \( \Omega_i \) to \( \Omega_0 \) for example 2 ........ 44
3.13 Corrected extrapolation from the solution in \( \Omega_i \) to \( \Omega_0 \) for example 2 ........ 44
3.14 MOF $m_1$ violates lower bound and needs to be shrunk ................. 47
3.15 MOF $m_1$ violates upper bound and needs to be shrunk ................. 47
3.16 MOF $m_1$ violates upper bound and needs to be expanded .............. 48

4.1 Mapping transformation in quadrilaterals ................................. 55
4.2 Tensor product of LGL nodes on a parallelogram ......................... 56
4.3 Illustrations of an enclosed parallelogram ................................. 57
4.4 Reordering of nodes in $I$ ................................................... 59

5.1 Demonstration of systematic refinements ................................. 64
5.2 Convergence tests of isentropic vortex .................................. 64
5.3 Order of accuracy results for isentropic vortex .......................... 65
5.4 Density profile of forward facing step case ............................... 65
5.5 RMI: A simplified planar description of the RMI .......................... 66
5.6 RMI: Demonstration of the structured mesh (left); density profiles at time $t = 0.2$ (middle) and time $t = 1.3$ (right), respectively .................. 67
5.7 RMI: Demonstration of the random mesh (left); density profiles at time $t = 0.2$ (middle) and time $t = 1.3$ (right), respectively .................. 67
5.8 RMI: Time evolution of the wavenumber spectrum of power density integrated along x direction .................................................. 68
5.9 Convergence tests of isodensity vortex .................................... 70
5.10 Order of accuracy results for isodensity vortex ........................... 70
5.11 $\nabla \cdot \mathbf{B}$ error over time using P3 NDG method ...................... 71
5.12 Three types of grids used in the tests. ..................................... 72
5.13 Initial condition at $t = 0$ and final solution at $t = 0.5$ for the diffusion test ................................................................. 73
5.14 Illustration of the perturbation on grid-a (Figure 5.12) with different resolutions ................................................................. 73
5.15 Convergence tests of the diffusion equation on three types of grids ... 74
5.16 Order of accuracy results for the diffusion equation on three types of grids ................................................................. 74
5.17 Initial condition at $t = 0$ and final solution at $t = 0.5$ for the advection-diffusion test ................................................................. 75
5.18 Convergence tests of advection-diffusion equation on three types of grids . 75
5.19 Order of accuracy results for advection-diffusion equation on three types of grids ................................................................. 76
5.20 Convergence tests of scalar convection-diffusion equation .............. 77
5.21 Order of accuracy results for scalar convection-diffusion equation ........ 77
5.22 Convergence tests of scalar shear-diffusion equation ..................... 78
5.23 Order of accuracy results for scalar shear-diffusion equation .......... 79
5.24 Convergence tests of compressible Navier-Stokes equations .......... 81
5.25 Order of accuracy results for compressible Navier-Stokes equations ... 81
5.26 Mach number plot with streamlines of subsonic flow over circular cylinder with $Re = 40$ ......................................................... 82

6.1 The growth rate $n$ changes with wavenumber $k$ for different Atwood number. 86
6.2 Density profile of single-mode RTI test at time $t = 15$ s .................. 89
6.3 Peak bubble to spike distance $h_{peak}$ over time of single-mode RTI .... 90
6.4 Density profile with $\phi = 0.5$ contour (a) of multimode RTI test at $t = 3.5$ .. 92
6.5 Comparison of density profiles ................................................. 95
6.6 Same as Figure 6.5 at $t = 1.7$ .................................................. 96
6.7 Same as Figure 6.5 at $t = 2.0$ .................................................. 97
6.8 Comparison of $B_x$ profiles for three cases described in Figure 6.5 at $t = 2.0$ . 98
6.9 Illustration of the mesh for the experimentally relevant RTI tests ........ 98
6.10 Comparison of mass density profiles for experimentally relevant RTI tests .. 99
6.11 Profiles of mass density and magnitude of magnetic field for experimentally relevant RTI tests of initial $B_x = 50$ T at $t_{\gamma RT} = 4.7$ ......................... 99
6.12 Profiles of mass density and magnitude of magnetic field for experimentally relevant RTI tests of initial $B_x = 100$ T at $t_{\gamma RT} = 6.7$ ......................... 100
6.13 Profiles of mass density and magnitude of magnetic field for experimentally relevant RTI tests of initial $B_x = 214$ T at $t_{\gamma RT} = 12.7$ ......................... 100
6.14 Maximum magnitude of magnetic field $|B|_{max}$ growth over time ($t_{\gamma RT}$) for experimentally relevant RTI tests of initial $B_x = 50$ T, $B_x = 100$ T, and $B_x = 214$ T .............................................. 101
6.15 Critical wavelength $\lambda_c$ growth over time ($t_{\gamma RT}$) for experimentally relevant RTI tests of initial $B_x = 50 \text{T}$, $B_x = 100 \text{T}$, and $B_x = 214 \text{T}$. $\lambda_c$ is calculated using $\lambda_c = \frac{B_{\text{max}}^2}{2 \mu_0 (\rho_h - \rho_l)}$ ................................................................. 101

6.16 Peak bubble to spike distance $h_{\text{peak}}$ growth over time ($t_{\gamma RT}$) for experimentally relevant RTI tests of initial $B_x = 0$, $B_x = 50 \text{T}$, $B_x = 100 \text{T}$, and $B_x = 214 \text{T}$ 102

6.17 Plasma $\beta$ evolution with peak bubble to spike distance $h_{\text{peak}}$ for experimentally relevant RTI tests of initial $B_x = 50 \text{T}$, $B_x = 100 \text{T}$, and $B_x = 214 \text{T}$ 102

6.18 Initial Reynolds number $Re$ plot along $y/L_y$ ................................................................. 103

6.19 Comparison of mass density profiles for experimentally relevant RTI tests without and with viscosity .......................................................... 103

6.20 Comparison of mass density profiles for experimentally relevant RTI tests without and with thermal conduction .......................................................... 104

6.21 Demonstration of the mesh used in radial implosion test with a coarse version 106

6.22 Density profile of implosion case without perturbation at time $t = 1.5 \text{s}$ using the Euler equations ................................................................. 106

6.23 Density profile of implosion case without perturbation at time $t = 2.5 \text{s}$ using the Euler equations ................................................................. 107

6.24 Density profile of implosion case without multimode perturbation at time $t = 1.5 \text{s}$ using the Euler equations ................................................................. 108

6.25 Density profile of implosion case with multimode perturbation at time $t = 2.5 \text{s}$ using the Euler equations ................................................................. 108

6.26 Density profile of implosion case with multimode perturbation at time $t = 2.5 \text{s}$ using the Euler equations ................................................................. 109

6.27 Density profile of implosion case with multimode perturbation at time $t = 2.5 \text{s}$ using the Navier-Stokes equations ................................................................. 110

6.28 Density profile of implosion case with multimode perturbation at time $t = 2.5 \text{s}$ using the Navier-Stokes equations ................................................................. 110

A.1 Visualization of FlowOverCylinder_longwake.geo ................................................................. 130

A.2 Visualization of FlowOverCylinder_longwake.msh ................................................................. 131

A.3 Fields in ParaView ................................................................. 142

A.4 Calculator in ParaView ................................................................. 143

A.5 Contour in ParaView ................................................................. 144
A.6 Descriptive Statistics in \textit{ParaView} \hfill 145
A.7 Statistical Model in \textit{ParaView} \hfill 146
## List of Tables

4.1 Comparison of storage requirements ........................................... 58

5.1 Drag coefficient and length of recirculation for subsonic flow over circular cylinder with $Re = 40$ ......................................................... 82

6.1 Simulation setup for single-mode RTI test ................................. 88

6.2 Simulation setup for multimode RTI test .................................... 91

6.3 Characteristic width of bubbles and spikes of multimode RTI test at $t = 3.5$ 93

6.4 Simulation setup for experimentally relevant multimode RTI test .... 94

6.5 Initial conditions of radial implosion test .................................. 105
Chapter 1

Introduction

In computational fluid dynamics and computational plasma physics, systems of partial differential equations (PDE) are investigated to solve large-scale, highly complex flow dynamics. There are different methods for solving partial differential equations numerically by discretizing the spatial derivatives and re-writing these equations in semi-discrete forms. Then an integration method for ordinary differential equations can be chosen to advance the equations in time. Among the spatial discretization methods, finite difference (FD), finite volume (FV), and finite element (FE) methods are most popular. FD method approximates the spatial derivative operator with differences between values at adjacent points. The main advantage of FD is its simplicity. However, it is ill-suited to deal with complex geometries [Hesthaven and Warburton, 2007]. This is addressed in the FV method, where the solution is approximated at the center of each element with the cell averaged value. This allows discontinuities of the solution at the element interfaces, which can be used to capture physical gradients or discontinuities. However, the FV method has some limitations on the extension to high-order discretization on general geometries. This is not the case for the classical FE method, where the solution is approximated with polynomials that can have high orders. However, the lack of an explicit semi-discrete form makes it less appealing. The discontinuous Galerkin (DG) method can be seen as a combination of FV and FE methods, as the solution in each element is approximated with a polynomial. However, discontinuity in solution at the cell interface is allowed.

In recent years, the DG method has been successfully applied to hyperbolic conservation laws [Bassi and Rebay, 1997, Cockburn and Shu, 1998b, Srinivasan, 2006, Srinivasan et al., 2011, Cockburn et al., 2012]. Due to its compactness, high-order accuracy, and versatility, the DG algorithm is favorable for applications to convection-diffusion problems,

\[
\frac{\partial q}{\partial t} + \nabla \cdot f + \nabla \cdot g = s, \quad (1.1)
\]

where \(q\) represents the conserved variable, \(f\) is the hyperbolic flux, \(g\) is the diffusive flux, and \(s\) represents the source term. In this dissertation, the primary focus is on solving neutral fluid
models, such as Euler equations and Navier-Stokes equations, and fluid plasma models, such as two-fluid plasma equations, and magnetohydrodynamics (MHD) equations. Viscosity, resistivity, and heat conduction are included in the MHD equations. Since the applications of interest may involve multi-fluid materials, interface tracking algorithms, such as level-set (LS) and volume of fluid (VOF) methods, are also discussed. A novel interface tracking method, mass of fluid (MOF) method, is proposed in this dissertation, where all equations can be written in conservative form, which simplifies the DG discretization significantly. These equations are described in Chapter 2.

A significant amount of literature exists on accurate and efficient implementations for the convection terms using the DG method. In this dissertation, the convection solver is based on the nodal DG (NDG) method described in Hesthaven and Warburton [2007]. Details of the formulation and important terminology are introduced in Chapter 3.

While the DG convection solver is well-developed, solving the diffusion terms, e.g. viscosity, resistivity, and heat conduction, in DG is non-trivial. The diffusive flux is not defined on the interface of elements as DG solution representations are only piecewise continuous. Approximating the diffusive flux as a simple arithmetic mean from both sides of the interface is not appropriate as it ignores the possible jump of the solutions. A number of numerical algorithms have been proposed in the DG community to approximate the diffusion operator with high-order accuracy, for example, Douglas and Dupont [1976], Arnold [1982], Cockburn and Shu [1998a], Peraire and Persson [2008], Liu and Yan [2009], and others. However, all the above methods require large computational effort.

In 2005, Van Leer and Nomura [2005] proposed a recovery-based DG algorithm to solve the diffusion operator, where a new polynomial that is smoothly defined across two adjacent elements is recovered from the two original polynomials with order of $P$. The new polynomial is of order $2P + 1$ and is indistinguishable from the original solutions defined across two cells in a weak sense. This recovery-based method is a more natural and accurate way of calculating the diffusive flux. This algorithm is further developed and applied on a two-dimensional structured mesh [Nourgaliev et al., 2010]. However, the accuracy of the scheme is affected not only by the diffusive parts but also the hyperbolic parts in the system. In fact, the order of accuracy is determined by the least accurate component in the system. Hence, a highly accurate diffusion solver does not increase the overall accuracy of the scheme in solving convection-diffusion problems, if the convection solver is of lower order. Also, constructing an appropriate basis function defined on the combination of two elements is an involved process. More recently, a reconstruction-based DG algorithm using Taylor basis functions is proposed in Luo et al. [2010]. In this algorithm, similar to the recovery DG algorithm, a smooth solution is reconstructed across two adjacent elements. Unlike the recovery DG algorithm, the reconstructed solution has the same polynomial order as the original solutions and is not indistinguishable from the original solutions in a weak sense. The reconstruction-based DG algorithm can solve the diffusion term with the same order of accuracy as the hyperbolic solver, making the scheme computationally efficient. Also, since the reconstructed polynomial has the same order as the underlying DG solution, it is not
necessary to carefully construct a basis function that is well-conditioned across two elements. The choice of Taylor basis simplifies the reconstruction process significantly although it suffers from ill-conditioning Vandermonde matrices.

Storage management and computational efficiency are playing increasingly significant roles in modern computational software especially for large-scale high-fidelity simulations. Conventional DG algorithms solve hyperbolic terms on a reference element, then transform the solution to physical elements. There are advantages with respect to computational efficiency and memory management if the reconstructed DG algorithm could be solved on a reference domain. Depending on the shape of the elements (triangle, quadrilateral, etc.), different memory requirements are dictated by the need to store the transformation Jacobians. Without careful treatment, this could result in higher cost of either memory or computation for recovery or reconstruction methods. Thus, solving the diffusion operator using DG in a stable, efficient, and accurate manner is still an open question. It is worth mentioning that recent developments have been made in the reconstructed DG algorithm to couple the direct DG method [Yang et al., 2018] with a first-order hyperbolic system (FOHS) [Lou et al., 2018]. However, the primary focus of the diffusion solver in this dissertation is on memory and computational efficiency while solving the diffusion term accurately. A new diffusion method, the affine reconstructed discontinuous Galerkin (aRDG) method, is described in Chapter 4 for solving the diffusion operator in convection-diffusion equations. The proposed numerical approach reconstructs a smooth solution in a parallelogram that is enclosed by the quadrilateral formed by two adjacent triangle elements. The interface between these two triangles is the diagonal of the enclosed parallelogram. Similar to triangles, the mapping of parallelograms from a physical domain to a reference domain is also an affine mapping. Thus, all computations can still be performed on the reference domain, which promotes efficiency in computation and storage. This reconstruction does not make assumptions on choice of polynomial basis. Reconstructed DG algorithms have previously been developed for modal implementations of the convection-diffusion equations. However, to the best of the author’s knowledge, this is the first practical guideline that has been proposed for applying the reconstructed algorithm on a nodal discontinuous Galerkin method.

DG methods can be used to numerically approximate neutral fluids and plasmas without further modification when the solution is smooth or only contains weak discontinuities. However, when strong discontinuities are present, Gibbs phenomenon [Wilbraham, 1848] appears in those regions and generates numerical instabilities. Slope or WENO limiters can be applied in this situation to ensure numerical stability. However, these two limiters do not guarantee the positivity of density and pressure, especially in plasma simulations related to high-energy-density (HED) hydrodynamics, where very large gradients could exist in density and temperature. A high-order positivity preserving (PP) limiter, originally proposed in Zhang and Shu [2010] with improvement in robustness described in Cheng et al. [2013], is also discussed. All three limiters are described in Chapter 3. In the same chapter, a novel high-order MOF bound limiter is introduced to ensure the marker function is bounded between [0, 1].
In this work, a computational framework PHORCE (Package for High ORder simulations of Convection-diffusion Equations) is developed. PHORCE is a multi-dimensional DG solver for general convection-diffusion equations on unstructured grids, that is written in C++ and fully parallelized using MPI. All simulations discussed in this dissertation have been performed using PHORCE. In PHORCE, equations that have been implemented currently are Euler equations, Navier-Stokes equations, MHD equations, two-fluid plasma equations, Maxwell equations, several scalar convection-diffusion equations with controllable coefficients for convection terms, and normal and shear diffusion terms, and the MOF equation.

This work is motivated by compressible hydrodynamics including in high-energy-density (HED) regimes. HED refers to the regime where the pressure is of 1 Mbar to 1000 Gbar, such as in some supernova explosions, taking the Crab Nebula as an example that is shown in Figure 1.1, or inertial confinement fusion (ICF) implosions. In Figure 1.2, a diagram of

![Image of Crab Nebula](image.jpg)

Figure 1.1: The Crab Nebula. Figure taken from Hester [2008].

ICF from Lawrence Livermore National Laboratory’s website is provided. The gold cylinder is called a hohlraum, which is about the size of a dime. Inside the hohlraum, a tiny capsule of fusion fuel made of deuterium and tritium is kept in the center by membranes. Laser beams shoot into the hohlraum from different directions to heat the inner surface of the hohlraum so that X-rays can be generated. The X-rays create a blow-off of the capsule surface and compress the inner fuel to initialize the implosion. Ideally, at the final time of the implosion, the fuel core reaches 100 times the density of lead and ignites at very high temperature. Thermonuclear burn then spreads rapidly and is expected to yield many times the input energy.

ICF implosions can be disrupted by hydrodynamic instabilities, such as the Rayleigh-Taylor instability (RTI). When a fluid with higher density is supported by another fluid with lower density, or when two fluids with different densities are accelerated towards each other, the instability that arises from the interface between the two fluids is called the Rayleigh-Taylor instability [Lord, 1900, Taylor, 1950, Chandrasekhar, 1961, Srinivasan et al., 2019]. In ICF implosions, RTI grows during the acceleration or deceleration phase of the implosion. The
growth of RTI increases the mix between the fuel and the ablator and disrupts the interface during compression. This decreases the convergence and negatively impact conditions necessary to achieve fusion ignition. In order to improve the energy gain in ICF, an understanding of the RTI growth and any possible growth suppression mechanism is important.

In Joggerst et al. [2014], ICF implosions have been simulated in a 2D cylindrical geometry using three codes, FLASH [Fryxell et al., 2000, Calder et al., 2002, Dubey et al., 2009], PPM [Ramaprabhu et al., 2012], and RAGE [Gittings et al., 2008]. Results from FLASH using a polar mesh and from PPM and RAGE using a Cartesian mesh are compared for an unperturbed implosion. Results from FLASH show that the curved interface is well maintained. However, results from PPM and RAGE show deformations of the curved interface due to the perturbation from the use of a Cartesian mesh. This is indicated in Figure 1.3. This motivates the unstructured mesh implementation in PHORCE, as randomized grid perturbations in unstructured meshes have minimal effects on the topology of complex configurations. Furthermore, an unstructured mesh allows for general geometry extending the capabilities to a number of other applications without the limited applicability of cylindrical and spherical codes.

Similar distortions of circular configurations due to the grid effects from Cartesian meshes are also reported in Srinivasan [2010] for simulations of a 3-D Field Reversed Configuration (FRC). An FRC is a compact fusion and space propulsion concept as shown in Figure 1.4. FRC is a toroidal magnetic confinement device that confines plasma on closed magnetic field lines with little or no toroidal field and high ratio of plasma pressure to magnetic pressure. FRC is considered a compact-toroidal device due to the absence of a central column. Multi-fluid plasma simulations of 3D FRCs using a Cartesian mesh exhibit distortions of the
Figure 1.3: Cross-code comparison of 2D implosion: fluid density profile at $t = 2.5\,\text{s}$ for \textsc{Flash} (a), \textsc{PPM} (b) and \textsc{Rage} (c). Figures taken from Joggerst et al. [2014].

equilibrium configuration producing azimuthal modes introduced due to preferential grid effect errors as shown in Figure 1.5. The FRC is included to highlight multiple scenarios in vastly different parameter regimes that could benefit from the random errors of a well-constructed unstructured mesh as opposed to the preferential errors of a Cartesian mesh.

Figure 1.4: Anatomy of an FRC [Steinhauer, 2011]

Code verification of and benchmark tests for neutral fluids and plasmas equation are performed and discussed in Chapter 5. Studies of Rayleigh-Taylor instabilities are carried using Euler equations, Navier-Stokes equations, and MHD equations. Discussions of the suppression mechanisms of Rayleigh-Taylor instabilities due to magnetic fields, resistivity, and thermal conductivity will be presented.

In this research, an unstructured DG package, \textsc{Phorce}, is developed to solve convection diffusion equations that describe the behaviors of neutral fluids and plasmas. \textsc{Phorce} is carefully
designed for, but not limited to, the study of compressible hydrodynamics in high-energy-density regimes. An accurate and efficient DG diffusion solver is proposed to approximate the diffusion terms that are important in the studies of RTI. A new interface tracking method, MOF, along with a bound limiter are designed to track the mass density fractions evolution in RTI, which is essentially important in the calculation of viscosity around the interface of RTI. Accuracy and robustness of PHORCE are demonstrated in Chapters 5 and 6. As PHORCE is a key product of this work and constitutes the major contribution here, the code architecture is outlined in Figure 1.6, and each class is described as,

**Shape:** File contains functions that are specific to NDG scheme, such as building quadrature points, computing Vandermonde matrices, calculating LIFT operators; Refer to Hesthaven and Warburton [2007] and Chapter 3 for details of these operators.

**RHS:** Class contains functions needed for calculating the right hand sides of the semi-discrete PDE;

**Time-stepping:** Class where time stepping algorithms are defined;

**Mesh:** Class contains all mesh relevant information and functions. Mesh class calls functions in **Shape**;

Figure 1.5: A cross-section of the ion density profile for a 3D FRC using the two-fluid plasma model with a Cartesian mesh on cylindrical domain. Figure taken from [Srinivasan, 2010]
Service: Class that handles input and output;

Hyper2D: The parent class that all the hydrodynamic systems are inherited from. The Hyper2DMOF class is a derived class from Hyper2D and is the parent class for all the systems that couple a hydrodynamic system of equations and MOF (VOF);

Hyper2DVar: Class that handles data storage and memory layout of the numerical solution;

APP: An application file that can be generated by users with specified objects, including Service, Hyper2D, Hyper2DVar, and Updater. A driver function is defined for each specific application, with user specified initial conditions and boundary conditions;

main: The main function that contains all the driver functions. The Mesh object is created in main and sent to the driver function (APP).

Furthermore, Appendices A, B, and C provide user and developer guides to encourage continued use and development of this framework.

1.1 Research Objectives

The over-arching objectives of this research are as follows:

1. Develop a parallel, unstructured nodal DG convection-diffusion package, PHORCE
2. Develop a novel diffusion algorithm to perform efficient and accurate affine reconstruction for the diffusion terms using an unstructured nodal DG method
3. Perform a thorough verification and benchmarking exercise of each component of the numerical algorithm for various equation systems implemented into PHORCE which include Euler equations, Navier-Stokes equations, MHD equations, two-fluid plasma equations, Maxwell equations, several scalar convection-diffusion equations with controllable coefficients for convection terms, and normal and shear diffusion terms, and the MOF equation.
4. Improve limiters to maintain effective accuracy with the capability to simulate highly disparate densities, temperatures, and diffusion coefficients in unprecedented HED simulations using an unstructured mesh
5. Perform simulations of the Rayleigh-Taylor instability in planar and implosion geometry in HED regimes in the presence of magnetic fields, viscosity, resistivity, and thermal conduction
Figure 1.6: Architecture of PHORCE
Chapter 2

Equation Systems

Multiple convection-diffusion systems are investigated in this research. This chapter focuses on describing the formulation of these systems. For neutral fluid models, Euler equations and Navier-Stokes equations will be described. For plasma fluid models, two-fluid plasma equations and ideal-MHD equations will be detailed. A divergence cleaning technique for ideal-MHD equations, proposed in Dedner et al. [2002], will also be included. Several diffusive terms, including viscosity, electrical conductivity, and thermal conduction will be detailed as well. Interface tracking methods, including level-set, and volume of fluid, will be discussed. A new interface tracking method, mass of fluid, is proposed for compressible flows. This method can be implemented in PHORCE efficiently as the introduced equations are written in conservative form and the introduced conserved variables are also physical conserved quantities. All systems of systems in this chapter can be written in forms of the conservation laws. For a $N_d$-dimensional system, the conservation laws can be generalized as:

\[
\frac{\partial Q}{\partial t} + \nabla \cdot F + \nabla \cdot G = S,
\]

or for implementation purposes, can be written as,

\[
\frac{\partial Q}{\partial t} + \frac{\partial F_i}{\partial x_i} + \frac{\partial G_i}{\partial x_i} = S, \quad i = 1, \ldots, N_d,
\]

where $Q$ are the conserved variables, $x_i = (x_1, x_2, \ldots, x_{N_d})$ are the coordinates on each direction, $F = (F_1, F_2, \ldots, F_{N_d})$ are the advective fluxes on each direction, $G = (G_1, G_2, \ldots, G_{N_d})$ are the diffusive fluxes on each direction, and $S$ are the source terms. For all systems of equations described in this chapter, 2-D is assumed although entries on the third direction in any vector field are still included, such as velocity on the $z$ direction $u_3$ (or $u_z$) or magnetic field on the $z$ direction $B_3$ (or $B_z$).
2.1 Eigenvalues and Eigenvectors

The classification of a system of quasi-linear partial differential equations can be determined by its eigenvalues [Anderson and Wendt, 1995, Hirsch, 2007]. If the eigenvalues are all real, the system is hyperbolic. If the eigenvalues are all complex, the system is elliptic. In this dissertation, only hyperbolic systems of equations are investigated. For a hyperbolic system, the eigenvalues indicate the speeds of propagation for the system characteristics. For simplicity purposes, let us consider the following 1-D hyperbolic conservation law,

\[
\frac{\partial Q}{\partial t} + \frac{\partial F}{\partial x} = 0, \tag{2.3}
\]

where the diffusion and source terms are dropped. The flux Jacobian matrix is defined as,

\[
A = \frac{\partial F}{\partial Q}. \tag{2.4}
\]

The eigenvalues of the flux Jacobian matrix is defined as the roots \(\lambda\) of the characteristic equations,

\[
\det(A - \lambda I) = 0, \tag{2.5}
\]

where \(I\) is the identity matrix. This equation can also be written in forms of the eigenvectors,

\[
LAR = \lambda I, \tag{2.6}
\]

with \(LR = I\), where \(L\) and \(R\) are the left and right eigenvectors of the system that transform \(A\) into a diagonal matrix that has eigenvalues on its diagonal entries. The descriptions of eigenvectors and eigenvalues for each system of equations will be provided in the following sections.

2.2 Euler Equations

The behavior of inviscid, adiabatic, and compressible neutral fluid is governed by the Euler equations. Euler equations commonly used in hydrodynamics can be written in conservative forms as shown in equation 2.1,

\[
\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho u) = 0, \tag{2.7}
\]
\[
\frac{\partial \rho u}{\partial t} + \nabla \cdot (\rho uu + pI) = 0, \tag{2.8}
\]
\[
\frac{\partial \epsilon}{\partial t} + \nabla \cdot [(\epsilon + p)u] = 0, \tag{2.9}
\]
where \( \rho \) is the mass density of the fluid, \( \mathbf{u} = (u_x, u_y, u_z) \) is the velocity vector of the fluid, \( p \) is the pressure, and \( \epsilon \) is the total energy,

\[
\epsilon \equiv \frac{p}{\gamma - 1} + \frac{1}{2} \rho u^2,
\]

where \( u^2 = u_x^2 + u_y^2 + u_z^2 \), and \( \gamma \) is the heat capacity ratio. For ideal gas, \( \gamma \) can be written in terms of the degrees of freedom \( f \) of a molecule,

\[
\gamma = 1 + \frac{2}{f}.
\]

For ideal gas, \( f \) is 5 (three for translation and two for rotation), thus \( \gamma \) is taken as 1.4. Equations 2.7, 2.8, and 2.9 describe the conservation of mass, momentum, and total energy, respectively. The conserved variables are normalized to unit volume and become mass density \( \rho \), momentum density \( \rho \mathbf{u} \), and total energy density \( \epsilon \). The eigenvalues of the flux Jacobian at the \( x \) direction, \( \frac{\partial F}{\partial Q} \), for the Euler equations are given as,

\[
\lambda = (u_x, u_y, u_x + a, u_x - a),
\]

where \( a = \sqrt{\gamma p/\rho} \) is the sound speed. The corresponding left and right eigenvectors are,

\[
L_1 = \begin{pmatrix}
\frac{1}{2} a^2 \left( \frac{1}{2} (\gamma - 1) u^2 + au_x \right) \\
-\frac{1}{2} a^2 (\gamma - 1) u_y
\end{pmatrix}^\top, \quad
L_2 = \begin{pmatrix}
-u_y \\
0
\end{pmatrix}^\top, \quad
L_3 = \begin{pmatrix}
2 - \frac{(\gamma - 1)H}{a^2} \\
\frac{(\gamma - 1)u_x}{a^2} \\
\frac{(\gamma - 1)u_y}{a^2} \\
-\frac{\gamma - 1}{a^2}
\end{pmatrix}^\top,
\]

\[
L_4 = \begin{pmatrix}
-u_z \\
0 \\
0
\end{pmatrix}^\top, \quad
L_5 = \begin{pmatrix}
\frac{1}{2} a^2 \left( \frac{1}{2} (\gamma - 1) u^2 - au_x \right) \\
-\frac{1}{2} a^2 ((\gamma - 1)u_x - a)
\end{pmatrix}^\top.
\]
and

\[ R_1 = \begin{pmatrix} 1 \\ u_x - a \\ u_y \\ u_z \\ H - au_x \end{pmatrix}, \quad R_2 = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ u_y \end{pmatrix}, \quad R_3 = \begin{pmatrix} 1 \\ u_x \\ u_y \\ u_z \\ \frac{1}{2}u^2 \end{pmatrix}, \quad R_4 = \begin{pmatrix} 0 \\ 0 \\ 1 \\ u_z \end{pmatrix}, \quad R_5 = \begin{pmatrix} 1 \\ u_x + a \\ u_y \\ u_z \\ H + au_x \end{pmatrix}, \]

respectively, where \( H = (\epsilon + p)/\rho, \quad L = (L_1, L_2, L_3, L_4, L_5)^T, \) and \( R = (R_1, R_2, R_3, R_4, R_5). \) The left and right eigenvectors are taken from the implementations in CLAWPACK [LeVeque, 2003].

The Euler equations serve as a base system for other more complicated systems that will be described later in this section.

### 2.3 Navier-Stokes Equations

When viscous flow with heat transfer is considered, Navier-Stokes equations are introduced by including viscosity and heat flux in Euler equations. Navier-Stokes equations are given in forms of equation 2.2, where

\[ Q = \begin{pmatrix} \rho \\ \rho u_j \\ \epsilon \end{pmatrix}, \quad F_i = \begin{pmatrix} \rho u_i \\ \rho u_i u_j + p \delta_{ij} \\ (\epsilon + p)u_i \end{pmatrix}, \quad G_i = \begin{pmatrix} 0 \\ -\Pi_{ij} \\ -u_j \Pi_{ij} + q_i \end{pmatrix}, \quad S = 0, \]

and the viscous stress tensor \( \Pi \) and heat flux \( q \) are given by

\[ \Pi_{ij} = \mu \left( \frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) - \frac{2}{3} \mu \nabla \cdot \mathbf{u} \delta_{ij}, \]

\[ q_i = -\kappa \frac{\partial T}{\partial x_i}, \]

where \( \delta_{ij} \) is defined as,

\[ \delta_{ij} = \begin{cases} 1, & \text{for } i = j \\ 0, & \text{for } i \neq j \end{cases}. \]

In neutral fluids, the temperature can be calculated as,

\[ T = \frac{p}{\rho R}, \]

where \( R \) is the gas constant. For dry air, \( R = 287 \text{ J kg}^{-1} \text{K}^{-1} \). The eigenvectors and eigenvalues for Navier-Stokes equations are the same as the Euler equations, since they share the same flux Jacobians.
2.4 Two-fluid Plasma Equations

When the hydrodynamics of plasma is of interest, two-fluid plasma equations can be used, where electrons and ions are each described by a system of Euler equations, such as

\[
\frac{\partial \rho_s}{\partial t} + \nabla \cdot (\rho_s u_s) = 0,
\]

\[
\frac{\partial \rho_s u_s}{\partial t} + \nabla \cdot (\rho_s u_s u_s + \rho_s I) = \frac{\rho_s q_s}{m_s} (E + u_s \times B),
\]

\[
\frac{\partial \epsilon_s}{\partial t} + \nabla \cdot [(\epsilon_s + \rho_s) u_s] = \frac{\rho_s q_s}{m_s} u_s \cdot E,
\]

where

\[
\epsilon_s \equiv \frac{p_s}{\gamma - 1} + \frac{1}{2} \rho_s u_s^2,
\]

where the subscript \(s\) refers to ion or electron species, \(E\) is the electrical field, and \(B\) is the magnetic field. \(E\) and \(B\) are described by Maxwell’s equations. The system of Maxwell’s equations is elliptic, which is non-trivial to solve. Hence, in PHORCE, a purely hyperbolic Maxwell model described in Munz et al. [2001] is implemented,

\[
\frac{\partial E}{\partial t} - c^2 \nabla \times B + \chi c^2 \nabla \Phi = -\mu_0 \frac{J}{\epsilon_0},
\]

\[
\frac{\partial B}{\partial t} + \nabla \times E + \gamma_m \nabla \Psi = 0,
\]

\[
\frac{1}{\chi} \frac{\partial \Phi}{\partial t} + \nabla \cdot E = \frac{\rho}{\epsilon_0},
\]

\[
\frac{1}{\gamma_m c^2} \frac{\partial \Psi}{\partial t} + \nabla \cdot B = 0,
\]

with

\[
J \equiv \sum_s \frac{q_s}{m_s} \rho_s u_s,
\]

\[
\varrho \equiv \sum_s \frac{q_s}{m_S} \rho_s,
\]

where \(\epsilon_0 = 8.854 \times 10^{-12} \text{ F m}^{-1}\) is the permittivity of free space, and \(\mu_0 = 4\pi \times 10^{-7} \text{ H m}^{-1}\) is the permeability of free space, \(c = 1/\epsilon_0 \mu_0\) is the speed of light, \(q_s\) is the charge of species \(s\). In the original Maxwell’s equation, \(\nabla \cdot B = 0\) and \(\nabla \cdot E = \frac{\varrho}{\epsilon_0}\). However, in numerical computations, due to finite sizes of the discretization, these two divergence conditions do not hold. Thus, two additional variables, \(\Phi\) and \(\Psi\) are introduced to correct (clean) the divergence errors of \(B\) and \(E\). \(\chi\) and \(\gamma_m\) are constants that characterize the divergence cleaning speeds of \(E\) and \(B\), respectively. The eigenvalues of the purely hyperbolic Maxwell’s equations are given as,

\[
\lambda = (c, c, -c, -c, \gamma_m c, -\gamma_m c, \chi c, -\chi c).
\]
Two-fluid plasma equations can be useful when the spatial scales in the system are on the order of the ion skin depth and time scales are on the order of the inverse ion cyclotron frequency. In two-fluid plasma model, the characteristic speeds range from the ion and electron speeds of sound to the speed of light [Srinivasan, 2010]. The fastest eigenvalue is the speed of light, which is too large to resolve in numerical simulations using an explicit time-stepping method. One often uses an artificial speed of light, or an implicit time-stepping scheme if physical speed of light is necessary. The two-fluid plasma equations are implemented in `twoFluid2D.h` and `twoFluid2D.cpp`.

### 2.5 Magnetohydrodynamics (MHD) Equations

#### 2.5.1 Ideal-MHD Equations

The ideal two-fluid plasma equations can be reduced to Hall MHD equations through certain approximations. These approximations are negligible electron inertia, infinite light speed, and quasi-neutrality [Srinivasan, 2010]. Hall MHD is further reduced to ideal-MHD with the assumptions of negligible Hall and diamagnetic drift terms. Ideal-MHD equations can be useful when the spatial scales in the system are much larger than the ion skin depth. The ideal-MHD equations are implemented in `MHD.h` and `MHD.cpp`, and described as,

\[
\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) = 0, \tag{2.31}
\]

\[
\frac{\partial \rho \mathbf{u}}{\partial t} + \nabla \cdot (\rho \mathbf{u} \mathbf{u} + p \mathbf{I} - \frac{B B}{\mu_0} + \frac{B^2}{2\mu_0} \mathbf{I}) = 0, \tag{2.32}
\]

\[
\frac{\partial \epsilon}{\partial t} + \nabla \cdot [(\epsilon + p + \frac{B^2}{2\mu_0}) \mathbf{u} - \frac{(B \cdot \mathbf{u})}{\mu_0} B] = 0, \tag{2.33}
\]

\[
\frac{\partial B}{\partial t} + \nabla \cdot (\mathbf{u} B - B \mathbf{u}) = 0, \tag{2.34}
\]

where

\[
\epsilon \equiv \frac{p}{\gamma - 1} + \frac{1}{2} \rho u^2 + \frac{1}{2} \frac{B^2}{\mu_0}, \tag{2.35}
\]

where \(B^2 = B_x^2 + B_y^2 + B_z^2\). The eigenvalues of the flux Jacobian at the \(x\) direction, \(\frac{\partial F_x}{\partial \mathbf{u}_x}\), for the ideal-MHD equations are given as,

\[
\lambda = (u_x, u_x, u_x + c_a, u_x - c_a, u_x + c_f, u_x - c_f, u_x + c_s, u_x - c_s), \tag{2.36}
\]

where \(c_a\) is Alfven speed, \(c_f\) and \(c_s\) are fast and slow magnetosonic speeds, respectively. These are defined as,

\[
c_a = \sqrt{\left(\frac{B_x^2}{\rho}\right)}, \tag{2.37}
\]
The corresponding left and right eigenvectors are,

\[
c_f = \frac{1}{2} \left( (c^2 + \frac{B^2}{\rho}) + \sqrt{\left( \frac{c^2 + B^2}{\rho} \right)^2 - \frac{4c^2B^2}{\rho}} \right)
\]

\[
c_s = \frac{1}{2} \left( (c^2 + \frac{B^2}{\rho}) - \sqrt{\left( \frac{c^2 + B^2}{\rho} \right)^2 - \frac{4c^2B^2}{\rho}} \right).
\]

(2.38)

(2.39)

The corresponding left and right eigenvectors are,

\[
L_1 = \begin{pmatrix}
\frac{1}{2} t_1 \alpha f a^2 u^2 + \frac{1}{2} t_2 [\alpha f a u_x \beta_x - \alpha_s c_s (\beta_y u_y + \beta_z u_z)] \\
-\frac{1}{2} t_1 \alpha f a^2 u_x - \frac{1}{2} t_2 \alpha f a \beta_x \\
-\frac{1}{2} t_1 \alpha f a^2 u_y - \frac{1}{2} t_2 \alpha f c_s \beta_y \\
-\frac{1}{2} t_1 \alpha f a^2 u_z - \frac{1}{2} t_2 \alpha f c_s \beta_z \\
\frac{1}{2} t_1 \alpha f a^2 \\
0 \\
\frac{1}{2} t_1 \alpha s \beta_y c_f (c_s^2 - \frac{2}{\gamma - 1} a^2) \sqrt{\rho} \\
\frac{1}{2} t_1 \alpha s \beta z c_f (c_s^2 - \frac{2}{\gamma - 1} a^2) \sqrt{\rho}
\end{pmatrix}^T,
\]

\[
L_2 = \begin{pmatrix}
-\frac{1}{2} (\beta_z u_y - \beta_y u_z) \beta_x \\
0 \\
\frac{1}{2} \beta_y \beta_x \\
0 \\
0 \\
\frac{1}{2} \beta_z \sqrt{\rho} \\
-\frac{1}{2} \beta_z \sqrt{\rho}
\end{pmatrix}^T,
\]

(2.40)

\[
L_3 = \begin{pmatrix}
\frac{1}{2} t_1 \alpha f c_f^2 u^2 + \frac{1}{2} t_2 [\alpha_s c_a u_x \beta_x - \alpha_f c_f (\beta_y u_y + \beta_z u_z)] \\
-\frac{1}{2} t_1 \alpha s c_f^2 u_x - \frac{1}{2} t_2 \alpha_s c_a \beta_x \\
-\frac{1}{2} t_1 \alpha s c_f^2 u_y - \frac{1}{2} t_2 \alpha_f c_f \beta_y \\
-\frac{1}{2} t_1 \alpha s c_f^2 u_z - \frac{1}{2} t_2 \alpha_f c_f \beta_z \\
\frac{1}{2} t_1 \alpha s c_f^2 \\
0 \\
-\frac{1}{2} t_1 \alpha f \beta_y c_f (c_f - \frac{2}{\gamma - 1} a^2) \sqrt{\rho} \\
-\frac{1}{2} t_1 \alpha f \beta z c_f (c_f - \frac{2}{\gamma - 1} a^2) \sqrt{\rho}
\end{pmatrix}^T,
\]

\[
L_4 = \begin{pmatrix}
1 - \frac{1}{2} t_1 (\alpha_f^2 a^2 + \alpha_s^2 c_f^2) u^2 \\
t_1 (\alpha_f^2 a^2 + \alpha_s^2 c_f^2) u_x \\
t_1 (\alpha_f^2 a^2 + \alpha_s^2 c_f^2) u_y \\
t_1 (\alpha_f^2 a^2 + \alpha_s^2 c_f^2) u_z \\
-t_1 (\alpha_f^2 a^2 + \alpha_s^2 c_f^2) \\
0 \\
t_1 \alpha_f^2 \alpha_s \beta_y c_f (c_f^2 - c_s^2) \sqrt{\rho} \\
t_1 \alpha_f^2 \alpha_s \beta z c_f (c_f^2 - c_s^2) \sqrt{\rho}
\end{pmatrix}^T.
\]
\[ \begin{pmatrix}
0 \\
0 \\
0 \\
0 \\
1 \\
0 \\
0 \\
1
\end{pmatrix}^T, \quad \begin{pmatrix}
\left(\frac{1}{4}t_1\alpha_s c_f^2 u^2 - \frac{1}{2}t_2 \left[ \alpha_s c_a u_x \beta_x + \alpha_f c_f (\beta_y u_y + \beta_z u_z) \right] \right)^T \\
-\frac{1}{2}t_1\alpha_s c_f^2 u_x + \frac{1}{2}t_2\alpha_s c_a \beta_x \\
-\frac{1}{2}t_1\alpha_s c_f^2 u_y + \frac{1}{2}t_2\alpha_f c_f \beta_y \\
-\frac{1}{2}t_1\alpha_s c_f^2 u_z + \frac{1}{2}t_2\alpha_f c_f \beta_z \\
\frac{1}{2}t_1\alpha_s c_f^2 \\
0 \\
-\frac{1}{2}t_1\alpha_f \beta_y c_f \left( c_f^2 - \frac{2}{\gamma - 1} a^2 \right) \sqrt{\rho} \\
-\frac{1}{2}t_1\alpha_f \beta_z c_f \left( c_f^2 - \frac{2}{\gamma - 1} a^2 \right) \sqrt{\rho}
\end{pmatrix}^T, \]

\[ \begin{pmatrix}
-\frac{1}{2}(\beta_z u_y - \beta_y u_z) \beta_x \\
0 \\
-\frac{1}{2} \beta_z \beta_x \\
\frac{1}{2} \beta_y \beta_x \\
0 \\
0 \\
\frac{1}{2} \beta_z \sqrt{\rho} \\
-\frac{1}{2} \beta_z \sqrt{\rho}
\end{pmatrix}^T, \quad \begin{pmatrix}
\left(\frac{1}{4}t_1\alpha_f c_f^2 u^2 - \frac{1}{2}t_2 \left[ \alpha_s c_a u_x \beta_x + \alpha_f c_f (\beta_y u_y + \beta_z u_z) \right] \right)^T \\
-\frac{1}{2}t_1\alpha_f a^2 u_x + \frac{1}{2}t_2\alpha_f a \beta_x \\
-\frac{1}{2}t_1\alpha_f a^2 u_y + \frac{1}{2}t_2\alpha_s a \beta_y \\
-\frac{1}{2}t_1\alpha_f a^2 u_z + \frac{1}{2}t_2\alpha_s a \beta_z \\
\frac{1}{2}t_1\alpha_f a^2 \\
0 \\
\frac{1}{2}t_1\alpha_s \beta_y c_f \left( c_s^2 - \frac{2}{\gamma - 1} a^2 \right) \sqrt{\rho} \\
\frac{1}{2}t_1\alpha_s \beta_z c_f \left( c_s^2 - \frac{2}{\gamma - 1} a^2 \right) \sqrt{\rho}
\end{pmatrix}^T, \]

and

\[ R_1 = \begin{pmatrix}
\alpha_f \\
\alpha_f (u_x - c_f) \\
\alpha_f u_y + \alpha_s \beta_y c_a \beta_x \\
\alpha_f u_z + \alpha_s \beta_z c_a \beta_x \\
\frac{1}{2}\alpha_f u^2 + h_{mf} \\
0 \\
\alpha_s \beta_a c_f / \sqrt{\rho} \\
\alpha_s \beta_a c_f / \sqrt{\rho}
\end{pmatrix}, \quad R_2 = \begin{pmatrix}
0 \\
0 \\
\beta_z \beta_x \\
-\beta_y \beta_x \\
(\beta_z u_2 - \beta_y u_3) \beta_x \\
0 \\
0 \\
\frac{\beta_z}{\sqrt{\rho}} \\
-\frac{\beta_y}{\sqrt{\rho}}
\end{pmatrix}, \quad R_3 = \begin{pmatrix}
\alpha_s \\
\alpha_s (u_1 - c_s) \\
\alpha_s u_2 - \alpha_f \beta_y a \beta_x \\
\alpha_s u_3 - \alpha_f \beta_z a \beta_x \\
\frac{1}{2}\alpha_s u^2 + h_{ms} \\
0 \\
-\alpha_f \beta_y a^2 / (c_f \sqrt{\rho}) \\
-\alpha_f \beta_z a^2 / (c_f \sqrt{\rho})
\end{pmatrix}. \]
\[
R_4 = \begin{pmatrix}
1 \\
u_x \\
u_y \\
u_z \\
\frac{1}{2} u^2 \\
0 \\
0 \\
0
\end{pmatrix}, \\
R_5 = \begin{pmatrix}
0 \\
0 \\
0 \\
0 \\
0 \\
1 \\
0 \\
0
\end{pmatrix}, \\
R_6 = \begin{pmatrix}
\alpha_s \\
\alpha_s (u_x + c_s) \\
\alpha_s u_y + \alpha_f \beta_y a \beta_x \\
\alpha_s u_z + \alpha_f \beta_z a \beta_x \\
\frac{1}{2} \alpha_s u^2 + h_{ps} \\
0 \\
-\alpha_f \beta_y a^2 / (c_f \sqrt{\rho}) \\
-\alpha_f \beta_z a^2 / (c_f \sqrt{\rho})
\end{pmatrix}, \\
R_7 = \begin{pmatrix}
0 \\
0 \\
-\beta_x \beta_x \\
\beta_y \beta_x \\
0 \\
\beta_z / \sqrt{\rho} \\
-\beta_y / \sqrt{\rho}
\end{pmatrix}, \\
R_8 = \begin{pmatrix}
\alpha_f \\
\alpha_f (u_x + c_f) \\
\alpha_f u_y - \alpha_s \beta_y c \beta_x \\
\alpha_f u_z - \alpha_s \beta_z c \beta_x \\
\frac{1}{2} \alpha_f u^2 + h_{pf} \\
0 \\
\alpha_s \beta_y c_f / \sqrt{\rho} \\
\alpha_s \beta_z c_f / \sqrt{\rho}
\end{pmatrix}, \quad (2.41)
\]

where

\[
\beta_x = \text{sign}(B_x), \quad (2.42)
\]
\[
\beta_y = \begin{cases} 
\frac{1}{\sqrt{2}} \frac{B_y}{\sqrt{B_y^2 + B_z^2}} & \text{if } B_y = B_z = 0 \\
\frac{1}{\sqrt{2}} \frac{B_y}{\sqrt{B_y^2 + B_z^2}} & \text{if } |B_y| + |B_z| > 0 
\end{cases}, \quad (2.43)
\]
\[
\beta_z = \begin{cases} 
\frac{1}{\sqrt{2}} \frac{B_z}{\sqrt{B_y^2 + B_z^2}} & \text{if } B_y = B_z = 0 \\
\frac{1}{\sqrt{2}} \frac{B_z}{\sqrt{B_y^2 + B_z^2}} & \text{if } |B_y| + |B_z| > 0 
\end{cases}, \quad (2.44)
\]
\[
\alpha_f = \begin{cases} 
1 & \text{if } B_y = B_z = 0 \& a^2 = \frac{B_x^2}{\rho} \\
\sqrt{\frac{|c_f^2 - c^2|}{|c_f^2 - c^2|}} & \text{otherwise}
\end{cases}, \quad (2.45)
\]
\[
\alpha_s = \begin{cases} 
1 & \text{if } B_y = B_z = 0 \& a^2 = \frac{B_x^2}{\rho} \\
\sqrt{\frac{|c_s^2 - a^2|}{|c_f^2 - c^2|}} & \text{otherwise}
\end{cases}, \quad (2.46)
\]
\[
t_1 = \frac{1}{\alpha_f a^2 \left( c_f^2 - \frac{\gamma - 2}{\gamma - 1} a^2 \right) + \alpha_s c_f^2 \left( c_s^2 - \frac{\gamma - 2}{\gamma - 1} a^2 \right)}, \quad (2.47)
\]
\[
\frac{1}{2} c_f \alpha_x + \alpha_s c_s \alpha_x \]

(2.48)

\[
h_{mf} = \frac{\alpha_f c_f^2}{\gamma - 1} - \alpha_f c_f u_x + \alpha_s c_a \alpha_x (\beta_y u_y + \beta_z u_3) + \frac{\gamma - 2}{\gamma - 1} \alpha_f (c_f^2 - a^2)
\]

(2.49)

\[
h_{pf} = \frac{\alpha_f c_f^2}{\gamma - 1} + \alpha_f c_f u_x - \alpha_s c_a \alpha_x (\beta_y u_y + \beta_z u_3) + \frac{\gamma - 2}{\gamma - 1} \alpha_f (c_f^2 - a^2)
\]

(2.50)

\[
h_{ms} = \frac{\alpha_s c_s^2}{\gamma - 1} - \alpha_s c_s u_x - \alpha_f a \alpha_x (\beta_y u_y + \beta_z u_3) + \frac{\gamma - 2}{\gamma - 1} \alpha_s (c_s^2 - a^2)
\]

(2.51)

\[
h_{ps} = \frac{\alpha_s c_s^2}{\gamma - 1} + \alpha_s c_s u_x + \alpha_f a \alpha_x (\beta_y u_y + \beta_z u_3) + \frac{\gamma - 2}{\gamma - 1} \alpha_s (c_s^2 - a^2).
\]

(2.52)

The left and right eigenvectors are taken from the implementations in CLAWPACK [LeVeque, 2003].

### 2.5.2 Generalized Lagrange Multiplier(GLM)-MHD Equations

Similar to Maxwell’s equation, \( \nabla \cdot \mathbf{B} = 0 \) is not guaranteed in the ideal-MHD equations. Dedner et al. [2002] details a divergence cleaning technique of \( \mathbf{B} \). The mixed GLM-MHD equations, which include both hyperbolic and parabolic divergence cleaning, are implemented in glmMHD.h and glmMHD.cpp and described as,

\[
\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) = 0,
\]

(2.53)

\[
\frac{\partial \rho \mathbf{u}}{\partial t} + \nabla \cdot (\rho \mathbf{u} \mathbf{u} + p \mathbf{I} - \frac{\mathbf{B} \mathbf{B}}{\mu_0} + \frac{\mathbf{B}^2}{2 \mu_0} \mathbf{I}) = \nabla \Pi,
\]

(2.54)

\[
\frac{\partial \epsilon}{\partial t} + \nabla \cdot [(\epsilon + p + \frac{\mathbf{B}^2}{2 \mu_0}) \mathbf{u} - \frac{(\mathbf{B} \cdot \mathbf{u})}{\mu_0} \mathbf{B}] = \nabla (\mathbf{u} \Pi) - \nabla (q_i),
\]

(2.55)

\[
\frac{\partial \mathbf{B}}{\partial t} + \nabla \cdot (\mathbf{u} \mathbf{B} - \mathbf{B} \mathbf{u}) + \nabla \psi = -\frac{1}{\mu_0} \nabla \times (\eta \nabla \times \mathbf{B}),
\]

(2.56)

\[
\frac{\partial \psi}{\partial t} + C_h^2 \nabla \cdot \mathbf{B} = -\frac{C_h^2}{C_p^2} \psi,
\]

(2.57)

where the diffusion terms are also included. Viscous stress tensor \( \Pi \) and heat flux \( q \) are described previously in Navier-Stokes equations. \( \eta \) is the electrical resistivity. The introduced variable \( \psi \) is the divergence cleaning variable. \( C_h \) and \( C_p \) are hyperbolic and parabolic cleaning speeds, respectively. The vector forms (equation 2.2) of GLM-MHD equations are
given as,

\[
Q = \begin{pmatrix}
\rho \\
\rho u_j \\
\epsilon \\
B_j \\
\psi
\end{pmatrix}, \quad
F_i = \begin{pmatrix}
\rho u_i + p \delta_{ij} - \frac{B_i B_j}{\mu_0} + \frac{B_i^2}{2 \mu_0} \delta_{ij} \\
(\epsilon + p + \frac{B_i^2}{2 \mu_0}) u_i - B \cdot \mathbf{u} \\
u_i B_j - u_j B_i + \psi \delta_{ij} \\
C_h B_i
\end{pmatrix}, \quad
G_i = \begin{pmatrix}
0 \\
-\Pi_{ij} \\
-q_i \\
\frac{\eta}{\mu_0} \left( \frac{\partial B_i}{\partial x_j} - \frac{\partial B_j}{\partial x_i} \right)
\end{pmatrix}, \quad
S = \begin{pmatrix}
0 \\
0 \\
0 \\
-\frac{C_r^2}{C_h^2} \psi
\end{pmatrix}.
\]

(2.58)

According to Dedner et al. [2002] and Hesthaven and Warburton [2007], the hyperbolic speed \( C_h \) is chosen to be,

\[
C_h = \frac{2 C_{\text{eff}} \min \Delta r_i}{3 \Delta t} \min_{\Omega_k} R_k,
\]

(2.59)

where \( r_i \) is the minimal nodal spacing in \( I \), and \( R_k \) is the radius of the inscribed circle of \( \Omega_k \). \( C_{\text{eff}} \) is the Courant-Friedrichs-Lewy condition [Courant et al., 1928], that describes the how many units of element spacing is traveled by any information in the system per time-step. For an explicit time-stepping scheme, \( C_{\text{eff}} \) is usually set smaller than 1. The parabolic speed \( C_p \) is calculated from the following equation,

\[
C_r = \frac{C_p}{C_h},
\]

(2.60)

where \( C_r \) is a user input that mirrors the ratio between hyperbolic and parabolic effects. The damping of the divergence errors is on the time scale \( C_p \sqrt{t} \), where the transport of the divergence errors is on the time scale \( C_h t \). If \( C_r = \infty \), the divergence error will only be transported through the hyperbolic term. Practically, \( C_r \) is chosen to be 0.18 according to Dedner et al. [2002].

For both equation systems 2.31-2.33 and 2.53-2.57, setting \( \mu_0 = 1 \) will convert them into normalized units, and setting \( \mu_0 = 4\pi \) will convert them into CGS units.

### 2.6 Viscosity, Resistivity, and Heat Conduction

For ideal-MHD equations, the ion viscosity \( \mu \), electrical resistivity \( \eta \), and electron thermal conductivity \( \kappa \) are considered. The formulas for these are described in Braginskii [1965], Chen et al. [1984], Huba [2006], and provided here

\[
\mu = 0.96 n_i k T_i \tau_i,
\]

(2.61)

\[
\kappa = 3.2 \frac{n_e k T_e \tau_e}{m_e},
\]

(2.62)

\[
\eta = \frac{m_e}{1.96 n_e q_e^2 \tau_e},
\]

(2.63)
where \( k = 1.38 \times 10^{-23} \text{JK}^{-1} \) is the Boltzmann constant, \( n_i \) and \( n_e \) are the ion and electron number densities, \( T_i \) and \( T_e \) are the ion and electron temperature, \( m_i \) and \( m_e = 0.91 \times 10^{-30} \text{kg} \) are the ion and electron mass, \( q_e = 1.6 \times 10^{-19} \text{C} \) is the electron charge, and collision times are calculated as,

\[
\tau_e = \frac{6\sqrt{2\pi^{3/2}e_0^2\sqrt{m_e} (kT_e)^{3/2}}}{\Lambda q_i^4 n_e} = 275243 \frac{T_{e}^{3/2}}{n_e}, \tag{2.64}
\]

\[
\tau_i = \frac{12\pi^{3/2}e_0^2\sqrt{m_i} (kT_i)^{3/2}}{\Lambda q_i^4 n_i} = 4.07837 \times 10^{20} \frac{\sqrt{m_i T_{i}^{3/2}}}{n_i Z^4 \Lambda}, \tag{2.65}
\]

where the ion charge is calculated as \( q_i = -Zq_e \), \( Z \) is the charge state, and the Coulomb Logarithm \( \Lambda \) calculated as,

\[
\Lambda = \ln(12\pi n_e \lambda_D^2), \tag{2.66}
\]

where \( \lambda_D \) is the Debye length and given as,

\[
\lambda_D \equiv \sqrt{\frac{\epsilon_0 kT_e}{n_e q_e^2}}. \tag{2.67}
\]

The Debye length is a measure of the shielding distance or thickness of the plasma sheath [Chen et al., 1984].

### 2.7 Interface Tracking Methods

Since PHORCE has been motivated by applications in high-energy-density hydrodynamics, where different materials with different ion densities, ion masses, and \( Z \) are involved in the calculation of viscosity, electrical conductivity, and thermal conductivity, an interface tracking method is needed. Two popular methods are the level-set (LS) and volume of fluid (VOF) methods. Both methods introduce a new variable \( \phi(x, t) \) into the system and solves and additional equation,

\[
\frac{\partial(\phi)}{\partial t} + \mathbf{u} \cdot \nabla \phi = 0. \tag{2.68}
\]

In LS, \( \phi \) is a function of the signed distance \( d \) to the fluid interface, while in VOF, \( \phi \) represents the volume fraction of one fluid. In LS, the interface location is directly available at \( \phi(d = 0) \), thus is useful when statistics of the interface is needed. In VOF, it is not obvious on how to obtain the interface location from the volume fraction of one fluid, especially for compressible flow [Karakus et al., 2018]. However, VOF is useful when there are quantities at the interface need to be calculated using the volume fraction data, e.g. viscosity.
2.7.1 Level Set Method

In LS, $\phi$ is a function of the distance $d$ to the interface,

$$\phi = \frac{1}{2} \left( 1 + \tanh \left( \frac{d}{2\epsilon} \right) \right),$$

(2.69)

where $\epsilon$ is the numerical width of the interface set by users, $d$ is a signed distance function to the fluid interface and initialized as,

$$d(h, t = 0) = h - h_0,$$

(2.70)

where $h$ is the location of interest, and $h_0$ is initial interface location. In numerical simulation, locating the interface when the gradient of the $\phi$ is very small could cause ill-conditioning of the algorithm and misrepresentation of the normal vector of the interface. To avoid this, a re-initialization equation is found to be necessary to solve in order to maintain the thickness of the interface and limit mass loss in LS method [Shukla et al., 2010],

$$\frac{\partial \phi}{\partial \tau} + \nabla \cdot (\phi (1 - \phi) \mathbf{n}) = \nabla \cdot (\epsilon (\nabla \phi \cdot \mathbf{n}) \mathbf{n}),$$

(2.71)

where $\mathbf{n}$ is the normal vector of the interface, and this equation is solved on a pseudo-time $\tau$ within a time-step $\Delta t$. Note that this equation has a diffusion term that requires a diffusion solver.

2.7.2 Volume of Fluid Method

In VOF, $\phi$ is the volume fraction of one material. No re-initialization is required for VOF, as no interface normal vector or distance function is involved. This makes VOF computationally inexpensive compared to LS method. Furthermore, for problems of interest in this work, viscosity (section 2.6) at the fluid interface is required and needs to be calculated in a fractional manner, which makes VOF more favorable to this work. The initialization of $\phi$ in VOF will be discussed in section 2.7.3.

2.7.3 Mass of Fluid Method

For incompressible flow, since $\nabla \cdot \mathbf{u} = 0$, equation 2.68 can be easily written in a conservative form,

$$\frac{\partial (\phi)}{\partial t} + \nabla \cdot (\phi \mathbf{u}) = 0,$$

(2.72)

where the conserved variable $\phi$ is the volume fraction of fluid 1, which is also a conserved quantity in incompressible flow.
However, this is not the case in compressible flow, where one has to solve equation 2.68. In Shukla et al. [2010], a standard multi-fluid compressible solver is formulated by replacing the continuity equation with equations 2.68 and,

\[ \frac{\partial (\rho_1 \phi)}{\partial t} + \nabla \cdot (\rho_1 \phi \mathbf{u}) = 0, \]  
\(2.73\)

\[ \frac{\partial (\rho_2 (1 - \phi))}{\partial t} + \nabla \cdot (\rho_2 (1 - \phi) \mathbf{u}) = 0, \]  
\(2.74\)

where \(\rho_1\) and \(\rho_2\) are the densities for two different materials. For other equations in the system, the density is calculated as \(\rho = \rho_1 \phi + \rho_2 (1 - \phi)\).

Solving equation 2.68 in DG is non-trivial, as it is written in a non-conservative form, which requires additional modifications to the discretization. In order to avoid solving equation 2.68, a new interface tracking method, mass of fluid (MOF) method, is proposed in this work by introducing \(\phi_m\) as the mass fraction of fluid 1. Then the continuity equation can be replaced by

\[ \frac{\partial (\rho \phi_m)}{\partial t} + \nabla \cdot (\rho \phi_m \mathbf{u}) = 0, \]  
\(2.75\)

\[ \frac{\partial (\rho (1 - \phi_m))}{\partial t} + \nabla \cdot (\rho (1 - \phi_m) \mathbf{u}) = 0. \]  
\(2.76\)

In this formulation, the conserved variables \(\rho \phi_m\) and \(\rho (1 - \phi_m)\) are the mass concentrations of fluid 1 and fluid 2 to the total mass density \(\rho\), respectively, which are also conserved quantities in compressible flow. It is very important to distinguish the initialization processes between VOF and MOF. For clarity, the volume fraction \(\phi\) in VOF is replaced with notation \(\phi_v\) from this point. Taking a case of planar Rayleigh-Taylor instability for example, assuming \(\rho_1(t_0)\) and \(\rho_2(t_0)\) are the mass densities of the heavy and light fluids, respectively, as shown in Figure 2.1, the density is initialized using a hyperbolic tangent function,

\[ \rho(y, t_0) = \rho_2(t_0) + \frac{1}{2} \left( \tanh \left( \frac{2\pi (y - y_0)}{\epsilon} \right) + 1 \right) (\rho_1(t_0) - \rho_2(t_0)), \]  
\(2.77\)

where \(y_0\) is the interface between fluid 1 and fluid 2. Since in VOF we have \(\rho = \rho_1 \phi_v + \rho_2 (1 - \phi_v)\), thus,

\[ \phi_v(t_0) = \frac{\rho - \rho_2(t_0)}{\rho_1(t_0) - \rho_2(t_0)}. \]  
\(2.78\)

From VOF to MOF, we also have,

\[ \rho \phi_m = \rho_1 \phi_v. \]  
\(2.79\)

\(\phi_m\) can then be initialized as,

\[ \phi_m(t_0) = \frac{\rho_1 \phi_v}{\rho} = \frac{\rho_1(\rho - \rho_2(t_0))}{\rho (\rho_1(t_0) - \rho_2(t_0))}. \]  
\(2.80\)
It is worth mentioning that $\rho_1$ and $\rho_2$ are not calculated over time in the MOF method. However, the mass fraction $\phi_m$ is needed in the calculation of viscosity in ideal-MHD equations, to account for different viscosities of the two fluids, $\mu_1$ and $\mu_2$, and different charge states, $Z_1$ and $Z_2$, across the interface. This mass fraction based viscous coefficient is given by,

$$\mu = \phi_m \mu_1 + (1 - \phi_m) \mu_2.$$  \hspace{1cm} (2.81)
Chapter 3

Nodal Discontinuous Galerkin Method for Hyperbolic Conservation Laws

When dealing with the challenge of solving hyperbolic partial differential equations numerically, one needs to represent the spatial derivative operator in a discrete form. There are many choices of doing this, such as finite difference, finite volume, and finite element methods. Among them, discontinuous Galerkin finite element method (DG-FEM) has the advantages of handling complex geometries, achieving high-order accuracy while maintaining locality of the scheme, and providing an explicit semi-discrete form that enables various choices of time advancing methods [Cockburn and Shu, 1989, Hesthaven and Warburton, 2007]. The DG algorithm implemented in PHORCE is based on nodal DG (NDG) method described in Hesthaven and Warburton [2007] and will be detailed in this chapter.

3.1 Governing Equation and Discretization

To explain the algorithm in detail, the hyperbolic conservation law is considered,

\[
\frac{\partial q(x, t)}{\partial t} + \nabla \cdot f(q(x, t), x, t) = s(x, t),
\]

\[q(x, 0) = q_0(x),\]

(3.1)

where \(q\) is the conserved quantity, \(f\) is the advective flux, and \(s\) is the source term.

In the DG method, \(q\) can be expressed as a direct sum of local piecewise polynomials

\[q^k(x, t) \simeq q_h(x, t) = \bigoplus_{k=1}^{K} q^k_h(x, t).\]

(3.3)
where $K$ is the total number of elements.

Replacing $q$ in equation 3.1 with $q_h$, multiplying with a test function $\phi_i$, and integrating over non-overlapping elements $\Omega_k$, where $k = 1, \ldots, K$, provides a typical DG treatment of hyperbolic equations,

$$\int_{\Omega_k} \left( \frac{\partial q_h^k}{\partial t} + \nabla \cdot f_h^k - s_h^k \right) \phi_i^k(x) d\Omega = 0. \quad (3.4)$$

A DG scheme can be obtained by integrating the second term in equation 3.4 by parts,

$$\int_{\Omega_k} \left( \frac{\partial q_h^k}{\partial t} \phi_i^k - f_h^k \cdot \nabla \phi_i^k - s_h^k \phi_i^k \right) d\Omega = -\int_{\partial \Omega_k} \hat{n} \cdot f^* \phi_i^k d\Omega, \quad (3.5)$$

where the DG representation of $f$ is discontinuous at the element interface, however the flux needs to be continuous at the interface. This is because for the conserved variable to remain conserved, the flux that leaves from an element through an interface should equal to the flux that enters its neighboring element that shares the same interface. Thus the flux in the surface integral is replaced by a numerical flux $f^*$, which is the solution of the Riemann problem. There are several choices of numerical fluxes that can be used to approximate $f^*$. The choice of numerical flux in PHORCE is Lax-Friedrichs flux [Toro, 2013], and is described as,

$$f^* = \frac{f(q_h^-) + f(q_h^+)}{2} + \frac{\alpha}{2} \hat{n}(q_h^- - q_h^+), \quad (3.6)$$

where $q_h^-$ and $q_h^+$ are the conserved variables evaluated at the interface from the solution polynomials of the elements, $\Omega_k$ and its neighbor, that are sharing this interface. $\hat{n}$ is the local outward pointing normal of element $\Omega_k$ on that interface. $\alpha$ is the maximum eigenvalue evaluated at the interface. Lax-Friedrichs is quite diffusive as a numerical flux, and may not be a good choice in finite volume (FV) method which the cell average representation of the solution makes itself already suffering from numerical diffusion. However, Lax-Friedrichs is a popular choice in DG method, which by nature is dispersive due to the high-order polynomial representation of the solution.

### 3.2 Nodal Discontinuous Galerkin Method

Following the NDG algorithm from Hesthaven and Warburton [2007], the test function and basis function are chosen to be the Lagrange polynomials, $\ell_i$. For 1-D elements, Lagrange polynomial is defined as,

$$\ell_i(x) = \prod_{j=1, j \neq i}^{N_p} \frac{x - x_i}{x_i - x_j}, \quad (3.7)$$

where $N_p$ is the number of quadrature nodes on each 1-D element, and $x_i, i = 1, \ldots, N_p$ are the coordinates of those quadrature nodes. For 2-D triangle elements, there is no explicit form
of the Lagrange polynomials. For a detailed description on how to construct the Lagrange polynomials on 2-D triangle elements, please refer to Hesthaven and Warburton [2007].

For the sake of simplicity, the subscript $h$ is dropped from now on. Then equation 3.5 can be rewritten as,

$$
\int_{\Omega_k} \left( \frac{\partial q^k}{\partial t} \ell^k_i - f^k \cdot \nabla \ell^k_i - s^k \ell^k_i \right) d\Omega = - \int_{\partial \Omega_k} \hat{n} \cdot f^* \ell^k_i d\Omega.
$$

Solutions on Legendre-Gauss-Lobatto (LGL) nodes [Abramowitz and Stegun, 1972] are chosen to be the expansion coefficients. Examples of LGL nodes on a triangle with orders from 1 to 9 are shown in figure 3.1.

Assume the polynomial order is $P$ and $x^k_j$, $j = 1, \cdots, N_p$ are the LGL nodes defined on $\Omega_k$, then the solution in $\Omega_k$ can be represented as the nodal expansion

$$
q^k(x, t) = \sum_{j=1}^{N_p} q^k(x^k_j, t) \ell^k_j(x),
$$

where $N_p = (P + 1)(P + 2)/2$ is the total number of nodes or unknowns in $\Omega_k$ and $q^k = [q^k(x^k_1, t), \ldots, q^k(x^k_{N_p}, t)]^T$. The modal expansion of the solution is introduced,

$$
q^k(x, t) = \sum_{j=1}^{N_p} \hat{q}^k_j(t) \psi^k_j(x),
$$

where $\hat{q}^k = [\hat{q}^k_1(t), \ldots, \hat{q}^k_{N_p}(t)]^T$ are the modal expansion coefficients and $\psi^k_j(x)$ are the orthonormal modal polynomial basis in $\Omega_k$. In PHORCE, $\psi_j(x)$ is chosen to be the normalized Legendre polynomial,

$$
\psi_j(x) = \tilde{P}_{j-1}(x),
$$

where $j = 1, \cdots, N_p$. For more details of how to construct $\tilde{P}_{j-1}(x)$ in triangle elements, please refer to Hesthaven and Warburton [2007]. The Vandermonde matrix $\mathcal{V}^k$ is defined as

$$
\mathcal{V}^k_{ij} = \psi^k_j(x_i),
$$

where $i, j = 1, \cdots, N_p$, such that

$$
q^k = \mathcal{V}^k \hat{q}^k.
$$

In the nodal DG method [Hesthaven and Warburton, 2007], all computations can be performed on the reference triangle $I = \{r = (r, s) | (r, s) \geq -1; r + s \leq 0\}$. Since the mapping for triangular elements is an affine transformation [Veblen and Young, 1918, Berger, 1987], the Jacobians of this mapping are constant in a triangle. This mapping is shown in Figure 3.2 and described in equations 3.14 and 3.15,

$$
x = -\frac{r + s}{2} v^1 + \frac{r + 1}{2} v^2 + \frac{s + 1}{2} v^3,
$$

with
Figure 3.1: Example of Legendre-Gauss-Lobatto (LGL) nodes on a triangle for orders from 1 to 9
\[(x_r, y_r) = \frac{v^2 - v^1}{2}, \quad (x_s, y_s) = \frac{v^3 - v^1}{2}. \quad (3.15)\]

The Jacobians of this mapping are described in equations 3.16 and 3.17,
\[r_x = \frac{y_s}{J}, \quad r_y = -\frac{x_s}{J}, \quad s_x = -\frac{y_r}{J}, \quad s_y = \frac{x_r}{J}, \quad (3.16)\]
\[J = x_r y_s - x_s y_r. \quad (3.17)\]

\[\Omega_k \quad v^3 \quad v^2 \quad (1,-1) \quad (-1,-1) \quad (-1,1) \quad \Omega_k \]

\[\mathbf{v}^1 \quad \mathbf{v}^2 \quad \mathbf{v}^3 \quad \mathbf{v}^4 \]

**Figure 3.2**: Affine transformation between physical element \(\Omega_k\) and reference element \(I\)

For the remainder of this chapter, any variable or matrix without the element index superscript \(k\) is defined on \(I\). Now, equation 3.8 can be written as
\[\mathcal{M}^k \frac{\partial q^k}{\partial t} - (\mathbf{S}^k)^T f^k - \mathcal{M}^k s^k = -\int_{\partial \Omega_k} \hat{n} \cdot \mathbf{f}^* \ell^k_i (x) d\Omega, \quad (3.18)\]
where \(f^k = [f^k(x^k_1, t), \ldots, f^k(x^k_{N_p}, t)]^T\), and the mass matrix, \(\mathcal{M}^k\), and stiffness matrix, \(\mathbf{S}^k\), are defined as
\[\mathcal{M}_{ij}^k = \int_{\Omega_k} \ell_i^k \ell_j^k d\Omega = J^k \int_I \ell_i \ell_j dI = J^k \mathcal{M}, \quad (3.19)\]
\[\mathbf{S}_{ij}^k = \int_{\Omega_k} \ell_i^k \nabla \ell_j^k d\Omega = \int_I \ell_i \nabla \ell_j dI = J^k \mathbf{S}, \quad (3.20)\]
respectively, where
\[
\mathbf{r}_x^k = \begin{bmatrix} r_x & s_x \\ r_y & s_y \end{bmatrix}^k.
\] (3.21)

Only reference mass, stiffness matrices, and geometric factors need to be stored. Equation 3.18 can then be written as a semi-discrete form,
\[
\frac{\partial q^k}{\partial t} = (\mathcal{M}^{-1} \mathcal{S}^T) \mathbf{f}^k + s^k - \sum_{f=1}^{3} \text{LIFT}^k_f (\mathbf{n}^k_f \cdot \mathbf{f}^*_f),
\] (3.22)

where \( \mathbf{f}^*_f = [\mathbf{f}^*_f(x_{f,1}, t), \ldots, \mathbf{f}^*_f(x_{f,N_{fp}}, t)]^T \), \( N_{fp} = P + 1 \) is the total number of nodes on one edge, and the lift operator is defined as,
\[
\text{LIFT}^k_f (\mathbf{n}^k_f \cdot \mathbf{f}^*_f) = \mathcal{M}^k \int_{\partial \Omega^k_f} \mathbf{n}^k_f \cdot \mathbf{f}^*_f \ell^{k,f}_j (\mathbf{x}) d\Omega
\]
\[
= J^k \mathcal{M}^{-1} \left( \int_{\partial \Omega^k_f} \ell^{k,f}_i d\partial \Omega \right) \mathbf{n}^k_f \cdot \mathbf{f}^*_f
\]
\[
= \frac{J^k_f}{J^k} \mathcal{M}^{-1} \left( \int_{\partial I^f} \ell^{I}_i d\partial I \right) \mathbf{n}^k_f \cdot \mathbf{f}^*_f
\]
\[
= \frac{J^k_f}{J^k} \text{LIFT}_f (\mathbf{n}^k_f \cdot \mathbf{f}^*_f),
\] (3.23)

where \( \ell^{k,f}_j \) is the basis function defined on edge \( f \) of element \( \Omega_k \). \( \ell^{I}_j \) is the basis function defined on edge \( f \) in \( I \). \( J^k_f \) is the transformation Jacobian along edge \( f \) of \( \Omega_k \). \( J^k_f \) can also be seen as the ratio between the length of \( \Omega^k_f \) and \( I^f \).

### 3.3 Initial Conditions

In this section, two different ways of initializing the DG presentation of solution \( q_h(\mathbf{x}) \) are discussed. The subscript \( h \) is re-introduced in this section in order to distinguish from the notation of exact initial solution \( q_e(\mathbf{x}) \). In PHORCE, the initialization is done by evaluation \( q_e(\mathbf{x}) \) on the LGL nodes \( x^k_i \), \( i = 1, \ldots, N_p \), in each element \( \Omega_k \),
\[
q_h(x^k_i) = q_e(x^k_i).
\] (3.24)

One immediate concern about this initialization is that this will introduce a conservation error when the order of \( q_e \) is higher than \( P \). For example, when \( q_e \) is a concave function in a triangle element, and \( q_e(\mathbf{v}^i) = a \), \( i = 1, 2, 3 \). Obviously, \( \bar{q}_e > a \), where \( \bar{q}_e \) is the cell average of the exact solution. However, a \( P1 \) \( (N_p = 3) \) initialization using equation 3.24 gives \( q_h(x_i) = a \), \( i = 1, 2, 3 \), which corresponds to an approximated cell average of \( \bar{q}_h = a \). A conservation error of \( (\bar{q}_h - \bar{q}_e) \) is introduced by this initialization method. To fix this, one can use a higher order approximation of the initial solution and project it on the
lower order computational nodes. This has been tested on the isentropic vortex using Euler equations. The details of isentropic vortex setups is described in section 5.2.1. Here, only two initializations will be compared. The exact solution of this case is an exponential function, which cannot be initialized exactly regardless what polynomial order is chosen. However, the initial conservation error can still be reduced significantly by using a high-order initial approximation. In this test, errors are compared between initialization polynomial orders of $P$ and $P + 5$, with $P = 1, 2, 3$, and are shown in Figure 3.3. The tests are performed on a sequence of grids with systematic refinements, that have 16, 64, 256, 1024, 4096 elements. A demonstration of mesh refinements is shown in Figure 5.1. The $L_2$ and $L_\infty$ norms of the errors are calculated. Details of the calculating $L_2$ and $L_\infty$ norms are discussed in chapter 5. Here we just point out that only small differences between the errors associated with two different initialization methods can be observed for $P = 1$ and on coarse refinement levels for $P = 2, 3$, and almost no differences between the errors can be observed on fine refinement levels for for $P = 2, 3$. This indicates that the initial conservation error is very small when the mesh resolution and the polynomial order are high. In addition, even when there is a difference between two initialization methods, the higher order initial approximation is always less accurate compared to the initialization described in equation 3.24. This is because the higher order initialization will introduce discontinuities in the initial conditions, which is corresponding to higher numerical diffusion when the mesh is coarse and numerical order $P$ is low. Furthermore, in cases the exact initial solution has sharp gradients and regions where the density or pressure is low, the “accurate” way of initialization could result in negative density or pressure in the initial conditions. Consequently, in PHORCE, the simulation is initialized using equation 3.24.

![Figure 3.3](image)

Figure 3.3: Convergence tests of isentropic vortex with different initialization methods on a sequence of grids with systematic refinements using (a) $P1$, (b) $P2$, and (c) $P3$ NDG algorithms. Formal orders of accuracy are indicated by the slopes with magenta lines.
3.4 Boundary Conditions

In PHORCE, the boundary conditions are applied on exterior traces of the solution $q$ on boundary edges. Details of how to set up the boundary conditions in PHORCE are described in Appendix A. To illustrate this, assuming $P2$ NDG method is used, let $q^+$ to be the exterior trace of a boundary edge, and $q^-$ to be the interior trace, as shown in Figure 3.4. $q^-$ and $q^+$ are evaluated at the same location on the interface, but from different solution. $q^-$ is evaluated from the numerical solution in the element that is adjacent to the boundary, while $q^+$ is evaluated based on the type of boundary condition that is applied. The information from the boundary propagates into the computational domain through the calculation of numerical flux, which has been described in equation 3.6.

Several types of boundary conditions have been implemented in PHORCE, and are described in this section.

3.4.1 Prescribed Boundary

Prescribed boundary refers to a predefined boundary value that is independent of the solution in the computational domain. For example, in the isentropic vortex test (section 5.2.1), the boundary condition is prescribed with an exact solution. This can be described as,

$$q^+ = q_e.$$  \hspace{1cm} (3.25)
3.4.2 Slip Wall

Slip wall refers to the boundary type that reflects fluid velocity that is normal to the wall but has no friction force, so the fluid velocity that is parallel to the wall can travel freely. This can be described as,

\[ u^+ - u^- = -u^+ - u^- \]  
\[ u^+ = u^- \]  
\[ (3.27) \]

3.4.3 Non-slip Wall

Non-slip wall refers to the boundary type that reflects fluid velocity that is normal to the wall and has friction force, so the fluid velocity that is parallel to the wall cannot slip through. This can be described as,

\[ u^+ - u^- = -u^+ - u^- \]  
\[ u^+ = 0 \]  
\[ (3.29) \]

3.4.4 Conducting Wall

For conducting wall, the magnetic field is zero on the normal direction to the wall and is not affected on the parallel direction to the wall. This can be described as,

\[ B^+ = B^- \]  
\[ (3.31) \]

3.4.5 Periodic Boundary

Periodic boundary condition (PBC) refers to the boundary that the information leaves on a side of the domain, re-appears on the other side. Two types of PBC are implemented in PHORCE, classical PBC and azimuthal PBC, and are described here.

Classical Periodic Boundary

In classical periodic boundary condition, the two periodic boundaries are parallel to each other, so the information that leaves on one side re-appears exactly on the other side. An example of classical PBC is shown in Figure 3.5, where the left and right boundaries are
periodic to each other, elements Ωₐ and Ωₐ are connected to each other through the PBC. This is described as,

\begin{align}
q^{a,+} &= q^{b,-}, \\
q^{b,+} &= q^{a,-}.
\end{align}

\[ \Omega \]

Figure 3.5: Illustration of classical periodic boundary condition setups

**Azimuthal Periodic Boundary**

In azimuthal periodic boundary condition, the two periodic boundaries are parallel to each other on the azimuthal direction, as illustrated in Figure 3.6. The scalar fields in the azimuthal PBC are the same as in classical PBC, as described in equations 3.32 and 3.33. The angle of the azimuthal PBC is θ. Elements Ωₐ and Ωₐ are connected to each other through the azimuthal PBC, so that all vector fields pass through this PBC will be rotated so that they are consistent on the azimuthal direction. Taking velocity field as an example, the azimuthal PBC is described as,

\begin{align}
\mathbf{u}^{a,+} &= \mathcal{R}(\theta)\mathbf{u}^{b,-}, \\
\mathbf{u}^{b,+} &= \mathcal{R}(-\theta)\mathbf{u}^{a,-},
\end{align}

where \( \mathcal{R}(\theta) \) is the rotation matrix that rotates a vector field counterclockwise by an angle of θ degrees, and is defined as,

\[ \mathcal{R}(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \]
3.5 Time Stepping

To advance the simulation in time, an explicit fourth-order five-stage strong stability-preserving Runge-Kutta (SSP-RK) scheme [Spiteri and Ruuth, 2002, Hesthaven and Warburton, 2007]
is applied. The scheme is given as,

\[
v^{(1)} = q^{n} + 0.3917522700392 \Delta t \mathcal{L}(q^{n}, t^{n}),
\]

\[
v^{(2)} = 0.44437049406734 q^{n} + 0.55562950593266 v^{(1)}
+ 0.36841059262959 \Delta t \mathcal{L}(v^{(1)}, t^{n} + 0.3917522700392 \Delta t),
\]

\[
v^{(3)} = 0.62010185138540 q^{n} + 0.37989814861460 v^{(2)}
+ 0.25189177424738 \Delta t \mathcal{L}(v^{(2)}, t^{n} + 0.58607968896780 \Delta t),
\]

\[
v^{(4)} = 0.17807995410773 q^{n} + 0.82192004589227 v^{(3)}
+ 0.54497475021237 \Delta t \mathcal{L}(v^{(3)}, t^{n} + 0.47454236302687 \Delta t),
\]

\[
q^{n+1} = 0.0068325884039 q^{n} + 0.51723167208978 v^{(2)}
+ 0.12759831133288 v^{(3)} + 0.34833675773694 v^{(4)}
+ 0.0846041638212 \Delta t \mathcal{L}(v^{(3)}, t^{n} + 0.47454236302687 \Delta t)
+ 0.22600748319395 \Delta t \mathcal{L}(v^{(4)}, t^{n} + 0.93501063100924 \Delta t),
\]

where \( \Delta t \) is the time-step, \( t^{n+1} = t^{n} + \Delta t \), \( q^{n} \) is the numerical solution at time \( t^{n} \), and

\[
\mathcal{L}(q, t) = \frac{\partial q}{\partial t}.
\]

\( \Delta t \) is calculated as

\[
\Delta t \leq \frac{2}{3} C_{cfl} \min \Delta r_i \min R_k,
\]

where \( r_i \), \( R_k \), and \( C_{cfl} \) were introduced in section 2.5.2.

### 3.6 Discussion of Limiters

DG methods can be used to numerically approximate neutral fluids and plasmas without further modification when the solution is smooth or only contains weak discontinuities. However, when strong discontinuities are present, Gibbs phenomenon [Wilbraham, 1848] appears in those regions and generates numerical instabilities. When this is the case, limiters, filters, and artificial viscosity can be applied to diffuse numerical oscillation and thus maintain numerical stability. For multi-dimensional equations, limiters are proven to be more effective and computationally affordable compared to the other two. In the later sections in this chapter, several multi-dimensional limiters along with troubled-cell detecting methods that are implemented in PHORCE will be described.

#### 3.6.1 Shock Sensor

Limiting is an interference to the computation. Applying limiters to regions with strong discontinuities can diffuse spurious oscillations in solutions and thus stabilize the numerical
simulation. However, if limiters are applied to regions where the solution is smooth, unwanted numerical errors will be introduced along with unnecessary computational effort. As a result, before applying limiters, detecting which regions need limiting is crucial. Two shock sensors have been implemented in PHORCE and will be detailed in this section.

**KXRCF Shock Sensor**

Several discontinuity detecting methods are studied and compared in Qiu and Shu [2005a]. The KXRCF shock sensor [Krivodonova et al., 2004] is found to be more effective and robust than others. This method detects troubled-cells by measuring the discontinuity at the cell interface. For each element $\Omega_j$, a troubled-cell indicator is calculated as,

$$I_j = \int_{\partial\Omega_j} (Q_j - Q_{nb_j}) d\Omega,$$  \hspace{1cm} (3.40)

where $Q_j$ is the numerical solution or a derived quantity in $\Omega_j$, and $Q_{nb_j}$ is the numerical solution or a derived quantity in the face neighboring cells of $\Omega_j$. If the solution is smooth, then $I_j$ is on the order of $O(h^{p+1})$. If the solution is discontinuous, then $I_j$ is on the order of $O(h)$. Hence,

$$I_j = \begin{cases} O(h^{p+1}), & Q \text{ is smooth}, \\ O(h), & Q \text{ is discontinuous}. \end{cases}$$  \hspace{1cm} (3.41)

In implementation, $I_j$ is normalized by the convergence rate and the solution in $\Omega_j$ as,

$$I_j = \frac{\int_{\partial\Omega_j} (Q_j - Q_{nb_j}) d\Omega}{h^{(p+1)/2} |\partial\Omega_j| \|Q_j\|}.$$  \hspace{1cm} (3.42)

The discontinuity detection becomes,

$$\begin{cases} \text{if } I_j < 1, & Q \text{ is smooth}, \\ \text{if } I_j > 1, & Q \text{ is discontinuous}. \end{cases}$$  \hspace{1cm} (3.43)

For systems of equations, the solution of density or pressure are often used to calculated the troubled-cell indicator.

**FS Shock Sensor**

FS shock sensor proposed by Fu and Shu [2017] recently is also investigated in this study. Unlike KXRCF shock sensor where the troubled-cell indicator is calculated by measuring the jump in solution at cell interfaces, the troubled-cell indicator in FS shock sensor is calculated
by measuring the differences in the cell averages of the solution evaluated on $\Omega_j$ from $Q_j$ and $Q_{nbj}$,

$$I_j = \frac{\sum_{i=1}^{N_b} |Q_j - \overline{Q}_i|}{\max_{i \in \{1,2,\ldots,N_b\}} \{|Q_j|, |\overline{Q}_i|\}},$$  \hspace{1cm} (3.44)

where $N_b$ is the number of neighboring cells of $\Omega_j$, $Q_{i \in \{1,2,\ldots,N_b\}} \subset Q_{nbj}$,

$$\overline{Q}_i = \frac{1}{|\Omega_j|} \int_{\Omega_j} Q_i(x) d\Omega,$$  \hspace{1cm} (3.45)

$$\overline{Q}_i = \frac{1}{|\Omega_i|} \int_{\Omega_i} Q_i(x) d\Omega,$$  \hspace{1cm} (3.46)

i.e., $\overline{Q}$ is the cell average of $*$ evaluated on the target cell $\Omega_j$, and $\overline{Q}$ is the cell average of $*$ evaluated on its own cell. Hence,

$$\begin{cases} 
\text{if } I_j < C_P, & Q \text{ is smooth}, \\
\text{if } I_j > C_P, & Q \text{ is discontinuous},
\end{cases}$$  \hspace{1cm} (3.47)

where $C_P$ is a constant that depends only on polynomial degree $P$. Following Fu and Shu [2017], $C_P$ is calculated as,

$$C_P = 0.015 \times 2^P.$$  \hspace{1cm} (3.48)

It is worth mentioning here that the extrapolation of the solution is nontrivial and will be discussed in later sections of this chapter. Although the extrapolation in the FS shock sensor requires slightly more computational effort compared to the KXRCF shock sensor, the former is found to be more accurate. In other words, the FS shock sensor reports less troubled-cells while maintaining numerical stability compared to the KXRCF shock sensor for all tests performed in this study. The list of troubled-cells is then provided to the chosen limiters.

### 3.6.2 Slope Limiter

After the list of troubled-cells is generated using a shock sensor, a limiter can be chosen to add numerical diffusion to troubled-cells thus maintaining numerical stability. As mentioned earlier, the limiting process is an interference and will introduce errors. Thus, the methodology of the limiter has a significant impact on the accuracy of numerical computation in troubled-cells.

Slope or moment limiters are widely used in the DG community [Hoteit et al., 2004, Krivodonova, 2007, Yang and Wang, 2009, Kuzmin, 2010, 2013, 2014], due to their simplicity and computational efficiency. A high-order slope limiter proposed by Moe et al. [2015] is implemented
in this framework. Assuming \( \Omega_0 \) is the element in which the solution needs to be limited, \( \Omega_1 \), \( \Omega_2 \), \( \Omega_3 \) are its face neighbors. This is illustrated in Figure 3.7. The algorithm is summarized as follows:

![Figure 3.7: The stencil of \( \{\Omega_i \in \{0,1,2,3\}\} \).](image)

1. For each element in \( \{\Omega_i \in \{0,1,2,3\}\} \) and each component \( \ell \) of primitive variables \( W = \{W^1, \cdots, W^{N_{eq}}\} \), where \( N_{eq} \) is the number of independent primitive variables (or number of equations) in the system, compute the cell maximum of \( W^\ell \) as \( W^\ell_{M_i} \), and cell minimum of \( W^\ell \) as \( W^\ell_{m_i} \).

2. For element \( \Omega_0 \) and each component \( \ell \) of primitive variables \( W \), compute the upper and lower bound over its neighbors,

\[
M^\ell_0 = \max \left\{ \tilde{W}^\ell_0 + \alpha(h), \max_{i \in \{1,2,3\}} \left\{ W^\ell_{M_i} \right\} \right\}, \tag{3.49}
\]

\[
m^\ell_0 = \min \left\{ \tilde{W}^\ell_0 - \alpha(h), \min_{i \in \{1,2,3\}} \left\{ W^\ell_{m_i} \right\} \right\}, \tag{3.50}
\]

where \( \alpha(h) = O(h^{1.5}) \) and needs to be tuned for different problems and thus calculated as,

\[
\alpha(h) = C_\alpha h^{1.5}, \tag{3.52}
\]

where \( C_\alpha \) is set empirically for each problem.
3. For element $\Omega_0$, compute,

$$ \theta_{M0} = \min_{\ell} \left\{ \phi \left( \frac{M_0^\ell - \bar{W}_0^\ell}{W_{M0} - \bar{W}_0} \right) \right\} , $$

$$ \theta_{m0} = \min_{\ell} \left\{ \phi \left( \frac{m_0^\ell - \bar{W}_0^\ell}{W_{m0} - \bar{W}_0} \right) \right\} , $$

where $\phi$ is calculated as,

$$ \phi(x) = \min \left\{ 1, \frac{x}{1.1} \right\} . \quad (3.55) $$

4. For element $\Omega_0$, compute,

$$ \theta_0 = \min \{ 1, \theta_{m0}, \theta_{M0} \} . $$

5. For element $\Omega_0$, the limited solution of conserved variables $Q$ is calculated as,

$$ Q^*_0 = \bar{Q}_0 + \theta_0 \left( Q_0 - \bar{Q}_0 \right) . $$

This limiter uses primitive variables to calculate the limiting coefficient $\theta$ as this performs better than using conserved variables according to Moe et al. [2015]. However, the limiter is still applied to conserved variables so that conservation is not violated.

For unstructured meshes, it is suggested in Moe et al. [2015] to consider all elements that are sharing the same vertices as neighbors. However, this will add additional elements in the halo communication in distributed memory parallelization, which makes this limiter less appealing. Thus, in this work, only face neighbors are considered. This will make this limiter slightly more diffusive. This slope limiter works well on cases with strong shocks, as the order of accuracy will degrade to 1st order in regions with shocks regardless of which limiter is chosen. Thus a computationally inexpensive limiter is a good choice. However, this limiter is found to be too diffusive to capture small scale features in tests with hydrodynamic instabilities. Also the parameter $C_\alpha$ is chosen empirically, and found to be quite sensitive to different systems of equations and even for different problems of the same system of equations. As a result, this limiter is only recommended for testing purposes in PHORCE.

### 3.6.3 WENO Limiter

Weighted essentially non-oscillatory (WENO) limiter, originated from the finite volume community [Liu et al., 1994, Jiang, 1995, Friedrich, 1998, Hu and Shu, 1999], and is applied to DG in Qiu and Shu [2004, 2005c,b], Zhu et al. [2008], Zhu and Qiu [2012], Zhong and Shu [2013]. In this work, a WENO limiter described in Zhu et al. [2013] is implemented and will
be detailed in this section. This limiter is a suitable choice as it only requires information from face neighbors for the WENO reconstruction, making it computationally more efficient compared to some other limiters. Generally, a DG WENO limiter has three steps, namely,

1. Identify the troubled-cells

2. Calculate the smoothness indicators for each troubled-cell and its neighbors

3. Reconstruct the polynomial solution in the troubled-cell using the original polynomials from the troubled-cell and its neighbors, and polynomial weights calculated from smoothness indicators

Step 1 has been described in Section 3.6.1. Step 2 and 3 will be discussed later in this section.

**Smoothness Indicator**

Once the troubled-cell is identified, a smoothness indicator needs to be calculated for the troubled-cell and its neighbors. The calculated smoothness indicator will then be used to determine the weights for each polynomial of the reconstruction. The calculation of the smoothness indicator is the most important step as it directly affects the accuracy of the WENO reconstruction and is also the most computationally expensive step in the WENO limiting process. In PHORCE, the smoothness indicator described in Jiang [1995], Zhu et al. [2013] is implemented. For troubled-cell $\Omega_0$, the smoothness indicators for the stencil $\{\Omega_i \in \{0,1,2,3\}\}$ in Figure 3.7 are calculated as follows,

$$SI_i = \sum_{l=1}^{P-1} |\Omega_0|^{l-1} \int_{\Omega_0} \left( \frac{\partial^l}{\partial x^l \partial y^l} \tilde{Q}_i(x,y) \right)^2 d\Omega, \quad (3.58)$$

where $\ell = l_1 + l_2$,

$$\tilde{Q}_i(x,y) = Q_i(x,y) - \frac{1}{|\Omega_0|} \int_{\Omega_0} Q_i(x,y) d\Omega + \frac{1}{|\Omega_0|} \int_{\Omega_0} Q_0(x,y) d\Omega, \quad (3.59)$$

where $\frac{1}{|\Omega_0|} \int_{\Omega_0} Q_i(x,y) d\Omega$ is the cell average of the extrapolation of $Q_i(x,y)$ on $\Omega_0$, while $\frac{1}{|\Omega_0|} \int_{\Omega_0} Q_0(x,y) d\Omega$ is the cell average of $Q_0(x,y)$ on $\Omega_0$.

**Polynomial Extrapolation**

The extrapolation of $Q_i(x,y)$ on $\Omega_0$ is nontrivial. One need to avoid have multiple nodes on line $s = 1$ in the reference domain $I$. This is because line $s = 1$ in the reference domain has...
$P_0$ projection. For more detail, please refer to Hesthaven and Warburton [2007]. To avoid this, the interface between the two triangles needs to be $\{(−1, −1), (1, −1)\}$ in the reference domain. This can be achieved through the rotational Vandermonde matrices described in Chapter 4. Here are some examples of poor extrapolations and their corrected ones. The solution in $\Omega_i$ is initialized as,

$$Q_i(x, y) = \sin \left( \frac{x}{\pi} + 0.2 \right) \cos \left( \frac{y}{\pi} - 0.244 \right). \quad (3.60)$$

**Example 1.** One edge of the $\Omega_0$ lies on the line $s = 1$ in the reference domain when the hypotenuse is the interface.

**Example 2.** One edge of the $\Omega_0$ lies on the line $s = 1$ in the reference domain when $\{(−1, −1), (−1, 1)\}$ is the interface.

Figures 3.8 (example 1) and 3.11 (example 2) are the configurations of $\Omega_i$ and $\Omega_0$ in physical domain and reference domain. Figures 3.9 (example 1) and 3.12 (example 2) are the solution in $\Omega_i$ and its extrapolation in $\Omega_0$ for $P1$ and $P5$. The extrapolated solution are incorrect. Figure 3.10 (example 1) and 3.13 (example 2) are the corrected versions of extrapolations.

![Figure 3.8](image-url)  
**Figure 3.8:** Configurations of $\Omega_i$ and $\Omega_0$ for example 1 in physical (left) and reference (right) domain.
WENO Limiting

Once the smoothness indicators are calculated for the stencil \( \{ \Omega_i \in \{0,1,2,3\} \} \), a positive weight can be calculated for each polynomial,

\[
\omega_i = \frac{\bar{\omega}_i}{\sum_{j=0}^{3} \bar{\omega}_j},
\]  
(3.61)
Figure 3.11: Configurations of $\Omega_i$ and $\Omega_0$ for example 2 in physical (left) and reference (right) domain.

Figure 3.12: Incorrect extrapolation from the solution in $\Omega_i$ to $\Omega_0$ for example 2 using P1 (left) and P5 (right) DG method.

Figure 3.13: Corrected extrapolation from the solution in $\Omega_i$ to $\Omega_0$ for example 2 using P1 (left) and P5 (right) DG method.

$$\mathcal{w}_i = \frac{\gamma_i}{(\epsilon + SI)^2},$$

(3.62)

where $i \in \{0, 1, 2, 3\}$, and $\epsilon$ is a small positive number to avoid the denominator being zero when $SI$ is zero, $\gamma_i$ is a constant weight for $\Omega_i$. In PHORCE, $\gamma_0 = 0.97$, $\gamma_1 = \gamma_2 = \gamma_3 = 0.01$. 
Now, the solution in troubled-cell $\Omega_0$ is replaced by a reconstructed polynomial,

$$Q^*_0 = \sum_{i=0}^{3} \omega_i \dot{Q}_i.$$  \hfill (3.63)

For a system of equations, where $Q_{i \in \{0,1,2,3\}} = [Q_i^1, \cdots, Q_i^{N_{eq}}]^T$ are the conserved variables, the above process is performed for each of the characteristic variables of the system. The conserved variables can be projected into the characteristic fields by,

$$C_i = LQ_i,$$  \hfill (3.64)

where $C_{i \in \{0,1,2,3\}} = [C_i^1, \cdots, C_i^{N_{eq}}]^T$ are the characteristic variables, $L$ is the array of left eigenvectors of the flux Jacobian matrix with respect to conserved variables, which is described in section 2.1. Once the limiting process is done, the limited characteristic variables, $C^*_0$, can then be projected into the physical space,

$$Q^*_0 = RC^*_0,$$  \hfill (3.65)

where $R$ is the array of the corresponding right eigenvectors.

### 3.6.4 Positivity Preserving Limiter

Slope and WENO limiters are applied when there are significant numerical oscillations in the simulation. However, these two limiters do not guarantee the positivity of density and pressure, especially in plasma simulations in high-energy-density regimes where very large gradients could exist in density and temperature. Thus, it is necessary to include a positivity preserving (PP) limiter. A high-order PP limiter, originally proposed in Zhang and Shu [2010], then further improved in robustness in Cheng et al. [2013], is implemented in PHORCE. For each element $\Omega_i$, PP limiter is formulated as follows.

1. Enforce positivity of density by replacing $\rho$ with

$$\rho^* = \theta_1 (\rho - \bar{\rho}) + \bar{\rho},$$  \hfill (3.66)

with

$$\theta_1 = \min_{x \in \Omega_i} \left\{ 1, \frac{\bar{\rho} - \epsilon}{\bar{\rho} - \rho(x)} \right\},$$  \hfill (3.67)

where $\epsilon$ is a small number, $\rho(x)$ are the nodal solutions in $\Omega_i$. 

2. Enforce positivity of pressure $p$ by replacing conserved variables $Q$ with,

$$Q^* = \theta_2(\hat{Q} - \bar{Q}) + \bar{Q},$$

(3.68)

where $\hat{Q}$ represents the original conserved variables with $\rho$ replaced by $\rho^*$, and $\theta_2$ is calculated as

$$\theta_2 = \begin{cases} 1, & \text{if } p(\hat{Q}) \geq 0 \\ \frac{\rho(\hat{Q})}{\rho(\hat{Q}) - \rho(Q)}, & \text{if } p(\hat{Q}) < 0 \end{cases}$$

(3.69)

This PP limiter is high-order accurate and computationally inexpensive, and is generic for any hydrodynamic system that is currently implemented in PHORCE. It also damps certain degrees of numerical oscillation. In Chapter 5, examples will be shown that this PP limiter alone is capable of maintaining numerical stability for some cases with a large amount of numerical oscillations.

### 3.6.5 MOF Bound Limiter

For simulations involving multi-fluid materials in compressible flow, the continuity equation is replaced by MOF equations 2.75 and 2.76. The mass fraction $\phi_m$ can be used to calculate quantities in the mixture region between the two fluids, as discussed in section 2.7.3. However, similar to LS and VOF method, MOF method suffers from the unbounded issue of the marker function $\phi_m$, i.e. $\phi_m$ is not bounded between $[0, 1]$. Motivated by PP limiter described in the above section, a high-order MOF bound limiter is proposed in this work.

The objectives are,

1. Ensuring $\phi_m$ is bounded in $[0, 1]$

2. Ensuring the conservation of $\rho\phi_m$. This will also ensure the conservation of $\rho(1 - \phi_m)$ as $\rho$ is conserved.

3. The limiting process only changes $\phi_m$ without modifying $\rho$ or other conserved quantities, e.g. momentum, total energy, etc.

Similar to PP limiter, when the lower bound is violated, a limiting coefficient needs to be calculated in order to shrink the solution polynomial of the conserved quantities while maintaining conservation (i.e., keeping the same cell averages). Unlike the PP limiter, where only lower bounds exist, there is also an upper bound of 1 in $\phi_m$. In order to also achieve objectives 2 and 3, $m_1 = \rho\phi_m$ is replaced by $m_1^*$ that is bounded by $[0, \rho]$ and also has the same cell average as $m_1$. Then $\phi_m$ is replaced by $\phi_m^* = \frac{m_1^*}{\rho}$. 
In order to bound $m_1^*$ in $[0, \rho]$ in each element, assuming $P = 1$ and one-dimensional for demonstration purposes, three cases are considered. In the following figures, the lower and upper bounds are indicated as red lines, the original solution of $m_1$ is the gray line, and limited solution $m_1^*$ is the blue line. The cell average of $m_1$ is the cell center value in P1 and is indicated as a black dot.

**Case 1.** $m_1$ violates the lower bound and needs to be shrunk as shown in Figure 3.14.

**Case 2.** $m_1$ violates the upper bound and needs to be shrunk as shown in Figure 3.15.

**Case 3.** $m_1$ violates the upper bound and needs to be expanded as shown in Figure 3.16.

The algorithm of MOF bound limiter is formulated using the following steps for each element $\Omega_i$,

1. Calculate the shrinking coefficient $\theta_c$, when lower bound is violated (Case 1), or upper bound is violated in the region of $m_1 > \bar{m}_1$ (Case 2), where $\bar{m}_1$ is the cell average of $m_1$

\[
\theta_c = \min\{\theta_1, \theta_2\},
\]  

(3.70)
where

\[
\theta_1 = \min_{x \in \Omega} \left\{ 1, \left| \frac{\bar{m}_1}{m_1 - m_1(x)} \right| \right\},
\]

(3.71)

\[
\theta_2 = \min_{x \in \Omega, \& m_1 > \bar{m}_1} \left\{ 1, \left| \frac{\rho(x) - \bar{m}_1}{m_1(x) - \bar{m}_1} \right| \right\}.
\]

(3.72)

2. Calculate the expanding coefficient \(\theta_e\), when upper bound is violated in the region of \(m_1 < \bar{m}_1\) (Case 3),

\[
\theta_e = \max_{x \in \Omega, \& m_1 < \bar{m}_1} \left\{ 1, \left| \frac{\bar{m}_1 - \rho(x)}{m_1(x) - \bar{m}_1} \right| \right\}.
\]

(3.73)

3. Calculate \(m_1^*\),

\[
m_1^* = \begin{cases} 
\theta_c(m_1 - \bar{m}_1) + \bar{m}_1, & \text{if } \theta_c < 1, \\
\theta_e(m_1 - \bar{m}_1) + \bar{m}_1, & \text{if } \theta_c = 1 \& \theta_e > 1.
\end{cases}
\]

(3.74)

4. Replace \(\phi_m\) with

\[
\phi_m^* = \begin{cases} 
\frac{m_1^*}{\rho}, & \text{if } \theta_c = 1 \text{ or } \theta_e = 1, \\
1 \text{ for } \phi_m > 1, & \text{otherwise}.
\end{cases}
\]

(3.75)

The MOF algorithm does not affect the total mass conservation even when \(\phi_m\) is unbounded. This is because the introduced equations \((2.75, 2.76)\) can be linearly combined to reconstruct the original continuity equation, where \(\phi_m\) does not appear. The total density \(\rho = \rho\phi_m + \rho(1 - \phi_m)\) is not affected by the solution of \(\phi_m\). With this MOF bound limiter, \(\phi_m\) is bounded between \([0, 1]\) while the conservation of the introduced conserved variables \(\rho\phi_m\) and \(\rho(1 - \phi_m)\) is also not violated.
3.6.6 Effect of Limiters on $\nabla \cdot B$ Errors

In the GLM-MHD equations described in section 2.5.2, $\psi$ is an introduced variable that deals with the divergence cleaning of the magnetic field $B$, and is not processed by limiters in PHORCE. For each time-step or RK stage, $\psi$ is calculated based on the error of $\nabla \cdot B$ from the previous time-step or RK stage before the limiting process. This is shown in the following pseudo-code.

**Algorithm 1:** Calculates $\psi$ before limiting other variables

```plaintext
// initializing conserved variables
Q = Q_0 = [ρ_0, ρ_0 u_0, ε_0, B_0, ψ_0]^T;
while t < t_{final} do
    // calculate right hand sides of the PDEs
    rhs = ∂Q/∂t;
    // update the conserved variables
    Q = Q + rhs * Δt;
    // apply limiters to the updated conserved variables except $\psi$
    [ρ, ρu, ε, B]^T = [ρ*, (ρu)*, ε*, B*]^T;
end
```

$B$ is modified in the limiting process, and an additional divergence error $\nabla B'$ may be introduced. $\psi$ is updated before the limiting process, thus will not be able to clean this $\nabla B'$ error in the rhs calculation in the next time-step. One way to deal with this error, is to calculate the updated $\psi$ after the limiting process, as indicated in the following pseudo-code.

**Algorithm 2:** Calculates $\psi$ after limiting other variables

```plaintext
// initializing conserved variables
Q = Q_0 = [ρ_0, ρ_0 u_0, ε_0, B_0, ψ_0]^T;
while t < t_{final} do
    // calculate right hand sides of the PDEs except the equation of $\psi$
    [rhs_ρ, rhs_ρu, rhs_ε, rhs_B]^T = [∂ρ/∂t, ∂ρu/∂t, ∂ε/∂t, ∂B/∂t]^T;
    // update the conserved variables except $\psi$
    [ρ, ρu, ε, B]^T = [rhs_ρ, rhs_ρu, rhs_ε, rhs_B]^T * Δt;
    // apply limiters to the updated conserved variables except $\psi$
    [ρ, ρu, ε, B]^T = [ρ*, (ρu)*, ε*, B*]^T;
    // calculate the right hand side of the equation of $\psi$, based on the updated and limited solution of other variables
    rhs_ψ = ∂ψ/∂t;
    // update $\psi$
    ψ = ψ + rhs_ψ * Δt;
end
```
This way, the updated $\psi$ has the limiter introduced divergence error $\nabla B'$ taken into consideration, and will be used to calculate the $\text{rhs}_{B}$. However, comparison of the first algorithm with the second shows that this limiter introduced divergence error is very small and does not grow with time since it will be addressed in the following time-step in the calculation of $\psi$. Also, the second algorithm is computationally inefficient compared to the first one, as it requires more expensive memory visit, i.e. revisiting the array of $Q$ and $\text{rhs}$. Thus, in PHORCE, the first algorithm is applied.
Chapter 4

Affine Reconstructed DG for Diffusion

As mentioned in Chapter 1, although a significant amount of literature exists on accurate and efficient DG implementations for the convection terms, an accurate and efficient DG diffusion solver is still a topic of on-going research. In this chapter, a new reconstruction-based algorithm, affine reconstructed DG (aRDG) algorithm, is proposed for solving the diffusion term accurately and efficiently. To the best of the author’s knowledge, this is the first practical guideline that has been proposed for applying the reconstructed algorithm on a nodal discontinuous Galerkin method. This is also the first algorithm using an affine transformation in reconstructed elements. The use of affine elements in the reconstruction makes the memory storage and computation more efficient as it avoids the higher order transformation (between reference domain and physical domain) in the reconstructed element.

4.1 Governing Equation and Discretization

4.1.1 Governing Equation

This work focuses on solving the diffusion term using a reconstructed DG method. The governing equation is the diffusion equation,

\[
\frac{\partial u}{\partial t} = \nabla \cdot (D \nabla u),
\]

(4.1)

where \( D \) is the diffusion coefficient. Without loss of generality, \( D \) is assumed to be a positive constant in space and time.
4.1.2 Discretization

In a DG scheme, the numerical solution can be expressed as a direct sum of local piecewise polynomials as

$$u(x, t) \simeq u_h(x, t) = \bigoplus_{k=1}^{K} u^k_h(x, t).$$

(4.2)

Similar to equations 3.4-3.5, the DG treatment is provided as,

$$
\int_{\Omega_k} \left( \frac{\partial u^k_h}{\partial t} \phi^i_k + D \nabla u^k_h \cdot \nabla \phi^i_k \right) d\Omega - D \int_{\partial \Omega_k} (\phi^i_k \hat{n} \cdot \nabla u^k) d\partial\Omega = 0.
$$

(4.3)

Since $u^k_h$ is discontinuous at the cell interface, the diffusive flux $\nabla u^k_h$ in the surface integral is not directly available on the boundary of $\Omega_k$. $\nabla u^k_h$ cannot be treated as an advective flux and approximated by a Riemann flux solver [van Leer et al., 2007, Luo et al., 2010], due to the lack of upwinding mechanism in diffusion. A number of numerical algorithms [Douglas and Dupont, 1976, Arnold, 1982, Cockburn and Shu, 1998a, Peraire and Persson, 2008, Liu and Yan, 2009] have been proposed in order to approximate the diffusive flux term accurately and consistently in DG. However, all these methods require substantial computational effort [Lou et al., 2018]. Nishikawa [2007, 2010] proposes an alternative way of approximating the diffusion term by introducing an additional hyperbolic equation, formulated as,

$$
\frac{\partial u}{\partial \tau} + \frac{\partial u}{\partial t} = \nabla \cdot \mathbf{v},
$$

(4.4)

$$
\frac{\partial \mathbf{v}}{\partial \tau} = \frac{1}{T_r} (\nabla u - \mathbf{v}),
$$

(4.5)

where $\mathbf{v} = [v_x, v_y]^T$ is the auxiliary variable, $T_r$ is called relaxation time. For each physical time-step, this system is also relaxed to a steady state with the pseudo time $\tau$. This approach is called first order hyperbolic system (FOHS) and is attractive due to its low computational effort compared to other methods. However, as reported in Mazaheri and Nishikawa [2016], FOHS approximates the diffusion term in DG with the order of accuracy of $P$, which is one order lower than the order of the conventional DG hyperbolic solver. Furthermore, it is unclear how this method can be applied to nodal DG. In Van Leer and Nomura [2005], a recovery-based DG algorithm to solve the diffusion term is proposed, where a new polynomial that is smoothly defined across two adjacent elements is recovered from the two original polynomials with order of $P$. The new polynomial is of order $2P + 1$ and is indistinguishable from the original solutions defined in each of the two cells in a weak sense. This means the new polynomial and the original polynomial are equivalent when both are multiplied by a test function and integrated over the cell. However, the accuracy of the scheme is affected not only by the diffusive parts but also the hyperbolic parts of the system. In fact, the order of accuracy is determined by the least accurate component. Hence, a highly accurate diffusion solver does not increase the overall accuracy of the scheme in solving convection-diffusion
problems, if the convection solver is of lower order. Also, constructing an appropriate basis function defined on the combination of two elements is an involved process. More recently, a reconstruction-based DG algorithm using Taylor basis functions was proposed in Luo et al. [2010], where the reconstructed polynomial has the same order as the original polynomials in each of the two adjacent elements. However, all computations are performed on the physical domain, which requires either substantial memory storage for the local mass and stiffness matrices or substantial computational effort for computing these matrices during run-time. Furthermore, it is also unclear how this algorithm extends to nodal DG. In this section, a new reconstruction-based algorithm is proposed, that is efficient in memory storage and computation, and also couples well with the nodal DG method that is used in PHORCE.

In equation 4.3, \( \nabla u_k^h \) in the surface integral is replaced by a reconstructed solution \( \nabla \tilde{u}^k \) that is smoothly defined at the interface. The details of this reconstruction algorithm will be discussed in section 4.3.5.

### 4.2 NDG Elemental Operation for Diffusion

Following the nodal DG algorithm [Hesthaven and Warburton, 2007] described in Chapter 3, the test function and basis function are chosen to be Lagrange polynomials, \( \ell_i \). For the sake of simplicity, the subscript \( h \) is dropped from now on. Then, equation 4.3 can be rewritten as

\[
\int_{\Omega_k} \left( \frac{\partial u_k^h}{\partial t} \ell_i^k + D \nabla u_k^h \cdot \nabla \ell_i^k \right) d\Omega - D \int_{\partial \Omega_k} \left( \ell_i^k \hat{n} \cdot \nabla \tilde{u}^k \right) d\partial\Omega = 0. \tag{4.6}
\]

For the remainder of this chapter, any variable or matrix without the element index superscript \( k \) is defined on a logical space element \( I \). Now, similar to equation 3.18, equation 4.6 can be written as

\[
\frac{\partial u^k}{\partial t} + D \left( M^{k-1} S^k T \cdot \nabla u^k \right) - D \sum_{f=1}^{3} \text{LIFT}_f^k \left( \hat{n}_f^k \cdot \nabla \tilde{u}_f^k \right) = 0, \tag{4.7}
\]

where the mass matrix and stiffness matrix are defined in equations 3.19 and 3.20, and \( u^k = [u^k(x_1^k, t), \ldots, u^k(x_{N_p}^k, t)]^T \). The lift operator is defined slightly differently here because of the diffusive flux (instead of advective flux),

\[
\text{LIFT}_f^k \left( \hat{n}_f^k \cdot \nabla \tilde{u}_f^k \right) = M^{k-1} \int_{\partial \Omega_k} \ell_i^k \hat{n}_f^k \cdot \nabla \tilde{u}_f^k d\partial\Omega. \tag{4.8}
\]

Here, the surface integration cannot be easily transformed to the reference domain, as the reconstructed element is not guaranteed to share the same mapping transformation of triangular elements, described in equation 3.14 and 3.15. This means, that this surface integration needs to be precalculated and stored on all elements, which is computationally inefficient. This will be discussed in the following section.
Now, equation 4.7 can be written as

$$\frac{\partial u^k}{\partial t} + D \left( M^{-1} S^T \cdot \nabla u^k \right) - D \sum_{f=1}^{3} \text{LIFT}_f^k \left( \hat{n}_f^k \cdot \nabla \tilde{u}_f^k \right) = 0. \quad (4.9)$$

If $D$ is not a constant, but a function of space and time, and also not isotropic (i.e. $D = (D_x, D_y)$), then equation 4.9 can be rewritten as

$$\frac{\partial u^k}{\partial t} + \left( M^{-1} S^T \cdot D \nabla u^k \right) - \sum_{f=1}^{3} \text{LIFT}_f^k \left( \hat{n}_f^k \cdot D \nabla \tilde{u}_f^k \right) = 0. \quad (4.10)$$

An alternative way of calculating the reconstructed solution for the surface term is,

$$\frac{\partial u^k}{\partial t} + \left( M^{-1} S^T \cdot D \nabla u^k \right) - \sum_{f=1}^{3} \text{LIFT}_f^k \left( \hat{n}_f^k \cdot D \nabla \tilde{u}_f^k \right) = 0. \quad (4.11)$$

This is summarized in the pseudo-code in Algorithm 3.

**Algorithm 3**: Reconstruction algorithm for non-constant diffusion coefficients using equation 4.11

// Perform the reconstruction of $u$ and get the reconstructed solution $\tilde{u}$ on the reconstructed element which consists of two triangles
\[ \tilde{u} = \text{RDG}(u^1, u^2); \]
// Calculate the gradients of $\tilde{u}$
\[ \nabla \tilde{u} = \text{Grad}(\tilde{u}); \]
// Perform the reconstruction of $D$ and get the reconstructed solution $\tilde{D}$ on the reconstructed element which consists of two triangles
\[ \tilde{D} = \text{RDG}(D^1, D^2); \]
// Calculate the product
\[ \tilde{D} \nabla \tilde{u} = \text{Product}(\tilde{D}, \nabla \tilde{u}) \]

Test results indicate minimal differences between the two reconstructed formulations described in equations 4.10 and 4.11. However, equation 4.10 is more desirable as it applies a reconstruction on the whole term, thus only equation 4.10 is applied in PHORCE.

### 4.3 Affine Reconstructed Algorithm

#### 4.3.1 Non-affine Mapping in Quadrilaterals

To obtain a reconstructed solution that is smoothly defined at the interface, the reconstruction needs to be performed on the combination of two triangles, which is a quadrilateral.
Hence, it is important to consider the mapping transformation between a quadrilateral element, \( \Omega^q \), and a reference square element, \( I^q = \{ R = (R, S) | -1 \leq (R, S) \leq 1 \} \). Here superscript \( q \) refers to quadrilateral. This mapping is described as,

\[
X = \frac{1}{4}(1-R)(1-S)v^1 + \frac{1}{4}(1+R)(1-S)v^2 + \frac{1}{4}(1+R)(1+S)v^3 + \frac{1}{4}(1-R)(1+S)v^4, \tag{4.12}
\]

where \( v^1, v^2, v^3, \) and \( v^4 \) are the coordinates of the four vertices of \( \Omega^q \). This is not always an affine mapping. Thus, assuming \( I(r) \) for the reference triangle and \( I^q(R) \) for the reference square element share the same coordinate system, then \( \Omega(x) \) and \( \Omega^q(X) \) are not in the same physical coordinate system. To demonstrate this, \( P4 \) (\( Pn \) denotes polynomial order \( n \)) tensor product nodal points in \( I^q \), as shown in Figure 4.1-a, are mapped to an arbitrary quadrilateral element \( \Omega^q_1 \) through equation 4.12, as shown in Figure 4.1-b. Note that the nodes on the diagonal of \( \Omega^q_1 \) are curved and do not represent the straight interface between the two triangles. Figure 4.1-c provides another example, where the diagonal of the quadrilateral in \( \Omega^q_2 \) is not curved but the nodes on diagonal are not symmetric. This shows that the diagonal of \( \Omega^q \) does not represent the interface between two triangular elements. This makes the reconstruction unfavorable as the surface integration described in equation 4.8 can then only be evaluated on the physical domain, which is inefficient for both computation and storage management.

**Figure 4.1:** Mapping transformation in quadrilaterals. (a) tensor product of LGL nodes on \( I^q \); (b) transformation from \( I^q \) to \( \Omega^q_1 \) that has a curved diagonal; (c) transformation from \( I^q \) to \( \Omega^q_2 \) that has a straight diagonal but with asymmetric nodes along the diagonal.

### 4.3.2 Enclosed Parallelogram

The mapping from equation 4.12 can be reduced to affine mapping when the physical quadrilateral \( \Omega^q \) is a parallelogram, which is shown in Figure 4.2. For any quadrilateral \( \Omega^q \) formed
by two adjacent triangles $\Omega_1$ and $\Omega_2$, one can always find an enclosed parallelogram $\Omega^p$ that shares the same diagonal with $\Omega^a$, which is also the interface between two triangles. This is demonstrated in Figure 4.3. Once $\Omega^p$ is found, the solution from $\Omega_1$ and $\Omega_2$ is projected onto the two smaller triangles $\Omega'_1$ and $\Omega'_2$ that form the parallelogram. Then the solution from these two triangles can be used to reconstruct a polynomial $\tilde{u}$ that is continuously defined in the parallelogram. This reconstruction can be done in the logical element $I^q = I + I^{-1}$, where $I^{-1} = \{r = (r, s) | (r, s) \leq 1; r + s \geq 0\}$, with solution of $\Omega'_1$ projected on $I$ and solution $\Omega'_2$ projected on $I^{-1}$, when the shared interface in $\Omega'_1$ and $\Omega'_2$ is the hypotenuse in $I$ and $I_q$. This is because the nodes on the diagonal of $\Omega^p$ are located exactly at the nodes on the interface of $\Omega'_1$ and $\Omega'_2$. In other words, the mapping transformation between $\Omega^p$ and $I^q$ is identical to the mapping transformation between $\Omega'$ and $I$. The formula for the projection is provided here but the reconstruction procedure will be discussed in detail in section 4.3.5.

Once the new vertices are found for $\Omega'_1$ and $\Omega'_2$, one can construct a projection Vandermonde matrix $\mathcal{V}_p$, that projects the modal expansion coefficients $\hat{u}$ on $\Omega$ to the nodal solution $u'$ on $\Omega'$,

$$u' = \mathcal{V}_p \hat{u}. \quad (4.13)$$

Now equation 4.8 can be rewritten as

$$\text{LIFT}_f^k \left( \hat{n}_f^k \cdot \nabla \tilde{u}_f^k \right) = \mathcal{M}^{-1} \left( \int_{\partial \Omega}^k \ell_i^k \hat{n}_f^k \cdot \nabla \tilde{u}_f^k d\partial \Omega \right)$$

$$= J^k \mathcal{M}^{-1} \left( \int_{\partial \Omega}^k \ell_i^k \hat{n}_f^k d\partial \Omega \right) \hat{n}_f^k \cdot \nabla \tilde{u}_f^k$$

$$= \frac{J_f^k}{J_k^f} \mathcal{M}^{-1} \left( \int_{\partial I}^f \ell_i^f \ell^f d\partial I \right) \hat{n}_f^k \cdot \nabla \tilde{u}_f^k$$

$$= \frac{J_f^k}{J_k^f} \text{LIFT}_f^k \left( \hat{n}_f^k \cdot \nabla \tilde{u}_f^k \right), \quad (4.14)$$
where $\tilde{\ell}_{r, f}^k$ is the basis function defined on the diagonal of the reconstructed enclosed parallelogram element, which is the same as the basis function defined on the edge of the triangle. $\ell_f^k$ is the basis function defined on edge $f$ in $I$. $J_f^k$ is the transformation Jacobian along edge $f$ of $\Omega_k$. $J_f^k$ can also be seen as the ratio between the length of $\Omega_f^k$ and $I_f$. $\nabla \tilde{u}_k^{f, 1, 2, 3}$ are $N_{fp} \times 1$ arrays of the gradients of the reconstructed nodal representations on the three edges of element $\Omega_k$. $N_{fp} = P + 1$ is the total number of nodes on one edge. $\nabla \tilde{u}_k^f(x)$ can be calculated as,

$$\nabla \tilde{u}(x)_f^k = \begin{bmatrix} \partial_x \tilde{u}_k^f \cr \partial_y \tilde{u}_k^f \end{bmatrix},$$

(4.15)

where the geometric factors are constant in a parallelogram, which requires much less storage compared to quadrilateral elements. Equation 4.9 now can be written as,

$$\frac{\partial u_k}{\partial t} + D \left( \mathcal{M}^{-1} S^T \cdot \nabla u_k \right) - D \sum_{f=1}^3 \frac{J_f^k}{J_k} \text{LIFT}_f \left( \hat{n}_f^k \cdot \nabla \tilde{u}_f^k \right) = 0,$$

(4.16)

in which all matrices are defined in $I$. This form has advantages for numerical implementation as the matrices can be precalculated while also using minimal storage.

Figure 4.3: Illustrations of an enclosed parallelogram found in two adjacent triangles. The enclosed parallelogram illustrated here represents a less challenging scenario as the enclosed parallelogram might be very small compared to the original quadrilateral. The size of the enclosed parallelogram is discussed in section 5.3.
4.3.3 Storage and Computational Efficiency

The use of an affine transformation on the reconstructed element has significant computational advantages. The geometric factors $J$ and $R_X$ are constant in an affine element, thus reducing the storage requirement significantly for affine elements. The comparison of the storage required for the geometric factors between parallelogram elements and quadrilateral elements is presented in Table 4.1. This storage is required for each interior edge of the mesh. The requirements for the mass matrices are also included in Table 4.1. For parallelogram elements, the computation can be performed on the reference domain, hence the mass matrix is only defined on the logical domain resulting in significantly lower storage requirements. For general quadrilateral elements, however, the transformation is different from that of triangular elements, hence the computation needs to be performed on the physical domain requiring the storage of the mass matrix for each element.

Table 4.1: Comparison of storage requirements (values correspond to the number of values stored for the geometric factors, mass and stiffness matrices) between parallelogram and quadrilateral elements. The storage indicated for $J$ and $R_X$ are required for each reconstructed element (each interior edge in the mesh). The storage indicated for $M$ is required only for reference element if the computation can be performed there, and is required for each reconstructed element if the computation needs to be performed on the physical domain.

<table>
<thead>
<tr>
<th></th>
<th>Parallelogram</th>
<th>Quadrilateral</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$P$</td>
<td></td>
</tr>
<tr>
<td>$P$</td>
<td>1 2 3 4 5</td>
<td>1 2 3 4 5</td>
</tr>
<tr>
<td>$N_P$</td>
<td>4 9 16 25 36</td>
<td>4 9 16 25 36</td>
</tr>
<tr>
<td>$J$</td>
<td>1 1 1 1</td>
<td>4 9 16 25 36</td>
</tr>
<tr>
<td>$R_X$</td>
<td>4 4 4 4 4</td>
<td>16 36 64 100 144</td>
</tr>
<tr>
<td>$M$</td>
<td>16 81 256 625 1296</td>
<td>16 81 256 625 1296</td>
</tr>
</tbody>
</table>

4.3.4 Reordering Nodes in the Reference Domain ($r, s$)

Every edge of $\Omega_k$ that has a neighboring element will need to be the hypotenuse in $I$ for the reconstruction. An immediate solution to this would be changing the ordering of the vertices $[v^1, v^2, v^3]$ in equation 3.14 to change the ordering of the nodes in $\Omega_k$, so that the target edge of $\Omega_k$ can be remapped to the hypotenuse of $I$. However, this needs to be done for two other edges of each element and requires either large computational effort if it is calculated during run-time or duplicated large storage if it is pre-calculated. This breaks the simplicity and efficiency of this scheme. A more efficient way to solve this is to change the ordering of nodes in $I$ to map its hypotenuse to the target edge in $\Omega$, without changing the ordering of nodes in $\Omega$. As this is in the reference domain ($r, s$), only two additional orderings of ($r, s$) need to be pre-calculated and stored. That is $[(1, -1), (-1, 1), (-1, -1)]$ if $(v^1, v^2)$ needs to
be the hypotenuse in $I$, and $[(-1, 1), (-1, -1), (1, -1)]$ if $(v^3, v^1)$ needs to be the hypotenuse in $I$. This is illustrated in Figure 4.4.

![Figure 4.4: Reordering of nodes in $I$ to map the hypotenuse $((1, -1), (-1, 1))$ on (a) edge $(v^1, v^2)$ or (b) edge $(v^3, v^1)$ in $\Omega$. The ordering of nodes in $\Omega$ remains unchanged.](image)

There are three orderings of $(r, s)$ in $I$ that can be used for performing the aRDG treatment on three edges of $\Omega$. Accordingly, three Vandermonde matrices $[V_r, V_{r+1}, V_{r+2}]$ can be generated to project the original nodal solutions $u$ on $\Omega$ to modal expansion coefficients $\hat{u}_f$ on $I$ so the desired edge $f$ matches the hypotenuse. This is described as

$$\hat{u}_f = V_{rf}^{-1}u.$$  \hspace{1cm} (4.17)

Combining equations 3.13, 4.13, and 4.17, the modal expansion coefficient $\hat{u}'_f$ is calculated in $\Omega'$, where the edge $f$ in $\Omega$ (or $\Omega'$) is the hypotenuse in $I$, from the nodal solution $u$ in $\Omega$, as

$$\hat{u}'_f = V^{-1}V_{r}V_{rf}^{-1}u.$$  \hspace{1cm} (4.18)

This expression can also be precomputed using any symbolic solver.

### 4.3.5 Reconstruction

The reconstruction process is performed using the modal solution, which is computed from the Vandermonde matrix and the nodal solutions in the two smaller triangles that form the enclosed parallelogram. Similar to the recovery [Van Leer and Nomura, 2005] and the reconstruction [Luo et al., 2010] methods, a new polynomial is constructed that is smoothly
defined across two adjacent cells,

\[
\int_{\Omega_1} \sum_{r=1}^{M_p} \tilde{u}_r \tilde{\psi}_r \psi_m d\Omega = \int_{\Omega_1} \sum_{r=1}^{N_p} \hat{u}_r^{1} \psi_r \psi_m d\Omega, \\
\int_{\Omega_2} \sum_{r=1}^{M_p} \tilde{u}_r \tilde{\psi}_r \psi_m d\Omega = \int_{\Omega_2} \sum_{r=1}^{N_p} \hat{u}_r^{2} \psi_r \psi_m d\Omega,
\]

(4.19)

where \( N_p = (P+1)(P+2)/2 \) is the number of modes in a triangle and \( M_p \) is the number of modes in the parallelogram, respectively. \( \hat{u}_r^{1} \) and \( \hat{u}_r^{2} \) are the modal solutions on the two smaller triangles \( \Omega_1 \) and \( \Omega_2 \). \( \tilde{u}_r \) is the reconstructed modal solution on the parallelogram. Using tensor product of Gauss-Legendre polynomial basis for the parallelogram, \( M_p = (P+1)(P+1) \). This system has \( 2N_p \) equations and \( M_p \) unknowns. This affine reconstruction method solves \( (P+1)^2 \) unknowns from \( (P+1)(P+2)/2 \) equations which differs from the \( (P+1)(P+2)/2 \) unknowns (potentially with additional higher order correction terms) in the work of Luo et al. [2010]. This system is solved using a least-squares method described in Luo et al. [2010]. The aRDG algorithm can be summarized in the pseudo-code in Algorithm 4.

**Algorithm 4: aRDG algorithm**

// \( \mathbf{u}^1 \) and \( \mathbf{u}^2 \) are nodal DG solutions on two adjacent elements \( \Omega_1 \) and \( \Omega_2 \).
// \( f^1 \) and \( f^2 \) are the local face indices of the interface in these two elements

// Calculate the Vandermonde Matrices that project \( \mathbf{u}^1 \) and \( \mathbf{u}^2 \) to \( \mathbf{u}^{1} \) and \( \mathbf{u}^{2} \) on the enclosed triangles
\( \mathcal{V}_{p1} = \text{getProjectVandermonde}(\Omega_1); \)
\( \mathcal{V}_{p2} = \text{getProjectVandermonde}(\Omega_2); \)

// Calculate the Vandermonde matrices that rotate \( \mathbf{u}^{1} \) and \( \mathbf{u}^{2} \), so that the interface is on the hypotenuse of \( \Omega_1 \) and \( \Omega_2 \) in the reference domain
\( \mathcal{V}_{r1} = \text{getRotateVandermonde}(f^1); \)
\( \mathcal{V}_{r2} = \text{getRotateVandermonde}(f^2); \)

// Calculate the rotated modal solution on the two enclosed triangles
\( \mathbf{\hat{u}}^1 = \mathbf{\psi}^{-1} \mathcal{V}_{p1} \mathcal{V}_{r1}^{-1} \mathbf{u}^{1}; \)
\( \mathbf{\hat{u}}^2 = \mathbf{\psi}^{-1} \mathcal{V}_{p2} \mathcal{V}_{r2}^{-1} \mathbf{u}^{2}; \)

// Perform the reconstruction
\( \mathbf{\tilde{u}} = \text{modalReconstruction}(\mathbf{\hat{u}}^1, \mathbf{\hat{u}}^2); \)
In this chapter, several code verification and benchmark tests are performed using PHORCE. The hyperbolic solver is verified for Euler equations and ideal-MHD equations. The diffusion solver is verified for multiple linear and non-linear scalar equations, and the Navier-Stokes equations. Benchmark tests are performed for Euler equations, ideal-MHD equations, and Navier-Stokes equations against results established in literature.

5.1 Order of Accuracy

Code verification aims to test the order of accuracy, or convergence rate, of the solver. For the hyperbolic solver, the Lax-Friedrichs [Toro, 2013] flux is applied. According to Hesthaven and Warburton [2007], the optimal order of accuracy of the NDG algorithm for a system is \( P + \frac{1}{2} \), when a general monotone flux is used. For the diffusion solver, the formal order of accuracy is \( P + 1 \). To calculate the order of accuracy, test cases with an analytical solution are needed. In cases where an analytical solution is not available, one can construct an analytical solution by introducing an artificial source term using the method of manufactured solutions (MMS) [Oberkampf and Roy, 2010].

In this chapter, the global \( L_2 \) and \( L_\infty \) norms of the error are investigated and calculated as follows,

\[
L_2 = \sqrt{\frac{\sum_{k=1}^{K} \int_{\Omega_k} [u^k - u_e]^2 \, d\Omega}{\sum_{k=1}^{K} |\Omega_k|}},
\]

\[
L_{\infty} = \max_{k=1}^{K} \frac{\int_{\Omega_k} |u^k - u_e| \, d\Omega}{|\Omega_k|},
\]

where \( u_e \) is the analytical solution. It is important to point out that the errors calculated in this chapter contain both the spatial and temporal discretization errors. Based on Oberkampf
and Roy [2010], the error norms are,
\[ \| \epsilon_{ht}^{h_x} \| = g_x h_x^\hat{p} + g_t h_t^\hat{q}, \] (5.3)
where \( g_x \) and \( g_t \) are constants, \( h_x \) is the spatial grid size, \( h_t \) is time-step size, \( \hat{p} \) is the order of accuracy for spatial discretization, and \( \hat{q} \) is the order of accuracy for temporal discretization. As mentioned in Oberkampf and Roy [2010], in order to evaluate \( \hat{p} \) correctly, time discretization \( h_t \) should be refined with the factor \( r_t \) calculated from the spatial refinement factor \( r_x \) and the formal numerical orders for space, \( p \), and for time, \( q \),
\[ r_t = (r_x)^{p/q}. \] (5.4)
Thus, \( \hat{p} \) can be calculated as
\[ \hat{p} = \frac{\ln \left( \frac{\| \epsilon_{ht}^{r_x h_x} \|}{\| \epsilon_{ht}^{h_x} \|} \right)}{\ln (r_x)}. \] (5.5)
Alternatively, one can fix \( h_t \), and equation 5.3 becomes,
\[ \| \epsilon_{ht}^{h_x} \| = g_x h_x^\hat{p} + \phi, \] (5.6)
where \( \phi = g_t h_t^\hat{q} \) is the fixed temporal error term. Then \( \hat{p} \) can be evaluated with three mesh refinement levels, e.g. coarse(\( r_x^2 h_x \)), medium(\( r_x h_x \)), and fine(\( h_x \)),
\[ \hat{p} = \frac{\ln \left( \frac{\| \epsilon_{ht}^{r_x h_x} \| - \| \epsilon_{ht}^{r_x^2 h_x} \|}{\| \epsilon_{ht}^{h_x} \| - \| \epsilon_{ht}^{r_x h_x} \|} \right)}{\ln (r_x)}. \] (5.7)
When \( g_t h_t^\hat{q} \) is significant smaller than \( g_x h_x^\hat{p} \), e.g. when \( \hat{q} > \hat{p} \) and \( h_t \) is fixed as a very small value, then only two mesh refinement levels are needed, and equation 5.7 is simplified as
\[ \hat{p} = \frac{\ln \left( \frac{\| \epsilon_{ht}^{r_x h_x} \|}{\| \epsilon_{ht}^{h_x} \|} \right)}{\ln (r_x)}. \] (5.8)
For all the code verification studies presented in this dissertation, the five-stage fourth-order SSP-RK scheme [Carpenter and Kennedy, 1994] is used for the time integration (\( p \leq q \)). The time-step \( h_t \) is calculated from the most restrictive mesh refinement level and is fixed for all meshes. \( g_t h_t^\hat{q} \) has been verified to always be significantly smaller than \( g_x h_x^\hat{p} \). Thus, the order of accuracy is calculated using equation 5.8.
5.2 Hyperbolic Solver

5.2.1 Euler Equations

Euler equations (2.7-2.9) only have hyperbolic terms, which is solved by the hyperbolic solver described in Chapter 3. Three case studies are performed here, the first is a code verification study, whereas the second and third are benchmark tests with slope limiter and WENO limiter, respectively.

Isentropic Vortex

An isentropic vortex case [Hesthaven and Warburton, 2007] is considered with an exact solution given by,

\[ u = 1 - \beta e^{(1-r^2)} \frac{y-y_0}{2\pi} \]
\[ v = \beta e^{(1-r^2)} \frac{x-x_0}{2\pi} \]
\[ \rho = \left( 1 - \frac{(\gamma - 1)}{16\gamma \pi^2} \beta^2 e^{2(1-r^2)} \right)^{\frac{1}{\gamma-1}} \]
\[ p = \rho^\gamma \]

where \( r = \sqrt{(x-x_0)^2 + (y-y_0)^2} \), \( x_0 = 5, y_0 = 0, \beta = 5, \) and \( \gamma = 1.4 \). The boundary condition is set to be exact solution.

The tests are performed on a sequence of grids with systematic refinements, that have 16, 64, 256, 1024, 4096 elements. To demonstrate systematic refinement, the first three grids are shown in Figure 5.1.

Results of the calculated error norms are presented in Figure 5.2, the results of the order of accuracy are shown in Figure 5.3. The observed order of accuracy are very close to the optimal rate of \( \mathcal{O}(h^{P+1/2}) \).

Forward Facing Step

This test involves supersonic (Mach 3) uniform flow over a forward facing step in a channel. This case was first studied in Woodward and Colella [1984] and then in Cockburn and Shu [1998b]. It is also reported in Hesthaven and Warburton [2007]. This test is interesting as the step corner generates a solution singularity, which makes for a good test of the slope limiter described in section 3.6.2. Initial and inflow boundary conditions are,

\[ (\rho, u, v, p) = (1.0, 3.0, 0.0, \gamma^{-1}) \]
Figure 5.1: Demonstration of systematic refinements.

Figure 5.2: Convergence tests of isentropic vortex on a sequence of grids with systematic refinements using (a) \( P_1 \), (b) \( P_2 \), and (c) \( P_3 \) NDG algorithms. Formal orders of accuracy are indicated by the slopes with magenta lines.

The density profile with Mach contours at time \( t = 4.0 \) is shown in Figure 5.4. There is no analytical solution available for this case, but the simulations are considered to be successful through a direct comparison with the results in Woodward and Colella [1984], Cockburn and Shu [1998b], and Hesthaven and Warburton [2007]. The slope limiter is computationally inexpensive and thus a good choice in this case, where strong shocks are present and no instabilities are involved. As mentioned in section 3.6.2, the order of accuracy will degrade to 1\textsuperscript{st} order in regions with strong shocks regardless of what limiter is chosen.
Figure 5.3: Order of accuracy results for isentropic vortex on a sequence of grids with systematic refinements using (a) $P_1$, (b) $P_2$, and (c) $P_3$ NDG algorithms. Formal orders of accuracy are indicated by the slopes with magenta lines.

Figure 5.4: Density profile of forward facing step case at time $t = 4.0$ using $P_1$ NDG algorithm with slope limiter. Mesh resolution: 67,000 random triangular elements with similar sizes.

Richtmyer-Meshkov Instability in a Blast Wave

The Richtmyer-Meshkov instability is an important instability in astrophysical and fusion plasmas. The Richtmyer-Meshkov instability (RMI) occurs when the interface of two fluids with different densities is accelerated by a shock wave [Londrillo and Zanna, 2000]. This is depicted in Fig. 5.5. The shock wave destabilizes the fluid interface and produces spikes and bubbles similar to those in the Rayleigh-Taylor instability (RTI). However, unlike the RTI where the direction from the heavier fluid to the lighter fluid needs to be the same
as the direction of the gravitational force, RMI occurs regardless of the direction of fluid density gradient with respect to the shock direction [Cloutman and Wehner, 1992]. RMI can be observed in inertial confinement fusion, magnetized target fusion, and astrophysical phenomena. It is important to understand the behavior of RMI in order to design the corresponding suppression mechanism.

In the following test, RMI is studied in a blast wave in a rectangular domain with $-0.5 \leq x \leq 0.5$, $-0.75 \leq y \leq 0.75$, and wall boundaries [Zachary et al., 1994]. The initial density is 1.0 and the initial velocities and magnetic fields are zero everywhere. Pressure is 0.1 at $r > 0.1$ and 10.0 at $r < 0.1$. Since the initial profile is circular in a rectangular domain, the choice of mesh determines how accurately the initial circle is captured. Grid level noise can alter the interface between the high pressure and low pressure fluids. Once the outgoing blast reflects from the boundary, it interacts with the perturbed fluid interface and causes RMI to grow. The choice of grid produces different initial perturbations to the fluid interface, and consequently different behavior of the RMI.

![RMI diagram](image)

Figure 5.5: RMI: A simplified planar description of the RMI. The RMI grows due to the interaction between a shock wave and an interface separating fluids of different densities. The resulting growth is in the form of bubbles and spikes similar to the RTI. In the geometry presented in Figures 5.6 and 5.7, the RMI grows because the reflected shock wave from the boundaries intersects the expanding fluid interface. [Srinivasan et al., 2019]

Here, two grids are used: a structured grid with 260,000 triangular elements and a random unstructured grid with 250,000 triangular elements. The left plots of Figs. 5.6 and 5.7 present the two grids with less refinement to allow the reader to view the grid. From the density profiles at $t = 0.2$ (middle plot) in Figs. 5.6 and 5.7, subtle differences are observed in the perturbation at the interface developing due to grid noise. For the structured mesh, due to the alignment of the mesh elements, the perturbation on the diagonal connecting the top-left to bottom-right is very different from the one on the diagonal connecting the bottom-left to top-right. For the random unstructured mesh, the perturbations along the circular interface

---

1 Simulation setup described in [https://www.astro.princeton.edu/~jstone/Athena/tests/blast/blast.html](https://www.astro.princeton.edu/~jstone/Athena/tests/blast/blast.html)
Figure 5.6: RMI: Demonstration of the structured mesh (left); density profiles at time $t = 0.2$ (middle) and time $t = 1.3$ (right), respectively. Mesh resolution is 260,000 triangular elements with triangles of the same shape. [Srinivasan et al., 2019]

Figure 5.7: RMI: Demonstration of the random mesh (left); density profiles at time $t = 0.2$ (middle) and time $t = 1.3$ (right), respectively. Mesh resolution is 250,000 triangular elements with similar sizes. [Srinivasan et al., 2019]
Figure 5.8: RMI: Time evolution of the wavenumber spectrum of power density integrated along x direction for the structured mesh (left) and the random mesh (middle) and time evolution of the peak bubble-to-spike distance (right). [Srinivasan et al., 2019]

are random. This leads to different topologies of the RMI late-in-time, as shown in the density profiles at \( t = 1.3 \) (right plots). The differences in RMI evolution due to the different grids are explored through the wavenumber spectra and the peak bubble-to-spike distance in Fig. 5.8. The left and middle plots of Fig. 5.8 present the wavenumber spectra as a function of time for the structured and random grids, respectively. The wavenumber spectrum is obtained by performing an integration in the x-direction where the time of 0.6 approximately corresponds to when the shock reaches the interface. Note that the wavenumber spectra do not seem to be significantly modified by choice of grid. However, the key differences are seen in the penetration of the bubbles/spikes as presented in the right plot of Fig. 5.8. The structured grid produces a solution with deeper penetration of the spikes and bubbles resulting in more mixing depth in the direction aligned with the grid. The random unstructured mesh produces more evenly distributed grid errors that lead to slower growth of RMI. The structured mesh generates more grid biasing in certain regions and directions. While it may not be obvious what the correct solution is for the nonlinear phase of such instabilities, it is safe to assume that a random grid with its random errors has a better chance at representing physically-relevant random perturbations compared to a rectangular structured grid attempting to capture dynamics in arbitrary geometries. This example is included to demonstrate that instabilities such as RMI are sensitive to the choice of meshes and one needs to pay attention to grid-based errors and biases introduced through the mesh that can alter the dynamics. For high-energy-density plasma applications, such differences in mixing and penetration depths can alter heat transfer significantly [Srinivasan and Tang, 2014b,a] and under- or over-predict temperatures and densities that may be achieved. This can have important consequences for studies of relevance to supernova and fusion.
5.2.2 GLM-MHD Equations

The GLM-MHD equations (2.53-2.57) without diffusion terms are tested using the hyperbolic solver described in Chapter 3. Two case studies are performed here, the first is a code verification study and the second is a benchmark test with WENO limiter.

Magnetized Isodensity Vortex

A magnetized isodensity vortex case is considered here for code verification of GLM-MHD equations. This case is extended from the isentropic vortex problem in Euler equations to ideal-MHD equations, and first studied in Balsara [2004]. This is a good test case for code verification since it involves a smooth and dynamically stable configuration that has an analytical solution. In this case, the computational domain is given by \([-5, 15] \times [-10, 10]\). The boundary conditions are set with an analytical solution. The initial unperturbed condition is given as \((\rho, v_x, v_y, v_z, P, B_x, B_y, B_z) = (1, 1, 1, 0, 1, 0, 0, 0)\). The vortex is initialized at the center of the domain with perturbations in in-plane velocity and magnetic fields:

\[
(\delta v_x, \delta v_y) = \frac{\beta}{2\pi} e^{\frac{1}{2}(1-r^2)}(-y, x) \quad (5.10a)
\]
\[
(\delta B_x, \delta B_y) = \frac{\kappa}{2\pi} e^{\frac{1}{2}(1-r^2)}(-y, x) \quad (5.10b)
\]

A gas pressure perturbation is introduced to ensure the dynamical balance:

\[
\delta P = \frac{1}{8\pi} \left( \frac{\kappa}{2\pi} \right)^2 (1 - r^2)e^{(1-r^2)} - \frac{1}{2} \left( \frac{\beta}{2\pi} \right)^2 e^{(1-r^2)} \quad (5.11)
\]

The tests are performed using cgs units by setting \(\mu_0 = 4\pi\) using the same sequence of grids with systematic refinements used for isentropic vortex tests described in section 5.2.1. The vortex is set up with \(\beta = 1\) and \(\kappa = \beta\sqrt{\mu_0\rho} = \sqrt{4\pi}\), which makes the Alfven speed equal to the rotational speed. Results of the calculated error norms are presented in Figure 5.9, the results of order of accuracy are shown in Figure 5.10. The observed orders of accuracy are close to the formal rate \(O(h^{P+1})\), except for the density in \(P2\), which is still close to the optimal rate \(O(h^{P+1/2})\). A test for the \(\nabla \cdot B\) error cleaning is also performed using \(P3\) NDG method, with ideal-MHD equations (without cleaning) and ideal GLM-MHD equations. The \(L_2\) and \(L_\infty\) of the \(\nabla \cdot B\) errors are calculated over time and presented in Figure 5.11. The \(\nabla \cdot B\) error increases over time without cleaning, and remains the same order of magnitude with cleaning.
5.3 Diffusion Solver

Numerical tests are performed on multiple linear and non-linear scalar equations with diffusion and the Navier-Stokes equations using $P_1$, $P_2$, and $P_3$ nodal DG algorithms with the aRDG method. Three types of grids, as shown in Figure 5.12, are tested. Grid-a and -b are $0 \leq x \leq 10$. Grid-b has the bottom-left corner moved to $(1.5, -3.5)$, the top-right corner moved to $(11.5, 6.5)$, and the center moved to $(6.5, 1.5)$. In grid-a, each quadrilateral combined by two adjacent triangles is a parallelogram, thus no error associated with area.
truncation will be generated through the reconstruction process. In grid-\( b \), large area truncation will occur on the diagonals of the domain, where the combination of two adjacent triangles forms a larger triangle with a larger area than the enclosed parallelogram on which the reconstruction is performed. In grid-\( c \), the bottom-left and top-right corners are moved so that larger area truncation to obtain an enclosed parallelogram for reconstruction will occur along the top-left, top-right, and bottom-right half of the diagonals. However, the size of each element is the same even though the shape is different. Among the four sections of the diagonals in grid \( c \), the top-right section has the largest truncated area when obtaining an enclosed parallelogram for reconstruction. Convergence studies are performed on a series of systematic refinements of these three grids. Series of grid-\( a \) has 32, 128, 512, 2048, and 8192 elements, while series of grid-\( b \) and -\( c \) have 16, 64, 256, 1024, 4096 elements.

5.3.1 Scalar Equations with Diffusion

**Diffusion Equation**

The diffusion equation described in equation 4.1 is solved on the three grids presented in Figure 5.12. At \( t = -D_0/D \), a solute of mass \( M = 1 \) is loaded at \((x_0, y_0)\), where \((x_0, y_0) = (5, 0)\) for grid-\( a \) and -\( b \), and \((x_0, y_0) = (6.5, 1.5)\) for grid-\( c \). The analytical solution is provided as

\[
  u_e = \left(\frac{M}{4\pi (Dt + D_0)}\right) e^{-\frac{(x-x_0)^2 + (y-y_0)^2}{4(Dt + D_0)}},
\]

where \( D = 1 \), and \( D_0 \) is set to be 2 to make it numerically feasible at \( t = 0 \). This reconstruction follows equation 4.9, as the diffusion coefficient is a constant. The initial condition

Figure 5.11: \( \nabla \cdot \mathbf{B} \) error over time using \( P3 \) NDG method, with ideal-MHD equations (without cleaning) and ideal GLM-MHD equations.
Tests are performed on the three types of grids in Figure 5.12 as well as a set of perturbed grids of grid-a. The perturbation is applied to the locations of all interior vertices with an amplitude of 5% of the grid spacing in random directions. An illustration of this perturbation is presented in Figure 5.14. Results of the convergence study are presented in Figure 5.15, and the order of accuracy plots are presented in Figure 5.16. The convergence rates of both the $L_2$ and $L_\infty$ errors for all three types of grids are close to the formal order of accuracy $\hat{P} = P + 1$ [Hesthaven and Warburton, 2007] for $P1$, $P2$, and $P3$ tests. The fact that convergence lines of grid-a, -b, and -c are close to each other also indicates that the area truncation in the aRDG process has minor impact on the accuracy of the scheme. When two triangles form a parallelogram, the density of degrees of freedom of the reconstructed solution remains the same. When the enclosed parallelogram truncates a large area from the original adjacent triangles that form a quadrilateral, the density of degrees of freedom in the enclosed parallelogram is increased, which could compensate for errors associated with the area truncation. That the errors calculated on the perturbed grids are very close to the errors on grid-a indicates that this algorithm is also not sensitive to grid perturbations.

**Scalar Advection-Diffusion Equation**

In order to test how well the aRDG diffusion solver couples with the well-benchmarked NDG hyperbolic solver, this test focuses on the scalar advection-diffusion equation,

$$\frac{\partial u}{\partial t} + \vec{a} \cdot \nabla u - Du^2 = 0.$$  \hspace{1cm} (5.13)
Figure 5.13: Initial condition at $t = 0$ and final solution at $t = 0.5$ for the diffusion test. $P3$ test on grid- $b$ with 4096 elements is presented here.

Figure 5.14: Illustration of the perturbation on grid- $a$ (Figure 5.12) with different resolutions. The perturbation is applied to the locations of all interior vertices with an amplitude of 5% of the grid spacing on random directions.

The analytical solution is given by,

$$u_e = \left( \frac{M}{4\pi(Dt + D_0)} \right) e^{-\frac{(x-a_xt-x_0)^2+(y-a_yt-y_0)^2}{4(Dt+D_0)}}.$$  

(5.14)

Similar to the diffusion test, equation 4.9 is applied for the reconstruction of the diffusion term here. A solute of mass is loaded at $(x_0, y_0)$ at $t = -D_0/D$, with $D = 1$ and $D_0 = 2$. $(x_0, y_0)$ is set to be $(4, -1.0)$ for all three types of grids (Figure 5.12), and a constant advection
Figure 5.15: Convergence tests of the diffusion equation on three types of grids (Figure 5.12) and the perturbed grid-a (a') using (a) P1, (b) P2, and (c) P3 NDG algorithms. Formal orders of accuracy are indicated by the slopes with magenta lines.

Figure 5.16: Order of accuracy results for the diffusion equation on three types of grids (Figure 5.12) and the perturbed grid-a (a') using (a) P1, (b) P2, and (c) P3 NDG algorithms. Formal orders of accuracy are indicated by the slopes with magenta lines.

speed $\vec{a} = (6, 6)$ is chosen so that the diffusive mass is traveling along the diagonal of the domain where truncation of area occurs in aRDG for grid-b and -c. This way, the $L_\infty$ of the error captures the error associated with area truncation in aRDG, if any.

The initial condition at $t = 0$ and the final solution at $t = 0.5$ are presented in Figure 5.17. Convergence tests are shown in Figure 5.18, and the order of accuracy are presented in Figure 5.19. Similar to the pure diffusion test case, the optimal convergence is achieved for all types of meshes and polynomial orders that are tested. Again, the convergence lines for all three grids are close to each other.
Convection-Diffusion Equation with Non-constant Coefficients

In order to test the robustness of the aRDG scheme on non-linear equations, a scalar convection-diffusion equation with spatially- and temporally-varying coefficients is employed here,

$$\frac{\partial C}{\partial t} + \frac{1}{2} \frac{\partial}{\partial x} (a_{0x} C C) + \frac{1}{2} \frac{\partial}{\partial y} (a_{0y} C C) - \frac{\partial}{\partial x} \left( D_{0x} C \frac{\partial C}{\partial x} \right) - \frac{\partial}{\partial y} \left( D_{0y} C \frac{\partial C}{\partial y} \right) = S_{MMS},$$

(5.15)
Figure 5.19: Order of accuracy results for advection-diffusion equation on three types of grids (Figure 5.12) using (a) $P_1$, (b) $P_2$, and (c) $P_3$ NDG algorithms. Formal orders of accuracy are indicated by the slopes with magenta lines.

where $(a_{0x}, a_{0y}) = (0.2, 0.2)$ and $(D_{0x}, D_{0y}) = (0.2, 0.2)$ are constants. The advection and diffusion coefficients are non-constant and do not assume isotropcity. Equation 4.10 is applied here for the reconstruction of the diffusion terms. The analytical solution is constructed by method of manufactured solutions (MMS) [Oberkampf and Roy, 2010], a standard method used for code verification. The analytical solution is constructed as,

$$C_e = C_0 + C_A \sin\left(\frac{\pi x}{L}\right) \cos\left(\frac{\pi y}{L}\right) \cos(\pi t), \quad (5.16)$$

where $C_0 = 2.0$, $C_A = 0.5$, and $L = 10.0$ are constant. The manufactured source term in equation 5.15 is,

$$S_{MMS} = \frac{\partial C_e}{\partial t} + \frac{1}{2} \frac{\partial}{\partial x} (a_{0x} C_e C_e) + \frac{1}{2} \frac{\partial}{\partial y} (a_{0y} C_e C_e) - \frac{\partial}{\partial x} \left(D_{0x} C_e \frac{\partial C_e}{\partial x}\right) - \frac{\partial}{\partial y} \left(D_{0y} C_e \frac{\partial C_e}{\partial y}\right), \quad (5.17)$$

which can be pre-calculated using a symbolic solver.

The results for the convergence tests are presented in Figure 5.20, and for the order of accuracy are presented in Figure 5.21. The convergence rates agree with the theoretical rates except for $P_2$, where the observed rate is slightly lower than the theoretical rates. This behavior is consistent with previous results [Oden et al., 1998]. Similar to the linear test cases presented, no significant difference is found between the results on different grids, which indicates that the truncation of the area to obtain an enclosed parallelogram for reconstruction does not introduce noticeable error into this system.
Figure 5.20: Convergence tests of scalar convection-diffusion equation with spatially- and temporally-varying coefficients on three types of grids (Figure 5.12) using (a) \( P_1 \), (b) \( P_2 \), and (c) \( P_3 \) NDG algorithms. Formal orders of accuracy are indicated by the slopes with magenta lines.

Figure 5.21: Order of accuracy results for scalar convection-diffusion equation with spatially- and temporally-varying coefficients on three types of grids (Figure 5.12) using (a) \( P_1 \), (b) \( P_2 \), and (c) \( P_3 \) NDG algorithms. Formal orders of accuracy are indicated by the slopes with magenta lines.

Shear Diffusion Equation with Non-constant Coefficients

Tests are performed on three types of grids (described in Figure 5.12) using the shear term in the diffusion equation to further benchmark the robustness of aRDG algorithm. Following
the work of Johnson and Johnsen [2019], the shear diffusion equation is described as,

$$\frac{\partial C}{\partial t} - \frac{\partial}{\partial x} \left( D_0 C \frac{\partial C}{\partial x} \right) - \frac{\partial}{\partial y} \left( D_0 C \frac{\partial C}{\partial y} \right) - \theta D_0 \left[ \frac{\partial}{\partial x} \left( C \frac{\partial C}{\partial y} \right) + \frac{\partial}{\partial y} \left( C \frac{\partial C}{\partial x} \right) \right] = S_{\text{MMS}}, \quad (5.18)$$

where $\theta = \frac{1}{6}$, $D_0 = 0.2$. The analytical solution is constructed the same as in equation 5.16. Similarly, the manufactured source term is,

$$S_{\text{MMS}} = \frac{\partial C_e}{\partial t} - \frac{\partial}{\partial x} \left( D_0 C_e \frac{\partial C_e}{\partial x} \right) - \frac{\partial}{\partial y} \left( D_0 C_e \frac{\partial C_e}{\partial y} \right) - \theta D_0 \left[ \frac{\partial}{\partial x} \left( C_e \frac{\partial C_e}{\partial y} \right) + \frac{\partial}{\partial y} \left( C_e \frac{\partial C_e}{\partial x} \right) \right]. \quad (5.19)$$

Equation 4.10 is applied here for the reconstruction of the diffusion term, and the convergence results are presented in Figure 5.22, and the order of accuracy are presented in Figure 5.23. In this study, noticeable differences in the convergence errors from three types of grids can be observed on $P1$ and $P2$ tests. Convergences rates agree well with theory, except in $P2$ tests, where the convergence rates on grid-b and grid-c are slower than the theoretical rate. The accuracy of aRDG appears to be more sensitive to area truncation necessary to obtain the enclosed parallelogram for $P2$ shear diffusion problems. However, no significant difference can be observed on different grids for $P3$ tests, and the computed convergence rates successfully predict the theory.

Figure 5.22: Convergence tests of scalar shear-diffusion equation with spatial and temporal varying coefficients on three types of grids (Figure 5.12) using (a) $P1$, (b) $P2$, and (c) $P3$ NDG algorithms. Formal orders of accuracy are indicated by the slopes with magenta lines.

The algorithms presented here and in Johnson and Johnsen [2019] exclusively use face neighbors of the element to perform the reconstruction and recovery, respectively. For complete consistency with accurately resolving the shear term in the diffusion equation, particularly as the shear term becomes significant, it is necessary to account for all face and vertex neighbors of the elements. However, a practical implementation including all vertex neighbors while
Figure 5.23: Order of accuracy results for scalar shear-diffusion equation with spatial and temporal varying coefficients on three types of grids (Figure 5.12) using (a) $P_1$, (b) $P_2$, and (c) $P_3$ NDG algorithms. Formal orders of accuracy are indicated by the slopes with magenta lines.

maintaining computational and storage efficiency is non-trivial for unstructured grids and is a subject of future work. The likely reason that the shear term here still produces sufficient order of accuracy is due to (i) the normal stresses being dominant as is the case in most physical systems and (ii) the fourth-order Runge-Kutta time-integration scheme sufficiently resolving the cross derivatives over the five stages for the problems tested.

5.3.2 Navier-Stokes Equations

This section applies the aRDG algorithm to the compressible Navier-Stokes equations (2.15), that have been described in Chapter 2. The molecular viscosity $\mu$ is calculated through Sutherland’s law [Sutherland, 1893],

$$\mu = \mu_0 \left( \frac{T}{T_0} \right)^{3/2} \frac{T_0 + S}{T + S},$$

(5.20)

where $\mu_0$ is the reference viscosity, $T_0$ is the reference temperature, and $S$ is an effective temperature, called the Sutherland constant. They are taken based on the air at moderate temperatures and pressures, $\mu_0 = 1.7894 \times 10^{-5} \text{kg m}^{-1} \text{s}^{-1}$, $T_0 = 273.11 \text{K}$, and $S = 110.56 \text{K}$. The thermal conductivity $\kappa$ is calculated as

$$\kappa = \frac{\gamma R_{\text{specific}}}{(\gamma - 1) Pr} \mu,$$

(5.21)

where $R_{\text{specific}} = 287.058 \text{J kg}^{-1} \text{K}^{-1}$ is the specific gas constant for dry air, and $Pr = 0.7$ is the Prandtl number.
Two sets of tests are performed. The first one is a code verification test and the second one is a model validation test.

**Method of Manufactured Solutions (MMS)**

Code verification is performed on grid-b (Figure 5.12) using an analytical solution constructed by MMS,

\[
\begin{align*}
\rho_e &= \rho_0 + \rho_A \sin \left( \frac{\pi x}{L} \right) \cos \left( \frac{\pi y}{L} \right) \cos(\pi t), \\
u_e &= u_0 - u_A \sin \left( \frac{\pi x}{L} \right) \cos \left( \frac{\pi y}{L} \right) \cos(\pi t), \\
v_e &= v_0 + v_A \cos \left( \frac{\pi x}{L} \right) \sin \left( \frac{\pi y}{L} \right) \cos(\pi t), \\
p_e &= p_0 + p_A \sin \left( \frac{\pi x}{L} \right) \cos \left( \frac{\pi y}{L} \right) \cos(\pi t),
\end{align*}
\]

where

<table>
<thead>
<tr>
<th></th>
<th>( \rho )</th>
<th>( u )</th>
<th>( v )</th>
<th>( p )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( *_0 )</td>
<td>3.0</td>
<td>1.0</td>
<td>1.0</td>
<td>2.0</td>
</tr>
<tr>
<td>( *_A )</td>
<td>0.5</td>
<td>0.4</td>
<td>0.6</td>
<td>0.5</td>
</tr>
</tbody>
</table>

Results of the calculated error norms are presented in Figure 5.24, and the results of order of accuracy are shown in Figure 5.25. The observed orders of accuracy for all three variables in \( P1 \) tests are slightly higher than the optimal rate. Results of \( P2 \) and \( P3 \) tests show good agreement with theory.

**Subsonic Flow past a Circular Cylinder**

Model validation is performed on a subsonic flow over cylinder case with \( Re = 40 \). A circular cylinder with a diameter of \( D \) is placed at the center of a domain of size \( 32D \times 16D \). In this simulation, \( D = 2 \times 10^{-5} \) m, the initial and inlet density is \( \rho_0 = 1 \) kg m\(^{-3}\), and initial and outlet pressure is \( P_0 = 1 \times 10^5 \) Pa. The inlet velocity is fixed as 43.1 m s\(^{-1}\), which corresponds to \( Re = 40 \). The computed Mach number is plotted in Figure 5.26 with streamlines indicating the recirculation. The drag coefficient and the length of the recirculation region are calculated and presented in Table 5.1, which agree well with Tseng and Ferziger [2003].
5.3.3 Summary of aRDG Tests

An affine reconstructed discontinuous Galerkin method has been described to solve the diffusion operator accurately and efficiently on unstructured grids of triangles described in Chapter 4. A practical guideline on how to apply this algorithm to the nodal discontinuous Galerkin method has been provided.
Table 5.1: Drag coefficient and length of recirculation for subsonic flow over circular cylinder with $Re = 40$

<table>
<thead>
<tr>
<th>$Re = 40$</th>
<th>Drag coefficient</th>
<th>Length of recirculation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current study</td>
<td>1.47</td>
<td>2.26$D$</td>
</tr>
<tr>
<td>Tseng and Ferziger [Tseng and Ferziger, 2003]</td>
<td>1.53</td>
<td>2.21$D$</td>
</tr>
</tbody>
</table>

Figure 5.26: Mach number plot with streamlines of subsonic flow over circular cylinder with $Re = 40$

All computations can be done on the reference domain, which couples well with the notable nodal discontinuous Galerkin scheme from Hesthaven and Warburton [2007]. Benchmark tests are performed on three types of grids with different refinement levels using $P1$, $P2$, and $P3$ NDG schemes for linear and non-linear scalar equations with diffusion and the Navier-Stokes equations. The observed orders of accuracy generally agree with the formal orders of accuracy for all tests. Some $P2$ results have a $O(h_2^3)$ convergence as described in Oden et al. [1998] which shows that the optimal order of accuracy of DG for diffusion is $O(h_x^{P+1})$ for odd $P$ and $O(h_x^P)$ for even $P$. By maintaining the same polynomial order for the described reconstruction method, the density of nodes in the reconstructed element on the physical domain is not decreased, which means discretization error is not increasing through this reconstruction. When two triangles form a parallelogram, the density of degrees of freedom of the reconstructed solution remains the same. When the enclosed parallelogram truncates a large area from the original adjacent triangles that form a quadrilateral, the density of degrees of freedom in the enclosed parallelogram is increased, which could compensate for errors associated with the area truncation. This may explain why the errors associated with all three types of grids are very close to each other for most of the tests presented, except for when the shear term is included in the diffusion. It is also straightforward to extend
the aRDG algorithm to other types of elements as long as an enclosed parallelogram can be found in adjacent elements. Future work will focus on extending the aRDG algorithm to three dimensional unstructured grids.
Chapter 6

Rayleigh-Taylor Instability

In this chapter, an important hydrodynamic instability in inertial confinement fusion (ICF) implosion, which is also relevant to astrophysical plasmas, is considered. When a fluid with higher density is supported by fluid with lower density, or when the interface between two fluids with different densities is accelerating, the instability that arises from the interface between two fluids, is called the Rayleigh-Taylor instability (RTI) [Lord, 1900, Taylor, 1950, Chandrasekhar, 1961, Srinivasan et al., 2019]. In a typical ICF target, three layers of different mass densities are involved: DT gas on the inside, CH ablator on the outside, and a DT ice fuel layer between the DT gas and CH ablator [Srinivasan and Tang, 2013]. Figure 1.2 presents an illustration of an ICF target. Due to surface imperfections at the interfaces between these three layers, RTI grows during the acceleration or deceleration phase of the implosion. In this chapter, the RTI evolutions in planar and radially converging geometries are studied using the MHD equations in PHORCE. The effects of magnetic fields, viscosity, resistivity, and heat conduction on the growth of RTI will be discussed.

6.1 Theory

The RTI growth in stratified flow is characterized by a gravitational acceleration $g$, the wavenumber of the perturbation $k$, and the Atwood number $A_t$. $A_t$ is defined as,

$$A_t = \frac{\rho_h - \rho_l}{\rho_h + \rho_l}$$

(6.1)

where $\rho_h$ and $\rho_l$ are the mass densities of the heavy fluid and light fluid, respectively. According to linear theory, a small perturbation would grow exponentially in time, at a growth rate,

$$\gamma = \sqrt{A_t k g}.$$  

(6.2)

However, if the density is smoothly varying between the two fluids (e.g. a hyperbolic tangent
profile), which is typical in numerical simulations in order to avoid a strong discontinuity in the initial conditions, then the growth rate is [Betti et al., 1998],

$$\gamma = \sqrt{\frac{A_t k g}{1 + A_t k L_m}},$$

(6.3)

where $L_m$ is the minimum density gradient scale length,

$$L_m = \min \left| \frac{\rho}{d\rho/dy} \right| .$$

(6.4)

### 6.1.1 Magnetic Field

When an externally applied magnetic field is present, the growth of RTI is suppressed thus mix is reduced, and the RTI growth rate becomes,

$$\gamma = \sqrt{\frac{A_t k g}{1 + A_t k L_m}} - \frac{(B \cdot k)^2}{2\pi \mu_0 (\rho_h + \rho_l)}.$$  

(6.5)

The magnetic field that is parallel to the fluid interface stabilizes the RTI for wavelengths below a critical wavelength,

$$\lambda_c = \frac{B^2}{2\mu_0 g (\rho_h - \rho_l)}.$$  

(6.6)

The magnetic field also grows exponentially into the non-linear phase of RTI due to the stretch-twist-fold MHD dynamo according to Childress and Gilbert [1995], Srinivasan and Tang [2013].

### 6.1.2 Viscosity

When viscosity is considered, the RTI growth is also affected. The growth rate of RTI in the presence of constant viscosity for incompressible flow has been reported in Chandrasekhar [1961]. According to Chandrasekhar [1961], the RTI growth rate changes with the wavenumber as

$$\gamma \rightarrow \begin{cases} \sqrt{(\alpha_h - \alpha_l)k}, & \text{for } k \to 0, \\ \sqrt{(\alpha_h - \alpha_l)/2k}, & \text{for } k \to \infty, \end{cases}$$  

(6.7)

where $\alpha_l = \rho_l / (\rho_l + \rho_h)$, $\alpha_h = \rho_h / (\rho_l + \rho_h)$, thus $\alpha_h - \alpha_l = A_l$, and $\alpha_l + \alpha_h = 1$. According to equation 6.7, $\gamma$ approaches 0 both when $k \to 0$ and $k \to \infty$. Thus, a mode of maximum instability exists, and this is approximated numerically in Chandrasekhar [1961] and provided in Figure 6.1.

However, in tests described later in this chapter, the viscosities are non-constant and calculated using equation 2.61, thus the RTI growth rate may not be well approximated from the
Figure 6.1: The growth rate $n$ (notated as $\gamma$ in this dissertation) changes with wavenumber $k$ for different Atwood number. $n$ ($\gamma$) is measured in the unit of $(g^2/\nu)^{\frac{1}{3}}$, and $k$ is measured in the unit of $(g/\nu^2)^{\frac{1}{3}}$, where $\nu$ is the kinetic viscosity. The curves labeled 1, 2, 3, 4, 5, 6, 7, 8 are for $A_t$ of 0.01, 0.05, 0.10, 0.15, 0.25, 0.5, 0.9, 1.0, respectively. Figures are taken from [Chandrasekhar, 1961].
results in Chandrasekhar [1961]. Furthermore, the fluids considered here are compressible. The RTI growth of superimposed viscous flow is better approximated by the theory described in Menikoff et al. [1977], Atzeni and Meyer-ter Vehn [2004],

\[ \gamma = \sqrt{\frac{A_t k g}{1 + A_t k L_m}} \left( \sqrt{1 + \omega} + \sqrt{\omega} \right), \] (6.8)

where \( \omega = \bar{\nu}^2 k^3 / A_t g \), and \( \bar{\nu} \) is a density averaged kinematic viscosity,

\[ \bar{\nu} = \frac{\mu_l + \mu_h}{\rho_l + \rho_h}, \] (6.9)

with the viscosities of two fluids, \( \mu_l \) and \( \mu_h \) calculated using equation 2.61. The RTI calculations performed in this chapter are for compressible fluids well into the non-linear phase of instability growth. Hence, theory alone is insufficient to describe the instabilities and numerical simulations are necessary.

### 6.1.3 Thermal Conduction

Thermal conduction transfers energy across the interface between hot and cold fluids. When spikes of the cold, high-density fluid enter the hot, low-density fluid, heat transfer ablates the spike and thus suppresses RTI. While this is beneficial for hydrodynamic mix reduction, it can be detrimental from the point of view of maintaining a high-temperature hot-spot due to substantial mix between cold and hot fuel.

### 6.1.4 Resistivity

Electrical resistivity does not affect the RTI growth directly. However, it is a diffusion mechanism of the magnetic field. The growth of magnetic field is suppressed at the fluid interface, thus the reduction of mix due to magnetic field becomes less significant according to equation 6.5, because the stabilizing component of the growth rate formula is decreased when the magnitude of B is decreased.

### 6.2 Growth Rate Validation of Single-mode RTI

A single-mode RTI is simulated here using the Euler equations in PHORCE to benchmark against the linear theory described in equation 6.2 (the initial density interface is set to be discontinuous since the Atwood number is small, which is numerically manageable in PHORCE). The simulation setup is provided in Table 6.1. Density profile at \( t = 15 \) is presented in Figure 6.2. The peak bubble to spike distance is measured over time and compared against
Table 6.1: Simulation setup for single-mode RTI test

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial interface location:</td>
<td>$y_0 = 0$</td>
</tr>
<tr>
<td>Atwood number:</td>
<td>$At = \frac{1}{3}$</td>
</tr>
<tr>
<td>Interface density profile:</td>
<td>discontinuous</td>
</tr>
<tr>
<td>Gravitational acceleration:</td>
<td>$g = 0.1 \text{ m s}^{-2}$</td>
</tr>
<tr>
<td>Domain width:</td>
<td>$L_x = 0.5 \text{ m}$</td>
</tr>
<tr>
<td>Domain height:</td>
<td>$L_y = 1.5 \text{ m}$</td>
</tr>
<tr>
<td>Left &amp; right boundary conditions:</td>
<td>periodic</td>
</tr>
<tr>
<td>Top &amp; bottom boundary conditions:</td>
<td>slip wall</td>
</tr>
<tr>
<td>Initial perturbation on velocity:</td>
<td>$v' = \frac{e^{-1000y^2}}{400} [1 + \cos(2\pi x/L_x)] [1 + \cos(4\pi y/L_y)]$</td>
</tr>
<tr>
<td>Initial pressure:</td>
<td>$p = p_0 - g\rho y$</td>
</tr>
<tr>
<td>Background pressure:</td>
<td>$p_0 = 8\rho_h g L_y$</td>
</tr>
</tbody>
</table>

linear theory, as shown in Figure 6.3. The growth rate calculated from the curve fitting of the resulting bubble-to-spike amplitude is 0.6168, which is 4.7% below the growth rate calculated from linear theory (equation 6.2), $\gamma = \sqrt{At kg} = 0.6472$. Thus, the benchmark of single-mode RTI growth calculated from simulation performed by PHORCE against linear theory is considered successful.

### 6.3 Statistical Study of Multimode RTI

A multimode, with mode numbers $m = 1$ to 23, RTI test is performed using Euler equations with the level-set method (described in section 2.7.1) in PHORCE. The simulation setup is provided in Table 6.2. Since the Atwood number is high, the interface density profile is initialized with a hyperbolic tangent profile for numerical purposes of handling the very sharp gradients initially,

$$\rho = \rho_l + \frac{1}{2} \left( \tanh \left( \frac{2\pi y}{w L_y} \right) + 1 \right) (\rho_h - \rho_l)$$

where $w$ is the width of the hyperbolic tangent profile. In order to maintain hydrostatic equilibrium, the initial pressure is calculated as,

$$p = p_0 - \int \rho g dy = p_0 - \frac{(\rho_l + \rho_h) g y}{2} - \frac{(\rho_h - \rho_l) g}{2} \ln \left( \cosh \left( \frac{2\pi y}{w L_y} \right) \right)$$

(6.11)
Figure 6.2: Density profile of single-mode RTI test at time $t = 15$ s

One may be tempted to choose a very small $w$ on a very refined mesh when enough computational resources are available. However, it is very important to note that the trigonometric function $\cosh$ can easily exceed the maximum magnitude of results that can be represented by a value of the return type in C++, which will then return $\text{inf}$. Large values returned by $\cosh$ are not an issue due to the logarithm of the value used here. But $\text{inf}$ will produce invalid results for the logarithm. A practical guideline is to keep $w \geq 0.01$, no matter how refined the mesh is. For $w = 0.01$,  

$$
\cosh \left( \frac{2\pi y_{\max}}{wL_y} \right) = 1.37 \times 10^{136},
$$

where $y_{\max} = 0.5L_y$.

In this test, statistics of the characteristic sizes of the bubbles and spikes in RTI at $t = 3.5$ s are generated using results from the marker function $\phi$. $\phi = 1$ in the heavy fluid, $\phi = 0$ in the light fluid, thus $\phi = 0.5$ can be treated as the boundaries of bubbles or spikes. The density profile with $\phi = 0.5$ contour is presented in Figure 6.4-(a). Marker functions are plotted on six vertical lines (a-f) and four horizontal lines (g-j) at $t = 3.5$, as shown in Figure
Figure 6.3: Peak bubble to spike distance $h_{\text{peak}}$ over time (normalized to growth time as $t_{\gamma_{\text{RT}}}$) of single-mode RTI. Top figure uses linear scale, bottom figure uses log scale for $h_{\text{peak}}$. The blue line is the simulation result, the orange line is an exponential curve fit to the simulation result, and the red line indicates the linear theory.

6.4-(b) and -(c). The characteristic sizes of the bubbles and spikes are included in Table 6.3, where the maximum bubble and spike widths are colored as red, while the minimum bubble and spike widths are colored as blue. Gray entries are invalid measurements, where the contours of $\phi$ are not closed, which cannot be treated as bubbles or spikes. This test
Table 6.2: Simulation setup for multimode RTI test

<table>
<thead>
<tr>
<th>Initial interface location:</th>
<th>$y_0 = 0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atwood number:</td>
<td>$At = 0.8$</td>
</tr>
<tr>
<td>Interface density profile:</td>
<td>$\rho = \rho_l + \frac{1}{2} \left( \tanh \left( \frac{2\pi y_0}{0.025L_y} \right) + 1 \right) (\rho_h - \rho_l)$</td>
</tr>
<tr>
<td>Gravitational acceleration:</td>
<td>$g = 0.5 \text{ m s}^{-2}$</td>
</tr>
<tr>
<td>Domain width:</td>
<td>$L_x = 0.5 \text{ m}$</td>
</tr>
<tr>
<td>Domain height:</td>
<td>$L_y = 0.75 \text{ m}$</td>
</tr>
<tr>
<td>Left &amp; right boundary conditions:</td>
<td>periodic</td>
</tr>
<tr>
<td>Top &amp; bottom boundary conditions:</td>
<td>slip wall</td>
</tr>
<tr>
<td>Initial perturbation on vertical velocity:</td>
<td>$v' = e^{-1000y^2/2400} \sum_{m=1}^{23} r_m \cos \left( 2\pi \left( \frac{mx}{L_x} + r_m \right) \right)$</td>
</tr>
<tr>
<td>Initial pressure:</td>
<td>$p = p_0 - \int p \rho g dy$</td>
</tr>
<tr>
<td>Background pressure:</td>
<td>$p_0 = 8\rho_h g L_y$</td>
</tr>
</tbody>
</table>

demonstrates the potential applications of interface tracking methods from PHORCE in the study of bubble/spike characteristics (size, counts, etc.) evolution during RTI growth. The statistics of bubble/spike sizes and interfacial mix is an important component in the study of overall hydrodynamic mix in inertial fusion and astrophysical plasmas. A study of chunk mixing of fuel and pusher materials and their sizes can impact fusion reactivity and result in yield reduction. This study is intended to provide statistics for kinetic calculations of yield reduction in ICF [McDevitt et al., 2018].

### 6.4 Effect of Resistivity

In this section, multimode RTI with the same setup described in Table 6.2 are tested with an in-plane magnetic field and constant uniform resistivity using a resistive GLM-MHD model (equation 2.53-2.57). The in-plane magnetic field is initialized to be $B = (0, 0.1, 0)$. The constant uniform resistivity is set to be 0.002, which is the largest resistivity one can chose without decreasing the explicit time-step calculated from the hyperbolic solver. The resistivity can be characterized by the Lundquist number,

$$L_u = \frac{\mu_0 L_x V_A}{\eta},$$

(6.13)

where $V_A$ is the Alfven speed described in Chapter 2. Similar to viscous flow, the resistive magnetized flow can be characterized by a magnetic Reynolds number,

$$R_m = \frac{\mu_0 L_x V}{\eta} = M_m L_u,$$

(6.14)
Figure 6.4: Density profile with $\phi = 0.5$ contour (a) of multimode RTI test at $t = 3.5$. Marker functions from six vertical lines (a-f) and four horizontal lines (g-j) indicated in (a) are plotted in (b) and (c).

where $V$ is fluid velocity, and $M_m$ is the magnetic Mach number,

$$M_m = \frac{V}{V_A}.$$  \hspace{1cm} (6.15)

Three simulations are performed, (a) hydrodynamic multimode RTI without magnetic field, (b) multimode RTI with an in-plane initial magnetic field $\mathbf{B} = (0.1, 0, 0)$, and (c) multimode RTI with an in-plane initial magnetic field $\mathbf{B} = (0.1, 0, 0)$ and a constant uniform constant resistivity $\eta = 0.002$, which corresponds to $R_m = 2.5$. The density profiles of three tests are shown in Figures 6.5, 6.6, and 6.7, at time $t = 1.4$, $1.7$, and $2.0$, respectively. In case (b),
Table 6.3: Characteristic width of bubbles and spikes of multimode RTI test at $t = 3.5$. Maximum bubble and spike widths are colored as red, while the minimum bubble and spike widths are colored as blue. The maximum spike width is measured only from the horizontal lines (f-j). Gray entries are invalid measurements, where cannot be treated as bubbles or spikes. The minimum width is marked as blue color and the maximum width is marked with red color.

<table>
<thead>
<tr>
<th>line</th>
<th>spike 1</th>
<th>spike 2</th>
<th>spike 3</th>
<th>spike 4</th>
<th>bubble 1</th>
<th>bubble 2</th>
<th>bubble 3</th>
<th>bubble 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0.0083</td>
<td>0.0143</td>
<td>0.0555</td>
<td></td>
<td>0.0135</td>
<td>0.0225</td>
<td>0.0960</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>0.0120</td>
<td>0.0097</td>
<td>0.0323</td>
<td></td>
<td>0.0525</td>
<td>0.1013</td>
<td>0.0278</td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>0.0097</td>
<td>0.0368</td>
<td></td>
<td></td>
<td>0.0165</td>
<td>0.0352</td>
<td></td>
<td></td>
</tr>
<tr>
<td>d</td>
<td>0.2250</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.0795</td>
<td></td>
<td></td>
</tr>
<tr>
<td>e</td>
<td>0.0217</td>
<td>0.0300</td>
<td></td>
<td></td>
<td>0.0383</td>
<td>0.1283</td>
<td></td>
<td></td>
</tr>
<tr>
<td>f</td>
<td>0.0187</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.0742</td>
<td></td>
<td></td>
</tr>
<tr>
<td>g</td>
<td>0.0325</td>
<td>0.0245</td>
<td>0.0870</td>
<td></td>
<td>0.1575</td>
<td>0.0780</td>
<td>0.1205</td>
<td></td>
</tr>
<tr>
<td>h</td>
<td>0.0225</td>
<td>0.0255</td>
<td>0.0295</td>
<td>0.0180</td>
<td>0.1355</td>
<td>0.0230</td>
<td>0.1210</td>
<td>0.1250</td>
</tr>
<tr>
<td>i</td>
<td>0.2705</td>
<td>0.1235</td>
<td></td>
<td></td>
<td>0.0210</td>
<td>0.0850</td>
<td></td>
<td></td>
</tr>
<tr>
<td>j</td>
<td>0.3955</td>
<td>0.0245</td>
<td></td>
<td></td>
<td>0.0350</td>
<td>0.0450</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

max spike width 0.1235 max bubble width 0.0850
min spike width 0.0083 min bubble width 0.0135

the magnetic field completely stabilizes the shorter wavelength modes, which agrees with the theory described in equation 6.6 of section 6.1.1. For case (c), the magnetic field growth around the RTI interface is diffused by resistivity, and thus the shorter wavelength growth is not stabilized. However, the overall bubble to spike distance is still shorter than case (a), as the background magnetic field can still reduce the RTI growth rate, according to the second term described in equation 6.5. This also agrees with the results presented in Figure 6.8 where the profiles of parallel magnetic field $B_x$ at $t = 2.0$ are presented. For comparison purposes, the $B_x = 0$ for case (a) is also included. One can see in case (b) that the magnetic field increases substantially around the interface of RTI, while in case (c) the magnetic field is diffused by the resistivity although a background magnetic field still exists.

6.5 Experimentally Relevant RTI

In this section, multimode RTI, with mode numbers $m = 1$ to 39, is tested in planar geometry using GLM-MHD solver in PHORCE with experimentally relevant parameters, described in Sauppe et al. [2019]. In Sauppe et al. [2019], the RTI is investigated in convergent geometry with cylindrical implosions, with high pressure in high density fluid. For the tests presented in this chapter, planar geometry is used, thus the light fluid has high pressure and high
<table>
<thead>
<tr>
<th>Simulation setup for experimentally relevant multimode RTI test</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Initial interface location:</strong></td>
</tr>
<tr>
<td>$y_0 = 0$</td>
</tr>
<tr>
<td><strong>Heavy fluid density:</strong></td>
</tr>
<tr>
<td>$\rho = 2842.6 \text{ kg m}^{-3}$</td>
</tr>
<tr>
<td><strong>Light fluid density:</strong></td>
</tr>
<tr>
<td>$\rho = 60 \text{ kg m}^{-3}$</td>
</tr>
<tr>
<td><strong>Ion mass:</strong></td>
</tr>
<tr>
<td>$m_i = 2.17438 \times 10^{-26} \text{ kg}$</td>
</tr>
<tr>
<td><strong>Charge state:</strong></td>
</tr>
<tr>
<td>$Z = 3.5$</td>
</tr>
<tr>
<td><strong>Atwood number:</strong></td>
</tr>
<tr>
<td>$At = 0.95866$</td>
</tr>
<tr>
<td><strong>Interface density profile:</strong></td>
</tr>
<tr>
<td>$\rho = \rho_l + \frac{1}{2} \left( \text{tanh} \left( \frac{2\pi y_0}{0.01L_y} \right) + 1 \right) (\rho_h - \rho_l) \text{ kg m}^{-3}$</td>
</tr>
<tr>
<td><strong>Gravitational acceleration:</strong></td>
</tr>
<tr>
<td>$g = 7.605 \times 10^{12} \text{ m s}^{-2}$</td>
</tr>
<tr>
<td><strong>Domain width:</strong></td>
</tr>
<tr>
<td>$L_x = 250 \mu\text{m}$</td>
</tr>
<tr>
<td><strong>Domain height:</strong></td>
</tr>
<tr>
<td>$L_y = 50 \mu\text{m}$</td>
</tr>
<tr>
<td><strong>Left &amp; right boundary conditions:</strong></td>
</tr>
<tr>
<td>periodic</td>
</tr>
<tr>
<td><strong>Top &amp; bottom boundary conditions:</strong></td>
</tr>
<tr>
<td>slip &amp; conducting wall</td>
</tr>
<tr>
<td><strong>Initial perturbation on velocity:</strong></td>
</tr>
<tr>
<td>$v' = \frac{e^{-1000(y/L_y)^2}}{100} \sum_{m=1}^{39} r_m \cos \left( \frac{2\pi (mx/L_x + r_m)}{L_x} \right) \text{ m s}^{-1}$</td>
</tr>
<tr>
<td><strong>Initial pressure:</strong></td>
</tr>
<tr>
<td>$p = p_0 - \int p g dy$</td>
</tr>
<tr>
<td><strong>Background pressure:</strong></td>
</tr>
<tr>
<td>$p_0 = 7.82634 \times 10^{11} \text{ Pa}$</td>
</tr>
</tbody>
</table>
Figure 6.5: Comparison of density profiles for top: hydrodynamic multimode RTI without magnetic field, middle: multimode RTI with an in-plane initial magnetic field \( B = (0, 1, 0, 0) \), and bottom: multimode RTI with an in-plane initial magnetic field \( B = (0, 1, 0, 0) \) and a constant uniform constant resistivity \( \eta = 0.002 \), at \( t = 1.4 \).

The setup in Table 6.4 is tested for \( B_x = 0, 50 \text{T}, 100 \text{T}, \) and \( 214 \text{T} \). A comparison of the mass density profiles for \( B_x = 0 \) and \( B_x = 50 \text{T} \) at \( t_{\gamma_{RT}} = 4.7 \) is presented in Figure 6.10. The temperature and heavy fluid has low pressure and low temperature in order to maintain hydrostatic equilibrium. The simulation setup is provided in Table 6.4. Tests are performed with different initial in-plane magnetic fields of \( B_x = 0, 50 \text{T}, 100 \text{T}, \) and \( 214 \text{T} \). Simulations of slightly different settings are also performed to study the effects of viscosity and thermal conduction on RTI growth. All simulations in this section are performed on a mesh shown in Figure 6.9 with 200,000 triangles, except for the cases with thermal conduction, which are tested on the same type of mesh with 32,000 triangles. Since the Atwood number \( A\tau = 0.95866 \) is very high, which corresponds to a density ratio of \( \rho_h/\rho_l = 47.38 \) at the fluid interface, the robustness of PHORCE is demonstrated in this section.

### 6.5.1 Effects of Magnetic Field

The setup in Table 6.4 is tested for \( B_x = 0, 50 \text{T}, 100 \text{T}, \) and \( 214 \text{T} \). A comparison of the mass density profiles for \( B_x = 0 \) and \( B_x = 50 \text{T} \) at \( t_{\gamma_{RT}} = 4.7 \) is presented in Figure 6.10. The
time is normalized by the growth time of $1/\gamma_{RT} = 1.5 \text{ ns}$, which corresponds to the growth rate for the mode with wavelength $\lambda = 103 \mu m$. The wavelengths of the initial perturbations range from $6.41 \mu m$ to $250 \mu m$. Similar to the results presented in section 6.4, modes with shorter wavelength are stabilized and the growth of the RTI is reduced. The profiles of density and the magnitude of magnetic field are presented for cases of initial $B_x = 50 \text{T}$ at $t\gamma_{RT} = 4.7$, $B_x = 100 \text{T}$ at $t\gamma_{RT} = 6.7$, and $B_x = 214 \text{T}$ at $t\gamma_{RT} = 12.7$ in Figures 6.11, 6.12, and 6.13, respectively. The initial magnetic field grows exponentially with RTI over time, as shown in Figure 6.14, where the maximum magnitude of magnetic field $|B|_{\text{max}}$ is tracked over time $t\gamma_{RT}$. The critical wavelength (equation 6.6) is calculated using $\lambda_c = \frac{B_{\text{max}}^2}{2\mu_0 \rho_o (\rho_h - \rho_l)}$ and presented in Figure 6.15. This is consistent with the results in Figure 6.11, 6.12, and 6.13, where all wavelengths below the critical wavelength are completely stabilized. Note that the stabilization is consistent as $B_x$ is increased. The peak bubble to spike distance $h_{\text{peak}}$ growth over time $t\gamma_{RT}$ is presented in Figure 6.16, where larger initial $B_x$ leads to a smaller growth rate, which agrees with the linear theory described in equation 6.5. The plasma $\beta$ is the ratio of the plasma pressure to magnetic pressure,

$$\beta = \frac{p}{B^2/2\mu_0}.$$  

(6.16)

The plasma $\beta$ evolution with $h_{\text{peak}}$ is presented in Figure 6.17.
6.5.2 Effects of Viscosity

Multimode RTI in the presence of viscosity is simulated with the same setup described in Table 6.4, except with a background pressure of $p_0 = 4.43064 \times 10^9$ Pa which is four times the original one. The viscosities for each fluid are first calculated using equation 2.61, and then the total viscosity is calculated using the mass fraction $\phi_m$ from MOF solver as described in equation 2.81. $p_0$ is modified so that the Reynolds number is small enough in the light fluid for the viscosity to have an impact. The initial Reynolds number is calculated as,

$$R_e = \frac{\rho (\gamma_{RT} L_y) L_y}{\mu}, \quad (6.17)$$

where $\gamma_{RT} = 1.5$ is the same as in section 6.5.1, and the characteristic speed is considered as $\gamma_{RT} L_y$. This result is presented in Figure 6.18, where the vertical location is normalized by $L_y$. $Re$ is approximately 83 in the light fluid, and $2.3 \times 10^7$ to $3.5 \times 10^7$ in the heavy fluid. Such a disparate $Re$ is typical of ICF regimes. The $Re$ of 83 is expected to have a substantial stabilization effect of short-wavelength RTI while the $Re$ of $2.3 \times 10^7$ is expected to have no impact. Hence, studying what happens to RTI stabilization when such disparate profiles exist is of interest. A comparison of the mass density profiles for inviscid and viscous
Figure 6.8: Comparison of $B_x$ profiles for three cases described in Figure 6.5 at $t = 2.0$.

Figure 6.9: Illustration of the mesh for the experimentally relevant RTI tests.

RTI tests at $t_{\gamma RT} = 4.7$ is presented in Figure 6.19. Note stabilization of the shortest wavelength modes and reduction of the peak bubble-to-spike amplitude in the case with viscosity compared to the case without.

### 6.5.3 Effects of Thermal Conduction

Multimode RTI in the presence of thermal conductivity calculated using equation 2.62 is simulated with the same setup described in Table 6.4, except the density is scaled up to
Figure 6.10: Comparison of mass density profiles for experimentally relevant RTI tests of initial $B_x = 0$ (top), and $B_x = 50$ T (bottom) at $t\gamma_{RT} = 4.7$.

Figure 6.11: Profiles of mass density (top) and magnitude of magnetic field (bottom) for experimentally relevant RTI tests of initial $B_x = 50$ T at $t\gamma_{RT} = 4.7$.

100 times of the original density in order to not significantly change the time-step $\Delta t$ in the explicit SSP-RK scheme used in PHORCE. To perform tests of realistic electron thermal conduction on ICF-relevant RTI parameters, an implicit or semi-implicit treatment is necessary. This is beyond the scope of this work and constitutes future work. However, some effect may be included by limiting thermal conductivity and studying its impact on RTI. This test is performed on a mesh shown in Figure 6.9 with 32,000 triangles. The comparison of the mass density profiles for RTI tests without and with thermal conduction are presented at
Figure 6.12: Profiles of mass density (top) and magnitude of magnetic field (bottom) for experimentally relevant RTI tests of initial $B_x = 100$ T at $t_{\gamma RT} = 6.7$.

Figure 6.13: Profiles of mass density (top) and magnitude of magnetic field (bottom) for experimentally relevant RTI tests of initial $B_x = 214$ T at $t_{\gamma RT} = 12.7$.

$t_{\gamma RT} = 6.7$ in Figure 6.20. As discussed in section 6.1.3, the energy transfer at the density interface ablates the spike from the cold fluid into the hot fluid. Short-wavelength modes are stabilized and the overall growth of RTI is reduced substantially.
Figure 6.14: Maximum magnitude of magnetic field $|\mathbf{B}|_{\text{max}}$ growth over time ($t_{\gamma RT}$) for experimentally relevant RTI tests of initial $B_x = 50 \text{T}$, $B_x = 100 \text{T}$, and $B_x = 214 \text{T}$.

Figure 6.15: Critical wavelength $\lambda_c$ growth over time ($t_{\gamma RT}$) for experimentally relevant RTI tests of initial $B_x = 50 \text{T}$, $B_x = 100 \text{T}$, and $B_x = 214 \text{T}$. $\lambda_c$ is calculated using $\lambda_c = \frac{B_{\text{max}}^2}{2 \mu_0 g (\rho_h - \rho_l)}$. 
6.6 Radial Implosion

Numerical simulations of high-energy-density implosion hydrodynamics relevant to inertial confinement fusion and astrophysics are challenging due to the large gradients in density, tem-
Figure 6.18: Initial Reynolds number $Re$ plot along $y/L_y$. $Re$ is about 83 in the light fluid, and $2.3 \times 10^7 \sim 3.5 \times 10^7$ in the heavy fluid.

Figure 6.19: Comparison of mass density profiles for experimentally relevant RTI tests without (top) and with viscosity (bottom) using the same setup in Table 6.4 (except the pressure is four times of the original pressure) at $t\gamma_{RT} = 4.7$.

perature, and pressure in these regimes that increase substantially as the implosions progress in time. While a number of 1-dimensional tools exist that are able to access these regimes, multi-dimensional simulations remain a challenge due to the growth of hydrodynamic in-
stabilities at the sharp interfaces [Clark et al., 2016, Srinivasan et al., 2012, Srinivasan and Tang, 2012, Wang et al., 2017, Srinivasan and Tang, 2013, Li et al., 2018, Srinivasan and Tang, 2014b], the need to resolve general geometries by mitigating the effects of grid shapes from affecting the dynamics [Joggerst et al., 2014], the highly disparate parameters that are encountered across relatively short spatial scales [Srinivasan and Tang, 2014a], and the need to resolve disparate spatial and temporal scales, to name a few. Furthermore, a majority of numerical simulations do not incorporate the highly disparate Reynolds numbers (and magnetic Reynolds numbers if including magnetic fields using MHD models) that occur in these regimes [Srinivasan and Tang, 2014a]. To address these challenges, this work demonstrates the application of the unstructured mesh aRDG algorithm developed here for implosion simulations in high-energy-density hydrodynamics employing highly disparate densities, temperatures, and viscosities over short spatial scales.

The radial implosion problem setup is adapted from Joggerst et al. [2014]. In Joggerst et al. [2014], the circular shape of the implosion without any perturbation is well maintained when using a spherical coordinate system. However, the circular shape of the implosion is changed by a structured mesh in Cartesian coordinate system, limiting the geometric flexibility of both types of coordinate systems in these codes. To explore this in the unstructured DG code using the aRDG algorithm for diffusion, simulations are performed on one quadrant of a circle.

Simulations are performed with unstructured meshes of approximately 3,000,000 and 1,000,000 triangular elements. The mesh elements are guided by a series of circles with size of the ele-
ment proportionally decreasing moving inward in radius until a radius well within the inner fluid, within which the element size remains similar. A lower resolution illustration of this mesh is presented in Figure 6.21. The inner region for \( r < 10 \text{ cm} \) in Figure 6.21 is a low-density region, followed by a high-density region for \( 10 \text{ cm} < r < 12 \text{ cm} \) with an Atwood number of 0.9 across the \( r = 10 \text{ cm} \) interface. For \( r > 12 \text{ cm} \) there is a low-density, high-pressure region that acts as a pusher for the implosion. An initial random multimode perturbation is applied at the interface between the inner region and the dense shell \( (r = 10 \text{ cm}) \). The initial conditions are also provided in Table 6.5.

Table 6.5: Initial conditions of radial implosion test

<table>
<thead>
<tr>
<th>Fluid</th>
<th>( \rho ) (kg m(^{-3}))</th>
<th>( p ) (Pa)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( R_1 = 10 \text{ cm} )</td>
<td>50</td>
<td>0.006</td>
</tr>
<tr>
<td>( R_2 = 12 \text{ cm} )</td>
<td>1000</td>
<td>0.006</td>
</tr>
<tr>
<td>( R_3 = 15 \text{ cm} )</td>
<td>100</td>
<td>0.6</td>
</tr>
</tbody>
</table>

The left and bottom boundary are set to be periodic in the azimuthal direction. The implosion is driven by a boundary that is initially located at \( r > 13 \text{ cm} \) and is moving radially inward. The density in the boundary is held constant at 100.0 kg/m\(^3\) and the pressure is held constant at 0.6 Pa for the first 0.5 seconds, then drops linearly to \( 0.6 \times 10^{-2} \text{ Pa} \) by \( t = 3.0 \text{ s} \). The location of the boundary is given by

\[
R_{bd}(t) = R_0(1 - 0.2t),
\]

where \( R_0 = 13 \text{ cm} \). The velocity in the boundary region is given by

\[
V_{bd}(r, t) = -0.2R_0 \left( \frac{r}{R_{bd}(t)} \right).
\]

The implosion, without any applied perturbation, is performed for the mesh with 3,000,000 elements and the density profiles at \( t = 1.5 \text{ s} \) and \( t = 2.5 \text{ s} \) are presented in Figure 6.22 and 6.23. The spikes shown on the outer region of the shell originate from grid level perturbations. Same small randomized grid noise can be observed at \( t = 2.5 \text{ s} \). It is important to point out that the circular shape of the implosion is well maintained using PHORCE which uses Cartesian coordinate system. This provides confidence that PHORCE with careful mesh generation can be used on problems that involve complex geometries and configurations.

This case is also tested on the same resolution with an initial multimode perturbation at the interface \( r = 10 \text{ cm} \) between the inner region and the dense shell. The perturbation is applied to the radial velocity along the azimuthal direction,

\[
v'_r = \frac{e^{-1000((r-R_1)/R_1)^2}}{1000} \sum_{m=1}^{40} \frac{\text{rand}_m}{\sqrt{m}} \cos(4m(\theta + 0.5\text{rand}_m\pi)),
\]

\( \text{rand}_m \)
Figure 6.21: Demonstration of the mesh used in radial implosion test with a coarse version.

Figure 6.22: Density profile of implosion case without perturbation at time $t = 1.5\,\text{s}$ using the Euler equations. Mesh resolution: 3,000,000 triangular elements.
where \( \text{rand}_{m=1,\ldots,40} \) are random numbers taken between \([0, 1]\). The density profiles at \( t = 1.5 \) s and \( t = 2.5 \) s are presented in Figure 6.24 and 6.25. Significant turbulent mixing due to the growth of the RTI can be observed at the inner interface. Note that the small scale features of the RTI mixing are well captured even with these large density gradients.

Simulations are performed on the mesh with 1,000,000 elements applying the aRDG algorithm to include disparate viscosities and explore their impact on the RTI growth during implosions. As a comparison, an inviscid simulation is performed on this mesh, and the density evolution at \( t = 2.5 \) s is presented in Figure 6.26. The initial perturbation applied on the radial velocity at \( r = R_1 \) is,

\[
v'_r = \frac{e^{-1000((r-R_1)/R_1)^2}}{3162} \sum_{m=1}^{40} \text{rand}_m \cos (4m(\theta + 0.5\text{rand}_m\pi)).
\]

(6.21)

MOF algorithm is used to apply the corresponding viscosity to the different sides of the interface, thus accounting for disparate Reynolds numbers across a sharp interface region. Viscosities are calculated using equation 2.81. The dense shell fluid viscosity is set as \( \mu_h = 1 \times 10^{-6} \text{kg m}^{-1} \text{s}^{-1} \), which corresponds to an inviscid regime. For the inner fluid, \( \mu_l = 1 \times 10^{-4} \text{kg m}^{-1} \text{s}^{-1} \) and \( \mu_l = 3 \times 10^{-4} \text{kg m}^{-1} \text{s}^{-1} \) are explored, which correspond to Reynolds numbers of approximately 1,300 and 400, respectively. The density evolution for these two viscous cases at \( t = 2.5 \) s are presented in Figures 6.27 and 6.28, respectively. RTI growth is impacted by viscosity where an inviscid simulation would permit development of turbulence while large viscosities stabilize short wavelength modes adjusting the flow to be more laminar. The effect of disparate viscosity across an interface with RTI growth, where
Figure 6.24: Density profile of implosion case without multimode perturbation at time $t = 1.5\,\text{s}$ using the Euler equations. Mesh resolution: 3,000,000 triangular elements.

Figure 6.25: Density profile of implosion case with multimode perturbation at time $t = 2.5\,\text{s}$ using the Euler equations. Mesh resolution: 3,000,000 triangular elements.
the bubbles grow into inviscid regions while spikes grow into viscous regions, constitutes open and important research in the field of high-energy-density hydrodynamics. While these simulations sufficiently demonstrate the capability of the aRDG algorithm to resolve disparate viscosities, even more extreme Reynolds number variation across an interface will constitute future physics studies.
Figure 6.27: Density profile of implosion case with multimode perturbation at time $t = 2.5$ s using the Navier-Stokes equations, with $\mu_l = 1 \times 10^{-4}$ kg m$^{-1}$ s$^{-1}$. Mesh resolution: 1,000,000 triangular elements.

Figure 6.28: Density profile of implosion case with multimode perturbation at time $t = 2.5$ s using the Navier-Stokes equations, with $\mu_l = 3 \times 10^{-4}$ kg m$^{-1}$ s$^{-1}$. Mesh resolution: 1,000,000 triangular elements.
Chapter 7

Conclusion

Although this work is motivated by physics in high-energy-density (HED) hydrodynamics, the emphasis is placed on numerical models that are generally applicable across a wide variety of fields and disciplines. A significant amount of content in this dissertation focuses on describing the underlying math and implementation details of the nodal discontinuous Galerkin (NDG) method [Hesthaven and Warburton, 2007]. This work results in PHORCE, a multi-dimensional computational framework for solving general convection-diffusion equations on unstructured grids. Considerable effort has been made on increasing the robustness of this framework in solving problems under extreme conditions, such as in initial confinement fusion (ICF) experiments. An accurate and efficient diffusion solver is proposed in this work and implemented in PHORCE to study the effects of diffusion terms on the growth of the Rayleigh-Taylor instability (RTI), a good understanding of which is necessary in order to improve the fusion gain in ICF.

Several convection-diffusion equations, such as the Euler equations, the Navier-Stokes equations, the two-fluid plasma equations, and the MHD equations, that describe the fundamental behavior of neutral fluids and plasmas are introduced in Chapter 2. The regimes of applicability of these models are discussed. Eigenvalues and eigenvectors are described in detail. Interface tracking methods, such as level-set and volume of fluid methods, are also described. The limitations, especially for DG implementations applied to compressible flows, are discussed. A mass of fluid (MOF) method is proposed to track the mass fractions of different fluids in the presence of compressibility. The MOF technique introduces an additional variable to bound and conserve mass while effectively tracking the interface. This is important for problems of interest in this work, as the viscosity at the RTI interface needs to be calculated from the mass fractions due to fluids of different masses and charge states separated by the interface.

The NDG method, on which the development of PHORCE is based, has been described theoretically and practically with significant details in Chapter 3. The spatial discretization of NDG on hyperbolic conservation laws has been provided and the elemental operations have
been discussed. Affine transformation is introduced to solve the semi-discrete forms of the
system on a reference domain with emphasis on computational efficiency. The implementation
details and the robustness of the slope limiter, WENO limiter, and positivity preserving
(PP) limiter are discussed. A MOF bound limiter, motivated by the PP limiter described in
Zhang and Shu [2010], Cheng et al. [2013], is proposed to bound the mass fraction $\phi$ between
$[0, 1]$, while ensuring mass conservation.

A new DG diffusion method, the affine reconstructed DG (aRDG) method, is proposed and
detailed in Chapter 4. The aRDG method is designed to solve the diffusion terms accurately
and efficiently on unstructured grids of triangles. The key feature of the aRDG method is
that the reconstruction is performed on the enclosed parallelogram of two adjacent triangles,
from which the transformation to reference domain is also an affine transformation. Thus,
a significant amount of storage and computational effort can be avoided compared to other
methods. A practical guideline on how to apply this algorithm to the nodal discontinuous
Galerkin method has been provided.

Code verification and benchmark cases of PHORCE are presented in Chapter 5. The hyperbolic
solver is verified for Euler equations and ideal-MHD equations, while the diffusion solver is
verified for multiple linear and non-linear scalar equations, and the Navier-Stokes equations.
Benchmark tests are performed for Euler equations, ideal-MHD equations, and Navier-Stokes
equations against established results in the literature.

Finally, the theory of Rayleigh-Taylor instability growth, under difference circumstances,
in the presence of magnetic fields, viscosity, resistivity, and thermal conduction, has been
described in Chapter 6. The RTI evolution in planar geometry is studied using the MHD
solver in PHORCE. The influence of resistivity on RTI growth is presented on a multi-mode
RTI test with a constant resistivity. The effects of magnetic fields, viscosity, and heat
conduction on RTI growth are discussed on simulation results with experimentally relevant
parameters, where very high Atwood number ($A_t = 0.95866$) is used. Robustness of PHORCE
is well demonstrated in these tests. MOF implementations are also tested in the cases
with viscosity, where a disparate Reynolds number ($Re$) as is the case in ICF, has been
shown to produce stable and interesting results. High $A_t$ and disparate $Re$ simulations are
challenging to perform. This work has demonstrated these challenging parameter regimes,
not just in simplified planar geometries, but also in converging implosion geometries using
an unstructured grid with a high-order accurate, newly-developed diffusion solver. Such a
contribution is novel and unprecedented in HED simulations.

Substantial work has been done in the development of PHORCE, substantial work remains to
be done as well. Several future projects are suggested for graduate students and researchers
involved in order to make PHORCE truly competitive:

1. Overall improvements to the DG method should be considered to improve on any errors
   introduced through aliasing. Some calculations in the nodal DG scheme may introduce
   aliasing errors. One cause includes the lack of exact integration which can potentially
be overcome through use of filters or through use of exact integration [Hesthaven and Warburton, 2007]. Another source of aliasing errors could be through the node-by-node division and multiplication that occurs in obtaining some primitive variables from conserved variables. Examples include obtaining velocity from momentum and density and obtaining temperature from other conserved variables. A potential manner to overcome this includes a weak form of division and multiplication [Hakim et al., 2019].

2. In PHORCE, the use of virtual functions in C++ may be expensive due to its indirection of function calls (look for the address of the function from the virtual method table before call it). Consider replacing these with templated functions instead.

3. There are some challenges with the diffusion solver when the Atwood number is high and the gradients are very large. The source of these computational challenges could be due to either the diffusion time-step which may not be as conservative as necessary or it could be due to the reconstructed gradients from aRDG producing spurious oscillations when the gradients of the variables have sharp gradients. Explore conservative time-steps for diffusion when using aRDG and also explore whether limiters are necessary for gradients of variables when using reconstruction on extreme jumps. The development of limiters for aRDG may be necessary for HED-relevant problems.

4. The WENO limiter works very well but is extremely costly, it increases the computational effort of the DG scheme by approximately a factor of four to eight, depending on how many troubled-cell are identified. Consider optimizing and speeding up the WENO limiter and also consider using cheaper slope (or other) limiters that may work almost as effectively with much less cost.

5. The present work provides some calculations with disparate Reynolds numbers across sharp gradients using interface tracking methods. In this work, a linear combination is used for the viscosities when combined with the mass fractions of the fluids. More sophisticated and physically-relevant combinations of the viscosities need to be considered across the interface in the future.

6. The extension of PHORCE to 3-D is important on problems where 2-D assumptions are invalid. This requires an extension of the aRDG method to 3-D as well.

7. Explicit time marching method is not practical for systems with wide ranges of temporal and spatial scales, such as the two-fluid plasma model, especially in 3-D. Thus, an efficient and robust implicit time-stepping scheme needs to be implemented. This can also help overcome the challenges of restrictive diffusive time-steps in ICF simulations. Suggestions include super time-stepping [Meyer et al., 2012, 2014] for diffusion terms, exact techniques for implicit time-stepping for the source terms of the two-fluid plasma model [Wang et al., 2019], and implicit techniques for Maxwell’s equations keeping in mind high-order accuracy of DG methods. Multirate methods [Sandu and Constantinescu, 2009] can be considered as well.
8. GPU-parallelized implementations with considerations of memory bandwidth in modern GPU chips can also be very helpful in order to push the boundaries of possible applications in PHORCE. Significant strides have been made in the development of libraries for heterogeneous architectures. These include OCCA [Medina et al., 2014], Kokkos [Edwards et al., 2014], and RAJA [Hornung and Keasler, 2014] which can be leveraged for MPI-GPU implementations.

9. A fairly optimized MPI architecture has been implemented in PHORCE. Some functions also include OpenMP which is easy to extend and generalize to all functions in PHORCE but careful optimization is needed. The extension to GPUs should be prioritized over OpenMP due to emerging computing clusters prioritizing GPU hardware.
Bibliography


Oswald Veblen and John Wesley Young. *Projective geometry*, volume 2. Ginn, 1918.


LiFeng Wang, WenHua Ye, XianTu He, JunFeng Wu, ZhengFeng Fan, Chuang Xue, HongYu Guo, WenYong Miao, YongTeng Yuan, JiaQin Dong, et al. Theoretical and simulation research of hydrodynamic instabilities in inertial-confinement fusion implosions. *SCIENCE CHINA Physics, Mechanics & Astronomy*, 60(5):055201, 2017.


Appendices
Appendix A

PHORCE User Guide

In this appendix, a general user guide of PHORCE is provided. This guide includes installation, mesh generation, input file setup, application driver set-up, and post-processing.

A.1 Installation

PHORCE can be installed on various operation systems. However, only the installation guide on Linux is provided here.

Once PHORCE is downloaded, certain libraries need to be installed first. Installation of PHORCE requires the following dependencies:

1. ParMETIS/METIS
   • For domain partitioning
   • Link: http://glaros.dtc.umn.edu/gkhome/metis/parmetis/overview
   • Version tested: 4.0.3

2. EIGEN
   • For matrix operation
   • Link: http://eigen.tuxfamily.org
   • Version tested: 3.3.7

3. zlib (binary IO and data compression)
   • For binary IO and data compression
   • Link https://www.zlib.net
• Version tested: 1.2.11

It is recommended to create a folder (e.g. $lib) to maintain all these dependencies. Assuming a C/C++ compiler, a MPI compiler, and Cmake are already installed, and the above three dependencies are downloaded, the installation guide for each of the dependencies is provided as follows:

**ParMETIS/METIS**

Once ParMETIS is downloaded and extracted to a folder (${parmetisPkg})

1. First install METIS

   ```
   $ mkdir ${libs}/metis
   $ cd ${parmetisPkg}/metis
   $ make config prefix=${libs}/metis
   $ make install
   ```

2. Then install ParMETIS

   ```
   $ mkdir ${libs}/parmetis
   $ cd ${parmetisPkg}/parmetis
   $ make config prefix=${libs}/parmetis
   $ make install
   $ cp ${PHORCE_parallel}/Documentation/parmetisdefs.h ←
   ← ${libs}/parmetis/include/
   ```

**EIGEN**

Once EIGEN is downloaded and extracted to a folder (${eigen}), there is no need to install EIGEN. One only needs to provide the correct path to the header files, which will be described later.

**zlib**

Once zlib is downloaded and extracted to a folder (${zlibPkg}), it can be installed as

1. If you are using GNU C/C++ compilers

   ```
   $ mkdir ${libs}/zlib
   $ cd ${zlibPkg}
   $ ./configure --prefix=${libs}/zlib
   $ make install
   ```
2. If you are using Intel C/C++ compilers

```
$ mkdir ${libs}/zlib_intel
$ cd ${zlibPkg}
$ ./configure --prefix=${libs}/zlib_intel
$ make install
```

Both version can be installed. During PHORCE installation, it will automatically switch to the right version of zlib based on the current loaded compilers.

Once all dependencies are installed, the following system environment variables need to be specified (idealy in ~/.bashrc):

```
$ export PHORCE_DIR='${PHORCE_parallel}/Code'
$ export PHORCE_Regression='${PHORCE_parallel}/Regression'
$ export Eigen_DIR='${libs}/eigen'
$ export ParMETIS_DIR='${libs}/parmetis'
$ export METIS_DIR='${libs}/metis'

#(only needed when you have installed zlib using GNU compilers)
$ export ZLIB_DIR='${libs}/zlib'

#(only needed when you have installed zlib using Intel compilers)
$ export intel_ZLIB_DIR='${libs}/zlib_intel'

$ PATH=${PHORCE_parallel}/bin:$PATH
```

After setting up these environment variables, PHORCE can be installed as

```
$ cd ${PHORCE_parallel}
$ make all -j [number of cores]
```

## A.2 Mesh Generation Using Gmsh

PHORCE has been developed to read the mesh generated by Gmsh-2.11.0. It is found that the mesh generated by the most recent version of Gmsh (currently 4.5.6) is not compatible with the mesh reader implemented in PHORCE. This is a subject of future work. In this section, mesh generation procedures will be described using Gmsh-2.11.0. The mesh generation can be done easily though Gmsh's graphical user interface. However, a graphical interface isn’t always conveniently available especially on supercomputer clusters. In this guide, mesh generation using Linux command line will be discussed.

The mesh file in Gmsh has an extension of .msh. To generate .msh file, a geometry file .geo needs to be generated first. This can be done through any text editor. The following is a sample .geo file:
// define user-specific variables
Fac = 1.0e-5;
cl_wall = 1.25*Fac;  // define the mesh size on the tunnel wall
cl_cyl = 0.25*Fac;  // define the mesh size on the cylinder
R1 = 1*Fac;  // Radius of the cylinder
X0 = 16*R1;  // x coordinate of the cylinder centroid
Y0 = 16*R1;  // y coordinate of the cylinder centroid

// define points: Point(id) = {x, y, z, mesh size}
Point(1) = {0, 0, 0, cl_wall};
Point(2) = {64*R1, 0, 0, cl_wall};
Point(3) = {64*R1, 32*R1, 0, cl_wall};
Point(4) = {0, 32*R1, 0, cl_wall};
Point(5) = {X0, Y0, 0, cl_cyl};
Point(6) = {X0-R1, Y0, 0, cl_cyl};
Point(7) = {X0, Y0+R1, 0, cl_cyl};
Point(8) = {X0+R1, Y0, 0, cl_cyl};
Point(9) = {X0, Y0-R1, 0, cl_cyl};
Point(10) = {X0-2*R1, Y0, 0, cl_cyl};
Point(11) = {X0, Y0-2*R1, 0, cl_cyl};
Point(12) = {64*R1, Y0-4*R1, 0, cl_cyl};
Point(13) = {64*R1, Y0+4*R1, 0, cl_cyl};
Point(14) = {X0, Y0+2*R1, 0, cl_cyl};

// define lines
// Line(id) = {starting point, ending point}
// Circle(id) = {starting point, centroid, ending point}
Line(1) = {1, 2};
Line(21) = {2, 12};
Line(22) = {12, 13};
Line(23) = {13, 3};
Line(3) = {3, 4};
Line(4) = {4, 1};
Circle(5) = {6, 5, 7};
Circle(6) = {7, 5, 8};
Circle(7) = {8, 5, 9};
Circle(8) = {9, 5, 6};
Line(11) = {12, 11};
Line(12) = {13, 12};
Line(13) = {14, 13};
Circle(14) = {10, 5, 14};
Circle(15) = {11, 5, 10};

// define line loops
// Line Loop(id) = {starting line, ..., ending line}
// adjacent lines should have consistent directions, i.e, the ending point of
// the first line, should be the starting point of the second line
Line Loop(1) = {1, 21, 11, 15, 14, 13, 23, 3, 4};
Line Loop(2) = {5, 6, 7, 8};
Line Loop(3) = {11, 12, 13, 14, 15};

// define surface from line loops: Plane Surface(id) = {line loops that enclose the surface}
// mesh will be build on surface
Plane Surface(6) = {1,3};
Plane Surface(7) = {3,2};

// define physical lines.
// this includes any non-interior lines (physical boundary or periodic boundary)
Physical Line("In") = {4};  // In == 1
Physical Line("Out") = {21, 22, 23};  // Out == 2
Physical Line("Wall") = {1,3};  // Wall == 3
Physical Line("Far") = {};  // Far == 4
Physical Line("Cyl") = {5,6,7,8};  // Cyl == 5
Physical Line("Dirichlet") = {};  // Dirichlet == 6
Physical Line("Neuman") = {};  // Neuman == 7
Physical Line("Slip") = {};  // Slip == 8
Physical Surface(7) = {-6,7};

where certain variables can be defined. A list of points needs to be defined, and the straight
lines (Line) and curved lines (Circle) can be specified. Line loops are generated so that
surfaces can be defined. The mesh will be built on the surfaces. Physical lines are defined so
that boundary information will be included in the .msh file. Note that for every line defined,
there is an ID and a physical (boundary) tag. The lines and physical tags are defined as
follows

| Line(line ID) = {starting point, ending point}; |
| Physical Line(physical tag) = {line ID}; |

In PHORCE, the physical tags are defined through enum variables, which are defined in
phorcevariables.h

```c
enum Boundary_name {
    BC_None = 0,
    BC_In = 1,
    BC_Out = 2,
    BC_Wall = 3,
    BC_Far = 4,
    BC_Cyl = 5,
    BC_Dirichlet = 6,
};
```
Once \texttt{.geo} file is created correctly, the \texttt{.msh} file can be generated by
\begin{verbatim}
$ gmsh FlowOverCylinder_longwake.geo -2
\end{verbatim}
where -2 means 2D mesh generation. The mesh can also be visualized in \texttt{Gmsh} as shown in Figure A.2

\section*{A.3 Input File}

After the mesh is generated, an input file needs to be specified by the user. The input file specifies a list of variables that are required by \texttt{PHORCE}, specific equation systems, and specific applications designed by the user. A variable can be specified in the following pattern

\begin{verbatim}
variable name: value;
\end{verbatim}
Figure A.2: Visualization of FlowOverCylinder_longwake.msh

Variable names are declared in code. Everything else in the input file that is neither a variable name, nor enclosed by "":" and ";" will be a comment. A sample input file for the RT simulations (section 6.5.1) is provided here.

```
// PHORCE INPUT FILE

// PHORCE required inputs

// specify the location of the .msh file
Mesh_File: ${Mesh_DIR}/unbiasedUniformRT_multiMode_flash200K.msh;

// Provide the full path to ${PHORCE_DIR}
PlotNodes: ${PHORCE_DIR}/Code/plotNodes;

// Case identifier that links to specific application driver specified by the user
// The list of case identifier is defined in main.cpp
Case: idealMHDmofRT;

UserSpecifiedModel: 1;   // 0 for ideal MHD; 1 for glmMHD
```
// Set to 1 to measure the total runtime
MeasureTime: 1;

// Partitioning method, 0 for parmetis, 1 for morton 2 for hilbert
// It is recommended to use parmetis whenever there is complex geometries in the
// domain, and mesh sizes are non-uniform. Hilbert partitioning is generally better
// on locality than Morton.
ParMethod: 2;

// PBC Setting
PBC_pair: 1; // number of boundary pairs that to be set as classic periodic BC
PBC_pair1_a: 1; // the physical tag for one boundary in the first PBC pair
PBC_pair1_b: 2; // the physical tag for the other boundary in the first PBC pair
PBC_pair2_a: ;
PBC_pair2_b: ;

RadPBC: 0; // set to 1 if there is a azimuthal periodic BC pair
RadPBC_pair_a: ; // the physical tag for one boundary in the azimuthal PBC pair
RadPBC_pair_b: ; // the physical tag for the other boundary in the azimuthal PBC pair

// Set to 1 to have dt to be fixed
fix_dt_flag: 0;
fixed_dt: 1.6e-4;

*** Limiter Setting ***
// A high-pass filter, set it to be 1 to turn it on
Filter_switch: 0;
filter_coeff: 0.9;

// An older implementation for the positivity limiter (not recommended)
Zhang_switch: 0;

// A newer implementation for the positivity limiter that is system independent (recommended)
ZhangStrong_switch: 0;

// Slope limiter (not recommended)
Moe_switch: 0; // 0: off, 1: on
Moe_alphaCoeff: 2000.0;
Moe_cutOff: 1.1; // > 1.0

// WENO limiter (turn on as needed)
Zhu_switch: 1;

*** Limiter Setting ***

// Polynomial Order (Spatial)
N: 1;

// CFL number
CFL: 0.7;

// Final time for the simulation
T_final: 1.0e-8;

// Set 1 to print time every time-step
printTime: 1;

// Write Interval (respect to time)
// T_writeDATA and T_writeVTU are the output intervals for the computational data
// and visualization data, respectively. T_write is the unified interval for both,
// i.e. T_writeDATA = min(T_write, T_writeDATA), T_writeVTU = min(T_write, T_writeVTU)
T_write: 1.0e-10;
T_writeDATA: 0.3;
T_writeVTU: 0.3;

// Restart from data set (frame number)
// 0 means turned off
Restart: 0;

// Output Folder Location
OutFolder: Out;

// Choose the VTU (ParaView) output format (ASCII, binary, compressed)
// Choose compressed for best IO performance. Only choose the other
// two for debugging or implementation purposes.
VTUformat: compressed;

// glmMHD parameters
Cr: 99999.0;
mu0: 12.5663706144e-7;
gas_gamma: 1.4;
lowerFluidMass: 2.17438e-26;
upperFluidMass: 2.17438e-26;
lowerFluidNumberDensity: 2.759407279316403e27;
middleTemperature: 3.933900278385380e2; //ev
BoltzmannConstant: 1.3807e-23;
Z1: 3.5;
Z2: 3.5;

hasResistivity: 0;
electricResistivity: 0.0002;

hasGravitation: 1;
gx: 0.0;
gy: 7.605e12; //0.5
gz: 0.0;

// Application specific inputs (created by the user) //
AtwoodNumber: 0.958657755116103;

Perturbation: 3; // 1 for single mode, 2 for 24 modes, 3 for 40 modes

DomainLength: 250.0e-6;
DomainHeight: 50.0e-6;
width_tanh: 0.01; // The ratio of the tanh width respected to the domain height
initialProfile: 1; // Choose different method for density initialization,
// 0 for step function, 1 for tanh

// magnetic field;
bx: 50.0;

Most PHORCE required variables defined here are either self-explanatory or well-explained by the comments. However, it is important to point out that for MOF coupled system, the MOF bound limiter (section 3.6.5) is implicitly applied to the MOF system, and the positivity limiter (ZhangStrong) is implicitly applied to the fluid system. When PHORCE is trying to load a variable but failed to find the variable name in the input file, a warning message will be printed in the console

** [WARNING]: Trying to load [variable name], but it is not found in your input file. You can ignore this if you are not using it

This usually means that the variable name is missing or there is a typo in the input file. If the variable name is missing, and is not a variable introduced by the user in the application code, nor a required input by PHORCE, then it usually represents a feature that is turned off by default. The user can turn this feature on by specifying this variable in the input file.

### A.4 Set Up an Application Code

Besides the input file, an application code also needs to be implemented by the user. Inside this application code, functions for initial and boundary conditions needs to be specified, and a driver needs to be implemented. In this section, the application code `idealMHD2DMOF_RT.cpp`, for the simulations in section 6.5.1, is taken as an example to explain the process.

#### A.4.1 Initial Conditions

Initial conditions need to be specified using the following pattern

```c++
void nameOfTheInitialConditionFunction(const Service *Srv,
```
const Mesh *mesh,
Hyper2D *Hyper,
Hyper2DVar *Var)
{
  // implementations of the initial conditions
}

For example, in idealMHD2DMOF_RT.cpp, the initial conditions are implemented as

void idealMHDmofRtIni(const Service *Srv,
const Mesh *mesh,
Hyper2D *Hyper,
Hyper2DVar *Var)
{
  const REAL Atwood = load_inputNum(Srv->inputFile, "AtwoodNumber");
  const int ptb = load_inputNum(Srv->inputFile, "Perturbation");
  // 1 for single mode, 2 for multimode (19), 3 for multimode (39)
  const REAL Lx = load_inputNum(Srv->inputFile, "DomainLength");
  const REAL Ly = load_inputNum(Srv->inputFile, "DomainHeight");
  const REAL hyper_width = load_inputNum(Srv->inputFile, "width_tanh");
  const int initialProfile = load_inputNum(Srv->inputFile, "initialProfile");
  const REAL mL = load_inputNum(Srv->inputFile, "IonMass1");
  const REAL mH = load_inputNum(Srv->inputFile, "IonMass2");
  const REAL nL = load_inputNum(Srv->inputFile, "lowerFluidNumberDensity");
  const REAL Z1 = load_inputNum(Srv->inputFile, "Z1");
  const REAL Z2 = load_inputNum(Srv->inputFile, "Z2");
  const REAL T0_ev = load_inputNum(Srv->inputFile, "middleTemperature");
  const REAL T0 = T0_ev * 11604.0;
  const REAL kb = load_inputNum(Srv->inputFile, "BoltzmannConstant");
  const int neq = Hyper->neq;
  const int Nelements = mesh->Nelements;
  const int Np = mesh->Np;
  REAL prm[neq];
  REAL eta = 1000.0;
  REAL g = load_inputNum(Srv->inputFile, "gy");
  const REAL rhoL = mL * nL; // lighter fluid
  const REAL rhoH = rhoL * (1.0 + Atwood) / (1.0 - Atwood);
  const REAL nH = rhoH / mH;
  const REAL rhoDiff = rhoH - rhoL;
const REAL bx = load_inputNum(Srv->inputFile, "bx");
const REAL by = 0.0;
const REAL bz = 0.0;

REAL p0 = (Z1+1)*nL*kb*T0;

REAL rand[40] = {0.879013904597178,
  0.988911616079589,
  0.000522375356945,
  0.865438591013025,
  0.612566469483999,
  0.989950205708831,
  0.527680069338442,
  0.479523385210219,
  0.801347605521952,
  0.227842935706042,
  0.498094291196390,
  0.900852488532005,
  0.574661219130188,
  0.845178185054037,
  0.738640291995402,
  0.58957035826476,
  0.246734525985975,
  0.666416217319468,
  0.083482813602623,
  0.625959785171583,
  0.660944557947342,
  0.729751855317221,
  0.890752116325322,
  0.982303222836306,
  0.254790156597005,
  0.224040030824219,
  0.667832727013717,
  0.844392156527205,
  0.344462411301042,
  0.780519652731358,
  0.675332065747000,
  0.006715314318477,
  0.602170487581795,
  0.386771194520985,
  0.915991244131425,
  0.001151057129107,
  0.462449159242329,
  0.424349039815375,
  0.460916366028964,
  0.770159728608609};

const REAL hp_size = 1.0/hyper_width;

for (iint e=0; e<Nelements; e++) {
\begin{verbatim}
const iint v0 = mesh->EToV(e,0);
const iint v1 = mesh->EToV(e,1);
const iint v2 = mesh->EToV(e,2);

REAL rho = 0.0;
REAL p = 0.0;
REAL f = 0.0;

if (initialProfile == 0) { // jump initialization
    const REAL y_aver = (mesh->Vy_global_raw[v0] + mesh->Vy_global_raw[v1] + mesh->Vy_global_raw[v2])/3.0;
    if (y_aver <= 0) {
        rho = rhoL;
    }
    if (y_aver > 0) {
        rho = rhoH;
    }
}

for (int n = 0; n < Np; n++) {
    const iint nid = e*Np+n;
    const iint vid = nid*neq;
    const REAL x = mesh->x_global_raw[nid];
    const REAL y = mesh->y_global_raw[nid];
    const REAL u = 0.0;

    if (initialProfile == 1) { // tanh initialization
        const REAL hp_scale = 0.5*(tanh(hp_size*y*2.0*PI/Ly)+1.0);
        rho = hp_scale*rhoDiff + rhoL;
        p = p0 - 0.5*(rhoL+rhoH)*g*y - 0.5*(rhoH-rhoL)*g*log(cosh(hp_size*y*2.0*PI/Ly))/(hp_size*2.0*PI/Ly);
    } else { // step initialization
        p = p0 - g*rho*y;
    }

    REAL v = 0.0;
    if (ptb==1) { // single mode
        v = 0.0025*(1.0+cos(2.0*PI*x/Lx))*(1.0+cos(4.5*PI*y/Ly))*exp(-eta*(y*y));
    } else if (ptb==2) {
        for (int m=0; m<20; m++) { // 19 modes
            v += 0.01*rand[m]*cos(2.0*PI*(m*x/Lx + rand[m]))*exp(-eta*(y*y/(Ly*Ly)));
        }
    } else if (ptb==3) {
        for (int m=0; m<40; m++) { // 39 modes
            v += 0.01*rand[m]*cos(2.0*PI*(m*x/Lx + rand[m]))*exp(-eta*(y*y/(Ly*Ly)));
        }
    }
}
\end{verbatim}
const REAL w = 0.0;

// initialize mof marker from the vof marker
const REAL fv = (rho - rhoL)/(rhoH - rhoL);
f = rhoH * fv / rho;

prm[IDRHO] = rho; prm[IDF] = f; prm[IDMTX] = u; prm[IDMTRY] = v;
prm[IDENRG] = p; prm[IDBX] = bx; prm[IDBY] = by; prm[IDBZ] = bz;
Hyper->Switch_Prm2Consrv(prm, Var->Q_raw + vid);
}

A.4.2 Boundary Conditions

Boundary conditions need to be specified using the following pattern

```c
void nameOfTheBoundaryConditionFunction(const Service *Srv,
    const Mesh *mesh,
    Hyper2D *Hyper,
    Hyper2DVar *Var)
{
    // implementations of the boundary conditions
}
```

For example, in `idealMHD2DMOF_RT.cpp`, the boundary conditions are implemented as

```c
void idealMHDmofRtBC(const Service *Srv,
    const Mesh *mesh,
    Hyper2D *Hyper,
    Hyper2DVar *Var)
{
    const int neq = Hyper->neq;
    for (int k = 0; k < mesh->Nelements; k++) {
        for (int face = 0; face < mesh->Nfaces; face++) {
            for (int n = 0; n < mesh->Nfp; n++) {
                const int idf = k * mesh->Nfaces * mesh->Nfp + face * mesh->Nfp + n;
                const int iP = mesh->vmapP(idf);
                const int iM = mesh->vmapM(idf);
                if (mesh->BCType(k, face) == BC_Wall) {
                    const REAL nx = mesh->nx(idf);
                    const REAL ny = mesh->ny(idf);
                    Var->QP_raw[iP*neq+IDRHOF] = Var->Q_raw[iM*neq+IDRHOF];
                    Var->QP_raw[iP*neq+IDRHO1MF] = Var->Q_raw[iM*neq+IDRHO1MF];
```
Var->QP_raw[idf*neq+IDMMTX] = (1.0 - 2.0*nx*nx)*Var->Q_raw[idM*neq+IDMMTX] - 2.0*nx*ny*Var->Q_raw[idM*neq+IDMMTY];
Var->QP_raw[idf*neq+IDMMTY] = (1.0 - 2.0*ny*ny)*Var->Q_raw[idM*neq+IDMMTY] - 2.0*nx*ny*Var->Q_raw[idM*neq+IDMMTX];
Var->QP_raw[idf*neq+IDBX] = (1.0 - 2.0*nx*nx)*Var->Q_raw[idM*neq+IDBX] - 2.0*nx*ny*Var->Q_raw[idM*neq+IDBY];
Var->QP_raw[idf*neq+IDBY] = (1.0 - 2.0*ny*ny)*Var->Q_raw[idM*neq+IDBY] - 2.0*nx*ny*Var->Q_raw[idM*neq+IDBX];
Var->QP_raw[idf*neq+IDMMTZ] = Var->Q_raw[idM*neq+IDMMTZ];
Var->QP_raw[idf*neq+IDBZ] = Var->Q_raw[idM*neq+IDBZ];
Var->QP_raw[idf*neq+IDENRG] = Var->Q_raw[idM*neq+IDENRG];
Var->QP_raw[idf*neq+IDPSI] = Var->Q_raw[idM*neq+IDPSI];
Var->Gx_raw[idf*neq+IDRHOF] = 0.0;
Var->Gx_raw[idf*neq+IDRHO1MF] = 0.0;
Var->Gx_raw[idf*neq+IDMMTX] = 0.0;
Var->Gx_raw[idf*neq+IDMMTY] = 0.0;
Var->Gx_raw[idf*neq+IDMMTZ] = 0.0;
Var->Gx_raw[idf*neq+IDENRG] = 0.0;
Var->Gx_raw[idf*neq+IDBX] = 0.0;
Var->Gx_raw[idf*neq+IDBY] = 0.0;
Var->Gx_raw[idf*neq+IDPSI] = 0.0;
Var->Gy_raw[idf*neq+IDRHOF] = 0.0;
Var->Gy_raw[idf*neq+IDRHO1MF] = 0.0;
Var->Gy_raw[idf*neq+IDMMTX] = 0.0;
Var->Gy_raw[idf*neq+IDMMTY] = 0.0;
Var->Gy_raw[idf*neq+IDMMTZ] = 0.0;
Var->Gy_raw[idf*neq+IDENRG] = 0.0;
Var->Gy_raw[idf*neq+IDBX] = 0.0;
Var->Gy_raw[idf*neq+IDBY] = 0.0;
Var->Gy_raw[idf*neq+IDPSI] = 0.0;
}
else{
    memcpy(Var->QP_raw+idf*neq, Var->Q_raw+idP*neq, neq*sizeof(REAL));
}
}
A.4.3 Application Driver

Once the initial and boundary conditions are defined, the application driver can be implemented as the following:

```c
void nameOfTheDriverFunction(Mesh *mesh, char *input_file){
    // create the service object
    Service *Srv = new Service(input_file, "{name of the output VTU ←
    → files}");

    // create the PDE (hyperbolic system) object
    Hyper2D *Hyper = new {PDE class}(input_file, mesh);

    // create the variable object based on the number of equations ←
    → (variables) in the PDE
    Hyper2DVar *Var = new Hyper2D{neq}Var(mesh, Hyper);

    // create the updater object
    Updater *ud = new Updater();

    // create the inspect list for the troubled-cell sensors
    // inspectList[0] = n = number of variables to be inspected by the ←
    → sensor
    // inspectList[1-n] = equation IDs for these n variables
    ...

    // sending those objects and the IC and BC functions to the specific ←
    → updater
    ...

    // free memory
    ...
}
```

For example, in `idealMHD2DMOF_RT.cpp`, the driver is implemented as

```c
void MHDmofRTDriver(Mesh *mesh, char *input_file){
    // here in the output folder, it will has "RT" in the name of .vtu, ←
    → .pvtu, and .pvd files
    // these files can be read by ParaView for postprocessing
    Service *Srv = new Service(input_file, "RT");

    // choose the glmMHD + MOF coupled PDE system
    Hyper2D *MHD = new glmMHD2DMOF(input_file, mesh);

    // glmMHD has 9 variables and MOF has 1 variable, so we have 10 ←
    → variables in the glmMHD+MOF coupled system (Hyper2D10Var)
    Hyper2DVar *Var = new Hyper2D10Var(mesh, MHD);

    Updater *ud = new Updater();
}
```
// choose density as the only variables to be considered when detecting the trouble-cell.
int *inspectList = new int[2];
inspectList[0] = 1; inspectList[1] = IDRHO;
ud->updating_rk45(input_file, Srv, mesh, MHD, Var, inspectList, 1,
idealMHDmofRtIni, idealMHDmofRtBC);

// free memory
delete Srv; Srv=NULL;
delete MHD; MHD=NULL;

Once the application code is implemented, a case identifier should be created in main.cpp, for example

```cpp
if (Case == "idealMHDmofRT") {
    MHDmofRtDriver(mesh, argv[1]);
}
```

Then the application code should be added to the `makefile` and PHORCE should to be re-compiled.

### A.5 Run the Code

Once the mesh and input file are created, the application code is implemented, and PHORCE is compiled, the simulation can be launched as

```bash
$ mpiexec -n ${number of cores} PHORCE.exe RT.in
```

### A.6 Post-processing and Visualization in ParaView

Once the simulation finishes, one can launch ParaView by

```bash
$ paraview ${Output Folder}/VTU/RT.pvd
```

Inside the VTU folder, there are three types of files, `RT.{Frame ID}.p{Partition ID}.vtu`, `RT.{Frame ID}.pvtu`, and `RT.pvd`. `RT.{Frame ID}.p{Partition ID}.vtu` files are the data files with user specified format (ASCII, binary, compressed), where the first number is the output frame ID and the second number is the partition ID. `RT.{Frame ID}.pvtu` file links all the partitioned `.vtu` with the same frame ID. `RT.pvd` links all the frames and associates each frame with a physical time.

More details on how to use ParaView can be found from their web-site https://www.paraview.org. In this section, only PHORCE related details be will presented. PHORCE will only output the
data for conserved variables and label them as \texttt{Field\_XX}, where \texttt{XX} is the index of the conserved variables. For example in the GLM-MHD and MOF coupled system we have:

\begin{verbatim}
#define IDRHO0 0 // Field\_00: \rho f
#define IDRHO1MF 1 // Field\_01: \rho (1-f)
#define IDMMTX 2 // Field\_02: x momentum
#define IDMMTY 3 // Field\_03: y momentum
#define IDMMTZ 4 // Field\_04: z momentum
#define IDENRG 5 // Field\_05: total energy
#define IDBX 6 // Field\_06: bx
#define IDBY 7 // Field\_07: by
#define IDBZ 8 // Field\_08: bz
#define IDPSI 9 // Field\_09: psi
\end{verbatim}

These fields can be selected in \texttt{ParaView} as shown in Figure A.3.

For the MOF coupled system, density $\rho$ is not directly available. $\rho = \texttt{Field\_00} + \texttt{Field\_01}$ can be generated through \texttt{ParaView}'s built-in calculator as shown in Figure A.4. Other quantities, such as MOF maker function, velocities, and pressure can also be generated using the same way.

For simulations using a MOF coupled system, one often needs to measure certain statistics of the interface. For example, in RT simulations, the peak bubble to spike distance $h_{\text{peak}}$ is
useful (for example in Figure 6.16). The peak bubble to spike distance can be calculated as

\[ h_{\text{peak}} = y_{\text{max}} - y_{\text{min}}, \quad (A.1) \]

where \( y_{\text{max}} \) and \( y_{\text{min}} \) are the maximum and minimum \( y \) coordinates of the interface. To get the information of the interface, one needs to calculate the MOF marker function first using the built-in calculator,

\[ f = \frac{\text{Field}_00}{\text{Field}_00 + \text{Field}_01}. \quad (A.2) \]

Then one needs to create a contour of \( f = 0.5 \) as shown in Figure A.5, which represents the interface. The contour contains the information of \( y_{\text{max}} \) and \( y_{\text{min}} \). One way to fetch this information is to first create a variable \( y = \text{coordsY} \) using the calculator. Then select the Descriptive Statistics filter and turn off all variables except \( y \), as shown in Figure A.6. This will create two outputs, Statistical Model and Assessed Data. In Statistical Model, the user can turn off all variables except Maximum and Minimum, as shown in Figure A.7. The user can plot the Maximum and Minimum over time and save the data for further post-processing. Note that this is not the only way to achieve this. ParaView’s built-in python shell is a great tool for automated post-processing. One can also use the trace feature in ParaView to learn different built-in python commands.
Figure A.5: Contour in ParaView
Figure A.6: Descriptive Statistics in ParaView
Figure A.7: Statistical Model in ParaView
Appendix B

PHORCE Developer Guide: Equation System

In this appendix, a developer guide on how to implement a new equation system is provided. In PHORCE, all equation systems are inherited from Hyper2D class. All functions in updater only take Hyper2D * as the equation system input argument. Inside Hyper2D, multiple virtual functions are declared.

```cpp
virtual unsigned nequ() const = 0;
virtual unsigned nwave() const = 0;
virtual void update_temporal_parameters(const REAL dt) {};
virtual void calc_flux(const unsigned d, const REAL* q, const REAL* qaux, REAL* f) {
    Log log;
    log.Error("calc_flux called but returned to Hyper class as it is not specified in the inheriated class");
}
virtual void calc_constCoeff_diffusive_flux(const REAL* q, const REAL* qaux, REAL* f) {
    Log log;
    log.Error("calc_constCoeff_diffusive_flux called but returned to Hyper class as it is not specified in the inheriated class");
}
virtual void calc_diffusive_flux_volume(const REAL* q, const REAL rx, const REAL sx, const REAL ry, const REAL sy, const REAL* qaux,
```
REAL* fd1,
REAL* fd2) {

Log log;
log.Error("calc_diffusive_flux_volume called but returned to Hyper ←
class as it is not specified in the inheriated class");
};

virtual void calc_diffusive_flux_surface(const REAL* q1,
const REAL* q2,
const MatrixXREAL NM1,
const MatrixXREAL NM2,
const REAL EP_rx,
const REAL EP_sx,
const REAL EP_ry,
const REAL EP_sy,
const REAL* qaux,
REAL* fd1,
REAL* fd2) {

Log log;
log.Error("calc_diffusive_flux_surface called but returned to Hyper ←
class as it is not specified in the inheriated class");
};

virtual void calc_source(const REAL* q, const REAL* qaux, REAL* s) ← 
{ hasInternalSource=0;};

virtual void calc_constGravitationSource(const REAL* qaux, const REAL* q, REAL* s) ← 
{ 
Log log;
log.Error("calc_constGravitationSource called but returned to Hyper ←
class as it is not specified in the inheriated class");
};

virtual void Switch_Consrv2Prm(const REAL* q, REAL* v){

Log log;
log.Error("Switch_Consrv2Prm called but returned to Hyper ←
is not specified in the inheriated class");
};

virtual REAL Switch_Consrv2P(const REAL* q){

Log log;
log.Error("Switch_Consrv2P called but returned to Hyper ←
is not specified in the inheriated class");
exit(1);
};

// virtual void Switch_Prm2Consrv(REAL* v, REAL* q) {};
virtual void Switch_Prm2Consrv(const REAL* v, REAL* q){

Log log;
log.Error("Switch_Prm2Consrv called but returned to Hyper ←
is not specified in the inheriated class");
};
virtual void Switch_Prm2Char(const REAL nx, const REAL ny, const REAL* v, REAL* ch) {
    Log log;
    log.Error("Switch_Prm2Char called but returned to Hyper class as it is not specified in the inherited class");
}

virtual void Switch_Char2Prm(const REAL nx, const REAL ny, const REAL* ch, REAL* v) {
    Log log;
    log.Error("Switch_Char2Prm called but returned to Hyper class as it is not specified in the inherited class");
}

virtual void Switch_Consrv2Char(const REAL nx, const REAL ny, const REAL* q, REAL* ch) {
    Log log;
    log.Error("Switch_Consrv2Char called but returned to Hyper class as it is not specified in the inherited class");
}

virtual void Switch_Char2Consrv(const REAL nx, const REAL ny, const REAL* ch, REAL* q) {
    Log log;
    log.Error("Switch_Char2Consrv called but returned to Hyper class as it is not specified in the inherited class");
}

virtual void calc_ConsrvLeftEigenVector(const VectorXREAL q, MatrixXREAL &lev) {
    Log log;
    log.Error("calc_ConsrvLeftEigenVector called but returned to Hyper class as it is not specified in the inherited class");
}

virtual void calc_ConsrvRightEigenVector(const VectorXREAL q, MatrixXREAL &rev) {
    Log log;
    log.Error("calc_ConsrvRightEigenVector called but returned to Hyper class as it is not specified in the inherited class");
}

virtual void Switch_Consrv2Char_Average(const REAL nx, const REAL ny, const REAL* qbar, const REAL* q, REAL* ch) {
    Log log;
    log.Error("Switch_Consrv2Char_Average called but returned to Hyper class as it is not specified in the inherited class");
}

virtual void Switch_Char2Consrv_Average(const REAL nx, const REAL ny, const REAL* qbar, const REAL* ch, REAL* q) {
    Log log;
    log.Error("Switch_Char2Consrv_Average called but returned to Hyper class as it is not specified in the inherited class");
}

void Switch_Prm2Char_Average(const REAL nx, const REAL ny, const REAL* qbar, const REAL* v, REAL* ch);
void Switch_Char2Prm_Average(const REAL nx, const REAL ny,
     const REAL* qbar, const REAL* ch, REAL* v);

virtual REAL calc_eigen_max(REAL* v){
    Log log;
    log.Error("calc_eigen_max called but returned to Hyper class as it is \not specified in the inherited class");
    exit(1);};

virtual REAL calc_eigen_max_normal(REAL* v, REAL nx, REAL ny){
    Log log;
    log.Error("calc_eigen_max_normal called but returned to Hyper \not as it is not specified in the inherited class");
    exit(1);};

virtual REAL calc_eigen_diffusive_max(REAL* v){
    Log log;
    log.Error("calc_eigen_diffusive_max called but returned to Hyper \not as it is not specified in the inherited class");
    exit(1);};

virtual REAL calc_eigen_diffusive_max_normal(REAL* v, REAL nx, REAL ny){
    Log log;
    log.Error("calc_eigen_diffusive_max_normal called but returned to \not Hyper class as it is not specified in the inherited class");
    exit(1);};

virtual void prmEigenSystem(const REAL nx,
     const REAL ny,
     const REAL *prm,
     REAL *ev,
     REAL *lev,
     REAL *rev){
    Log log;
    log.Error("prmEigenSystem called but returned to Hyper class as it is \not not specified in the inherited class");
    exit(1);};

virtual ~Hyper2D();

The detail description for each virtual function is available in Hyper2D.h. To implement a new equation system, virtual functions need to be defined. Here an example header file for Euler equation with three velocity components (Euler2D5eq) is given

//======================================================================================//
// Euler2D5eq.h
// Declare class Euler2D5eq, which inherited from Hyper2D
//======================================================================================//

#ifndef EULER2D5EQ_H
#define EULER2D5EQ_H

#include "Hyper2D.h"

class Euler2D5eq : public Hyper2D
{...

protected:
    //... properties

public:
    Euler2D5eq();
    ~Euler2D5eq();

    // virtual functions

    virtual void Switch_Char2Prm_Average(const REAL nx, const REAL ny, const REAL* qbar, const REAL* ch, REAL* v);
    virtual REAL calc_eigen_max(REAL* v);
    virtual REAL calc_eigen_max_normal(REAL* v, REAL nx, REAL ny);
    virtual REAL calc_eigen_diffusive_max(REAL* v);
    virtual REAL calc_eigen_diffusive_max_normal(REAL* v, REAL nx, REAL ny);
    virtual void prmEigenSystem(const REAL nx, const REAL ny, const REAL *prm, REAL *ev, REAL *lev, REAL *rev);
    virtual ~Euler2D5eq();

private:
    //... private members and methods
};

#endif // EULER2D5EQ_H
#include <Equations/Hyper2D.h>
#include <HandyTools/handytools.h>

class Euler2D5eq: public Hyper2D
{
  public:

  Euler2D5eq(Mesh *mesh);
  Euler2D5eq(Mesh *mesh, const char * input_file);

  ~Euler2D5eq();

  /* Virtual Functions in Hyper2D.h */
  unsigned nequ() const { return 5; }
  unsigned nwave() const { return 3; }

  void calc_flux(const unsigned d, const REAL * q, const REAL * qaux, REAL * f);

  void Switch_Consrv2Prm(const REAL * q, REAL * v);

  REAL Switch_Consrv2P(const REAL * q);

  void Switch_Prm2Consrv(const REAL * v, REAL * q);

  void Switch_Prm2Char(const REAL nx, const REAL ny, const REAL * v, REAL * ch);

  void Switch_Char2Prm(const REAL nx, const REAL ny, const REAL * v, REAL* ch);

  void Switch_Consrv2Char(const REAL nx, const REAL ny, const REAL * q, REAL * ch);

  void Switch_Char2Consrv(const REAL nx, const REAL ny, const REAL * ch, REAL * q);

  void calc_ConsrvLeftEigenVector(const VectorXREAL q, MatrixXREAL & lev);

  void calc_ConsrvRightEigenVector(const VectorXREAL q, MatrixXREAL & rev);

  void Switch_Consrv2Char_Average(const REAL nx, const REAL ny, const REAL * qbar, const REAL * q, REAL * ch);

  void Switch_Char2Consrv_Average(const REAL nx, const REAL ny, const REAL * qbar, const REAL * ch, REAL * q);
Detailed implementation for each of these functions can be found in Euler2D5eq.cpp. For the implementation of other equation systems, the developer is expected to follow this example. Once Euler2D5eq.h and Euler2D5eq.cpp are implemented, then Euler2D5eq.cpp needs to be added to the makefile.
Appendix C

PHORCE Developer Guide: Diffusion Terms

For equation systems with diffusion terms, diffusive volume and surface fluxes need to be implemented in the equation class. The aRDG algorithm is already described in Chapter 4. In this appendix, only implementation details will be provided. Consider the convection-diffusion equation 2.2 again,

\[
\frac{\partial Q}{\partial t} + \frac{\partial F_i}{\partial x_i} + \frac{\partial G_i}{\partial x_i} = S, \quad i = 1, \ldots, N_d, \tag{C.1}
\]

where \(G_i\) are the diffusive fluxes. After the aRDG treatment, the diffusive part becomes,

\[
\left(M^{-1}S^T \cdot G_i \right) - \sum_{f=1}^{3} \text{LIFT}_f^k \left( \hat{n}_f \cdot \tilde{G}_{if} \right), \tag{C.2}
\]

Note that the element index \(k\) is dropped for the sake of simplicity. The first term is the diffusive volume integration and the second term is the diffusive surface integration. The calculations of \(G_i\) and \(G_{if}\) need to be implemented by the developer in functions `calc_diffusive_flux_volume` and `calc_diffusive_flux_surface`, respectively. The implementation of Navier-Stokes equations is taken as an example in this appendix. In Navier-Stokes equations (section 2.3), the diffusive flux is

\[
G_i = \begin{pmatrix}
0 \\
-\Pi_{ij} \\
-u_j \Pi_{ij} + q_i
\end{pmatrix} \tag{C.3}
\]

C.1 Implementation of `calc_diffusive_flux_volume`

calc_diffusive_flux_volume is declared as a virtual function in `Hyper2D.h`,

153
The implementation of diffusive volume flux is the same as the implementation of advective volume flux, which is very straightforward. `calc_diffusive_flux_volume` in `NavierStokes2D` is implemented as

```cpp
virtual void calc_diffusive_flux_volume(
    const REAL* q, // conserved variables [input]
    const REAL rx, // geometric factor [input]
    const REAL sx, // geometric factor [input]
    const REAL ry, // geometric factor [input]
    const REAL sy, // geometric factor [input]
    const REAL* qaux, // reserved as an additional input handle [input]
    REAL* fd1, // diffusive flux on x direction [output]
    REAL* fd2 // diffusive flux on y direction [output]
);
```

```cpp
void NavierStokes2D :: calc_diffusive_flux_volume(const REAL* q,
    const REAL rx,
    const REAL sx,
    const REAL ry,
    const REAL sy,
    const REAL* qaux,
    REAL* fd1,
    REAL* fd2)
{
    // Gx Gy : Gx = Dx*grad(Cx), Gy = Dy*grad(Cy); where grad can be any differential form
    // create EIGEN wrapper for diffusive fluxes (output)
    Map<MatrixXREAL> fd1_mat(fd1, neq, Np);
    Map<MatrixXREAL> fd2_mat(fd2, neq, Np);

    unsigned mu=1, mv=2, mw=3;

    // copy q into a EIGEN matrix
    MatrixXREAL qmat(neq, Np);
    memcpy(qmat.data(), q, Np*neq*sizeof(REAL));

    // calculate the components needed and store them in EIGEN vector
    const RowVectorXd rho = qmat.row(0);
    const RowVectorXd invRho = rho.cwiseInverse();
    const RowVectorXd u = qmat.row(mu).cwiseProduct(invRho);
    const RowVectorXd v = qmat.row(mv).cwiseProduct(invRho);
    const RowVectorXd w = qmat.row(mw).cwiseProduct(invRho);
    const RowVectorXd U2 = u.cwiseAbs2() + v.cwiseAbs2() + w.cwiseAbs2();
    const RowVectorXd E = qmat.row(4);
    const RowVectorXd P = gas_gamma1*(E-0.5*rho.cwiseProduct(U2));
    const RowVectorXd T = P.cwiseProduct(invRho)/gas_R;

    // calculate viscosity
    const RowVectorXd gas_mu = calc_viscosity_Sutherland(Np, T, gas_T0, gas_mu0, gas_S);
```
C.2 Implementation of calc_diffusive_flux_surface

calc_diffusive_flux_surface is declared as a virtual function in Hyper2D.h,

```cpp
virtual void calc_diffusive_flux_surface(
const REAL* q1, // conserved variables on element 1 [input]
const REAL* q2, // conserved variables on element 2 [input]
const MatrixXREAL NM1, // projection matrix for element 1
const MatrixXREAL NM2, // projection matrix for element 2
const REAL EP_rx, // geometric factor for enclosed parallelogram [input]
const REAL EP_sx, // geometric factor for enclosed parallelogram [input]
);```

// calculate thermal conductivity
const REAL mu2kappa = gas_R*gas_gamma/(gas_gamma1*gas_Pr);
const RowVectorXREAL kappa = gas_mu*mu2kappa;

// calculate transposes of physical differentiation matrices
const MatrixXREAL DxTrans = (rx*Dr+sx*Ds).transpose();
const MatrixXREAL DyTrans = (ry*Dr+sy*Ds).transpose();

// calculate the derivatives needed for fd1 and fd2
const RowVectorXREAL dudx = u*DxTrans;
const RowVectorXREAL dvdx = v*DxTrans;
const RowVectorXREAL dTdx = T*DxTrans;

// fill fd1 and fd2
fd1_mat.row(0) = RowVectorXREAL::Zero(Np);
fd1_mat.row(mu) = gas_mu.cwiseProduct(-fourThirds*dudx + twoThirds*dvdy);
fd1_mat.row(mv) = -gas_mu.cwiseProduct(dvdx+dudy);
fd1_mat.row(mw) = RowVectorXREAL::Zero(Np);
fd1_mat.row(4) = -(gas_mu.cwiseProduct(u.cwiseProduct(fourThirds*dudx-fourThirds*dvdy)+v.cwiseProduct(dvdx+dudy))
                     -kappa.cwiseProduct(dTdx);

fd2_mat.row(0) = RowVectorXREAL::Zero(Np);
fd2_mat.row(mu) = -gas_mu.cwiseProduct(dvdx+dudy);
fd2_mat.row(mv) = gas_mu.cwiseProduct(twoThirds*dudx-fourThirds*dvdy);
fd2_mat.row(mw) = RowVectorXREAL::Zero(Np);
fd2_mat.row(4) = -(gas_mu.cwiseProduct(u.cwiseProduct(dvdx+dudy)
                     +v.cwiseProduct(-twoThirds*dudx+fourThirds*dvdy))
                     -kappa.cwiseProduct(dTdy);
```
const REAL EP_ry, // geometric factor for enclosed parallelogram [input]
const REAL EP_sy, // geometric factor for enclosed parallelogram [input]
const REAL* qaux, // reserved as an additional input handle [input]
REAL* fd1, // diffusive flux on x direction [output]
REAL* fd2 // diffusive flux on y direction [output]
);

Note that NM1 and NM2 are the pre-calculated matrix products, $\mathcal{V}^{-1}\mathcal{V}_p\mathcal{V}^{-1}$ and $\mathcal{V}^{-1}\mathcal{V}_p\mathcal{V}^{-1}$, described in Algorithm 4. This matrix projects the nodal solution of a triangle element to the modal solution of the enclosed half parallelogram, which is an enclosed triangle. The enclosed triangle is rotated in the logical domain so that its hypotenuse matches the interface in the physical space. These matrices are stored in mesh class and named as Nodal2RotEnPaModal. Implementation of calc_diffusive_flux_surface can be summarized by the pseudo-code in Algorithm 5.

---

**Algorithm 5:** Implementation of calc_diffusive_flux_surface

// Consider $G$ has the form of $D\nabla C$

// Step 1: calculate $D$ in both elements

$D_1 = \text{calc\_diffusion\_coeff}(Q_1);$

$D_2 = \text{calc\_diffusion\_coeff}(Q_2);$

// Step 2: calculate the nodal projection of $D$ in the enclosed triangles (rotated + truncated) from the original nodal solutions

$Dp_1 = \text{TriangleNodal2EnclosedTriangleNodal}(Q_1, NM1, \mathcal{V});$

$Dp_2 = \text{TriangleNodal2EnclosedTriangleNodal}(Q_2, NM2, \mathcal{V});$

// Step 3: calculate the reconstructed nodal solution of $C$ in the enclosed parallelogram

$Cp = \text{aRDGEnclosedTriangleNodal2EnclosedParallelogramNodal}(Q_1, Q_2);$

// Step 4: calculate gradient of $C$ using $Cp$

$\nabla Cp = \text{Grad}(Cp);$

// Step 5: project $\nabla Cp$ to the LGL nodes in those two enclosed triangles using two precalculated matrices (declared and defined in the parent class Hyper2D)

$(\nabla Cp_1, \nabla Cp_2) = \text{EnclosedParallelogramNodal2EnclosedTriangleNodal}(\nabla Cp);$

// Step 6: calculate $G$ in two enclosed triangles

$Gp_1 = Dp_1\nabla Cp_1;$

$Gp_2 = Dp_2\nabla Cp_2;$

---
// Step 7: convert $G_{p1}$ and $G_{p2}$ to modal solution
$\hat{G}_{p1} = \mathcal{V}^{-1}G_{p1}$;
$\hat{G}_{p2} = \mathcal{V}^{-1}G_{p2}$;

// Step 8: calculate the reconstructed nodal solution of $G$ from $\hat{G}_{p1}$ and $\hat{G}_{p2}$
$G_{p} = \text{aRDGEnclosedTriangleModal2EnclosedParallelogramNodal}(\hat{G}_{p1}, \hat{G}_{p2})$;

// Step 9: copy the solution. Note: only solution on the diagonal (interface) nodes are needed.
$G_{f} = \text{fetchDiagonalSolution}(G_{p})$;

Following this, calc_diffusive_flux_surface in NavierStokes2D is implemented as

```c
void NavierStokes2D :: calc_diffusive_flux_surface ( const REAL * q1,
                                    const REAL * q2,
                                    const MatrixXREAL NM1,
                                    const MatrixXREAL NM2,
                                    const REAL EP_rx,
                                    const REAL EP_sx,
                                    const REAL EP_ry,
                                    const REAL EP_sy,
                                    const REAL * qaux,
                                    REAL * fd1,
                                    REAL * fd2 )
{
  unsigned mu =1 , mv =2 , mw =3;

  // create EIGEN wrapper for diffusive fluxes (output)
  Map<MatrixXREAL> fd1_mat(fd1, neq, Nfp);
  Map<MatrixXREAL> fd2_mat(fd2, neq, Nfp);

  // transposes of the physical differentiation matrices of the enclosed parallelogram

  // transposes of the Nodal2RotEnPaModal matrices (see definition in mesh.h)
  const MatrixXREAL NM1Trans = NM1.transpose();
  const MatrixXREAL NM2Trans = NM2.transpose();

  // copy original solutions to EIGEN matrices.
  MatrixXREAL q1_mat(neq, Np), q2_mat(neq, Np);
  memcpy(q1_mat.data(), q1, Np*neq*sizeof(REAL));
  memcpy(q2_mat.data(), q2, Np*neq*sizeof(REAL));

  // calculate each state variables on the original nodes
  // (need velocities and temperature to be reconstructed)
```

const RowVectorXREAL rho1 = q1_mat.row(0);
const RowVectorXREAL invRho1 = rho1.cwiseInverse();
const RowVectorXREAL u1 = q1_mat.row(mu).cwiseProduct(invRho1);
const RowVectorXREAL v1 = q1_mat.row(mv).cwiseProduct(invRho1);
const RowVectorXREAL w1 = q1_mat.row(mw).cwiseProduct(invRho1);
const RowVectorXREAL sqU1 = u1.cwiseAbs2() + v1.cwiseAbs2() + w1.cwiseAbs2();
const RowVectorXREAL E1 = q1_mat.row(4);
const RowVectorXREAL P1 = gas_gamma1 * (E1 - 0.5 * rho1.cwiseProduct(sqU1));
const RowVectorXREAL T1 = P1.cwiseProduct(invRho1) / gas_R;

const RowVectorXREAL rho2 = q2_mat.row(0);
const RowVectorXREAL invRho2 = rho2.cwiseInverse();
const RowVectorXREAL u2 = q2_mat.row(mu).cwiseProduct(invRho2);
const RowVectorXREAL v2 = q2_mat.row(mv).cwiseProduct(invRho2);
const RowVectorXREAL w2 = q2_mat.row(mw).cwiseProduct(invRho2);
const RowVectorXREAL sqU2 = u2.cwiseAbs2() + v2.cwiseAbs2() + w2.cwiseAbs2();
const RowVectorXREAL E2 = q2_mat.row(4);
const RowVectorXREAL P2 = gas_gamma1 * (E2 - 0.5 * rho2.cwiseProduct(sqU2));
const RowVectorXREAL T2 = P2.cwiseProduct(invRho2) / gas_R;

/* Step 1 */
/* calculate viscosity */
const RowVectorXREAL gas_mu1 = calc_viscosity_Sutherland(Np, T1, gas_T0, gas_mu0, gas_S);
const RowVectorXREAL gas_mu2 = calc_viscosity_Sutherland(Np, T2, gas_T0, gas_mu0, gas_S);
/* calculate thermal conductivity */
const REAL mu2kappa = gas_R * gas_gamma / (gas_gamma1 * gas_Pr);
const RowVectorXREAL kappa1 = gas_mu1 * mu2kappa;
const RowVectorXREAL kappa2 = gas_mu2 * mu2kappa;

/* Step 2 */
/* calculate the nodal projection of the enclosed triangles (rotated ↔ + truncated) */
/* from the original nodal solutions (velocities, viscosity, thermal ↔ conductivity) */
const RowVectorXREAL u_p1 = triNodal2etriNodal2D(u1, NM1Trans, VTrans);
const RowVectorXREAL v_p1 = triNodal2etriNodal2D(v1, NM1Trans, VTrans);
const RowVectorXREAL gas_mu_p1 = triNodal2etriNodal2D(gas_mu1, NM1Trans, VTrans);
const RowVectorXREAL kappa_p1 = triNodal2etriNodal2D(kappa1, NM1Trans, VTrans);
const RowVectorXREAL u_p2 = triNodal2etriNodal2D(u2, NM2Trans, VTrans);
const RowVectorXREAL v_p2 = triNodal2etriNodal2D(v2, NM2Trans, VTrans);
const RowVectorXREAL gas_mu_p2 = triNodal2etriNodal2D(gas_mu2, NM2Trans, VTrans);
const RowVectorXREAL kappa_p2 = triNodal2etriNodal2D(kappa2, NM2Trans, VTrans);

/* Step 3 */
// calculate the reconstructed nodal solution (of velocities and temperature) in the enclosed parallelogram
const RowVectorXREAL up_rdg = triNodal2epNodalRDG2D(N, Np, Nqp, 1, u1, u2, NM1Trans, NM2Trans, VqTrans);
const RowVectorXREAL vp_rdg = triNodal2epNodalRDG2D(N, Np, Nqp, 1, v1, v2, NM1Trans, NM2Trans, VqTrans);
const RowVectorXREAL Tp_rdg = triNodal2epNodalRDG2D(N, Np, Nqp, 1, T1, T2, NM1Trans, NM2Trans, VqTrans);

/* Step 4 */
// calculate the nodal gradients of the reconstructed solution
const RowVectorXREAL dupdx_rdg = up_rdg * (DxqTrans);
const RowVectorXREAL dupdy_rdg = up_rdg * (DyqTrans);
const RowVectorXREAL dvpdx_rdg = vp_rdg * (DxqTrans);
const RowVectorXREAL dvpdy_rdg = vp_rdg * (DyqTrans);
const RowVectorXREAL dTpdx_rdg = Tp_rdg * (DxqTrans);
const RowVectorXREAL dTpdy_rdg = Tp_rdg * (DyqTrans);

/* Step 5 */
// project the nodal gradients of the reconstructed solution on the two sets of triangle LGL nodes.
const RowVectorXREAL dup1dx_rdg = dupdx_rdg * Vpt1invVqTrans;
const RowVectorXREAL dup1dy_rdg = dupdy_rdg * Vpt1invVqTrans;
const RowVectorXREAL dvp1dx_rdg = dvpdx_rdg * Vpt1invVqTrans;
const RowVectorXREAL dvp1dy_rdg = dvpdy_rdg * Vpt1invVqTrans;
const RowVectorXREAL dTp1dx_rdg = dTpdx_rdg * Vpt1invVqTrans;
const RowVectorXREAL dTp1dy_rdg = dTpdy_rdg * Vpt1invVqTrans;

const RowVectorXREAL dup2dx_rdg = dupdx_rdg * Vpt2invVqTrans;
const RowVectorXREAL dup2dy_rdg = dupdy_rdg * Vpt2invVqTrans;
const RowVectorXREAL dvp2dx_rdg = dvpdx_rdg * Vpt2invVqTrans;
const RowVectorXREAL dvp2dy_rdg = dvpdy_rdg * Vpt2invVqTrans;
const RowVectorXREAL dTp2dx_rdg = dTpdx_rdg * Vpt2invVqTrans;
const RowVectorXREAL dTp2dy_rdg = dTpdy_rdg * Vpt2invVqTrans;

/* Step 6 */
// calculate the products
RowMajorMatrixXREAL fd1_p1(neq, Np), fd1_p2(neq, Np), fd2_p1(neq, Np), fd2_p2(neq, Np);
fd1_p1.row(0) = RowVectorXREAL::Zero(Np);
fd1_p1.row(mu) = gas_mu_p1.cwiseProduct(-fourThirds*dup1dx_rdg + twoThirds*dvp1dy_rdg);
fd1_p1.row(mv) = -gas_mu_p1.cwiseProduct(dvp1dx_rdg + dup1dy_rdg);
fd1_p1.row(mw) = RowVectorXREAL::Zero(Np);
fd1_p1.row(4) = -gas_mu_p1.cwiseProduct(u_p1.cwiseProduct(fourThirds*dup1dx_rdg - twoThirds*dvp1dy_rdg)

fd1_p2.row(0) = RowVectorXREAL::Zero(Np);
fd1_p2.row(mu) = gas_mu_p1.cwiseProduct(-fourThirds*dup2dx_rdg + twoThirds*dvp2dy_rdg);
fd1_p2.row(mv) = -gas_mu_p1.cwiseProduct(dvp2dx_rdg + dup2dy_rdg);
fd1_p2.row(mw) = RowVectorXREAL::Zero(Np);
fd1_p2.row(4) = -gas_mu_p1.cwiseProduct(u_p1.cwiseProduct(fourThirds*dup2dx_rdg - twoThirds*dvp2dy_rdg)

fd2_p1.row(0) = RowVectorXREAL::Zero(Np);
fd2_p1.row(mu) = gas_mu_p1.cwiseProduct(-fourThirds*dup1dx_rdg + twoThirds*dvp1dy_rdg);
fd2_p1.row(mv) = -gas_mu_p1.cwiseProduct(dvp1dx_rdg + dup1dy_rdg);
fd2_p1.row(mw) = RowVectorXREAL::Zero(Np);
fd2_p1.row(4) = -gas_mu_p1.cwiseProduct(u_p1.cwiseProduct(fourThirds*dup1dx_rdg - twoThirds*dvp1dy_rdg)

fd2_p2.row(0) = RowVectorXREAL::Zero(Np);
fd2_p2.row(mu) = gas_mu_p1.cwiseProduct(-fourThirds*dup2dx_rdg + twoThirds*dvp2dy_rdg);
fd2_p2.row(mv) = -gas_mu_p1.cwiseProduct(dvp2dx_rdg + dup2dy_rdg);
fd2_p2.row(mw) = RowVectorXREAL::Zero(Np);
fd2_p2.row(4) = -gas_mu_p1.cwiseProduct(u_p1.cwiseProduct(fourThirds*dup2dx_rdg - twoThirds*dvp2dy_rdg)
\[ \rightarrow +v_{p1}.cwiseProduct(dvp1dx_{rdg}+dup1dy_{rdg}) - \rightarrow \kappa_{p1}.cwiseProduct(dTp1dx_{rdg}) \]

\[ fd2_{p1}.row(0) = RowVectorXREAL::Zero(Np); \]
\[ fd2_{p1}.row(mu) = -gas_{mu,p1}.cwiseProduct(dvp1dx_{rdg}+dup1dy_{rdg}); \]
\[ fd2_{p1}.row(mv) = gas_{mu,p1}.cwiseProduct(twoThirds*dvp1dx_{rdg} \rightarrow \rightarrow fourThirds*dp2dy_{rdg}); \]
\[ fd2_{p1}.row(mw) = RowVectorXREAL::Zero(Np); \]
\[ fd2_{p1}.row(4) = \leftarrow \leftarrow -gas_{mu,p1}.cwiseProduct(u_{p1}.cwiseProduct(dvp1dx_{rdg}+dup1dy_{rdg}) \rightarrow \rightarrow -\kappa_{p1}.cwiseProduct(twoThirds*dp2dy_{rdg}); \]

\[ fd1_{p2}.row(0) = RowVectorXREAL::Zero(Np); \]
\[ fd1_{p2}.row(mu) = gas_{mu,p2}.cwiseProduct(-fourThirds*dup2dx_{rdg} + \leftarrow \rightarrow twoThirds*dp2dy_{rdg}); \]
\[ fd1_{p2}.row(mv) = -gas_{mu,p2}.cwiseProduct(dvp2dx_{rdg}+dp2dy_{rdg}); \]
\[ fd1_{p2}.row(mw) = RowVectorXREAL::Zero(Np); \]
\[ fd1_{p2}.row(4) = \leftarrow \leftarrow -gas_{mu,p2}.cwiseProduct(u_{p2}.cwiseProduct(fourThirds*dup2dx_{rdg} \rightarrow \rightarrow -twoThirds*dp2dy_{rdg}) \rightarrow \rightarrow +v_{p2}.cwiseProduct(dvp2dx_{rdg}+dup2dy_{rdg}) \rightarrow \rightarrow \kappa_{p2}.cwiseProduct(dp2dx_{rdg}); \]

\[ fd2_{p2}.row(0) = RowVectorXREAL::Zero(Np); \]
\[ fd2_{p2}.row(mu) = -gas_{mu,p2}.cwiseProduct(dvp2dx_{rdg}+dp2dy_{rdg}); \]
\[ fd2_{p2}.row(mv) = gas_{mu,p2}.cwiseProduct(twoThirds*dp2dy_{rdg} \rightarrow \rightarrow fourThirds*dp2dy_{rdg}); \]
\[ fd2_{p2}.row(mw) = RowVectorXREAL::Zero(Np); \]
\[ fd2_{p2}.row(4) = \leftarrow \leftarrow +v_{p2}.cwiseProduct(u_{p2}.cwiseProduct(dp2dx_{rdg}+dp2dy_{rdg}) \rightarrow \rightarrow \kappa_{p2}.cwiseProduct(dp2dy_{rdg}); \]

/* Step 7 */
// then convert into modal solution.
const RowMajorMatrixXREAL fd1_{p1}_hat = fd1_{p1}*invVTrans;
const RowMajorMatrixXREAL fd1_{p2}_hat = fd1_{p2}*invVTrans;
const RowMajorMatrixXREAL fd2_{p1}_hat = fd2_{p1}*invVTrans;
const RowMajorMatrixXREAL fd2_{p2}_hat = fd2_{p2}*invVTrans;

/* Step 8 */
// calculate the reconstructed nodal solution of the products in the enclosed parallelogram.
MatrixXREAL fpd1_mat(neq, Nqp), fpd2_mat(neq, Nqp);
fpd1_mat.row(0) = RowVectorXREAL::Zero(Nqp);
fpd2_mat.row(0) = RowVectorXREAL::Zero(Nqp);
// skipping mass equation, as no diffusion term there.
fpd1_mat.block(1,0,neq-1,Nqp) =
etriModal2epNodalRDG2D(N, Np, Nqp, neq-1,
    fd1_p1_hat.block(1,0,neq-1,Np),
    fd1_p2_hat.block(1,0,neq-1,Np),
    VqTrans);
fpd2_mat.block(1,0,neq-1,Nqp) =
etriModal2epNodalRDG2D(N, Np, Nqp, neq-1,
    fd2_p1_hat.block(1,0,neq-1,Np),
    fd2_p2_hat.block(1,0,neq-1,Np),
    VqTrans);

/* Step 9 */
// copy the reconstructed solution on the surface nodes to fd1 fd2;
for (int n=0; n<Nfp; n++) {
    for (int eq=0; eq<neq; eq++) { //eq=0;
        const int id = n*neq+eq;
        const int fvid = QDMask(n); // local interface node

        fd1[id]= fpd1_mat(eq, fvid);
        fd2[id]= fpd2_mat(eq, fvid);
    }
}