

University of Cincinnati

Date: 9/27/2013

I, Marshall C Galbraith , hereby submit this original work as part of the requirements for
the degree of Doctor of Philosophy in Aerospace Engineering.

It is entitled:

A Discontinuous Galerkin Chimera Overset Solver

Student's name: Marshall C Galbraith

This work and its defense approved by:

Committee chair: Paul Orkwis, Ph.D.

Committee member: John A. Benek, Ph.D.

Committee member: Shaaban Abdallah, Ph.D.

Committee member: Mark Turner, Sc.D.



6837

A Discontinuous Galerkin Chimera Overset Solver

A dissertation submitted to the

Graduate School

of the University of Cincinnati

in partial fulfillment of the

requirements for the degree of

Doctor of Philosophy

in the Department of Aerospace Engineering and Engineering Mechanics

of the College of Engineering

by

Marshall Christopher Galbraith

B.S., University of Cincinnati, 2006

M.S., University of Cincinnati, 2009

Date 9/27/2013

Committee Chair: Paul D. Orkwis, Ph.D.

Abstract

This work summarizes the development of an accurate, efficient, and flexible Computational Fluid Dynamics computer code that is an improvement relative to the state of the art. The improved accuracy and efficiency is obtained by using a high-order discontinuous Galerkin (DG) discretization scheme. In order to maximize the computational efficiency, quadrature-free integration and numerical integration optimized as matrix-vector multiplications is employed and implemented through a pre-processor (PyDG). Using the PyDG pre-processor, a C++ polynomial library has been developed that uses overloaded operators to design an efficient Domain Specific Language (DSL) that allows expressions involving polynomials to be written as if they are scalars. The DSL, which makes the syntax of computer code legible and intuitive, promotes maintainability of the software and simplifies the development of additional capabilities.

The flexibility of the code is achieved by combining the DG scheme with the Chimera overset method. The Chimera overset method produces solutions on a set of overlapping grids that communicate through an exchange of data on grid boundaries (known as artificial boundaries). Finite volume and finite difference discretizations use fringe points, which are layers of points on the artificial boundaries, to maintain the interior stencil on artificial boundaries. The fringe points receive solution values interpolated from overset grids. Proper interpolation requires fringe points to be contained in overset grids. Insufficient overlap must be corrected by modifying the grid system. The Chimera scheme can also exclude regions of grids that lie outside the computational domain; a process commonly known as hole cutting.

The Chimera overset method has traditionally enabled the use of high-order finite difference and finite volume approaches such as WENO and compact differencing schemes, which require structured meshes, for modeling fluid flow associated with complex geometries. The large stencil associated with these high-order schemes can significantly complicate the inter-grid communication and hole cutting processes. Unlike these high-order schemes, the DG method always retains a small stencil regardless of the order of approximation.

The small stencil of the DG method simplifies the inter-grid communication scheme as well as hole cutting procedures. The DG-Chimera scheme does not require a separate interpolation method because the DG scheme represents the solution as cell local polynomials. Hence, the DG-Chimera method does not require fringe points to maintain the interior stencil across inter-grid boundaries. Thus, inter-grid communication can be established as long as the receiving boundary is enclosed by or abuts the donor mesh. This makes the inter-grid communication procedure applicable to both Chimera and zonal meshes. The small stencil implies hole cutting can be performed without regard to maintaining a minimum stencil and thereby greatly simplifies hole cutting. Hence, the DG-Chimera scheme has the potential to greatly simplify the overset grid generation process. Furthermore, the DG-Chimera scheme is capable of using curved cells to represent geometric features. The curved cells resolve issues associated with linear Chimera viscous meshes used for finite volume and finite difference schemes. Finally, the convergence rate of the Chimera schemes is dramatically increased by linearization of the inter-grid communication.

Acknowledgments

First and foremost, I would like to thank my adviser Dr. Paul Orkwis for giving me the freedom and support to pursue my ambitions. His insights proved to be most valuable during trying times of this dissertation, and I could always count on him when needed. I am also deeply grateful to Dr. John Benek for suggesting the idea of combining the Discontinuous Galerkin method with the Chimera scheme. Jack not only provided insight regarding Chimera schemes, but he also served as a mentor of the Discontinuous Galerkin method and other algorithms. In working with Jack, I thoroughly enjoyed the countless hours of discussions of color coded mathematical expressions on the white board. Not only did Paul and Jack expand my mathematical knowledge, they significantly contributed to my growth in regard to technical writing by dedicating untold hours reviewing this and other manuscripts. My other two committee members Dr. Mark Turner and Dr. Shaaban Abdallah also deserve considerable recognition. Mark's deep understanding of fluid dynamics and sparse matrix solvers was invaluable. Discussions in Mark's office were always insightful and entertaining, and he always found ways to get me involved in turbomachinery projects. However, it is Shaaban that I have to thank for inspiring my initial efforts to developing CFD codes. His creativity in regard to CFD algorithms served as inspiration for my pursuit of this dissertation.

I am thankful to the dedicated researchers at the Computational Sciences Center of Wright Patterson Air Force Base (WPAFB) for their guidance and support throughout my college career. Notably, Dr. Miguel Visbal, Dr. Donald Rizzetta, Dr. Philip Morgan, Dr. Michael White and Dr. Scott Sherer provided guidance throughout the confusing, frustrating, and discouraging earliest days as a young aerospace engineer student in the field of computational fluid dynamics. I continue to benefit from their wisdom and experience to this day. I also wish to thank Dr. Carl Tilmann and Dr. Micheal Ol for taking me under their wing when I first arrived at WPAFB. Special recognition is reserved for Dr. Kirti N. Ghia and Dr. Urmila Ghia at the University of Cincinnati for introducing me to the field of computational fluid dynamics my freshman year

of college. I would not be where I am today without their continued dedication to their students.

My understanding of the Discontinuous Galerkin method can be attributed to Dr. Donald French, Dr. Krzysztof Fidkowski, Dr. Bram van Leer, Dr. Mauricio Osorio, and Dr. Marcus Lo. Don in particular dedicated numerous hours teaching me both finite element methods as well as sparse linear algebra analysis. He was always willing to entertain my mathematical inquiries, even when it seemed like we were speaking two different languages. Besides mentoring me personally, his math classes geared towards engineering students were always practical and informative. I am also deeply grateful to Krzysztof for allowing me access to his Discontinuous Galerkin code. It was only by studying his work that I was able to appreciate the importance of, amongst other things, a complete linearization of the discrete partial differential equations. His contributions to the completion of this dissertation are likely greater than he knows.

I'd like to also offer gratitude to the members of the Gas Turbine Simulation Laboratory at the University of Cincinnati. Robert Ogden is always ready to replace a broken hard drive or fix a corrupted graphics card driver. He is most certainly overworked but not underappreciated. Kiran Siddappaji and Ahmed Nemnem both spent a great deal of effort assisting me with generating turbomachinery cascade blades. Furthermore, I would still be designing input files if it weren't for the programming skills of Robert Knapke. Not only has he used the code as the basis for his own research, he has significantly contributed to the overall design of the code. I could not ask for a more reliable and capable colleague. Notably, I have to also thank Michael List. Mike is both a great colleague and friend. Not only did he help me with visualization, generating smooth airfoil shapes, coding practices, and other CFD tools, he provided hot meals and a roof over my head during my work at WPAFB. Thanks to Dave Car for my introduction to the Python programming language, and I learned a great deal from Dave's programming skills. I am sincerely grateful towards David Car and Micheal List for developing the Blockit pre-processing framework. Without their hard work and insightful discussions, none of this work would have been possible.

Besides the insights by Dr. John Benek in regard to the Discontinuous Galerkin Chimera scheme, Robert Haimes provided a number of invaluable discussions. He also contributed ideas for the hole cutting processes presented in this dissertation. Suggestions by Bob and Dr. Steve Allmaras in regards to the mass flux errors significantly contributed to the quality of my dissertation. I am also indebted to Dr. Ralph Noack for assistance with the SUGGAR software.

I wish to thank my friends and family for their dedication over the years. I want to thank my father, Robert Galbraith, for inspiring me to pursue a scientific career and for introducing me to the field of

aerospace engineering as a child. I also am deeply thankful to my mother, Ann-Christine Eriksson, for teaching me to think one step ahead and keep my feet firmly planted on the ground. I want them both to know that they have done their job well. My brother, Daniel Galbraith, has always been there for me, and he is always ready to have a brotherly discussion. I also want to thank my stepmother, Beth Galbraith, and my sister, Emily Galbraith, for their support. Most of all, I would like to thank my loving girlfriend, Melissa Muchmore, for standing by me throughout the many years it took for me to complete this dissertation. Spending countless hours working on the code would not have been feasible without her support and belief in me.

I am deeply grateful for the financial support that I received over the past several years. This work was made possible by the United States Department of Defense (DoD) through the National Defense Science & Engineering Graduate Fellowship (NDSEG) Program and the cooperative education program at WPAFB.

Finally, I would like to summarize my experience as a graduate student with the following quote:

“Flying is the second greatest thrill known to man.... Landing is the first!”

–Unknown Author

Contents

Abstract	ii
Acknowledgments	v
Contents	viii
List of Figures	xii
List of Tables	xix
Nomenclature	xxi
1 Introduction	1
1.1 Motivation	1
1.2 High-Order Discretization Methods	5
1.2.1 Finite Difference	6
1.2.2 Finite Volume	7
1.2.3 Spectral Volume	8
1.2.4 Discontinuous Finite Elements	9
1.3 The Chimera Overset Method	10
1.4 Dissertation Overview	13
2 Discontinuous Galerkin Method for Conservation Form Equations	18
2.1 Discontinuous Galerkin Method	18
2.1.1 Newton's Method	23

2.2	Conservation Form Equations	26
2.2.1	Scalar Equations	26
2.2.2	Scalar Boundary Conditions	28
2.2.3	Fluid Flow Equations	29
2.2.4	Fluid Flow Boundary Conditions	45
2.2.5	Artificial Viscosity	58
2.2.6	Artificial Viscosity Boundary Conditions	64
2.3	Summary	65
3	Discontinuous Galerkin Solver Implementation	66
3.1	Linear Algebra Libraries	66
3.1.1	Template Expressions	67
3.1.2	Dense Linear Algebra Library	74
3.1.3	Block-Sparse Iterative Matrix Solver Library	77
3.2	Flux and Flux Jacobian Integration	78
3.2.1	Cell Curvilinear Coordinates	80
3.2.2	Analytical Quadrature-free Integration	88
3.3	Solution Advancement	114
3.3.1	Convergence to Steady State	114
3.3.2	Time Integration	116
3.4	Verification Test Cases	118
3.4.1	Scalar Equations	119
3.4.2	Inviscid Flow	138
3.4.3	Viscous Flow	151
3.5	Summary	161
4	A Discontinuous Galerkin Chimera Scheme	163
4.1	Artificial Boundaries	163
4.1.1	Implicit Artificial Boundaries	171
4.1.2	Implementation Strategy	175
4.2	Hole Cutting with Curved Cells	181

4.2.1	Details of the Cutting Surface Flagging Process	184
4.2.2	Cutting Groups	191
4.2.3	Hole Cutting Examples	193
4.2.4	Summary	199
4.3	Verification Test Cases	199
4.3.1	Scalar Equations	199
4.3.2	Inviscid Flow	209
4.3.3	Curved Geometry with Viscous Flows	242
4.4	Parallel Performance	263
4.4.1	SKF 1.1 Airfoil	266
4.4.2	Steady Circular Cylinder at Re=40	272
4.4.3	Summary	280
5	Conclusion and Future Work	282
5.1	Summary and Conclusions	282
5.2	Future Work	284
Bibliography		287
A Non-Dimensionalization of the Total Enthalpy Equation		303
B Comparison of Reciprocal Density Projections via Model and Quadrature Analysis		305
B.1	Modal Approximation	306
B.2	Quadrature Projection	306
B.3	Preliminary Approximation Theory	307
B.4	Modal Approximation Error Estimate	308
B.5	Quadrature Approximation Error Estimate	308
B.6	Proof of Quadrature Error Estimate	309
C Iterative Sparse Matrix Solver Preconditioners		310
C.1	Lower-Upper Symmetric Gauss Seidel (LU-SGS)	310
C.2	Incomplete Lower-Upper (ILU)	312

C.2.1	ILU(0) 2D	313
C.2.2	ILU(0) 3D	316
C.2.3	ILU(1) 2D	318
C.2.4	ILU(1) 3D	320
D	Solution to Multidimensional Burger's Equation	324
E	Bassi and Rebay Scheme 2 Linearization	326

List of Figures

1.1	Artificial Boundary Communication	12
1.2	Large Stencil Hole Cutting Issues	13
1.3	Example of Inaccurate Interpolation associated with Overset Meshes on Curved Geometries	15
2.1	Graphical Explanation of the Notation for Cell Volume and Boundary Integrals	19
3.1	Expression Tree for The Linear Algebra Expression in Eq. 3.1	68
3.2	Nodal and modal representation of cells.	82
3.3	Nodes used to form 3D Cells	82
3.4	Illustration of the Two-Dimensional Unit Cell	88
3.5	Table of Tensor Product of Basis Functions where $\psi_{ij} = \psi_i(\xi) \psi_j(\eta)$	92
3.6	Modal Coefficient Ordering of Nested Orders of the Approximation	92
3.7	Convergence Rates for the One-dimensional Linear Poisson Equation	120
3.8	One-Dimensional Linear Poisson Equation Solutions with Five Cells (Black Line - Analytical Solution)	121
3.9	Convergence Rates for the Two-dimensional Linear Poisson Equation	122
3.10	Two-Dimensional Linear Poisson Equation Solutions with 8×8 Cells	123
3.11	Convergence Rates for the One-dimensional Non-Linear Poisson Equation	125
3.12	One-Dimensional Non-Linear Poisson Equation Solutions with Five Uniform Cells (Black Line - Analytical Solution)	126
3.13	One-Dimensional Non-Linear Poisson Equation Solutions with Five Clustered Cells (Black Line - Analytical Solution)	127
3.14	Convergence Rates for the One-dimensional Linear Advection Diffusion Equation	129

3.15 One-Dimensional Linear Advection Diffusion Equation Solutions with Five Uniform Cells (Black Line - Analytical Solution)	130
3.16 One-Dimensional Linear Advection Diffusion Equation Solutions with Five Clustered Cells (Black Line - Analytical Solution)	131
3.17 Convergence Rates for the Two-dimensional Linear Advection Diffusion Equation	132
3.18 Two-Dimensional Linear Advection Diffusion Equation Solutions with 8×8 Uniform Cells	133
3.19 Convergence Rates for the One-dimensional Burger's Equation	135
3.20 One-Dimensional Burger's Equation Solutions with Five Uniform Cells	136
3.21 Convergence Rates for the Two-dimensional Burger's Equation (Black Line - Analytical Solution)	137
3.22 Two-Dimensional Burger's Equation Solutions with 8×8 Uniform Cells	138
3.23 Smooth Bump Geometry	139
3.24 Smooth Bump Spatial Order of Accuracy Verification with XFLOW	140
3.25 Gaussian Smooth Bump Solutions with 6×2 Grid	141
3.26 Gaussian Smooth Bump Solutions with 96×32 Mesh and $N = 4$	141
3.27 Gaussian Smooth Bump Quasi-Newton Convergence History for each Grid Size	142
3.28 Gaussian Smooth Bump Spatial Order of Accuracy and Execution Time Comparison with Commercial Codes	143
3.29 Grids for Ringleb Flow	146
3.30 Ringleb Flow Spatial Order of Accuracy	147
3.31 Mach Number Contours Computed for Ringleb Flow with 6×2 Grid	147
3.32 Oblique Shock Mesh	148
3.33 Oblique Shock Surfaces of Pressure Coefficient with and without Artificial Viscosity	149
3.34 Oblique Shock Pressure Coefficient on the Lower Wall	150
3.35 Oblique Shock Quasi-Newton Convergence History	151
3.36 Computational domain for channel flow	153
3.37 Fully Developed Channel Flow	153
3.38 Computational Domain for Laminar Blasius Boundary Layer Flow	154
3.39 Blasius boundary layer flow $Re = 10^6$	155
3.40 Lid Driven Cavity	156

3.41 Streamline for the Lid Driven Cavity, (20×20) Cells, $Re = 100$	157
3.42 Streamline for the Lid Driven Cavity, (30×30) Cells, $Re = 1000$	158
3.43 Streamline for the Lid Driven Cavity, (40×40) Cells, $Re = 3200$	159
3.44 Lid driven cavity velocity components	160
4.1 Interior Boundary Integration	164
4.2 Overlapping grids	164
4.3 Obtaining Conservative Variables from the Blue Mesh	164
4.4 Sufficient Overlap for Zonal Type Interfaces	167
4.5 Insufficient Overlap for Zonal Type Interfaces and Corrections for Sufficient Overlap	167
4.6 Overlapping grids	168
4.7 Obtaining Conservative Variables from the Black Mesh	170
4.8 Example Mesh for Implicit Chimera Artificial Boundary	171
4.9 Example Chimera Mesh Used to Detail the Implementation of the Artificial Boundary Communication Scheme	175
4.10 Details of the Donor Implementation	177
4.11 Details of the Receiver Implementation	177
4.12 Schematic of the Periodic Boundary Condition Implementation	179
4.13 Schematic of the MPI Implementation	180
4.14 Cutting Surfaces Used to Flag Boundary Between Hole and Field Cells (Blue - Hole Cells, Red - Field Cells, White - Undetermined)	182
4.15 Open Cutting Surface Leading to a Leak in the Fill Process (Blue - Hole Cells, Red - Field Cells, White - Undetermined)	183
4.16 Half Circle using Background Grid Boundary to Close Cutting Volume (Blue - Hole Cells, Red - Field Cells, White - Undetermined)	184
4.17 Cutting Surfaces with Small Segments in Size Relative to the Background Grid Cells (Blue - Hole Cells, Red - Field Cells, White - Undetermined)	185
4.18 Cutting Surfaces with Segments Much Larger in Size Relative to the Background Grid Cells (Blue - Hole Cells, Red - Field Cells, White - Undetermined)	186
4.19 Nelder-Mead Method	188

4.20 Nelder-Mead Degenerate Situation were Vertices Remain on Cell Boundary	189
4.21 Nelder-Mead with Multiple Local Minimums	190
4.22 Resolving Contradictory Flagging of a Cell from Multiple Faces	191
4.23 Example of Improper Hole Cutting with Two Overlapping Cutting Surfaces	191
4.24 Examples of Normal Vectors used to Flag Field and Hole Nodes	192
4.25 Final Holes in the Double Cylinder Mesh	193
4.26 SKF 1.1 Airfoil	194
4.27 SKF 1.1 Airfoil Finite Volume Chimera Mesh	195
4.28 SKF 1.1 Airfoil with Flap Meshes	196
4.29 Staggered Tube Bank Overset Mesh	197
4.30 Hole Cutting Groups for the Stagger Tube Bank (Green Edges are the Cutting Boundaries) .	198
4.31 Coarsest Zonal Mesh for Scalar Equation Order of Accuracy Calculations	200
4.32 Coarsest Chimera Meshes for Scalar Equation Order of Accuracy Calculations	200
4.33 Convergence Rates for the Linear Poisson Equation using Zonal and Chimera Meshes . . .	201
4.34 Two-Dimensional Linear Poisson Equation Solutions with Zonal Mesh with 8×8 Cells . .	202
4.35 Two-Dimensional Linear Poisson Equation Solutions with Chimera Mesh with 25% Overlap and 8×8 Cells	203
4.36 Convergence Rates for the Linear Advection Diffusion Equation using Zonal and Chimera Meshes	204
4.37 Two-Dimensional Linear Advection Diffusion Equation Solutions with a Zonal Mesh with 8×8 Cells	205
4.38 Two-Dimensional Linear Advection Diffusion Equation Solutions with a Chimera Mesh with 25% Overlap and 8×8 Cells	206
4.39 Convergence Rates for Burger's Equation using Zonal and Chimera Meshes	207
4.40 Two-Dimensional Burger's Equation Solutions with Zonal Mesh with 8×8 Cells	208
4.41 Two-Dimensional Burger's Equation Solutions with Chimera Mesh with 25% Overlap and 8×8 Cells	209
4.42 Smooth Bump Geometry	212
4.43 Smooth Bump Zonal Meshes	213

4.44 Observed Order of Accuracy using the Zonal Meshes with Different Number of Quadrature Nodes	214
4.45 Smooth Bump Zonal Mesh Mass Flux Error	215
4.46 Smooth Bump with 3 Domains	216
4.47 Smooth Bump Spatial Order of Accuracy with 3 Domains	216
4.48 Smooth Bump with 3 Domains Mass Flux Error	217
4.49 Turbomachinery Cascade Blade Meshes	218
4.50 Turbomachinery Cascade Blade, ($M_\infty = 0.25$)	219
4.51 Channel with 10% Circular Arc Meshes	220
4.52 Channel with 10% Circular Arc, ($M_\infty = 0.675$)	221
4.53 Diffuser Meshes	222
4.54 Normal Shock Pressure Coefficient	223
4.55 SKF 1.1 Airfoil Meshes	226
4.56 SKF 1.1 Airfoil, ($M_\infty = 0.4$, $\alpha = 2.5^\circ$)	227
4.57 SKF 1.1 Airfoil, ($M_\infty = 0.76$, $\alpha = 2.5^\circ$)	228
4.58 Supersonic Inviscid Cylinder Meshes, ($M_\infty = 2.0$)	229
4.59 Circular Cylinder Pressure Coefficient, ($M_\infty = 2$)	231
4.60 SKF 1.1 Airfoil with Flap Meshes	233
4.61 SKF 1.1 Airfoil with Flap Pressure Coefficient, ($M_\infty = 0.2$, $\alpha = 3^\circ$)	234
4.62 Isentropic Convecting Vortex Meshes, ($M_\infty = 0.5$)	237
4.63 Convecting Isentropic Vortex after 12 Characteristic Times, ($M_\infty = 0.5$)	238
4.64 Analytical 3D Body	239
4.65 Analytical 3-D Body Meshes	240
4.66 Analytical 3-D Body Lift and Drag and Pressure Coefficient	241
4.67 Example of Inaccurate Interpolation associated with Overset Meshes on Curved Geometries	243
4.68 Generic First Stage Launch Vehicle	244
4.69 Launch Vehicle Nose Section	244
4.70 Finite Volume Meshes	245
4.71 Discontinuous Galerkin $N_g = 3$ Polynomial Mapping Meshes	246

4.72 Finite Volume C_p Contours on the Generic Nose Section (Single Mesh - Black Lines, Chimera Mesh - Gray Lines)	247
4.73 Finite Volume Entropy Rise Contours on the Generic Nose Section (Single Mesh - Black Lines, Chimera Mesh - Gray Lines)	249
4.74 Discontinuous Galerkin C_p Contours on the Generic Nose Section	249
4.75 Discontinuous Galerkin Entropy Rise Contours on the Generic Nose Section	250
4.76 Chimera Mesh Overlapping Region	250
4.77 $ \bar{V} $ Contour in the Overlapping Region	251
4.78 Velocity Magnitude One Point off the Wall	251
4.79 C_p Contours in the Overlapping Region (Single Mesh - Black Lines, Chimera Mesh - White/Gray Lines)	252
4.80 Entropy Rise Contours in the Overlapping Region	252
4.81 Boundary Layer on Common Grid Line	253
4.82 Finite Volume Meshes	254
4.83 Discontinuous Galerkin $N_g = 3$ Polynomial Mapping Meshes	255
4.84 Finite Volume C_p Contours on the Circular Cylinder (Single Mesh - Black Lines, Chimera Mesh - White Lines)	257
4.85 Finite Volume Entropy Rise Contours on the Circular Cylinder (Single Mesh - Black Lines, Chimera Mesh - Gray Lines)	258
4.86 Discontinuous Galerkin C_p Contours on the Circular Cylinder ($N = 3$)	258
4.87 Discontinuous Galerkin Entropy Rise Contours on the Circular Cylinder ($N = 3$)	259
4.88 Streamlines of Wake Separation Bubble	260
4.89 Finite Volume Chimera Mesh Overlapping Region	261
4.90 Discontinuous Galerkin Chimera Mesh Overlapping Region	262
4.91 Boundary Layer on Common Grid Line	263
4.92 SKF 1.1 Airfoil Meshes	265
4.93 SKF 1.1 Airfoil Iteration Speedup with Explicit Artificial Boundaries	267
4.94 SKF 1.1 Airfoil Solution Speedup with Explicit Artificial Boundaries	267
4.95 SKF 1.1 Airfoil Convergence History with Explicit Artificial Boundaries	268
4.96 SKF 1.1 Airfoil Convergence History with Implicit Artificial Boundaries	269

4.97 SKF 1.1 Airfoil Iteration Speedup with Implicit Artificial Boundaries	270
4.98 SKF 1.1 Airfoil Solution Speedup with Implicit Artificial Boundaries	270
4.99 SKF 1.1 Airfoil Solution Time (Dashed Lines show Ideal Times)	271
4.100 SKF 1.1 Airfoil Solution Speedup with Implicit Artificial Boundaries Relative to Explicit Artificial Boundaries	272
4.101 Cylinder $Re = 40$ Meshes	274
4.102 Cylinder Iteration Speedup with Explicit Artificial Boundaries	275
4.103 Cylinder Solution Speedup with Explicit Artificial Boundaries	275
4.104 Cylinder $Re = 40$ Convergence History with Explicit Artificial Boundaries	276
4.105 Cylinder Iteration Speedup with Implicit Artificial Boundaries	277
4.106 Cylinder Solution Speedup with Implicit Artificial Boundaries	277
4.107 Cylinder Convergence History with Implicit Artificial Boundaries	278
4.108 Cylinder Solution Time (Dashed Lines show Ideal Times)	279
4.109 SKF 1.1 Airfoil Solution Speedup with Implicit Artificial Boundaries Relative to Explicit Artificial Boundaries	280
E.1 BR2 lifting operators for cell i	326

List of Tables

3.1	Template Expression for Multiplication of a Scalar and a Vector	69
3.2	Generator Functions for Multiplication of a Scalar and a Vector	70
3.3	Class used to Identify Vector Datatypes	70
3.4	Templated Class Representing a Binary Summation Operation in the Expression Tree	71
3.5	Operator Tags for Addition and Subtraction	71
3.6	Generator Functions For Vector Summation	72
3.7	Vector Class	73
3.8	C++ Code for Computing the Expression in Eq. 3.1	74
3.9	Template Expression Data Type Representing the Expression Tree in Eq. 3.1	74
3.10	Block Tri-Diagonal Solver Based on the Thomas Algorithm Using Template Expressions . .	77
3.11	PyDG Code for Performing Polynomial Expansion Products	90
3.12	PyDG Code for Converting an Integrated Polynomial Expression to a Linear Algebra Expression	91
3.13	PyDG Code to Generate the Polynomial Product of Eq. 3.47	94
3.14	C++ Code for Computing the Square of a Polynomial	95
3.15	C++ Syntax for Computing a Polynomial Cubed	95
3.16	C++ Code for Projecting Non-polynomial Functions	96
3.17	C++ Syntax for Computing a Polynomial Inverse	99
3.18	C++ Syntax for Computing the Cartesian Derivative of a Polynomial	100
3.19	C++ Syntax for Computing Weak Volume Integral	101
3.20	PyDG Code to Generate the Integral in Eq. 3.19	102
3.21	C++ Syntax for Computing a Boundary Integral	103

3.22	PyDG Code to Generate the Boundary Integral in Eq. 3.69	104
3.23	C++ Syntax for Computing the Jacobian of the Weak Volume Integral	106
3.24	PyDG Code to Generate the Integral in Eq. 3.78	107
3.25	C++ Syntax for Computing a Boundary Integral	109
3.26	PyDG Code to Generate the Boundary Integral in Eq. 3.69	110
4.1	Example of Implicit Chimera Artificial Boundary Algorithm	174
4.2	Cylinder Wake Vortex Length	260
4.3	SKF 1.1 Airfoil Degrees of Freedom	266
4.4	Cylinder $Re = 40$ Degrees of Freedom	275

Nomenclature

General

\vec{F} Flux Tensor

\vec{F}^a Advective Flux Tensor

\vec{F}^d Diffusive Flux Tensor

∇ Gradient vector, $\left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z} \right)$

t Temporal Dimension

x, y, z Cartesian Coordinates

Discretization

η_e Number of Faces of a Cell

A_e Area of the Cell e

δ_Q Newton Solver Update Vector

Δt_e Time Step of the Cell e

$\vec{\mathcal{P}}$ Polynomial Expansion of the Flux Tensor

Γ_e Spatial Boundary of the Cell e

h Representative Cell Size (1/DOF for 1D, $\sqrt{1/DOF}$ for 2D)

h_e Characteristic Length of the Cell e

h_x, h_y, h_z	Cartesian Cell Bounding Box Dimensions
$\ u\ $	L^2 -norm, $\sqrt{\int_{\Omega} u^2 d\Omega}$
\vec{n}	Spatial Cell Boundary Unit Normal Vector
N	Spatial Polynomial Order of the Solution Approximation
N_g	Spatial Polynomial Order of a Cell Geometric Mapping
N_{GQ}	Number of Gauss Quadrature Nodes Used for Artificial Boundary Integration
Ω_e	Volume Domain of the Cell e
ϕ	Numerical Dissipation Vector
P_n	n^{th} -order One-dimensional Legendre Polynomial
ψ	Polynomial Basis/Test Function
Q	Dependent Variable Vector
\vec{R}	Sum of Lifting Operators \vec{r} for BR2 Viscous Scheme
\vec{r}	Lifting Operator for BR2 Viscous Scheme
s	Cell Face Local Curvilinear Coordinate
s_k	Gauss-Quadrature Normalize Coordinate
T	Coordinate Jacobian Matrix
w_k	Gauss-Quadrature Weight
$\vec{\xi}$	Cell Local Coordinate (ξ, η, ζ)
\vec{X}	Cartesian Coordinate (x, y, z)
ξ, η, ζ	Cell Local Curvilinear Coordinates
\vec{X}_k	Cartesian Reciever Node Coordinate

Fluids

α	Angle of Attack
β_{x^i}	Viscous Work Terms
C	Airfoil Chord Length
c	Speed of Sound
C_d	Lift Coefficient, $D/(Cq_\infty)$
C_l	Lift Coefficient, $L/(Cq_\infty)$
C_p	Pressure Coefficient $(P - P_\infty)/q_\infty$
C_x	Horizontal Force Coefficient, $F_x/(Cq_\infty)$
C_y	Vertical Force Coefficient, $F_y/(Cq_\infty)$
D	Drag
ε	Artificial Viscosity Coefficient
$\tilde{\varepsilon}$	Limited Artificial Viscosity Coefficient
$\tilde{\varepsilon}_{hi}, \tilde{\varepsilon}_{low}$	Bounds on the Limited Artificial Viscosity Coefficient
\vec{F}^{av}	Artificial Viscosity Flux Tensor
F_x	Horizontal Force
F_y	Vertical Force
γ	Ratio of Specific Heats
H	Total Enthalpy
L	Lift
λ	Characteristic Speed

$\Delta\dot{m}$	Mass Flux Error, $(\dot{m}_{out} - \dot{m}_{in})/\dot{m}_{in}$
M_∞	Reference Mach Number
μ	Dynamic Viscosity
p	Static Pressure
P_∞	Reference Pressure $1/(\gamma M_\infty^2)$
Pr	Prandtl Number
Q	Conservative Variable Vector
q_∞	Reference Dynamic Pressure $\frac{1}{2}\rho\vec{V}^2$
Re	Reynolds Number
ρ	Density
ρE	Total Energy
$\rho u, \rho v, \rho w$	Cartesian Momentum Components in x, y, and z Directions
s	Entropy (p/ρ^γ)
s_0, κ	Mean Value and Range for the Shock Sensor Limiter
\tilde{S}_1	Sutherland Temperature (198.6° R)
$S_\varepsilon(Q)$	Source Term for Artificial Viscosity Poisson Equation
\hat{s}	Entropy Rise $(s - s_\infty)/s_\infty$
\tilde{s}_k	Limited Shock Sensor
S_k	Shock Sensor
T	Temperature
$\tau_{x^i x^j}$	Viscous Stress Tensor

u, v, w Cartesian Velocity Components in x, y, and z Directions

\vec{V} Velocity Vector [u, v, w]

Scalar Equation

\vec{c} Direccion Vector [c_x, c_y, c_z]

$\mu(u)$ Viscosity Coefficient

u Dependent Variable

Hole Cutting

d Distance Between a Point on a Cell and a Point on a Cutting Face

s Cutting Surface Segment Parametric Coordinate

x_c, y_c Cartesian Coordinates of a Call

x_s, y_s Cartesian Coordinates of a Cutting Surface Segment

Subscript

∞ Reference Quantities

Superscript

$-$, $+$ Cell Interior and Exterior Values

$\bar{\cdot}$ Cell Mean Value

$\hat{\cdot}$ Dimensional Quantity

Acronyms

CFD Computational Fluid Dynamics

CFL Courant-Freidrichs-Lewy Number

DG Discontinuous Galerkin

DOF Degrees of Freedom

GMRES Generalized Minimum Residual

Chapter 1

Introduction

1.1 Motivation

Fluid dynamics plays a vital role in a wide array of applications ranging from the study of astrophysics to the aeronautical, automotive, and medical industries. The study of fluid dynamics has traditionally been founded on experimental research. In the aeronautical industry, experiments range from large scale wind-tunnels capable of providing measurements on full scale aircraft to small water tunnels. Regardless of the size of the experiment, they are expensive and suffer from relatively slow turn around time. In addition, there are often limitations in measurement instrumentation to probe all aspects of the flow field.

The increase of computational power over the past several decades has caused a paradigm shift in which aerodynamic design is no longer experimentally based but rather is computationally based. Numerical solutions to the Navier-Stokes equations, a set of partial differential equations that model Newtonian fluids, is known as Computational Fluid Dynamics (CFD). CFD can be used as a predictive tool to reduce the design space, aid in determining what measurements should be taken for a particular experiment, or compliment experimental measurements when an experiment is limited by measurement access/capabilities or flow conditions. The requirements on validity, accuracy, numerical efficiency, and flexibility in mesh generation are continuously increasing as CFD becomes a more integral part of the design process. This has lead to the development of sophisticated solution algorithms and computer software that is executed on massively parallel machines. Despite great advancements in CFD codes, engineers are still seeking mathematical models with increased accuracy and solution algorithms with increased efficiency.

In recent years, research in the development of CFD algorithms has shifted to high-order schemes,

i.e., schemes that are third-order or higher. Research has shown that high-order methods can produce flow field estimates that are within engineering tolerances with reduced computational requirements compared to conventional 2^{nd} -order accurate methods.[1, 2, 3] In addition, high-order methods are suited particularly well for Large-Eddy Simulations (LES).[4, 1, 5, 6, 7] Unlike Reynolds Averaged Navier-Stokes (RANS) models, which model the dissipation of the entire spectrum of turbulent scales, LES simulations directly compute the unsteady larger scale turbulence structures that can be resolved with a given numerical mesh, and only model the dissipation of the scales that are not resolved by the mesh. While LES calculations are expensive due to their inherent unsteady nature and increased mesh resolution requirements, they have the potential to further the understanding of turbulent flows and aid in the improvement of less expensive RANS models. Researchers have found that high-order numerical methods with low dispersion errors improve accuracy while increasing computational efficiency of LES calculations. The reduced computational cost comes in the form of reduced grid resolution requirements compared to grids for 2^{nd} -order methods.

The high-order Discontinuous Galerkin (DG) method has been growing in popularity over the past decade.[8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 2, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28] The DG method is a finite element method that provides a natural extension of the Finite Volume method to high orders of accuracy. The high order of accuracy is obtained by explicitly representing the numerical solution within the cell of a grid as a polynomial expansion. Rather than expanding the difference stencil, as is done in Finite Volume and Finite Difference schemes, the DG method obtains higher orders of accuracy by increasing the order of the polynomial expansion. Thus, the DG method retains a small stencil where cells only communicate with their immediate neighbors. The DG method relies on discontinuous test functions that cause the numerical solution to be continuous within the cell, but allows the solution to be discontinuous across cell boundaries. This makes the DG method well suited for shock capturing. As the DG method is an extension to the Finite Volume scheme, DG methods are able to take advantage of the numerous numerical flux schemes and limiters developed for the Finite Volume method. In particular, algorithms in computer codes that rely on a nodal DG formulation will closely resemble that of a Finite Volume solver.

The nodal and modal formulations are two approaches used to implement a code based on a DG discretization. The nodal approach is currently the most commonly used, and allows developers to use a framework similar to traditional Finite Volume codes from the computer science perspective. In the nodal method, integrals of fluxes are evaluated using numerical quadrature. The fluxes are computed from the dependent variables at quadrature nodes within the cell and the integral is evaluated by a weighted sum of

the nodal fluxes. The nodal method is favored for the Continuous Finite Element Method (FEM). While the FEM scheme only requires the evaluation of cell volume integrals, the DG method requires both cell volume and cell boundary integrals. Unfortunately, no single set of quadrature nodes exists that are capable of evaluating both cell volume and boundary integrals without loss of integration accuracy. The most common solution is to evaluate the polynomial expansions at the separate quadrature nodes for the cell volume and boundary integrals. While this solution maintains the accuracy of the integration, it comes at a significant computational expense.

The modal scheme relies on some form of analytical integration, or numerical integration performed through a pre-processor or during the initialization phase of the solver.[29, 30, 22, 23, 31] The modal DG method offers a reduced operation count at the cost of increased complexity in formulation and implementation of the solver. For example, Atkins et al.[29] evaluated inner products in the governing equations during the initialization of the solver and stored the result in a matrix form. C-macros were then developed to perform matrix multiplications that would utilize the sparsity pattern of the matrix to improve efficiency. The C-macros require knowledge of the sparsity pattern of the matrix, and a macro must be specialized for each choice in test functions. As generating a family of C-macros is tedious and error prone, the author used symbolic manipulation software combined with a pre-processor to automatically evaluate polynomial inner products and generate the equivalent code produced by the C-macros. This type of automation greatly increases the legibility and hence maintainability of the code.

High-order DG methods utilize high-order curved cell representation of curved geometry in order to obtain accurate solutions.[14] Finite Difference and Finite Volume schemes are restricted to using linear cells and grid refinement is the only method available for improving geometric representation of a curved geometry. However, for Finite Element schemes, high-order representation of the geometry can be explicitly defined through the use of cells with curved boundaries. Unfortunately, generating curved cells for unstructured meshes is not trivial. For example, Persson and Peraire[26] solved a set of equations inspired by Lagrangian Solid Mechanics to obtain curved cell boundaries that conform to the geometry while producing computationally acceptable cells. The construction of curved cells is significantly simpler for structured meshes. On a structured mesh, linear cells can be agglomerated to generate the curved elements without involving a solution of partial differential equations. If the cell stretching is not too severe, a structured mesh will not produce degenerate curved cells.

In addition to the complications associated with generating curved cells on unstructured meshes, Babuska

and Aziz demonstrated that the accuracy of the finite element approximation degrades as the maximum angle of triangular elements approaches 180° asymptotically.[32] Several unstructured mesh generation techniques have been developed to avoid generating triangles with large angles, such as the minimum-maximum triangulation[33, 34], and hybrid schemes that introduce quadrilateral elements in regions of high anisotropy.[35, 36, 21] Quadrilateral elements do not exhibit the same loss of accuracy due to anisotropy as tetrahedral elements as they do not degenerate after successive refinement in one direction. In addition, quadrilateral elements are more accurate on the highly stretched grids that are necessary for viscous flows.[37] Finally, implicit solvers are more complicated to implement on unstructured meshes compare to structured meshes where the cell connectivity is regular.

Unfortunately, a single structured mesh does not provide the same flexibility to accommodate complex geometry as unstructured meshes. The Chimera grid system was introduced by Benek et al. [38] for the Euler equations to increase the flexibility of structured meshes to represent complex geometries by defining a set of overlapping computational subdomains. The boundaries of the subdomains that are interior to the computational domain and do not coincide with the domain boundary are called artificial boundaries. For finite volume and finite difference schemes, additional points exterior to the artificial boundaries are required to maintain the interior difference stencil. These points are called fringe points; they form a fringe exterior to the subdomains. Fringe points in a Chimera overset scheme are equivalent to ghost points used to maintain the interior stencil across grid boundaries in a multi-block scheme[39, 40, 41]. The difference is that fringe points are explicitly included in the grid system during the grid generation process, whereas ghost points are implicitly created during an initialization process and are coincident with points in the neighboring grids. The values of the dependent variables at the fringe points are obtained by interpolation from overset grids. The interpolation provides the coupling mechanism between the overset grids. Sufficient overlap between grids is required for proper interpolation to the fringe points. Insufficient overlap can result in reduced order of accuracy in the interpolation or a failure to establish proper interpolation. Fringe points without proper interpolation are often denoted as orphan points[42, 43]. The grid system must be adjusted if orphan points are present, typically in a manual fashion, until no orphan points exist.

The high-order DG method combined with the Chimera overset method offers solutions to many problems associated with the implementation of the Chimera method using traditional high-order Finite Difference and Finite Volume schemes. Most notably, the compact stencil of the DG method eliminates the need for fringe points that are required to preserve the difference stencil across artificial boundaries for high-order

Finite Difference and Finite Volume schemes. Furthermore, a large interpolation stencil is not required for artificial boundaries because the higher-order information is retained within the polynomial representation of the approximate solution. The DG-Chimera artificial boundary scheme also naturally reduces to the scheme of the interior faces for the case of coincident abutting faces. Hence, a DG-Chimera grid system is always valid so long as no physical gaps exist between the grids.

The use of curved cells also has the potential to solve inter-grid communication issues associated with overlapping viscous meshes on curved geometries. The grid boundary of a Finite volume or Finite difference grid is defined by straight lines that are secants to the curved surface of the geometry. Thus, the cells near the surface can actually reside inside the body, especially when the grids are tightly clustered to the surface. Therefore, an artificial boundary point that is located on the surface of the body may reside in a donor cell that is several cells removed from the body and not adjacent to it. As a result, the artificial boundary will be updated from the incorrect donor cell. This error is purely a result of the linear representation of the curved geometry. The use of curved cells offers a simple solution to this problem by explicitly approximating the curvature of the geometry.

The Chimera scheme also allows for the exclusion of computational cells (known as hole cutting) to remove cells that lie outside the computational domain. For example, consider a body fitted mesh for a sphere placed at the center of a much larger Cartesian background grid. All the cells from the background grid that reside within the body of the sphere would be excluded from the computation by using hole cutting. Hole cutting is a useful tool for fitting structured meshes on complex geometries. However, maintaining an adequate difference stencil for high-order Finite Difference and Finite Volume schemes is not trivial. Sherer et al. [44] implemented a pre-processor with extensive logic required to maintain adequate stencils for a high-order FD scheme. This issue is naturally resolved by the DG-Chimera scheme as it lacks fringe points and uses a minimal stencil.

1.2 High-Order Discretization Methods

This section summarizes the development of some of the more common high-order methods applied to CFD. It highlights their benefits and weaknesses leading to the decision to focus on the Discontinuous Gelerkin method for this work.

In the following discussion, the order of accuracy of a scheme refers to the rate at which the error

decays as $h \rightarrow 0$, where h is a measure of the grid spacing. The industry standard currently relies primarily on second-order accurate schemes, i.e. $O(h^2)$. A scheme is typically considered high-order when the convergence rate is 3^{rd} -order or higher, $O(h^q)$, where $q \geq 3$. Higher-order schemes can achieve acceptable engineering error tolerances with fewer degrees of freedom and reduced computational time compared to 2^{nd} -order Finite Volume and Finite Difference schemes[2, 1, 45]. Several discretization techniques capable of achieving high-orders of accuracy are discussed in the following sections.

1.2.1 Finite Difference

Finite Difference schemes establish a numerical system of equations by substituting discrete numerical derivatives for the analytical derivatives in the conservation law equations. The numerical derivatives are formed as differences between the numerical solution at a node and its neighbors. The first extension of Godunov's [46] 1^{st} -order accurate finite difference scheme that can resolve shocks while maintaining monotonicity is van Leer's Monotonic Upstream-centered Scheme for Conservation Laws (MUSCL)[47, 48, 49, 50, 51], and has become the industry standard for many 2^{nd} - and 3^{rd} -order accurate codes. A less known but similar scheme was also independently developed by Kolgan[52, 53] in the same time period. The MUSCL scheme relies on non-linear limiters to maintain monotonicity and eliminate the Gibbs phenomena, where oscillations of the numerical solution occur in the vicinity of solution discontinuities. Since the introduction of the MUSCL scheme, much effort has been dedicated to devising a variety of limiters.

The 2^{nd} -order accuracy of the MUSCL scheme is achieved by reconstructing linear solutions based on neighboring cell average values. The higher-order Essentially Non-oscillatory (ENO) scheme was introduced by Harten et al.[54]. The ENO scheme achieves high-order accuracy by reconstructing high-order non-oscillatory polynomials using limiters and neighboring cell averaged values. The order of accuracy of the ENO scheme was later improved by the introduction of the Weighted Essentially Non-oscillatory (WENO) scheme by Liu et al.[55]. Several flavors of the WENO scheme have since been developed.[56] Higher order of accuracy of Finite Difference scheme typically comes at the cost of a larger stencil, i.e., a single node needs to communicate with an increasing number of neighboring nodes located at a greater distance. The stencil size is one node greater than the obtained order of accuracy, i.e., a 4^{th} -order accurate scheme requires five nodes. These large stencils impose restrictions on grid regularity for stability reasons. In addition, they complicate the implementation of parallel algorithms.

An alternative approach that exhibits spectral convergence rates is the compact difference scheme de-

veloped by Lele[57]. The compact scheme reduces the stencil by using an implicit formulation to compute the numerical derivatives, and is hence primarily restricted to structured meshes. For example, a 6^{th} -order scheme requires the solution of a tri-diagonal system along a grid line of a structured mesh with a stencil of five points, and an 8^{th} -order discretization requires the solution of a penta-diagonal system. The compact scheme, coupled with a filter for stabilization, has been successfully applied to a variety of flow fields using curvilinear meshes and Chimera grids.[58, 59, 5, 60, 61, 62, 63] However, the treatment of shocks with the compact scheme is still an area of active research.[64, 65] In addition, despite the smaller stencil, the compact scheme is sensitive to grid irregularities[66]. In practice, for stability reasons, the grid is required to be near C_1 continuous along grid lines and cell stretching is often limited to 10%. Additional stability is achieved by reducing the order of accuracy near grid boundaries, which is compensated by increasing the mesh resolution. For parallel execution, inter-grid communication is treated with high-order one sided stencils.[67]

1.2.2 Finite Volume

The Finite Volume (FV) scheme discretizes the conservation law equations by integrating them over a control volume, and then transforming the volume integral into a boundary integral of the fluxes using Green's theorem. The numerical value within the mesh cells is explicitly taken to be the average value of the approximate solution, and the boundary fluxes are replaced with numerical approximate upwind fluxes, or approximate Riemann solvers[68, 69]. The method was first applied to fluid dynamics problem by McDonald[70] and MacCormack and Paullay[71]. Unlike the finite difference scheme, the FV scheme lends itself to meshes that consist of elements other than quadrilaterals and hexahedrals, such as triangles, tetrahedrons, prisms, and pyramids, as it only requires numerical approximations of the flux at cell interfaces. However, schemes that construct numerical upwind fluxes that rely solely on information from the cells sharing a common cell interface are 1^{st} -order accurate.

FV schemes rely on reconstructing high-order polynomial representations of the solution based on neighboring cells, much like the finite difference MUSCL scheme, in order to achieve high orders of accuracy. In three spatial dimensions, reconstruction for high-order stencils grows non-linearly, requiring 27 cells on a Cartesian mesh, 50 to 70 cells for 3^{rd} -order, and over 120 cells for 4^{th} -order.[72, 21] The coefficients are often stored for performance reasons, thus increasing memory requirements. The large stencil also increases the complexity and quantity of data that needs to be transferred for a parallel solver, and increases

the difficulties in implementing implicit iterative algorithms and boundary conditions[2, 73, 66]. Higher than linear reconstruction is not often used in practice because of these restrictions.

1.2.3 Spectral Volume

A Spectral Volume (SV) discretization uses a piecewise continuous polynomial to represent the approximate solution within a cell [74, 75]. The cell is subdivided into sub-cells called control volumes and the approximate solution is stored as a mean value for each control volume. Similar to a traditional FV method, the cell polynomial solution is reconstructed with Lagrange polynomials using the control volume mean values. The SV method is equivalent to a Petrov-Galerkin method where the test function and polynomial expansions are different. In this case, the polynomial expansions are the Lagrange polynomials, and the test functions are a set of sub-cell piecewise constant polynomials. The mean values of the control volumes are updated using the same principals as a traditional FV method. An approximate Riemann solver is used to compute the flux on shared cell faces that are between cells. However, fluxes on faces that are shared between control volumes within the same cell are evaluated from a reconstructed polynomial of the flux vector. An approximate Riemann solver is not necessary for control volume faces within a cell as the reconstructed flux vector is continuous by definition. However, the reconstructed polynomial flux vector on control volume faces within a cell must be integrated with appropriate quadrature rules. In higher dimensions, the quadrature significantly increases the computational cost of the SV method. Harris et al. [76] developed a Quadrature Free SV method (QFSV) by using a near optimal set to sub-cell control volumes [77] that can be integrated once to improve the efficiency of the SV method.

The SV method shares many traits with the Discontinuous Galerkin (DG) method. Both methods rely on a cell piecewise polynomial representation of the approximate solution, they are also conservative, compact, suitable for parallelization, and can handle complex geometries. Numerical experiments have shown that, in general, the DG discretization achieves lower magnitude errors, while the SV method can be updated with larger time steps [74, 78]. In two spatial dimensions, the 2^{nd} -order SV method is significantly faster than the DG method; however, the efficiency of the 3^{rd} - and 4^{th} -order SV schemes are comparable to the corresponding DG methods in two spatial dimensions [79].

1.2.4 Discontinuous Finite Elements

The Discontinuous Galerkin (DG) method is a high-order accurate discretization scheme that has received growing interest. The method was originally developed for the 1st-order neutron transport problem [80, 8], and was later extended to non-linear hyperbolic conservation laws[81, 9, 12, 10, 11, 82, 14, 83, 17, 84]. The scheme represents the approximate solution using piecewise cell local polynomials that are continuous within a given cell, but the discrete solution is allowed to be discontinuous between cell faces. Similar to the finite volume scheme, the DG method relies on an integral form of the conservation law equations. The integral form is obtained by multiplying the conservation law equations by a set of test functions and integrating over a control volume. The resulting volume integral is then transformed into a boundary and a volume integral using Gauss's theorem to convert the conservation law into a weak form. The volume integral can be evaluated without modification. The flux in the boundary integral is substituted with the same numerical upwind fluxes, or approximate Riemann solvers, used for the finite volume discretization[18, 85, 68, 86]. Thus, the traditional 1st-order Finite Volume scheme is recovered if the discrete solution is chosen to be represented by piecewise constants. The DG method lends itself to both structured and unstructured meshes due to its integral form.

Like the continuous finite element discretization, increasing the order of the piecewise polynomial approximation leads to higher order accuracy without increasing the size of the stencil. The DG method is less sensitive to the grid irregularities that plague the finite difference schemes, as the DG stencil only relies on the immediate cell neighbors.[37] In addition, the cell based DG finite element has a local mass matrix that is uncoupled from neighboring elements, unlike continuous node-based finite elements that produce a large, dense, mass matrix for the complete finite element system. The uncoupled mass matrix means the DG solution can be marched in time with either explicit, such as Runge-Kutta [17], or implicit time integration schemes[20, 87]. Furthermore, the implicit time integration will have a sparser matrix compared to the continuous finite element discretization.

The potential discontinuities at cell interfaces have led to complications when developing schemes for 2nd-order, or viscous, partial differential equations. As a result several schemes now exist. Some of the more popular discretization techniques for the viscous terms are the Bassi and Rebay 2 (BR2) [13, 24], Symmetric Interior Penalty Method (SIP)[88, 89], Local DG (LDG) [15], Compact DG (CDG) [25], and Recovery[90] schemes. The BR2 method is used in this work.

Similar to all other high-order schemes, the DG method suffers from Gibbs phenomena, where spurious oscillations in the solution occur in the vicinity of solution discontinuities, i.e., shocks and contact discontinuities. Similar to the Finite Difference and Finite Volume methods, the Gibbs phenomena in DG methods can be mitigated with either limiters[81, 91, 92, 93, 94, 95] or artificial viscosity.[19, 96, 27, 97] The limiters are based on the same principals put forth by the MUSCL scheme by van Leer. The oscillations in the solution are limited by lowering the order of accuracy, in this case the order of the polynomial solution approximation, in the vicinity of a shock. The regions containing a shock are determined through a shock detection scheme. The limiters are able to take advantage of the polynomial derivatives to create a Hermite WENO scheme that maintains the same stencil as the non-limited DG method.[92, 93, 94] The artificial dissipation method adds a diffusive second derivative term in the vicinity of a shock to damp the oscillations. Like the limiters, the artificial dissipation requires a shock detector to ensure that diffusion is not applied in smooth regions of the flow. The artificial dissipation schemes typically rely on user inputs to control the magnitude of dissipation. Excess dissipation will smear the solution while insufficient dissipation will not eliminate the spurious oscillations.

Including turbulence models is still an area of active research for DG methods. However, both the $k - \omega$ [98] and SA[99] have been successfully implemented. The implementation of a turbulence model is beyond the scope of this dissertation.

The ability to use curved elements is a significant reason for choosing the DG method for this dissertation.[26, 100] The curved elements resolve the issue of overlapping linear element meshes possibly producing incorrect interpolation stencils when mapped to a curved geometry.

1.3 The Chimera Overset Method

This section covers the historical development of the Chimera overset method and highlights both the benefits and drawbacks of the method. Three issues with the Finite Difference and Finite Volume Chimera schemes are highlighted where the Discontinuous Galerkin scheme offers solutions: Orphan Fringe Points, Hole Cutting that maintains the minimum stencil, and non-co-located artificial boundaries on curved geometries.

The Chimera scheme was first applied to solving the Euler equations by Benek et al. [38]. Their work was inspired by that of Magnus and Yoshihara[101] who used a body fitted grid to impose boundary

conditions for a curved geometry. The Chimera scheme is a generalization of their work. The scheme was later extended to three-dimensions by Benek et al.[102], and finally extended to three-dimensional viscous flows by Benek et al.[103]. The scheme produces a solution on a system of grids that communicate through an exchange of boundary data, known as artificial boundaries, in overlapping regions. The arbitrary overlapping of grids allows the mesh generator to focus on resolving individual components of the geometry independently. This provides great flexibility for the mesh generation process to focus resources on regions of interest. The Chimera method also enables swapping geometric features by replacing specific grids in the overall system of grids. This feature can be useful in a design trade study.[104] In addition, the method is suitable for simulations of bodies in relative motion, such as store separation, where body fitted meshes move with the body through a background mesh.[105, 106, 107]

The stencil of the interior scheme must be maintained on the artificial boundaries in order to obtain a continuous solution across artificial boundaries. For Finite Volume and Finite Difference schemes, this is achieved through the use of fringe points. Fringe points are additional points added to a mesh to facilitate the inter-grid communication and maintain the interior stencil across the artificial boundary. Consider an explicit 4^{th} -order differencing scheme that relies on a five point stencil. Figure 1.1a illustrates the additional fringe points required to maintain the interior five point stencil at nodes 1 and N on two abutting one-dimensional meshes. The fringe points are generated as part of the grid generation process. In this example, the fringe points directly coincide with the interior points of the neighboring grid, and the solution values at the interior nodes can be directly transferred to the fringe points. However, the fringe points are not required to coincide with interior points. As shown in Fig. 1.1b, fringe points can be updated with interpolated values from interior nodes of a donor grid. The example in Fig. 1.1b relies on a 4^{th} -order interpolation to maintain the formal order of accuracy of the scheme. However, if sufficient overlap is not provided between the two meshes, as shown in Fig. 1.1c, the fringe points cannot rely solely on interior points to establish the interpolation stencil. The use of fringe points in the interpolation stencil gives rise to a cyclic dependency where the boundary information is no longer properly updated. Fringe points that are unable to locate an adequate donor stencil are marked as Orphan Points. The overlap between the two meshes must be altered such that the fringe points obtain an appropriate donor stencil. In more general three-dimensional situations, the resolution of Orphan Points is often not trivial and can be a tedious and time consuming process. The order of accuracy of a Finite Difference or Finite Volume scheme is typically increased by increasing the stencil size. The larger stencil requires both more fringe points and a large

stencil interpolation scheme of similar order accuracy, a requirement which further complicates the process of resolving Orphan Points.[108, 44]

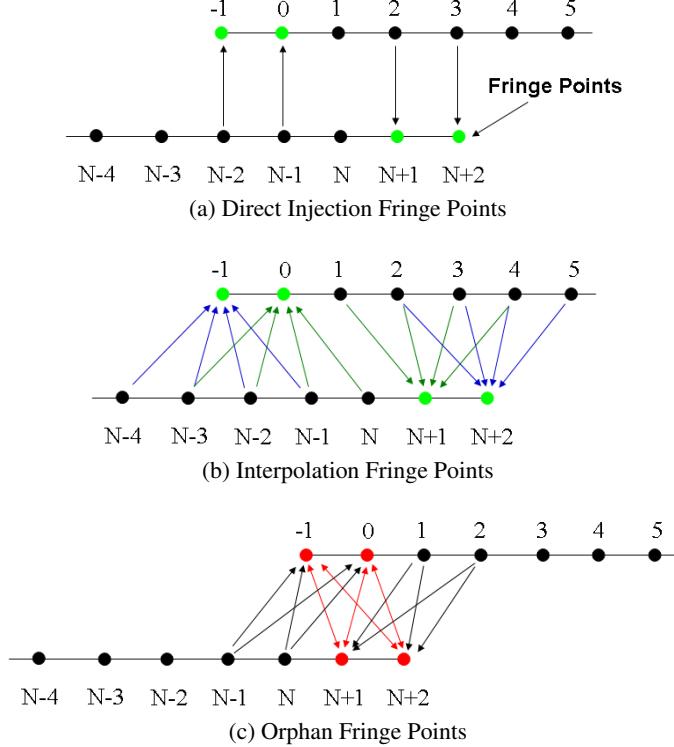


Figure 1.1: Artificial Boundary Communication

Benek et al.[38] also introduced the concept of hole cutting with the Chimera method [38]. Hole cutting excludes a region of grid points from the computational domain. Such regions are, for example, points located inside a closed body of an external flow field such as that shown in Fig. 1.2. Figure 1.2a shows an external flow field involving two cylinders in close proximity. All points in the background mesh located within the cylinder meshes need to be excluded from the computational domain. An initial hole, which assumes a 2nd-order accurate solver, is shown in Fig. 1.2b. However, this hole leaves regions where the available stencil is too small for a high-order accurate solver.[108, 44] The hole must be modified so that the minimum stencil of the high-order solver is satisfied throughout the entire mesh, as shown in Fig. 1.2c. Ensuring that the holes satisfy the minimum stencil size for a high-order code can be challenging for a more complicated three-dimensional configuration. [44]

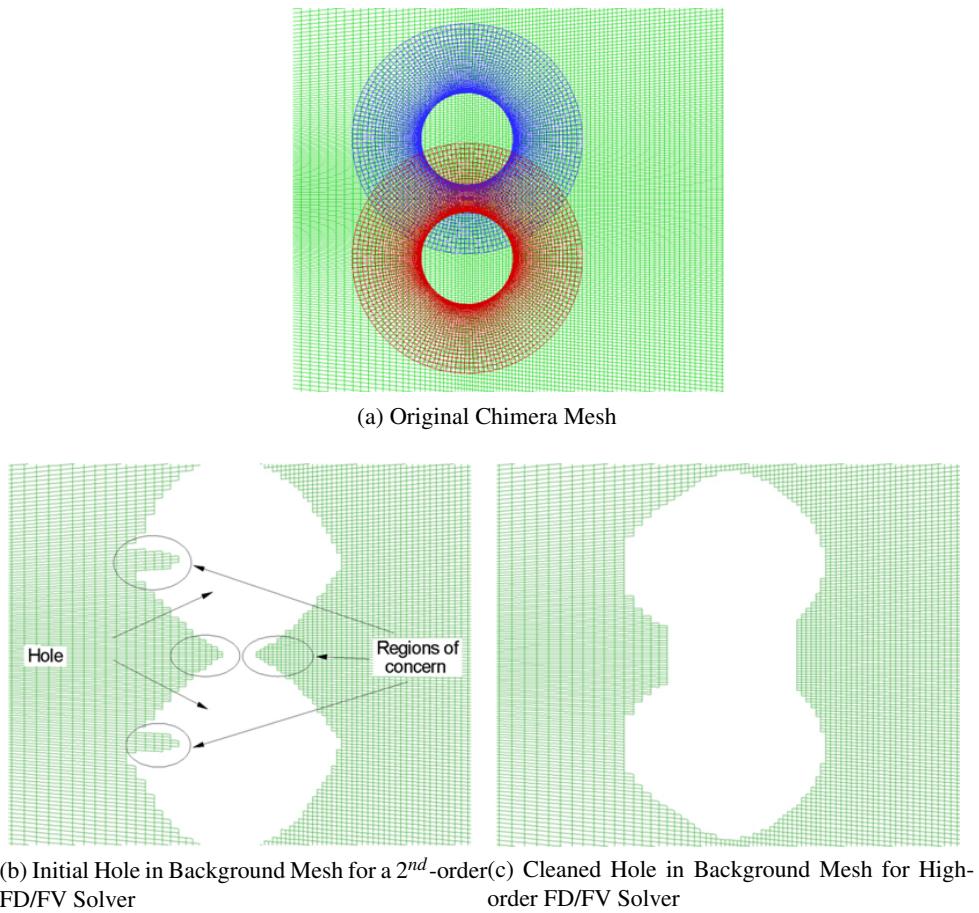


Figure 1.2: Large Stencil Hole Cutting Issues

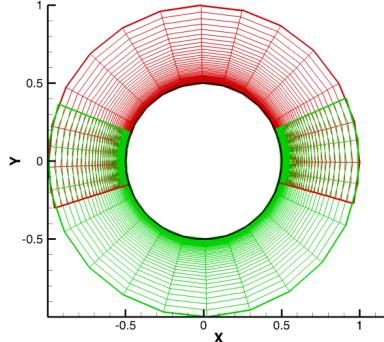
1.4 Dissertation Overview

The goal of this dissertation is the development of a Computational Fluid Dynamics code that is an improvement in accuracy and efficiency relative to the state of the art. The improvement in accuracy is achieved by using the Discontinuous Galerkin discretization method, and the improvement in efficiency is achieved by using a complete linearization of the discrete partial differential equations and by using computationally efficient C++ Template Expressions. A further improvement in computational efficiency is explored through the use of Quadrature-free integration where integrals are evaluated analytically using symbolic manipulation software.

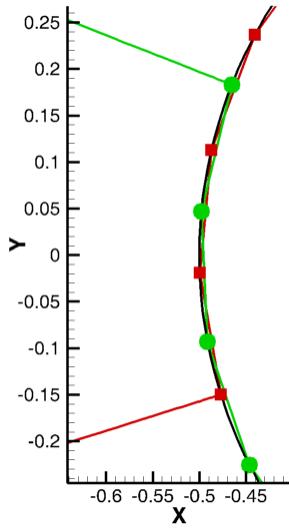
Flexibility in mesh generation is achieved through the development of a Chimera scheme suitable for

the Discontinuous Galerkin discretization. Using Chimera overset meshes with a Discontinuous Galerkin discretization solves many of the issues associated with Chimera schemes that use a Finite Volume or Finite Difference discretization. The predominant issue with Finite Volume and Finite Difference Chimera meshes is establishing the communication between overset grids and artificial boundaries. The large stencil of high-order Finite Difference and Finite Volume schemes require a larger number of fringe points in order to maintain the interior stencil on artificial boundaries. In addition, a high-order interpolation scheme, which also has a large stencil, is needed to interpolate values from the neighboring grid to the fringe points. Proper interpolation may not exist if grids do not overlap sufficiently. In this situation, the fringe points are often denoted as orphan points and the grid system must be adjusted, typically manually, until no orphan points exist in the grid system. Ensuring that no Orphan points exist in a grid system can be a tedious and time consuming task. The process of locating appropriate interpolation is further complicated when holes are cut in the grids, because the holes reduce the number of points in neighboring grids that can be used for interpolation. Hence, a robust hole cutting process also considers the interpolation when deciding which points to remove from the grid.

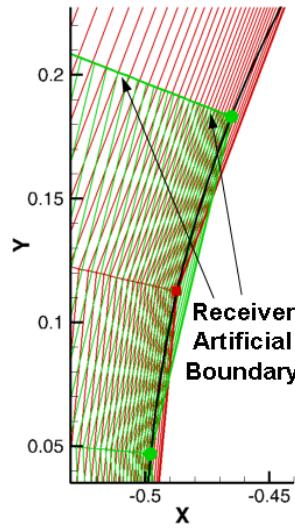
Both of the issues associated with orphan points and hole cutting with Finite Volume and Finite Difference Chimera schemes are resolved by the DG-Chimera scheme simply because the Discontinuous Galerkin method has a small stencil. No fringe points are required to facilitate the interior scheme on the artificial boundaries because the Discontinuous Galerkin method communicates with neighboring cells through cell face integrals. In addition, the DG-Chimera scheme does not require an interpolation scheme with a large stencil to transfer values from overset grids to the artificial boundaries. Instead, the cell local polynomial representation of the DG method serves as the interpolation scheme. Furthermore, as the volume of the overlapping region approaches zero, the DG-Chimera scheme naturally reduces to a zonal method[109]. As a result, there is no requirement in the extent of the overlap of the grids in the Chimera mesh with a DG-Chimera discretization; only that the grids overlap or abut. Therefore, a system of grids will not have orphans so long as no physical gaps between grids exists. This also simplifies the hole cutting process as holes can be cut without regard to the interpolation, and instead must only ensure that no gaps between the grids result from the holes that are cut.



(a) Circular Cylinder Chimera Mesh



(b) Surface Node Locations in the Leading Edge Overset Region



(c) Offset Receiver Node Due to Linear Cells

Figure 1.3: Example of Inaccurate Interpolation associated with Overset Meshes on Curved Geometries

The Chimera mesh shown in Fig. 1.3 was contrived to demonstrate issues with using chimera meshes with non-co-located artificial boundaries on curved geometries. The nodes on each mesh used to represent the cylinder in Fig. 1.3a are placed directly on the surface of the cylinder. Traditional Finite Difference and Finite Volume schemes rely on linear cells to represent geometric features. On concave surfaces, the face of a linear cell will lie within the actual geometry it is supposed to represent. As a result, since the nodes of the two meshes are not co-located, the surface nodes of the green mesh are offset from the face of the red mesh, and vice versa, as shown in Fig. 1.3b. In addition, the receiver node on the green mesh shown in Fig. 1.3c actually lies within the 4th cell off the surface of the red mesh, as Finite Difference and Finite Volume codes

typically require a large number of cells normal to the wall to capture a viscous boundary layer. This offset not only applies to the surface node, but also several layers of nodes away from the cylinder in the green mesh. Thus, an interpolation scheme that relies on the red donor cells encompassing the green nodes will transfer an incorrect boundary layer profile to the green mesh. Much effort has been put into developing the “Projection” features in PEGASUS5[43, 42] and SUGGAR++[110] to account for the offset between the two meshes.

Unlike Finite Difference and Finite Volume schemes, the DG method allows the use of curved cells. The curved cells give a better approximation to the curvature of a concave surface. As long as the polynomial order of the curved cells is adequate, the cell boundary will not fall within the surface of the geometry. In addition, the DG method requires significantly fewer cells than the Finite Difference and Finite Volume schemes to represent a viscous boundary layer. Thus, the DG method is capable of resolving the problems demonstrated in Fig. 1.3 without resorting to the projection techniques used in PEGASUS5 and SUGGAR++.

The specific contributions of this dissertation are as follows:

- The use of C++ Template Expressions to design modular code with the computational efficiency comparable to code written in the FORTRAN programming language.
- A set of libraries suitable for a Quadrature-free Discontinuous Galerkin discretization method.
- The development of a Discontinuous Galerkin Chimera scheme.
- A hole cutting process suitable for the DG-Chimera scheme that accounts for curved cells.
- A novel approach to the linearization of the equations associated with the artificial boundaries of the DG-Chimera scheme.

Details of the Discontinuous Galerkin discretization and the fluxes and flux Jacobians associated with the set of partial differential equations currently included in the code are given in Chapter 2. Strategies for the implementation of the code including a description of the Quadrature-free integration process is given in Chapter 3. Solutions to scalar partial differential equations as well as the Euler and Navier-Stokes equations are also presented in Chapter 3 as verification that the code is properly implemented. The Discontinuous Galerkin Chimera scheme is presented in detail in Chapter 4. This chapter also includes an outline of the implementation strategy for the code and details of implicit artificial boundaries where the equations

associated with the artificial boundaries are linearized. The implementation of the DG-Chimera scheme is verified with by solving a set of scalar partial differential equations as well as the Euler and Naver-Stokes equations. The use of curved elements to represent the geometry with grids adjacent to the body is also demonstrated in Chapter 4, as well as the improved computational efficiency obtained by using implicit artificial boundaries. Finally, conclusions and suggestions for future work are given in Chapter 5.

Chapter 2

Discontinuous Galerkin Method for Conservation Form Equations

The Discontinuous Galerkin method is a discretization scheme for solving partial differential equations written in weak conservation form. This chapter outlines the details of the Discontinuous Galerkin method in a general form followed by the specific partial differential equations included in the code. The code utilizes Newton's method, which relies on a linearization of the partial differential equations, to advance the solution for both linear and non-linear partial differential equations. Hence, the flux Jacobians of the of the partial differential equations are also presented; as well as the linearization of boundary conditions.

2.1 Discontinuous Galerkin Method

The Discontinuous Galerkin method used here is suitable for solving 1st and 2nd-order partial differential equations written in weak conservation form as

$$\frac{\partial Q}{\partial t} + \nabla \cdot \vec{F}(Q, \nabla Q) + S(Q, \nabla Q) = 0, \quad (2.1)$$

where Q is the dependent variable vector, $\vec{F}(Q, \nabla Q)$ is the flux, and $S(Q, \nabla Q)$ is a source term. Equation 2.1 is in strong conservation form when $S(Q, \nabla Q) = 0$. The computational domain is subdivided into non-overlapping cells where the dependent variable vector, Q , is expressed as an expansion in cell polynomial basis functions $\hat{\psi}$ of order N . Discontinuous Galerkin schemes are commonly formulated on cells

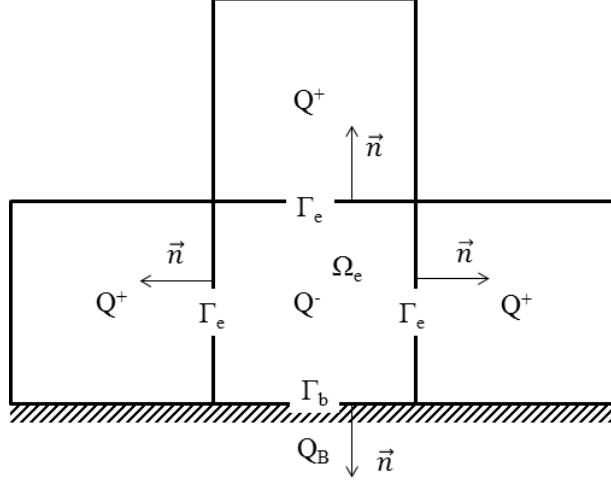


Figure 2.1: Graphical Explanation of the Notation for Cell Volume and Boundary Integrals

that triangles or quadrilaterals in two-dimensions, and tetrahedra or hexahedra in three-dimensions. The flexibility in mesh generation for a code based on a Discontinuous Galerkin discretization is often achieved with unstructured meshes based on triangles and tetrahedra. However, the present code instead relies on structured meshes composed of quadrilaterals and hexahedra combined with the Chimera overset method to achieve geometric flexibility. The notation used here to define the Discontinuous Galerkin discretization is illustrated in Fig. 2.1. The domain of an interior cell e is defined as Ω_e , and the cell boundaries interior to the computational domain of that cell are Γ_e . Cell boundaries located on the boundary of the computational domain are denoted as Γ_b . The cell polynomial expansion of the dependent vector Q on the interior of cell e is denoted with the superscript $(\cdot)^-$, while the superscript $(\cdot)^+$ denotes a polynomial on a cell that shares a common boundary with cell e .

For higher dimension, the polynomial expansion of the dependent variable vector is expressed as a tensor product of the one-dimensional polynomial since the code relies on quadrilateral and hexahedra elements. The polynomial expansions in one-, two-, and three- dimensions are

$$\begin{aligned}
Q(x, t) &= \sum_{i=0}^N Q_i(t) \hat{\psi}_i(x) && 1D \\
Q(x, y, t) &= \sum_{i=0}^N \sum_{j=0}^N Q_{ij}(t) \hat{\psi}_i(x) \hat{\psi}_j(y) && 2D \\
Q(x, y, z, t) &= \sum_{i=0}^N \sum_{j=0}^N \sum_{k=0}^N Q_{ijk}(t) \hat{\psi}_i(x) \hat{\psi}_j(y) \hat{\psi}_k(z) && 3D
\end{aligned} \tag{2.2}$$

where $Q_i(t)$, $Q_{ij}(t)$, and $Q_{ijk}(t)$ are the modes of the respective expansions. For the sake of generality, the polynomial expansion in any dimension is expressed as a single sum

$$Q(\vec{x}) = \sum_{i=0}^{N_m} Q_i \hat{\psi}_i(\vec{x}) \tag{2.3}$$

where $\hat{\psi}_{i(j,k,l)}(\vec{x}) \equiv \hat{\psi}_{jkl}(x, y, z) = \hat{\psi}_j(x) \hat{\psi}_k(y) \hat{\psi}_l(z)$, and the modes are implicitly assumed to be a function of time. Thus, for the same order of expansion in each dimension, let, $N_m \equiv N$, $N_m \equiv N^2$, and $N_m \equiv N^3$ for one-, two-, and three-dimensions , respectively.

A number of different polynomial functions can be chosen as basis functions for the polynomial expansion. A simple choice is monomials, i.e., $\hat{\psi}_i(x) = x^i$. However, this choice of basis function leads to an ill-conditioned discretization.[111] This problem can be solved by using an orthogonal set of basis functions. The orthogonal Legendre polynomials [112] are used here as they have a weighting function of unity. Equation 2.1 is put in weak integral form via multiplication by a column vector of test functions (which are also Legendre polynomials) $\hat{\psi} = [\hat{\psi}_i(\vec{x})]^T$, $\forall i \in [0, N_m]$, and applying Gauss's theorem to obtain

$$\int_{\Omega_e} \hat{\psi} \frac{\partial Q}{\partial t} d\Omega + \int_{\Gamma_e} \hat{\psi} \vec{F} \cdot \vec{n} d\Gamma - \int_{\Omega_e} \nabla \hat{\psi} \cdot \vec{F} d\Omega + \int_{\Omega_e} \hat{\psi} \vec{S} d\Omega = 0. \tag{2.4}$$

Here, the outward normal vector, \vec{n} , in the boundary integral is defined as a unit normal vector and the area of the surface integral is represented by $d\Gamma$. In addition, polynomials in the integrand of the boundary integral are evaluated on the cell boundary Γ_e . The boundary integral flux terms are computed from the dependent variables evaluated on both sides of the cell boundary. This provides the mechanism to couple the solution across cell boundaries. However, the advective fluxes and diffusive fluxes must be treated differently. Let $\vec{F}(Q, \nabla Q) = \vec{F}^a(Q) + \vec{F}^d(Q, \nabla Q)$, where $\vec{F}^a(Q)$ and $\vec{F}^d(Q, \nabla Q)$ are the advective and diffusive fluxes,

respectively. The specific details of the advective and diffusive fluxes are defined in Section 2.2 for the applied partial differential equations. The cell boundary integral for the advective fluxes in Eq. 2.4 is approximated with the average of the flux across the boundary and an upwinding term, ϕ , as

$$\int_{\Gamma_e} \hat{\psi} \vec{F}^a \cdot \vec{n} d\Gamma \Rightarrow \int_{\Gamma_e} \hat{\psi} \frac{1}{2} \left[(\vec{F}^a(Q^+) + \vec{F}^a(Q^-)) \cdot \vec{n} - \phi(Q^+, Q^-) \right] d\Gamma, \quad (2.5)$$

The volume integral of the advective fluxes does not require modification.

The discontinuities at cell interfaces have lead to complications when developing schemes for 2nd-order (e.g., viscous) partial differential equations. As a result, several different schemes exist in the literature to address this issue. Some of the more popular discretization techniques for viscous terms are the Bassi and Rebay 2 (BR2)[13, 24], Symmetric Interior Penalty Method (SIP)[88, 89], Local DG (LDG)[15], and Compact DG (CDG)[25] schemes.

The BR2 scheme is used here as it has a compact stencil that only relies on information from cells adjacent to cell e and has been demonstrated to obtain solutions with relatively small discontinuities on cell boundaries for smooth flow problems on coarser meshes[113]. The boundary and volume integrals associated with the diffusive fluxes are modified by adding a lifting operator, \vec{r} , in accord with the BR2 discretization scheme. This yields

$$\begin{aligned} & \int_{\Gamma_e} \hat{\psi} \vec{F}^d \cdot \vec{n} d\Gamma - \int_{\Omega_e} \nabla \psi \cdot \vec{F}^d d\Omega \Rightarrow \\ & \int_{\Gamma_e} \hat{\psi} \frac{1}{2} \left[\vec{F}^d(Q^+, \nabla Q^+ + \eta_e \vec{r}^+) + \vec{F}^d(Q^-, \nabla Q^- + \eta_e \vec{r}^-) \right] \cdot \vec{n} d\Gamma - \int_{\Omega_e} \nabla \hat{\psi} \cdot \vec{F}^d(Q^-, \nabla Q^- + \vec{R}) d\Omega \end{aligned} \quad (2.6)$$

where $\eta_e > 0$ is a stabilization coefficient taken to be number of faces of the cell.[16] The lifting operator is defined by the boundary integral over a single face, Γ_k , of the cell e as

$$\int_{\Omega_e} \hat{\psi} \vec{r}_k d\Omega = \int_{\Gamma_k} \hat{\psi} \frac{1}{2} (Q^+ - Q^-) \vec{n} d\Gamma. \quad (2.7)$$

Hence there are four lifting operators \vec{r}_k for a quadrilateral cell, and six lifting operator for a hexahedron cell. \vec{R} is the sum of the lifting operators \vec{r}_k on the cell e .

Boundary conditions are weakly imposed with the boundary integral of the advective and diffusive fluxes on the computational domain boundary Γ_b . The fluxes are computed using a boundary condition vector, Q_B , and boundary condition gradient vector, ∇Q_B . The boundary condition vector, Q_B , is a function of the interior dependent variable vector, Q^- , and imposed boundary condition values, Q_b , and the boundary condition gradient vector is a function of the interior variable vector gradient, ∇Q^- , and the imposed boundary gradient, ∇Q_b . The specific details of Q_B , Q_b , ∇Q_B , and ∇Q_b (defined in Section 2.2) depend on the type of boundary condition imposed. Specifically, the integral of the advective flux on the domain boundary is

$$\int_{\Gamma_b} \hat{\psi} \vec{F}^a \cdot \vec{n} d\Gamma \Rightarrow \int_{\Gamma_b} \hat{\psi} \vec{F}^a (Q_B(Q^-, Q_b)) \cdot \vec{n} d\Gamma. \quad (2.8)$$

where as the integral of the diffusive flux on the domain boundary is

$$\int_{\Gamma_b} \hat{\psi} \vec{F}^d \cdot \vec{n} d\Gamma \Rightarrow \int_{\Gamma_b} \hat{\psi} \vec{F}^d (Q_B(Q^-, Q_b), \nabla Q_B(\nabla Q^-, \nabla Q_b) + \eta_e \vec{r}_b) \cdot \vec{n} d\Gamma. \quad (2.9)$$

For Dirichlet boundary conditions, the lifting operator, \vec{r}_b , on the boundaries Γ_b is computed by replacing the external conservative variables, Q^+ , in Eq. 2.7 with the boundary condition variable vector, Q_B . For Neumann boundary conditions, \vec{r}_b is set to zero. Hence, the equation for \vec{r}_b is

$$\int_{\Omega_e} \hat{\psi} \vec{r}_b d\Omega = \begin{cases} \int_{\Gamma_b} \hat{\psi} (Q_B(Q^-, Q_b) - Q^-) \vec{n} d\Gamma & \text{Dirichlet} \\ 0 & \text{Neumann} \end{cases}, \quad (2.10)$$

and the boundary gradient vector is

$$\nabla Q_B(\nabla Q^-, \nabla Q_b) = \begin{cases} \nabla Q^- & \text{Dirichlet} \\ \nabla Q_b & \text{Neumann} \end{cases}. \quad (2.11)$$

The discrete formulation of Eq. 2.1 is

$$\begin{aligned}
& \int_{\Omega_e} \hat{\psi} \frac{\partial Q}{\partial t} d\Omega + \mathcal{R}(Q) = \\
& \quad \int_{\Omega_e} \hat{\psi} \frac{\partial Q}{\partial t} d\Omega + \\
& \int_{\Gamma_e} \hat{\psi} \frac{1}{2} \left[(\vec{F}^a(Q^+) + \vec{F}^a(Q^-)) \cdot \vec{n} - \phi(Q^+, Q^-) \right] d\Gamma - \int_{\Omega_e} \nabla \hat{\psi} \cdot \vec{F}^a(Q^-) d\Omega + \\
& \quad \int_{\Gamma_b} \hat{\psi} \vec{F}^a(Q_B(Q^-, Q_b)) \cdot \vec{n} d\Gamma + \\
& \int_{\Gamma_e} \hat{\psi} \frac{1}{2} \left[\vec{F}^d(Q^+, \nabla Q^+ + \eta_e \vec{r}^+) + \vec{F}^d(Q^-, \nabla Q^- + \eta_e \vec{r}^-) \right] \cdot \vec{n} d\Gamma - \\
& \quad \int_{\Omega_e} \nabla \hat{\psi} \cdot \vec{F}^d(Q^-, \nabla Q^- + \vec{R}) d\Omega + \\
& \int_{\Gamma_b} \hat{\psi} \vec{F}^d(Q_B(Q^-, Q_b), \nabla Q_B(\nabla Q^-, \nabla Q_b) + \eta_e \vec{r}_b) \cdot \vec{n} d\Gamma + \\
& \int_{\Omega_e} \hat{\psi} S(Q^-, \nabla Q^- + \vec{R}) d\Omega = 0 \quad (2.12)
\end{aligned}$$

where $\mathcal{R}(Q)$ is the sum of all the spatial integrals. On interior cells the integrals on the computational domain boundary, Γ_b , are zero.

2.1.1 Newton's Method

Newton's method is an effective method for finding solutions that satisfy Eq. 2.12[114, 115, 20]. However, a complete linearization, including boundary conditions, of Eq. 2.12 is required to obtain a robust method with quadratic convergence properties. The system of linear equations

$$\frac{\partial \mathcal{R}(Q^n)}{\partial Q} \Delta Q^n = -\mathcal{R}(Q^n), \quad (2.13)$$

where Q is a vector of all cell dependent variables in the computational domain, is used to compute ΔQ^n , which is the update vector used to update the entire solution vector as $Q^{n+1} = Q^n + \Delta Q^n$ for each Newton iteration n . ΔQ is the global change in Q , which consists of the update vector of a given cell increment, ΔQ^- , and its neighbors cells, ΔQ^+ . Later, ΔQ^\pm is used to clarify the origin of the Jacobian entries of $\frac{\partial \mathcal{R}(Q^n)}{\partial Q}$. For a structured mesh, the Jacobian matrix $\frac{\partial \mathcal{R}(Q^n)}{\partial Q}$ forms a block tri-, block penta-, and block hepta-diagonal matrix for one-, two-, and three-dimensional problems respectively. The cell increment ΔQ^- is multiplied with the diagonal of the matrix $\frac{\partial \mathcal{R}(Q^n)}{\partial Q}$, which is obtained from the linearization of $\mathcal{R}(Q^n)$ with respect

to the cell polynomial expansion Q^- . The off-diagonal block matrices in $\frac{\partial \mathcal{R}(Q^n)}{\partial Q}$, which are multiplied by increments ΔQ^+ , are obtained from the linearization of $\mathcal{R}(Q^n)$ with respect to cell neighboring polynomial expansions Q^+ . The linearization of the fluxes of each term in Eq. 2.12 are as follows:

- Time Integral

$$\frac{\partial \Delta Q^-}{\partial t} \quad (2.14)$$

- Advection Boundary Integral

$$\frac{1}{2} \left[\left(\frac{\partial \vec{F}^a(Q^+)}{\partial Q^+} \Delta Q^+ + \frac{\partial \vec{F}^a(Q^-)}{\partial Q^-} \Delta Q^- \right) \cdot \vec{n} - \left(\frac{\partial \phi(Q^+, Q^-)}{\partial Q^+} \Delta Q^+ + \frac{\partial \phi(Q^+, Q^-)}{\partial Q^-} \Delta Q^- \right) \right] \quad (2.15)$$

- Advection Volume Integral

$$\frac{\partial \vec{F}^a(Q^-)}{\partial Q^-} \Delta Q^- \quad (2.16)$$

- Advection Boundary Condition

$$\frac{\partial \vec{F}^a(Q_B(Q^-, Q_b))}{\partial Q_B} \frac{\partial Q_B(Q^-, Q_b)}{\partial Q^-} \Delta Q^- \quad (2.17)$$

- Diffusion Boundary Integral

$$\begin{aligned} & \frac{\partial \vec{F}^d(Q^+, \nabla Q^+ + \eta_e \vec{r}^+)}{\partial Q^+} \Delta Q^+ + \\ & \frac{\partial \vec{F}^d(Q^+, \nabla Q^+ + \eta_e \vec{r}^+)}{\partial \nabla Q^+} \left(\frac{\partial \nabla Q^+}{\partial Q^+} \Delta Q^+ + \eta_e \frac{\partial \vec{r}^+}{\partial Q^+} \Delta Q^+ + \eta_e \frac{\partial \vec{r}^+}{\partial Q^-} \Delta Q^- \right) + \\ & \frac{\partial \vec{F}^d(Q^-, \nabla Q^- + \eta_e \vec{r}^-)}{\partial Q^-} \Delta Q^- + \\ & \frac{\partial \vec{F}^d(Q^-, \nabla Q^- + \eta_e \vec{r}^-)}{\partial \nabla Q^-} \left(\frac{\partial \nabla Q^-}{\partial Q^-} \Delta Q^- + \eta_e \frac{\partial \vec{r}^-}{\partial Q^+} \Delta Q^+ + \eta_e \frac{\partial \vec{r}^-}{\partial Q^-} \Delta Q^- \right) \end{aligned} \quad (2.18)$$

- Diffusion Volume Integral

$$\begin{aligned} & \frac{\partial \vec{F}^d(Q^-, \nabla Q^- + \eta_e \vec{r}^-)}{\partial Q^-} \Delta Q^- + \\ & \frac{\partial \vec{F}^d(Q^-, \nabla Q^- + \vec{R})}{\partial \nabla Q^-} \left(\frac{\partial \nabla Q^-}{\partial Q^-} \Delta Q^- + \frac{\partial \vec{R}}{\partial Q^+} \Delta Q^+ + \frac{\partial \vec{R}}{\partial Q^-} \Delta Q^- \right) \end{aligned} \quad (2.19)$$

- Diffusion Boundary Condition

$$\begin{aligned} & \frac{\partial \vec{F}^d(Q_B, \nabla Q_B + \eta_e \vec{r}_b)}{\partial Q_B} \frac{\partial Q_B(Q^-, Q_b)}{\partial Q^-} \Delta Q^- + \\ & \frac{\partial \vec{F}^d(Q_B, \nabla Q_B + \eta_e \vec{r}_b)}{\partial \nabla Q_B} \left(\frac{\partial \nabla Q_B(\nabla Q^-, \nabla Q_b)}{\partial \nabla Q^-} \frac{\partial \nabla Q^-}{\partial Q^-} \Delta Q^- + \eta_e \frac{\partial \vec{r}_b(Q^-, Q_B)}{\partial Q^-} \Delta Q^- + \right. \\ & \quad \left. \eta_e \frac{\partial \vec{r}_b(Q^-, Q_B)}{\partial Q_B} \frac{\partial Q_B(Q^-, Q_b)}{\partial Q^-} \Delta Q^- \right) \end{aligned} \quad (2.20)$$

- Source Term

$$\begin{aligned} & \frac{\partial S(Q^-, \nabla Q^- + \vec{R})}{\partial Q^-} \Delta Q^- + \\ & \frac{\partial S(Q^-, \nabla Q^- + \vec{R})}{\partial \nabla Q^-} \left(\frac{\partial \nabla Q^-}{\partial Q^-} \Delta Q^- + \frac{\partial \vec{R}}{\partial Q^-} \Delta Q^- + \frac{\partial \vec{R}}{\partial Q^+} \Delta Q^+ \right) \end{aligned} \quad (2.21)$$

The Jacobians associated with the boundary conditions are written as a matrix product using the chain rule to simplify their derivation. Consider the advective boundary condition Jacobian in Eq. 2.17. Deriving the Jacobian of the convective flux evaluated with the boundary condition conservative variable vector, $Q_B(Q^-, Q_b)$, with respect to the interior conservative variable vector, Q^- , i. e.,

$$\frac{\partial \vec{F}^c(Q_B(Q^-, Q_b))}{\partial Q^-} \Delta Q^-, \quad (2.22)$$

can result in a complicated expression even when symbolic manipulation software is utilized. In comparison, the Jacobian

$$\frac{\partial \vec{F}^c(Q_B(Q^-, Q_b))}{\partial Q^B}, \quad (2.23)$$

is the Jacobian required for the interior scheme and the Jacobian

$$\frac{\partial Q_B(Q^-, Q_b)}{\partial Q^-}, \quad (2.24)$$

is always significantly simpler to derive in comparison with Eq. 2.22. The Jacobian in Eq. 2.24 is also used in the diffusion boundary condition Jacobian in Eq. 2.20.

2.2 Conservation Form Equations

This section outlines the set of partial differential equations, and their Jacobians, currently included in the code. The generality of the code allows for the inclusion of any number of partial differential equations written in conservation form. A set of scalar equations with increasing complexity are included for verification purposes, but the compressible Navier-Stokes equations suitable for modeling fluid flow are of primary interest in this work.

2.2.1 Scalar Equations

The various scalar equations are used to verify that aspects of the code are properly implemented.

2.2.1.1 Poisson's Equation

Poisson's equation is a diffusion equation that can model phenomena such as temperature distribution in a solid. It is given as

$$\nabla \cdot \vec{F}^d(u, \nabla u) + S(Q, \nabla Q) = 0. \quad (2.25)$$

This equation is used in this case to verify proper implementation of the diffusive fluxes in the code. The diffusive flux is given by

$$\vec{F}^d(u, \nabla u) = -\mu(u) \nabla u, \quad (2.26)$$

and the Jacobians of the flux are

$$\begin{aligned} \frac{\partial \vec{F}^d(u, \nabla u)}{\partial u} &= -\frac{\partial \mu(u)}{\partial u} \nabla u, \\ \frac{\partial \vec{F}^d(u, \nabla u)}{\partial \nabla u} &= -\mu(u) [I]. \end{aligned} \quad (2.27)$$

2.2.1.2 Local Lax-Friedrichs

The scalar equations in the following two sections include linear and non-linear advection fluxes with the diffusive flux of Poisson's equation. In each equation, the advective fluxes are upwinded using the dissipation flux of the Local Lax-Friedrichs scheme[12]

$$\phi(u^+, u^-) = \max \left(\left| \frac{\partial \vec{F}^a(u^+)}{\partial u} \right|, \left| \frac{\partial \vec{F}^a(u^-)}{\partial u} \right| \right) (u^+ - u^-), \quad (2.28)$$

which relies on the Jacobian of the advective flux. The Jacobians for the Local Lax-Friedrichs flux are

$$\begin{aligned} \frac{\partial \phi(u^+, u^-)}{\partial u^+} &= \begin{cases} \text{sign} \left(\frac{\partial \vec{F}^a(u^+)}{\partial u^+} \right) \frac{\partial^2 \vec{F}^a(u^+)}{\partial (u^+)^2} (u^+ - u^-) + \left| \frac{\partial \vec{F}^a(u^+)}{\partial u^+} \right| & \text{if } \frac{\partial \vec{F}^a(u^+)}{\partial u^+} > \frac{\partial \vec{F}^a(u^-)}{\partial u^-} \\ 0 & \text{otherwise} \end{cases} \\ \frac{\partial \phi(u^+, u^-)}{\partial u^-} &= \begin{cases} \text{sign} \left(\frac{\partial \vec{F}^a(u^-)}{\partial u^-} \right) \frac{\partial^2 \vec{F}^a(u^-)}{\partial (u^-)^2} (u^+ - u^-) - \left| \frac{\partial \vec{F}^a(u^-)}{\partial u^-} \right| & \text{if } \frac{\partial \vec{F}^a(u^-)}{\partial u^-} > \frac{\partial \vec{F}^a(u^+)}{\partial u^+} \\ 0 & \text{otherwise} \end{cases}, \quad (2.29) \end{aligned}$$

and require the Jacobian and the Hessian of the advective flux.

2.2.1.3 Linear Advection and Diffusion Equation

The steady linear advection and diffusion equation is

$$\nabla \cdot \vec{F}^a(u) + \nabla \cdot \vec{F}^d(u, \nabla u) + S(Q, \nabla Q) = 0, \quad (2.30)$$

where the advective flux is

$$\vec{F}^a(u) = \vec{c}u, \quad \text{where } \vec{c} = \begin{bmatrix} c_x, & c_y, & c_z \end{bmatrix}, \quad (2.31)$$

and the diffusive flux is given in Eq. 2.26. The Jacobian and Hessian of the advective flux are

$$\begin{aligned} \frac{\partial \vec{F}^a(u)}{\partial u} &= \vec{c} \\ \frac{\partial^2 \vec{F}^a(u)}{\partial u^2} &= 0, \quad (2.32) \end{aligned}$$

and the diffusive Jacobians are given in Eq. 2.27.

2.2.1.4 Burger's Equation

The steady multidimensional nonlinear Burger's equation is

$$\nabla \cdot \vec{F}^a(u) + \nabla \cdot \vec{F}^d(u, \nabla u) = 0, \quad (2.33)$$

where the advective flux is

$$\vec{F}^a(u) = \vec{c} \frac{1}{2} u^2, \quad \text{where } \vec{c} = \begin{bmatrix} c_x, & c_y, & c_z \end{bmatrix}, \quad (2.34)$$

and the diffusive flux is given in Eq. 2.26. The Jacobian and Hessian of the nonlinear advective flux are

$$\begin{aligned} \frac{\partial \vec{F}^a(u)}{\partial u} &= \vec{c} u \\ \frac{\partial^2 \vec{F}^a(u)}{\partial u^2} &= \vec{c}, \end{aligned} \quad (2.35)$$

and the diffusive Jacobians are given in Eq. 2.27. Burger's equation is useful for verification as it resembles the fluid flow equations.

2.2.2 Scalar Boundary Conditions

Only two types of boundary conditions are used for the scalar equations; Dirichlet and Neumann. The Dirichlet boundary conditions impose the value of the scalar and the Neumann boundary condition imposes the gradient of the scalar.

2.2.2.1 Dirichlet

The value and gradient of a Dirichlet boundary condition are give by

$$\begin{aligned} u_B(u^-, u_b) &= u_b \\ \nabla u_B(\nabla u^-, \nabla u_b) &= \nabla u^-. \end{aligned} \quad (2.36)$$

The Jacobian of the boundary value and gradient with respect to the interior value and gradient are

$$\begin{aligned}\frac{\partial u_B(u^-, u_b)}{\partial u^-} &= 0 \\ \frac{\nabla u_B(\nabla u^-, \nabla u_b)}{\nabla u^-} &= [I].\end{aligned}\tag{2.37}$$

2.2.2.2 Neumann

For a Neumann boundary condition the value and gradient of the boundary are

$$\begin{aligned}u_B(u^-, u_b) &= u^- \\ \nabla u_B(\nabla u^-, \nabla u_b) &= \nabla u_b,\end{aligned}\tag{2.38}$$

and the Jacobians with respect to the interior value and gradient are

$$\begin{aligned}\frac{\partial u_B(u^-, u_b)}{\partial u^-} &= 1 \\ \frac{\nabla u_B(\nabla u^-, \nabla u_b)}{\nabla u^-} &= 0.\end{aligned}\tag{2.39}$$

2.2.3 Fluid Flow Equations

The equations for modeling fluid flow are derived based on the principals of conservation of mass, Newton's Second Law, and the First Law of Thermodynamics. Applying the principal of conservation of mass to a fluid flow results in an equation known as the continuity equation. The momentum equation is a vector equation that results from applying Newton's Second Law to a fluid flow. The application of the First Law of Thermodynamics to fluid flow is known as the energy equation. An additional relationship between the fluid flow properties and equations relating fluid viscosity and thermal conductivity with thermodynamics properties are required in order to close the system of equations derived from the conservation principals. Equations that relate the thermodynamic properties are known as equations of state, they relate the thermodynamic variables pressure density and temperature. Equations relating fluid viscosity and thermal conductivity are related to the thermodynamic variables using kinetic theory.

This set of conservation equations are commonly referred to as the Navier-Stokes equations. A subset of the Navier-Stokes equations that neglects the influence of viscous forces is known as the Euler equations.

2.2.3.1 Navier-Stokes Equations

The Navier-Stokes equations are presented here in non-dimensional form where the reference quantities are

$$t = \frac{\hat{t}\hat{V}_\infty}{\hat{L}}, \quad \vec{x} = \frac{\hat{x}}{\hat{L}}, \quad \vec{V} = \frac{\hat{\vec{V}}}{\hat{V}_\infty}, \quad \rho = \frac{\hat{\rho}}{\hat{\rho}_\infty}, \quad E, p = \frac{\hat{E}, \hat{p}}{\hat{\rho}_\infty \hat{V}_\infty^2}, \quad e_i = \frac{\hat{e}_i}{\hat{V}_\infty^2}, \quad \mu, \lambda_1 = \frac{\hat{\mu}, \hat{\lambda}_1}{\hat{\mu}_\infty}, \quad \kappa = \frac{\hat{\kappa}}{\hat{\kappa}_\infty}, \quad (2.40)$$

and $\hat{\cdot}$ indicates a dimensional quantity. The non-dimensionalization does not modify the inviscid fluxes, but it does introduce reference quantities into the viscous terms. The non-dimensional Navier-Stokes equations are written in divergence, or conservative, form as

$$\frac{\partial Q}{\partial t} + \nabla \cdot \vec{F}^a(Q) + \vec{F}^d(Q, \nabla Q) = 0 \quad (2.41)$$

where the conservative variables are $Q = \begin{bmatrix} \rho, & \rho u, & \rho v, & \rho w, & \rho E \end{bmatrix}^T$, the inviscid fluxes are

$$\vec{F}^a(Q) = \begin{bmatrix} F_x^c, & F_y^c, & F_z^c \end{bmatrix} = \begin{bmatrix} \begin{pmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ \rho uw \\ \rho uH \end{pmatrix}, & \begin{pmatrix} \rho v \\ \rho vu \\ \rho v^2 + p \\ \rho vw \\ \rho vH \end{pmatrix}, & \begin{pmatrix} \rho w \\ \rho wu \\ \rho wv \\ \rho w^2 + p \\ \rho wH \end{pmatrix} \end{bmatrix}, \quad (2.42)$$

where $H = \frac{\rho E + p}{\rho}$ is the total enthalpy,

$$p = (\gamma - 1) \left(\rho E - \frac{1}{2} \rho \vec{V}^2 \right) \quad (2.43)$$

is the static pressure, and $\vec{V} = \begin{bmatrix} u, & v, & w \end{bmatrix}$ is the velocity vector. The viscous fluxes are

$$\vec{F}^d(Q, \nabla Q) = -\frac{1}{Re} \begin{bmatrix} F_x^d, & F_y^d, & F_z^d \end{bmatrix} = -\frac{1}{Re} \left[\begin{pmatrix} 0 \\ \tau_{xx} \\ \tau_{xy} \\ \tau_{xz} \\ \beta_x \end{pmatrix}, \begin{pmatrix} 0 \\ \tau_{yx} \\ \tau_{yy} \\ \tau_{yz} \\ \beta_y \end{pmatrix}, \begin{pmatrix} 0 \\ \tau_{zx} \\ \tau_{zy} \\ \tau_{zz} \\ \beta_z \end{pmatrix} \right], \quad (2.44)$$

where $\tau_{x^i x^j}$ are the stress tensor components, and β_{x^i} are the work terms defined by

$$\begin{aligned} \tau_{xx} &= 2\mu u_x + \lambda_1 \nabla \cdot \vec{V} & \tau_{xy} &= \tau_{yx} = \mu (u_y + v_x) \\ \tau_{yy} &= 2\mu v_y + \lambda_1 \nabla \cdot \vec{V} & \tau_{xz} &= \tau_{zx} = \mu (u_z + w_x) \\ \tau_{zz} &= 2\mu w_z + \lambda_1 \nabla \cdot \vec{V} & \tau_{yz} &= \tau_{zy} = \mu (v_z + w_y) \\ \beta_{x^i} &= \frac{\mu}{M_\infty^2 Pr_\infty (\gamma - 1)} \frac{\partial T}{\partial x^i} + u \tau_{x^i x} + v \tau_{x^i y} + w \tau_{x^i z} \end{aligned} \quad (2.45)$$

where the static temperature, T , is obtained from the ideal gas law

$$T = \gamma M_\infty^2 \frac{p}{\rho} \quad (2.46)$$

Sutherland's law is used to model the viscosity coefficient μ

$$\mu(T) = T^{3/2} \left(\frac{1 + S_1}{T + S_1} \right) \quad (2.47)$$

where $S_1 = \hat{S}_1 / \hat{T}_\infty$, $\hat{S}_1 = 198.6^\circ$ R is Sutherland's law coefficient, and λ_1 is the 2nd viscosity coefficient and is taken to be $\lambda_1 = -\frac{2}{3}\mu$ in accordance with the Stokes hypothesis. All calculations in this dissertation assume air and use $Pr_\infty = 0.72$ and $\hat{T}_\infty = 522.6^\circ$ R, which implies $S_1 = 0.38$. The gradients in velocity are expressed in terms of Q and ∇Q using the chain rule as

$$\begin{aligned}
\nabla u &= \frac{\partial u}{\partial \rho} \nabla \rho + \frac{\partial u}{\partial \rho u} \nabla (\rho u), \\
\nabla v &= \frac{\partial v}{\partial \rho} \nabla \rho + \frac{\partial v}{\partial \rho v} \nabla (\rho v), \\
\nabla w &= \frac{\partial w}{\partial \rho} \nabla \rho + \frac{\partial w}{\partial \rho w} \nabla (\rho w).
\end{aligned} \tag{2.48}$$

Note that the velocity vector is a function of the conservative variable vector, i.e.

$$\vec{V} = \frac{\rho \vec{v}}{\rho} \tag{2.49}$$

Hence, the velocity Jacobian required by the velocity gradients is

$$\frac{\partial \vec{V}}{\partial Q} \Rightarrow \begin{cases} \frac{\partial \vec{V}}{\partial \rho} = \frac{\partial}{\partial \rho} \left(\frac{\rho \vec{v}}{\rho} \right) = -\frac{\rho \vec{v}}{\rho^2} \\ \frac{\partial v_i}{\partial (\rho v_j)} = \frac{\partial}{\partial (\rho v_j)} \left(\frac{\rho v_i}{\rho} \right) = \begin{cases} \frac{1}{\rho} & i = j \\ 0 & i \neq j \end{cases}, \\ \frac{\partial \vec{V}}{\partial (\rho E)} = 0 \end{cases} \tag{2.50}$$

Similarly, the temperature gradient is expressed in terms of Q and ∇Q using the chain rule as

$$\nabla T = \frac{\partial T}{\partial \rho} \nabla \rho + \frac{\partial T}{\partial (\rho u)} \nabla (\rho u) + \frac{\partial T}{\partial (\rho v)} \nabla (\rho v) + \frac{\partial T}{\partial (\rho w)} \nabla (\rho w) + \frac{\partial T}{\partial (\rho E)} \nabla (\rho E), \tag{2.51}$$

where the temperature Jacobian is

$$\frac{\partial T}{\partial Q} \Rightarrow \begin{cases} \frac{\partial T}{\partial \rho} = \gamma M_\infty^2 \left(-\frac{p}{\rho^2} + \frac{1}{\rho} \frac{\partial p}{\partial \rho} \right) \\ \frac{\partial T}{\partial \rho \vec{V}} = \gamma M_\infty^2 \frac{1}{\rho} \frac{\partial p}{\partial (\rho \vec{V})} \\ \frac{\partial T}{\partial \rho E} = \gamma M_\infty^2 \frac{1}{\rho} \frac{\partial p}{\partial (\rho E)} \end{cases}. \tag{2.52}$$

The temperature Jacobian requires the pressure Jacobian

$$\frac{\partial p}{\partial Q} \Rightarrow \begin{cases} \frac{\partial p}{\partial \rho} = -(\gamma - 1) \frac{\partial}{\partial \rho} \left(\frac{1}{2} \rho \vec{V}^2 \right) \\ \frac{\partial p}{\partial \rho \vec{V}} = -(\gamma - 1) \frac{\partial}{\partial (\rho \vec{V})} \left(\frac{1}{2} \rho \vec{V}^2 \right) \\ \frac{\partial p}{\partial \rho E} = (\gamma - 1) \end{cases}, \quad (2.53)$$

which requires the kinetic energy Jacobian

$$\frac{\partial}{\partial Q} \left(\frac{1}{2} \rho \vec{V}^2 \right) \Rightarrow \begin{cases} \frac{\partial}{\partial \rho} \left(\frac{1}{2} \rho \vec{V}^2 \right) = -\frac{1}{2} \vec{V}^2 \\ \frac{\partial}{\partial (\rho \vec{V})} \left(\frac{1}{2} \rho \vec{V}^2 \right) = -\vec{V} \\ \frac{\partial}{\partial (\rho E)} \left(\frac{1}{2} \rho \vec{V}^2 \right) = 0 \end{cases}. \quad (2.54)$$

Deriving Jacobians for systems of equations can be complex, and is often performed with a brute force approach with symbolic manipulation software.[114] The resulting code generated with the symbolic manipulation software is often complex and difficult to verify as correct. The derivation of the Jacobians is significantly simplified by repeatedly applying the chain rule instead of applying the symbolic manipulator directly to the flux. Using the chain rule breaks the derivation into smaller more manageable steps, and common reoccurring terms become evident. The intermediate terms in the Jacobians are cell quantities that do not need to be stored. This approach of repeatedly applying the chain rule was employed by Fidkwoiski as part of the development of the XFLOW discontinuous Galerkin solver.[116]

Applying the chain rule, the Jacobian with respect to the conservative variable vector for the advective fluxes are

$$\begin{aligned}
\frac{\partial F_x^a(Q)}{\partial Q} &= \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ \rho u \frac{\partial u}{\partial \rho} + \frac{\partial p}{\partial \rho} & u + \rho u \frac{\partial u}{\partial(\rho u)} + \frac{\partial p}{\partial(\rho u)} & \frac{\partial p}{\partial(\rho v)} & \frac{\partial p}{\partial(\rho w)} & \frac{\partial p}{\partial(\rho E)} \\ \rho u \frac{\partial v}{\partial \rho} & v & u & 0 & 0 \\ \rho u \frac{\partial w}{\partial \rho} & w & 0 & u & 0 \\ \rho u \frac{\partial H}{\partial \rho} & H + \rho u \frac{\partial H}{\partial(\rho u)} & \rho u \frac{\partial H}{\partial(\rho v)} & \rho u \frac{\partial H}{\partial(\rho w)} & \rho u \frac{\partial H}{\partial(\rho E)} \end{pmatrix}, \\
\frac{\partial F_y^a(Q)}{\partial Q} &= \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ \rho v \frac{\partial u}{\partial \rho} & v & u & 0 & 0 \\ \rho v \frac{\partial v}{\partial \rho} + \frac{\partial p}{\partial \rho} & \frac{\partial p}{\partial(\rho u)} & v + \rho v \frac{\partial v}{\partial(\rho v)} + \frac{\partial p}{\partial(\rho v)} & \frac{\partial p}{\partial(\rho w)} & \frac{\partial p}{\partial(\rho E)} \\ \rho v \frac{\partial w}{\partial \rho} & 0 & w & v & 0 \\ \rho v \frac{\partial H}{\partial \rho} & \rho v \frac{\partial H}{\partial(\rho u)} & H + \rho v \frac{\partial H}{\partial(\rho v)} & \rho v \frac{\partial H}{\partial(\rho w)} & \rho v \frac{\partial H}{\partial(\rho E)} \end{pmatrix}, \\
\frac{\partial F_z^a(Q)}{\partial Q} &= \begin{pmatrix} 0 & 0 & 0 & 1 & 0 \\ \rho w \frac{\partial u}{\partial \rho} & w & 0 & u & 0 \\ \rho w \frac{\partial v}{\partial \rho} & 0 & w & v & 0 \\ \rho w \frac{\partial w}{\partial \rho} + \frac{\partial p}{\partial \rho} & \frac{\partial p}{\partial(\rho u)} & \frac{\partial p}{\partial(\rho v)} & w + \rho w \frac{\partial w}{\partial(\rho w)} + \frac{\partial p}{\partial(\rho w)} & \frac{\partial p}{\partial(\rho E)} \\ \rho w \frac{\partial H}{\partial \rho} & \rho w \frac{\partial H}{\partial(\rho u)} & \rho w \frac{\partial H}{\partial(\rho v)} & H + \rho w \frac{\partial H}{\partial(\rho w)} & \rho w \frac{\partial H}{\partial(\rho E)} \end{pmatrix}. \quad (2.55)
\end{aligned}$$

The inviscid flux Jacobians require the velocity vector and pressure Jacobians given in Eqs. 2.50 and 2.53 as well as the total enthalpy Jacobian

$$\frac{\partial H}{\partial Q} \Rightarrow \begin{cases} \frac{\partial H}{\partial \rho} = \rho E \frac{\partial}{\partial \rho} \left(\frac{1}{\rho} \right) + \frac{\partial}{\partial \rho} \left(\frac{1}{\rho} \right) p + \frac{1}{\rho} \frac{\partial p}{\partial \rho} = -\frac{(\rho E + p)}{\rho^2} + \frac{1}{\rho} \frac{\partial p}{\partial \rho} = -\frac{H}{\rho} + \frac{1}{\rho} \frac{\partial H}{\partial \rho} \\ \frac{\partial H}{\partial(\rho V)} = \frac{1}{\rho} \frac{\partial p}{\partial(\rho V)} \\ \frac{\partial H}{\partial(\rho E)} = \frac{1}{\rho} + \frac{1}{\rho} \frac{\partial p}{\partial(\rho E)} = \frac{\gamma}{\rho} \end{cases}. \quad (2.56)$$

In the case of the advective flux Jacobians in Eq. 2.55, the velocity vector, pressure, and total enthalpy Jacobians could be substituted into Eq. 2.55 and simplified. However, this is not generally a simple process for all Jacobians. The Jacobians for the viscous flux are significantly more complex relative to the inviscid flux Jacobians, and cannot be greatly simplified due to the large number of terms in the Jacobians. Hence, the viscous flux Jacobians are simpler to express in terms of repeated application of the chain rule.

The viscous flux Jacobian with respect to the conservative variable vector is

$$\frac{\partial F_i^d(Q)}{\partial Q} = \begin{pmatrix} 0 \\ \frac{\partial \tau_{x^i x}}{\partial Q} \\ \frac{\partial \tau_{x^i y}}{\partial Q} \\ \frac{\partial \tau_{x^i z}}{\partial Q} \\ \frac{\partial \beta_{x^i}}{\partial Q} \end{pmatrix} \quad (2.57)$$

Jacobians of the diagonal elements of the stress tensor are

$$\begin{aligned} \frac{\partial \tau_{xx}}{\partial Q} &= 2\frac{\partial \mu}{\partial Q} u_x + 2\mu \frac{\partial u_x}{\partial Q} + \frac{\partial \lambda_1}{\partial Q} \nabla \cdot \vec{V} + \lambda_1 \frac{\partial \nabla \cdot \vec{V}}{\partial Q} \\ \frac{\partial \tau_{yy}}{\partial Q} &= 2\frac{\partial \mu}{\partial Q} v_y + 2\mu \frac{\partial v_y}{\partial Q} + \frac{\partial \lambda_1}{\partial Q} \nabla \cdot \vec{V} + \lambda_1 \frac{\partial \nabla \cdot \vec{V}}{\partial Q} \\ \frac{\partial \tau_{zz}}{\partial Q} &= 2\frac{\partial \mu}{\partial Q} w_z + 2\mu \frac{\partial w_z}{\partial Q} + \frac{\partial \lambda_1}{\partial Q} \nabla \cdot \vec{V} + \lambda_1 \frac{\partial \nabla \cdot \vec{V}}{\partial Q}, \end{aligned} \quad (2.58)$$

and the Jacobians of the off diagonal elements of the stress tensor are

$$\begin{aligned} \frac{\partial \tau_{xy}}{\partial Q} &= \frac{\partial \tau_{yx}}{\partial Q} = \frac{\partial \mu}{\partial Q} (u_y + v_x) + \mu \frac{\partial}{\partial Q} (u_y + v_x) \\ \frac{\partial \tau_{xz}}{\partial Q} &= \frac{\partial \tau_{zx}}{\partial Q} = \frac{\partial \mu}{\partial Q} (u_z + w_x) + \mu \frac{\partial}{\partial Q} (u_z + w_x) \\ \frac{\partial \tau_{yz}}{\partial Q} &= \frac{\partial \tau_{zy}}{\partial Q} = \frac{\partial \mu}{\partial Q} (v_z + w_y) + \mu \frac{\partial}{\partial Q} (v_z + w_y). \end{aligned} \quad (2.59)$$

The stress tensor Jacobians require the Jacobian of the viscosity and 2nd viscosity coefficient as well as the Jacobian of the gradients of the velocity components. The viscosity and 2nd viscosity coefficient Jacobians are

$$\begin{aligned} \frac{\partial \mu}{\partial Q} &= \frac{\partial \mu(T)}{\partial T} \frac{\partial T}{\partial Q} \\ \frac{\partial \lambda_1}{\partial Q} &= -\frac{2}{3} \frac{\partial \mu}{\partial Q}, \end{aligned} \quad (2.60)$$

which requires the temperature Jacobian given in Eq. 2.52 and the Jacobian of Sutherland's law with respect

to temperature given as

$$\frac{\partial \mu(T)}{\partial T} = \frac{3}{2} T^{1/2} \left(\frac{1+S_1}{T+S_1} \right) - T^{3/2} \left(\frac{1+S_1}{(T+S_1)^2} \right). \quad (2.61)$$

The Jacobian of the velocity gradient, which is

$$\frac{\partial \nabla(V_i)}{\partial Q} \Rightarrow \begin{cases} \frac{\partial \nabla(V_i)}{\partial \rho} = \frac{\partial^2 V_i}{\partial \rho^2} \nabla \rho + \frac{\partial^2 V_i}{\partial(\rho \vec{V}) \partial \rho} \nabla (\rho V_i) \\ \frac{\partial \nabla(V_i)}{\partial(\rho \vec{V})} = \frac{\partial^2 V_i}{\partial(\rho \vec{V}) \partial \rho} \nabla \rho + \frac{\partial^2 V_i}{\partial(\rho \vec{V})^2} \nabla (\rho V_i) \\ \frac{\partial \nabla(V_i)}{\partial(\rho E)} = 0 \end{cases}, \quad (2.62)$$

where V_i represents the i^{th} component of the velocity vector, requires the Hessian of the velocity vector, given as

$$\begin{aligned} \frac{\partial^2 V_i}{\partial \rho^2} &= \frac{\partial}{\partial \rho} \left(-\frac{(\rho V)_i}{\rho^2} \right) = 2 \frac{(\rho V)_i}{\rho^3} \\ \frac{\partial^2 V_i}{\partial(\rho V)_j \partial \rho} &= \begin{cases} \frac{\partial}{\partial(\rho V)_j} \left(-\frac{(\rho V)_i}{\rho^2} \right) = -\frac{1}{\rho^2} & i=j \\ 0 & i \neq j \end{cases} \\ \frac{\partial^2 V_i}{\partial(\rho \vec{V})^2} &= \frac{\partial}{\partial(\rho \vec{V})} \left(\frac{1}{\rho} \right) = 0, \end{aligned} \quad (2.63)$$

because the velocity gradient in Eq. 2.48 is a function of the velocity Jacobian. The Jacobian of the work term in the viscous flux is

$$\begin{aligned} \frac{\partial \beta_{x^i}}{\partial Q} &= \frac{1}{M_\infty^2 Pr_\infty (\gamma - 1)} \left(\frac{\partial \mu}{\partial Q} \frac{\partial T}{\partial x^i} + \mu \frac{\partial}{\partial Q} \frac{\partial T}{\partial x^i} \right) \\ &+ \frac{\partial u}{\partial Q} \tau_{x^i x} + u \frac{\partial \tau_{x^i x}}{\partial Q} \\ &+ \frac{\partial v}{\partial Q} \tau_{x^i y} + v \frac{\partial \tau_{x^i y}}{\partial Q} \\ &+ \frac{\partial w}{\partial Q} \tau_{x^i z} + w \frac{\partial \tau_{x^i z}}{\partial Q}, \end{aligned} \quad (2.64)$$

which requires the Jacobian of the viscosity coefficient given in Eq. 2.60, velocity vector give in Eq. 2.50,

the stress tensor given in Eqs. 2.58 and 2.59, and the temperature gradient given as

$$\frac{\partial}{\partial Q} \nabla T = \frac{\partial^2 T}{\partial \rho \partial Q} \nabla \rho + \frac{\partial^2 T}{\partial (\rho u) \partial Q} \nabla (\rho u) + \frac{\partial^2 T}{\partial (\rho v) \partial Q} \nabla (\rho v) + \frac{\partial^2 T}{\partial (\rho w) \partial Q} \nabla (\rho w) + \frac{\partial^2 T}{\partial (\rho E) \partial Q} \nabla (\rho E). \quad (2.65)$$

The temperature gradient Jacobian in Eq. 2.65 requires the Hessian of the temperature, which is given in three parts. The components of the temperature Hessian with respect to the density is

$$\frac{\partial^2 T}{\partial \rho \partial Q} \Rightarrow \begin{cases} \frac{\partial^2 T}{\partial \rho^2} = \gamma M_\infty^2 \left(2 \frac{p}{\rho^3} - 2 \frac{1}{\rho^2} \frac{\partial p}{\partial \rho} + \frac{1}{\rho} \frac{\partial^2 p}{\partial \rho^2} \right) \\ \frac{\partial^2 T}{\partial \rho \partial (\rho \vec{V})} = \gamma M_\infty^2 \left(-\frac{1}{\rho^2} \frac{\partial p}{\partial (\rho \vec{V})} + \frac{1}{\rho} \frac{\partial^2 p}{\partial \rho \partial (\rho \vec{V})} \right) \\ \frac{\partial^2 T}{\partial \rho \partial (\rho E)} = \gamma M_\infty^2 \left(-\frac{1}{\rho^2} \frac{\partial p}{\partial (\rho E)} + \frac{1}{\rho} \frac{\partial^2 p}{\partial \rho \partial (\rho E)} \right) = -\gamma M_\infty^2 \frac{1}{\rho^2} \frac{\partial p}{\partial (\rho E)} \end{cases}. \quad (2.66)$$

The temperature Hessian with respect to the momentum vector components is

$$\frac{\partial^2 T}{\partial (\rho \vec{V}) \partial Q} \Rightarrow \begin{cases} \frac{\partial^2 T}{\partial (\rho \vec{V}) \partial \rho} = \frac{\partial^2 T}{\partial \rho \partial (\rho \vec{V})} \\ \frac{\partial^2 T}{\partial (\rho \vec{V})^2} = \gamma M_\infty^2 \frac{1}{\rho} \frac{\partial^2 p}{\partial (\rho \vec{V})^2} \\ \frac{\partial^2 T}{\partial (\rho \vec{V}) \partial (\rho E)} = \gamma M_\infty^2 \frac{1}{\rho} \frac{\partial^2 p}{\partial (\rho E) \partial (\rho \vec{V})} = 0 \end{cases}, \quad (2.67)$$

and the temperature Hessian components with respect to the total energy is

$$\frac{\partial^2 T}{\partial (\rho E) \partial Q} \Rightarrow \begin{cases} \frac{\partial^2 T}{\partial (\rho E) \partial \rho} = \frac{\partial^2 T}{\partial \rho \partial (\rho E)} \\ \frac{\partial^2 T}{\partial (\rho E) \partial (\rho \vec{V})} = 0 \\ \frac{\partial^2 T}{\partial (\rho E)^2} = 0 \end{cases}. \quad (2.68)$$

The temperature Hessian also requires the Hessians of the pressure and kinetic energy which are

$$\begin{aligned}
\frac{\partial^2 p}{\partial \rho^2} &= (\gamma - 1) \frac{\partial^2}{\partial \rho^2} \left(\frac{1}{2} \rho \vec{V}^2 \right), \\
\frac{\partial p}{\partial \rho \partial (\rho \vec{V})} &= (\gamma - 1) \frac{\partial^2}{\partial \rho \partial (\rho \vec{V})} \left(\frac{1}{2} \rho \vec{V}^2 \right), \\
\frac{\partial^2 p}{\partial \rho \partial (\rho E)} &= 0, \\
\frac{\partial^2 p}{\partial (\rho E) \partial (\rho \vec{V})} &= 0,
\end{aligned} \tag{2.69}$$

and

$$\begin{aligned}
\frac{\partial^2}{\partial \rho^2} \left(\frac{1}{2} \rho \vec{V}^2 \right) &= \frac{\vec{V}^2}{\rho}, \\
\frac{\partial^2}{\partial \rho \partial (\rho V)_i} \left(\frac{1}{2} \rho \vec{V}^2 \right) &= -\frac{V_i}{\rho}.
\end{aligned} \tag{2.70}$$

The viscous flux Jacobian with respect to the gradient of the conservative variable vector is

$$\frac{\partial F_i^d(Q)}{\partial \nabla Q} = \begin{pmatrix} 0 \\ \frac{\partial \tau_{x,i,x}}{\partial \nabla Q} \\ \frac{\partial \tau_{x,i,y}}{\partial \nabla Q} \\ \frac{\partial \tau_{x,i,z}}{\partial \nabla Q} \\ \frac{\partial \beta_{i,i}}{\partial \nabla Q} \end{pmatrix}. \tag{2.71}$$

The viscous flux Jacobian in Eq. 2.71 requires the Jacobian of the stress tensor and work term with respect to the gradient of the conservative variables. The Jacobians of the normal terms stress terms of the stress tensor are

$$\begin{aligned}
\frac{\partial \tau_{xx}}{\partial \nabla Q} &= 2\mu \frac{\partial u_x}{\partial \nabla Q} + \lambda_1 \frac{\partial \nabla \cdot \vec{V}}{\partial \nabla Q} \\
\frac{\partial \tau_{yy}}{\partial \nabla Q} &= 2\mu \frac{\partial v_y}{\partial \nabla Q} + \lambda_1 \frac{\partial \nabla \cdot \vec{V}}{\partial \nabla Q} \\
\frac{\partial \tau_{zz}}{\partial \nabla Q} &= 2\mu \frac{\partial w_z}{\partial \nabla Q} + \lambda_1 \frac{\partial \nabla \cdot \vec{V}}{\partial \nabla Q},
\end{aligned} \tag{2.72}$$

and the Jacobians of the shear stress terms are

$$\begin{aligned}
\frac{\partial \tau_{xy}}{\partial \nabla Q} &= \frac{\partial \tau_{yx}}{\partial \nabla Q} = \mu \left(\frac{\partial u_y}{\partial \nabla Q} + \frac{\partial v_x}{\partial \nabla Q} \right) \\
\frac{\partial \tau_{xz}}{\partial \nabla Q} &= \frac{\partial \tau_{zx}}{\partial \nabla Q} = \mu \left(\frac{\partial u_z}{\partial \nabla Q} + \frac{\partial w_x}{\partial \nabla Q} \right) \\
\frac{\partial \tau_{yz}}{\partial \nabla Q} &= \frac{\partial \tau_{zy}}{\partial \nabla Q} = \mu \left(\frac{\partial v_z}{\partial \nabla Q} + \frac{\partial w_y}{\partial \nabla Q} \right).
\end{aligned} \tag{2.73}$$

The stress tensor Jacobians in Eqs. 2.72 and 2.73 only require the Jacobian of the gradient of the velocity vector components, which is given as

$$\frac{\partial V_{x^j}^i}{\partial \nabla Q} \Rightarrow \begin{cases} \frac{\partial V_{x^j}^i}{\partial \rho_{x^j}} = \begin{cases} \frac{\partial V_i}{\partial \rho} & i = j \\ 0 & i \neq j \end{cases} \\ \frac{\partial V_{x^j}^i}{\partial (\rho V)_{x^j}^{i^j}} = \begin{cases} \frac{\partial V_i}{\partial (\rho V)_i} & i = j \\ 0 & i \neq j \end{cases} \end{cases}. \tag{2.74}$$

where $V_{x^j}^i$ is the j^{th} derivative of the i^{th} velocity component. The Jacobian of the work terms with respect to the gradient of the conservative variable vector is

$$\frac{\partial \beta_{x^i}}{\partial \nabla Q} = \frac{\mu}{M_\infty^2 Pr_\infty (\gamma - 1)} \frac{\partial}{\partial \nabla Q} \left(\frac{\partial T}{\partial x^i} \right) + u \frac{\partial \tau_{x^i x}}{\partial \nabla Q} + v \frac{\partial \tau_{x^i y}}{\partial \nabla Q} + w \frac{\partial \tau_{x^i z}}{\partial \nabla Q}, \tag{2.75}$$

which requires the Jacobians of the stress tensor given in Eqs. 2.72 and 2.73 and the Jacobian of the temperature gradient with respect to the gradient of the conservative variable vector. The temperature gradient Jacobian with respect to the conservative variable vectors,

$$\frac{\partial \nabla T}{\partial \nabla Q} = \begin{cases} \frac{\partial \nabla T}{\partial \nabla \rho} = \frac{\partial T}{\partial \rho} \\ \frac{\partial \nabla T}{\partial \nabla (\rho V)_i} = \frac{\partial T}{\partial (\rho V)_i} \\ \frac{\partial \nabla T}{\partial \nabla (\rho E)} = \frac{\partial T}{\partial (\rho E)} \end{cases}, \quad (2.76)$$

is solely a function of the temperature Jacobian with respect to the conservative variables given in Eq. 2.52.

2.2.3.2 Roe's Approximate Riemann Formulation

Roe's approximate Riemann solver[117] is used as the upwind dissipation flux. This upwind flux is written in a compact form as

$$\phi(Q^+, Q^-) = |\tilde{A}(Q^+, Q^-)| (Q^+ - Q^-) = |\lambda_3| \Delta Q + \begin{pmatrix} C_1 \\ C_1 \tilde{V} + C_2 \vec{n} \\ C_1 \tilde{H} + C_2 \tilde{V} \cdot \vec{n} \end{pmatrix}, \quad (2.77)$$

where $\Delta Q = (Q^+ - Q^-)$ is the jump in the conservative variables across a cell face. This compact form was used as part of the development of the PROJECT-X[118] discontinuous Galerkin solver and is documented in the theory guide for Fidkowski's XFLOW discontinuous Galerkin solver.[116] The terms C_1 and C_2 in Eq. 2.77 are

$$\begin{aligned} C_1 &= \frac{G_1}{\tilde{c}^2} s_1 + \frac{G_2}{\tilde{c}} s_2 \\ C_2 &= \frac{G_1}{\tilde{c}} s_2 + G_2 s_1 \end{aligned} \quad (2.78)$$

where terms G_1 and G_2 in Eq. 2.78 are

$$\begin{aligned} G_1 &= (\gamma - 1) \left(\frac{1}{2} \tilde{V}^2 \Delta \rho - \tilde{V} \cdot \Delta(\rho \vec{V}) + \Delta(\rho E) \right) \\ G_2 &= -(\tilde{V} \cdot \vec{n}) \Delta \rho + \Delta(\rho \vec{V}) \cdot \vec{n}, \end{aligned} \quad (2.79)$$

and the terms s_1 and s_2 in Eq. 2.78 are

$$\begin{aligned} s_1 &= \frac{1}{2}(|\lambda_1| + |\lambda_2|) - |\lambda_3| \\ s_2 &= \frac{1}{2}(|\lambda_1| - |\lambda_2|). \end{aligned} \quad (2.80)$$

The absolute value of the characteristic velocities, λ_i , in Eq 2.80 using an entropy fix formulation[119] is

$$|\lambda_i| = \begin{cases} \frac{1}{2} \left(\varepsilon \tilde{c} + \frac{\lambda_i^2}{\varepsilon \tilde{c}} \right) & \text{if } -\varepsilon \tilde{c} < \lambda_i < \varepsilon \tilde{c} \\ \sqrt{\lambda_i^2} & \text{otherwise} \end{cases}. \quad (2.81)$$

where $\varepsilon = 0.01$. The entropy fix absolute value is also used for λ_3 in Eq. 2.77. The Roe averaged eigenvalues, defined as

$$\begin{aligned} \lambda_1 &= \tilde{\vec{V}} \cdot \vec{n} + \tilde{c} \\ \lambda_2 &= \tilde{\vec{V}} \cdot \vec{n} - \tilde{c} \\ \lambda_3 &= \tilde{\vec{V}} \cdot \vec{n}, \end{aligned} \quad (2.82)$$

are a function of the Roe averaged speed of sound, velocity, and total enthalpy defined by

$$\begin{aligned} \tilde{c}^2 &= (\gamma - 1) \left(\tilde{H} - \frac{1}{2} \tilde{\vec{V}}^2 \right), \\ \tilde{\vec{V}} &= \frac{\sqrt{\rho^-} \vec{V}^- + \sqrt{\rho^+} \vec{V}^+}{\sqrt{\rho^-} + \sqrt{\rho^+}}, \\ \tilde{H} &= \frac{\sqrt{\rho^-} H^- + \sqrt{\rho^+} H^+}{\sqrt{\rho^-} + \sqrt{\rho^+}}. \end{aligned} \quad (2.83)$$

The Jacobian of the Roe dissipation flux is obtained through repeated application of the chain rule. The Jacobian of Roe dissipation flux in Eq. 2.77 with respect to both Q^+ and Q^- is

$$\frac{\partial \phi(Q^+, Q^-)}{\partial Q^\pm} = \frac{\partial |\lambda_3|}{\partial Q^\pm} \Delta Q + |\lambda_3| \frac{\partial \Delta Q}{\partial Q^\pm} + \begin{pmatrix} \frac{\partial C_1}{\partial Q^\pm} \\ \frac{\partial C_1}{\partial Q^\pm} \tilde{V} + C_1 \frac{\partial \tilde{V}}{\partial Q^\pm} + \frac{\partial C_2}{\partial Q^\pm} \vec{n} \\ \frac{\partial C_1}{\partial Q^\pm} \tilde{H} + C_1 \frac{\partial \tilde{H}}{\partial Q^\pm} + \frac{\partial C_2}{\partial Q^\pm} \tilde{V} \cdot \vec{n} + C_2 \frac{\partial \tilde{V} \cdot \vec{n}}{\partial Q^\pm} \end{pmatrix}. \quad (2.84)$$

where the Jacobians of the jump in the conservative variables are

$$\begin{aligned} \frac{\partial \Delta Q}{\partial Q^+} &= \begin{bmatrix} 1, & 1, & 1, & 1, & 1 \end{bmatrix} \\ \frac{\partial \Delta Q}{\partial Q^-} &= - \begin{bmatrix} 1, & 1, & 1, & 1, & 1 \end{bmatrix}, \end{aligned} \quad (2.85)$$

the Jacobians of the variables C_1 and C_2 are

$$\begin{aligned} \frac{\partial C_1}{\partial Q^\pm} &= \frac{\partial}{\partial Q^\pm} \left(\frac{G_1}{\tilde{c}^2} \right) s_1 + \frac{G_1}{\tilde{c}^2} \frac{\partial s_1}{\partial Q^\pm} + \frac{\partial}{\partial Q^\pm} \left(\frac{G_2}{\tilde{c}} \right) s_2 + \frac{G_2}{\tilde{c}} \frac{\partial s_2}{\partial Q^\pm} \\ \frac{\partial C_2}{\partial Q^\pm} &= \frac{\partial}{\partial Q^\pm} \left(\frac{G_1}{\tilde{c}} \right) s_2 + \frac{G_1}{\tilde{c}} \frac{\partial s_2}{\partial Q^\pm} + \frac{\partial G_2}{\partial Q^\pm} s_1 + G_2 \frac{\partial s_1}{\partial Q^\pm}, \end{aligned} \quad (2.86)$$

and the Jacobians of the ratios of G_i over \tilde{c} and \tilde{c}^2 are

$$\begin{aligned} \frac{\partial}{\partial Q^\pm} \left(\frac{G_1}{\tilde{c}} \right) &= \frac{\partial G_1}{\partial Q^\pm} \left(\frac{1}{\tilde{c}} \right) - \frac{\partial \tilde{c}}{\partial Q^\pm} \left(\frac{G_1}{\tilde{c}^2} \right) \\ \frac{\partial}{\partial Q^\pm} \left(\frac{G_2}{\tilde{c}} \right) &= \frac{\partial G_2}{\partial Q^\pm} \left(\frac{1}{\tilde{c}} \right) - \frac{\partial \tilde{c}}{\partial Q^\pm} \left(\frac{G_2}{\tilde{c}^2} \right) \\ \frac{\partial}{\partial Q^\pm} \left(\frac{G_1}{\tilde{c}^2} \right) &= \frac{\partial G_1}{\partial Q^\pm} \left(\frac{1}{\tilde{c}^2} \right) - 2 \frac{\partial \tilde{c}}{\partial Q^\pm} \left(\frac{G_1}{\tilde{c}^3} \right). \end{aligned} \quad (2.87)$$

The Jacobians of G_1 and G_2 are

$$\begin{aligned}
\frac{\partial G_1}{\partial Q^\pm} &= (\gamma - 1) \left(\frac{1}{2} \frac{\partial \tilde{\vec{V}}^2}{\partial Q^\pm} \Delta \rho + \frac{1}{2} \tilde{\vec{V}}^2 \frac{\partial \Delta \rho}{\partial Q^\pm} - \frac{\partial \tilde{\vec{V}}}{\partial Q^\pm} \cdot \Delta(\rho \vec{V}) - \tilde{\vec{V}} \cdot \frac{\partial \Delta(\rho \vec{V})}{\partial Q^\pm} + \frac{\partial \Delta(\rho E)}{\partial Q^\pm} \right) \\
\frac{\partial G_2}{\partial Q^\pm} &= -\frac{\partial \tilde{\vec{V}} \cdot \vec{n}}{\partial Q^\pm} \Delta \rho - (\tilde{\vec{V}} \cdot \vec{n}) \frac{\partial \Delta \rho}{\partial Q^\pm} + \frac{\partial \Delta(\rho \vec{V}) \cdot \vec{n}}{\partial Q^\pm},
\end{aligned} \tag{2.88}$$

and the Jacobians of s_1 and s_2 are

$$\begin{aligned}
\frac{\partial s_1}{\partial Q^\pm} &= \frac{1}{2} \left(\frac{\partial |\lambda_1|}{\partial Q^\pm} + \frac{\partial |\lambda_2|}{\partial Q^\pm} \right) - \frac{\partial |\lambda_3|}{\partial Q^\pm} \\
\frac{\partial s_2}{\partial Q^\pm} &= \frac{1}{2} \left(\frac{\partial |\lambda_1|}{\partial Q^\pm} - \frac{\partial |\lambda_2|}{\partial Q^\pm} \right).
\end{aligned} \tag{2.89}$$

The Jacobian of the absolute value operator with the entropy fix is

$$\frac{\partial |\lambda_i|}{\partial Q^\pm} = \begin{cases} \frac{1}{2} \epsilon \frac{\partial \tilde{c}}{\partial Q^\pm} + \frac{\lambda_i}{\epsilon \tilde{c}} \frac{\partial \lambda_i}{\partial Q^\pm} - \frac{1}{2} \frac{\lambda_i^2}{\epsilon \tilde{c}^2} \frac{\partial \epsilon \tilde{c}}{\partial Q^\pm} & \text{if } \sqrt{\lambda_i^2} < \epsilon \tilde{c}, \\ \frac{\lambda_i}{\sqrt{\lambda_i^2}} \frac{\partial \lambda_i}{\partial Q^\pm} & \text{otherwise} \end{cases}, \tag{2.90}$$

and the Jacobians of the Roe averaged eigenvalues are

$$\begin{aligned}
\frac{\partial \lambda_1}{\partial Q^\pm} &= \frac{\partial \tilde{\vec{V}} \cdot \vec{n}}{\partial Q^\pm} + \frac{\partial \tilde{c}}{\partial Q^\pm} \\
\frac{\partial \lambda_2}{\partial Q^\pm} &= \frac{\partial \tilde{\vec{V}} \cdot \vec{n}}{\partial Q^\pm} - \frac{\partial \tilde{c}}{\partial Q^\pm} \\
\frac{\partial \lambda_3}{\partial Q^\pm} &= \frac{\partial \tilde{\vec{V}} \cdot \vec{n}}{\partial Q^\pm}.
\end{aligned} \tag{2.91}$$

The Jacobians for the Roe averaged speed of sound are

$$\begin{aligned}
\frac{\partial \tilde{c}}{\partial Q^\pm} &= \frac{1}{2\tilde{c}} \frac{\partial \tilde{c}^2}{\partial Q^\pm} \\
\frac{\partial \tilde{c}^2}{\partial Q^\pm} &= (\gamma - 1) \left(\frac{\partial \tilde{H}}{\partial Q^\pm} - \frac{1}{2} \frac{\partial \tilde{\vec{V}}^2}{\partial Q^\pm} \right),
\end{aligned} \tag{2.92}$$

where the Jacobian of the Roe averaged velocity squared is

$$\frac{1}{2} \frac{\partial \tilde{V}^2}{\partial Q^\pm} \Rightarrow \begin{cases} \frac{1}{2} \frac{\partial \tilde{V}^2}{\partial \rho^\pm} = 0 \\ \frac{1}{2} \frac{\partial \tilde{V}^2}{\partial (\rho u)^\pm} = \tilde{u} \frac{\partial \tilde{u}}{\partial (\rho u)^\pm} \\ \frac{1}{2} \frac{\partial \tilde{V}^2}{\partial (\rho v)^\pm} = \tilde{v} \frac{\partial \tilde{v}}{\partial (\rho v)^\pm} \\ \frac{1}{2} \frac{\partial \tilde{V}^2}{\partial (\rho w)^\pm} = \tilde{w} \frac{\partial \tilde{w}}{\partial (\rho w)^\pm} \\ \frac{1}{2} \frac{\partial \tilde{V}^2}{\partial (\rho E)^\pm} = 0 \end{cases} . \quad (2.93)$$

All the expressions for the Jacobians in Eqs. 2.84 through 2.93 associated with the Roe upwind dissipation flux have been the same regardless if they were with respect to Q^+ or Q^- , with the exception of Eq. 2.85. The Jacobians with respect to Q^+ or Q^- in Eqs. 2.84 through 2.93 will differ in value as the Jacobians for the Roe averaged velocity and enthalpy with respect to Q^+ or Q^- differ. The Jacobians for the Roe averaged velocity are

$$\begin{aligned} \frac{\partial \tilde{V}}{\partial Q^-} &\Rightarrow \begin{cases} \frac{\partial \tilde{V}}{\partial \rho^-} = \frac{\partial}{\partial \rho^-} \left(\frac{\sqrt{\rho^-}}{\sqrt{\rho^-} + \sqrt{\rho^+}} \right) \vec{V}^- + \frac{\sqrt{\rho^-}}{\sqrt{\rho^-} + \sqrt{\rho^+}} \frac{\partial \vec{V}^-}{\partial \rho^-} + \frac{\partial}{\partial \rho^-} \left(\frac{\sqrt{\rho^+}}{\sqrt{\rho^-} + \sqrt{\rho^+}} \right) \vec{V}^+ \\ \frac{\partial \tilde{V}}{\partial (\rho \vec{V})^-} = \frac{\sqrt{\rho^-}}{\sqrt{\rho^-} + \sqrt{\rho^+}} \frac{\partial \vec{V}^-}{\partial (\rho \vec{V})^-} \\ \frac{\partial \tilde{V}}{\partial (\rho E)^-} = 0 \end{cases}, \\ \frac{\partial \tilde{V}}{\partial Q^+} &\Rightarrow \begin{cases} \frac{\partial \tilde{V}}{\partial \rho^+} = \frac{\partial}{\partial \rho^+} \left(\frac{\sqrt{\rho^+}}{\sqrt{\rho^-} + \sqrt{\rho^+}} \right) \vec{V}^+ + \frac{\sqrt{\rho^+}}{\sqrt{\rho^-} + \sqrt{\rho^+}} \frac{\partial \vec{V}^+}{\partial \rho^+} \\ \frac{\partial \tilde{V}}{\partial (\rho \vec{V})^+} = \frac{\sqrt{\rho^+}}{\sqrt{\rho^-} + \sqrt{\rho^+}} \frac{\partial \vec{V}^+}{\partial (\rho \vec{V})^+} \\ \frac{\partial \tilde{V}}{\partial (\rho E)^+} = 0 \end{cases}, \end{aligned} \quad (2.94)$$

which require the Jacobian of the velocity vector given in Eq. 2.50. Similarly, the Jacobians for the Roe averaged enthalpy are

$$\begin{aligned} \frac{\partial \tilde{H}}{\partial Q^-} &\Rightarrow \begin{cases} \frac{\partial \tilde{H}}{\partial \rho^-} = \frac{\partial}{\partial \rho^-} \left(\frac{\sqrt{\rho^-}}{\sqrt{\rho^-} + \sqrt{\rho^+}} \right) H^- + \frac{\sqrt{\rho^-}}{\sqrt{\rho^-} + \sqrt{\rho^+}} \frac{\partial H^-}{\partial \rho^-} + \frac{\partial}{\partial \rho^-} \left(\frac{\sqrt{\rho^+}}{\sqrt{\rho^-} + \sqrt{\rho^+}} \right) H^+ \\ \frac{\partial \tilde{H}}{\partial (\rho V)^-} = \frac{\sqrt{\rho^-}}{\sqrt{\rho^-} + \sqrt{\rho^+}} \frac{\partial H^-}{\partial (\rho V)^-} \\ \frac{\partial \tilde{H}}{\partial (\rho E)^-} = \frac{\sqrt{\rho^-}}{\sqrt{\rho^-} + \sqrt{\rho^+}} \frac{\partial H^-}{\partial (\rho E)^-} \end{cases}, \\ \frac{\partial \tilde{H}}{\partial Q^+} &\Rightarrow \begin{cases} \frac{\partial \tilde{H}}{\partial \rho^+} = \frac{\partial}{\partial \rho^+} \left(\frac{\sqrt{\rho^+}}{\sqrt{\rho^-} + \sqrt{\rho^+}} \right) H^+ + \frac{\sqrt{\rho^+}}{\sqrt{\rho^-} + \sqrt{\rho^+}} \frac{\partial H^+}{\partial \rho^+} \\ \frac{\partial \tilde{H}}{\partial (\rho V)^+} = \frac{\sqrt{\rho^+}}{\sqrt{\rho^-} + \sqrt{\rho^+}} \frac{\partial H^+}{\partial (\rho V)^+} \\ \frac{\partial \tilde{H}}{\partial (\rho E)^+} = \frac{\sqrt{\rho^+}}{\sqrt{\rho^-} + \sqrt{\rho^+}} \frac{\partial H^+}{\partial (\rho E)^+} \end{cases}, \quad (2.95) \end{aligned}$$

which require the Jacobian of the enthalpy given in Eq. 2.56. The Jacobians of the weights with ρ^- in the numerator of the Roe averaging are

$$\begin{aligned} \frac{\partial}{\partial \rho^-} \left(\frac{\sqrt{\rho^-}}{\sqrt{\rho^-} + \sqrt{\rho^+}} \right) &= \frac{1}{2} \frac{\sqrt{\rho^+}}{\sqrt{\rho^-} (\sqrt{\rho^-} + \sqrt{\rho^+})^2}, \\ \frac{\partial}{\partial \rho^+} \left(\frac{\sqrt{\rho^-}}{\sqrt{\rho^-} + \sqrt{\rho^+}} \right) &= -\frac{1}{2} \frac{\sqrt{\rho^-}}{\sqrt{\rho^+} (\sqrt{\rho^-} + \sqrt{\rho^+})^2}, \quad (2.96) \end{aligned}$$

and the weights with ρ^+ in the numerator are the negative of the Jacobians with ρ^- in the numerator, i. e.,

$$\begin{aligned} \frac{\partial}{\partial \rho^-} \left(\frac{\sqrt{\rho^+}}{\sqrt{\rho^-} + \sqrt{\rho^+}} \right) &= -\frac{\partial}{\partial \rho^-} \left(\frac{\sqrt{\rho^-}}{\sqrt{\rho^-} + \sqrt{\rho^+}} \right), \\ \frac{\partial}{\partial \rho^+} \left(\frac{\sqrt{\rho^+}}{\sqrt{\rho^-} + \sqrt{\rho^+}} \right) &= -\frac{\partial}{\partial \rho^+} \left(\frac{\sqrt{\rho^-}}{\sqrt{\rho^-} + \sqrt{\rho^+}} \right). \quad (2.97) \end{aligned}$$

2.2.4 Fluid Flow Boundary Conditions

The conservative variable vector for imposing boundary conditions, defined as

$$Q_B(Q^-, Q_b) = \begin{pmatrix} \rho_B(\rho^-, \rho_b) \\ \rho u_B(\rho u^-, \rho u_b) \\ \rho v_B(\rho v^-, \rho v_b) \\ \rho w_B(\rho w^-, \rho w_b) \\ \rho E_B(\rho E^-, \rho E_b) \end{pmatrix}, \quad (2.98)$$

is a function of the conservative variable vector interior to a cell, Q^- , and a set of imposed quantities, Q_b . The specific functions in Eq. 2.98 depend on the type of boundary condition. The boundary condition gradient vector

$$\nabla Q_B(\nabla Q^-, \nabla Q_b) = \begin{pmatrix} \nabla \rho_B(\nabla \rho^-, \nabla \rho_b) \\ \nabla \rho u_B(\nabla \rho u^-, \nabla \rho u_b) \\ \nabla \rho v_B(\nabla \rho v^-, \nabla \rho v_b) \\ \nabla \rho w_B(\nabla \rho w^-, \nabla \rho w_b) \\ \nabla \rho E_B(\nabla \rho E^-, \nabla \rho E_b) \end{pmatrix}, \quad (2.99)$$

is a function of the gradient interior to the cell, ∇Q^- , and imposed gradients, ∇Q_b . In general, the Jacobian of the boundary condition conservative variable vector with respect to the interior conservative variable vector is

$$\frac{\partial Q_B(Q^-, Q_b)}{\partial Q^-} = \begin{pmatrix} \frac{\partial \rho_B}{\partial \rho^-} & \frac{\partial \rho_B}{\partial (\rho u)^-} & \frac{\partial \rho_B}{\partial (\rho v)^-} & \frac{\partial \rho_B}{\partial (\rho w)^-} & \frac{\partial \rho_B}{\partial (\rho E)^-} \\ \frac{\partial \rho u_B}{\partial \rho^-} & \frac{\partial \rho u_B}{\partial (\rho u)^-} & \frac{\partial \rho u_B}{\partial (\rho v)^-} & \frac{\partial \rho u_B}{\partial (\rho w)^-} & \frac{\partial \rho u_B}{\partial (\rho E)^-} \\ \frac{\partial \rho v_B}{\partial \rho^-} & \frac{\partial \rho v_B}{\partial (\rho u)^-} & \frac{\partial \rho v_B}{\partial (\rho v)^-} & \frac{\partial \rho v_B}{\partial (\rho w)^-} & \frac{\partial \rho v_B}{\partial (\rho E)^-} \\ \frac{\partial \rho w_B}{\partial \rho^-} & \frac{\partial \rho w_B}{\partial (\rho u)^-} & \frac{\partial \rho w_B}{\partial (\rho v)^-} & \frac{\partial \rho w_B}{\partial (\rho w)^-} & \frac{\partial \rho w_B}{\partial (\rho E)^-} \\ \frac{\partial \rho E_B}{\partial \rho^-} & \frac{\partial \rho E_B}{\partial (\rho u)^-} & \frac{\partial \rho E_B}{\partial (\rho v)^-} & \frac{\partial \rho E_B}{\partial (\rho w)^-} & \frac{\partial \rho E_B}{\partial (\rho E)^-} \end{pmatrix}. \quad (2.100)$$

Some boundary conditions use specific equations to obtain the values ρ_B , p_B , and \vec{V}_B that are used to compute the conservative variables. The equations, and associated Jacobians, for the conservative variables based on ρ_B , p_B , and \vec{V}_B are given here as they are common for these type of boundary conditions. The momentum vector equation is

$$\rho \vec{V}_B = \rho_B \vec{V}_B, \quad (2.101)$$

and total energy equation is

$$\rho E_B = \frac{p_B}{\gamma - 1} + \frac{\rho \vec{V}_B^2}{2}. \quad (2.102)$$

The Jacobian for the momentum vector

$$\begin{aligned} \frac{\partial \rho u_B}{\partial Q^-} &= \frac{\partial \rho_B}{\partial Q^-} u_B + \rho_B \frac{\partial u_B}{\partial Q^-}, \\ \frac{\partial \rho v_B}{\partial Q^-} &= \frac{\partial \rho_B}{\partial Q^-} v_B + \rho_B \frac{\partial v_B}{\partial Q^-}, \\ \frac{\partial \rho w_B}{\partial Q^-} &= \frac{\partial \rho_B}{\partial Q^-} w_B + \rho_B \frac{\partial w_B}{\partial Q^-}, \end{aligned} \quad (2.103)$$

and Jacobian for the energy equation

$$\frac{\partial \rho E_B}{\partial Q^-} = \frac{1}{\gamma - 1} \frac{\partial p_B}{\partial Q^-} + u_B \frac{\partial \rho u_B}{\partial Q^-} + v_B \frac{\partial \rho v_B}{\partial Q^-} + w_B \frac{\partial \rho w_B}{\partial Q^-} - \frac{\partial \rho_B}{\partial Q^-} \frac{\vec{V}_B^2}{2} \quad (2.104)$$

with respect to the interior conservative variable vector requires the Jacobians $\frac{\partial \rho_B}{\partial Q^-}$, $\frac{\partial p_B}{\partial Q^-}$, and $\frac{\partial \vec{V}_B}{\partial Q^-}$.

The Jacobian for the boundary condition gradient vector in Eq. 2.11 with respect to the interior gradient vector always results in a set of diagonal matrices. The diagonal matrices have a unit value for Dirichlet boundary conditions, and a value of zero for Neumann boundary conditions. For example, the boundary condition gradient vector for an adiabatic wall is

$$\nabla Q_B (\nabla Q^-, \nabla Q_b) = \begin{pmatrix} 0 \\ \nabla \rho u^- \\ \nabla \rho v^- \\ \nabla \rho w^- \\ 0 \end{pmatrix}, \quad (2.105)$$

which has the Jacobian with respect to the interior gradient vector

$$\begin{aligned}
& \frac{\partial \nabla Q_B(\nabla Q^-, \nabla Q_b)}{\partial \nabla Q^-} = \frac{\partial}{\partial \nabla Q^-} \begin{pmatrix} 0 \\ \nabla \rho u^- \\ \nabla \rho v^- \\ \nabla \rho w^- \\ 0 \end{pmatrix} \\
&= \left(\begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \right). \quad (2.106)
\end{aligned}$$

Since this pattern is general, the Jacobians for the boundary condition gradient vector will not be included with each boundary condition outlined in the following.

2.2.4.1 Walls

No-slip walls are imposed with $\vec{V}_B = 0$ and a slip wall is imposed with $\vec{V}_B \cdot \vec{n} = 0$. However, these conditions are equivalent for weakly imposed boundary conditions. Weak boundary conditions are imposed through the normal flux

$$\vec{F}^a(Q) \cdot \vec{n} = \begin{pmatrix} \rho \vec{V} \cdot \vec{n} \\ \rho u \vec{V} \cdot \vec{n} + p n_x \\ \rho v \vec{V} \cdot \vec{n} + p n_y \\ \rho w \vec{V} \cdot \vec{n} + p n_z \\ \rho H \vec{V} \cdot \vec{n} \end{pmatrix}. \quad (2.107)$$

Imposing either $\vec{V}_B = 0$ or $\vec{V}_B \cdot \vec{n} = 0$ in the normal flux, and using the pressure from the interior cell, results in the boundary condition flux is

$$\vec{F}^a(Q_B(Q^-, Q_b)) \cdot \vec{n} = \begin{pmatrix} 0 \\ p^- n_x \\ p^- n_y \\ p^- n_z \\ 0 \end{pmatrix}. \quad (2.108)$$

Hence, both slip and no-slip walls are imposed with the boundary condition variable vector where $\vec{V}_B = 0$

$$Q_B(Q^-, Q_b) = \begin{pmatrix} \rho^- \\ 0 \\ 0 \\ 0 \\ \rho E^- - \frac{\rho^-}{2} (\vec{V}^-)^2 \end{pmatrix}. \quad (2.109)$$

The Jacobian of the boundary condition conservative vector with respect to Q^- is

$$\frac{\partial Q_B(Q^-, Q_b)}{\partial Q^-} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ -\frac{1}{2} (\vec{V}^-)^2 & -u^- & -v^- & -w^- & 1 \end{pmatrix}. \quad (2.110)$$

An adiabatic no-slip wall also imposes $\nabla T = 0$. By recognizing that $\frac{\partial T}{\partial(\rho \vec{V})} \propto \vec{V}$ and that $\vec{V}_B = 0$, Eq. 2.51 is set to zero by imposing $\nabla \rho_B = 0$ and $\nabla \rho E_B = 0$. Hence, the boundary condition gradient vector is

$$\nabla Q_B(\nabla Q^-, \nabla Q_b) = \begin{pmatrix} 0 \\ \nabla \rho u^- \\ \nabla \rho v^- \\ \nabla \rho w^- \\ 0 \end{pmatrix}. \quad (2.111)$$

2.2.4.2 Total Temperature and Total Pressure Inflow

A total temperature and total pressure boundary condition is appropriate for internal flow problems. The boundary condition conservative variable vector is a function of a specified total temperature, T_T , a total pressure, p_T , and a flow direction unit vector \vec{n}_b . The non-dimensional total temperature equation

$$T_T = T + (\gamma - 1) M_\infty^2 \frac{\vec{V}^2}{2}, \quad (2.112)$$

is derived from the total enthalpy equation assuming constant C_p . The derivation for the non-dimensional total temperature is given in Appendix A. The boundary condition static temperature is obtained using Eq. 2.112 with the imposed total temperature T_T and the interior velocity as

$$T_B = T_T - (\gamma - 1) M_\infty^2 \frac{(\vec{V}^-)^2}{2}. \quad (2.113)$$

The isentropic relation is used to obtain the boundary condition static pressure

$$p_B = p_T \left(\frac{T_B}{T_T} \right)^{\frac{\gamma}{\gamma-1}}, \quad (2.114)$$

and the static density is obtained from the ideal gas law equation (Eq. 2.46)

$$\rho_B = \gamma M_\infty^2 \frac{p_B}{T_B} \quad (2.115)$$

The boundary condition velocity vector is the interior velocity magnitude in the direction imposed by the unit vector \vec{n}_b

$$\vec{V}_B = |\vec{V}^-| \vec{n}_b. \quad (2.116)$$

The boundary condition momentum vector and total energy are given by Eqs. 2.101 and 2.102.

The boundary condition conservative variable vector in Eq. 2.98 is obtained from ρ_B , $\vec{\rho}V_B$, and ρE_B in Eqs. 2.115, 2.101, and 2.102 respectively. There are no requirements on gradient values for the total temperature and total pressure boundary condition so the boundary condition gradient vector is

$$\nabla Q_B (\nabla Q^-, \nabla Q_b) = \nabla Q^-. \quad (2.117)$$

The Jacobian of the boundary condition static temperature

$$\frac{\partial T_B}{\partial Q^-} = -(\gamma - 1) M_\infty^2 \frac{1}{2} \frac{\partial (\vec{V}^-)^2}{\partial Q^-}, \quad (2.118)$$

requires the Jacobian of the interior velocity squared

$$\frac{\partial (\vec{V}^-)^2}{\partial Q^-} \Rightarrow \begin{cases} \frac{\partial (\vec{V}^-)^2}{\partial \rho^-} = -2 \frac{(\vec{V}^-)^2}{\rho^-} \\ \frac{\partial (\vec{V}^-)^2}{\partial (\rho \vec{V}^-)} = 2 \frac{\vec{V}^-}{\rho^-} \\ \frac{\partial (\vec{V}^-)^2}{\partial (\rho E^-)} = 0 \end{cases}. \quad (2.119)$$

The boundary condition static temperature Jacobian

$$\frac{\partial p_B}{\partial Q^-} = p_T \left(\frac{T_B}{T_T} \right)^{\frac{\gamma}{(\gamma-1)}} \frac{\gamma}{(\gamma-1) T_B} \frac{\partial T_B}{\partial Q^-} = \frac{\gamma}{(\gamma-1)} \frac{p_B}{T_B} \frac{\partial T_B}{\partial Q^-}, \quad (2.120)$$

requires the boundary condition static temperature Jacobian given in Eq. 2.118. The Jacobian for the static density

$$\frac{\partial \rho_B}{\partial Q^-} = \gamma M_\infty^2 \left(\frac{\partial p_B}{\partial Q^-} \frac{1}{T_B} + p_B \frac{\partial}{\partial Q^-} \left(\frac{1}{T_B} \right) \right), \quad (2.121)$$

requires the Jacobian for the boundary condition static pressure (Eq. 2.120) and the Jacobian for the inverse of the boundary condition static temperature

$$\frac{\partial}{\partial Q^-} \left(\frac{1}{T_B} \right) = \frac{-1}{T_B^2} \frac{\partial T_B}{\partial Q^-}. \quad (2.122)$$

The Jacobian for the velocity vector

$$\frac{\partial \vec{V}_B^i}{\partial Q^-} = \frac{1}{2 |\vec{V}^-|} \frac{\partial (\vec{V}^-)^2}{\partial Q^-} \vec{n}_b^i, \quad (2.123)$$

requires the Jacobian of the velocity vector squared give in Eq. 2.119. Note that the Jacobian for the velocity

vector requires the inverse of the interior velocity vector. Hence, the limit of Eq. 2.123 as $|\vec{V}^-| \rightarrow 0$, namely

$$\lim_{|\vec{V}^-| \rightarrow 0} \frac{\partial \vec{V}_B^i}{\partial Q^-} = \begin{cases} \lim_{|\vec{V}^-| \rightarrow 0} -\frac{1}{|\vec{V}^-|} \frac{(\vec{V}^-)^2}{\rho^-} \vec{n}_b^i = 0 \\ \lim_{|\vec{V}^-| \rightarrow 0} \frac{1}{|\vec{V}^-|} \frac{\vec{V}^-}{\rho^-} \vec{n}_b^i = \frac{1}{\rho^-} \vec{n}_b^i \\ \frac{\partial \vec{V}_B^i}{\partial (\rho E^-)} = 0 \end{cases}, \quad (2.124)$$

is used when $|\vec{V}^-| = 0$. The Jacobians for the boundary condition static density, static pressure, and velocity given in Eqs. 2.121, 2.120, and 2.123 are used along with the Jacobians for the momentum vector and total energy given in Eqs. 2.103 and 2.104 are used to obtain the conservative variable vector Jacobian in Eq. 2.100. The Jacobian of the boundary condition gradient vector is a set of identity matrices.

2.2.4.3 Subsonic Fixed Momentum Inflow

The subsonic fixed momentum inflow boundary condition imposes a prescribed density and momentum vector, and extrapolates the interior pressure. The boundary condition conservative variable vector is

$$Q_B(Q^-, Q_b) = \begin{pmatrix} \rho_b \\ \rho u_b \\ \rho v_b \\ \rho w_b \\ \frac{p^-}{\gamma-1} + \frac{\rho_b}{2} (\vec{V}_b)^2 \end{pmatrix}, \quad (2.125)$$

and the boundary condition conservative variable vector Jacobian is

$$\frac{\partial Q_B(Q^-, Q_b)}{\partial Q^-} = \frac{1}{\gamma-1} \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ \frac{\partial p^-}{\partial \rho^-} & \frac{\partial p^-}{\partial (\rho u^-)} & \frac{\partial p^-}{\partial (\rho v^-)} & \frac{\partial p^-}{\partial (\rho w^-)} & \frac{\partial p^-}{\partial (\rho E^-)} \end{pmatrix}, \quad (2.126)$$

where the pressure Jacobian is given in Eq. 2.53. The boundary condition gradient vector is simply

$$\nabla Q_B (\nabla Q^-, \nabla Q_b) = \nabla Q^- . \quad (2.127)$$

2.2.4.4 Fixed Back Pressure Outflow

The fixed back pressure boundary condition imposes a back pressure and extrapolates density and momentum. The boundary condition conservative variable vector is

$$Q_B (Q^-, Q_b) = \begin{pmatrix} \rho^- \\ \rho u^- \\ \rho v^- \\ \rho w^- \\ \frac{p_B}{\gamma-1} + \frac{\rho^-}{2} (\vec{V}^-)^2 \end{pmatrix}, \quad (2.128)$$

and the Jacobian of the boundary condition conservative variable vector is

$$\frac{\partial Q_B (Q^-, Q_b)}{\partial Q^-} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ -\frac{1}{2} (\vec{V}^-)^2 & u^- & v^- & w^- & 0 \end{pmatrix}. \quad (2.129)$$

With no requirements on the boundary condition gradients, the boundary condition gradient vector is

$$\nabla Q_B (\nabla Q^-, \nabla Q_b) = \nabla Q^- . \quad (2.130)$$

2.2.4.5 Supersonic Inflow

This boundary condition is only appropriate if the Mach number for the inflow boundary condition is above unity. The supersonic inflow boundary condition imposes all values in the boundary condition conservative variable vector

$$Q_B(Q^-, Q_b) = \begin{pmatrix} \rho_b \\ \rho u_b \\ \rho v_b \\ \rho w_b \\ \rho E_b \end{pmatrix}. \quad (2.131)$$

Hence, the Jacobian of the conservative variable vector is

$$\frac{\partial Q_B(Q^-, Q_b)}{\partial Q^-} = 0. \quad (2.132)$$

With no requirements on the boundary condition gradients, the boundary condition gradient vector is

$$\nabla Q_B(\nabla Q^-, \nabla Q_b) = \nabla Q^-. \quad (2.133)$$

2.2.4.6 Supersonic Outflow

The supersonic outflow boundary condition is appropriate when the Mach number of the interior cell is above unity. The boundary condition conservative variable vector takes all values from the interior cell

$$Q_B(Q^-, Q_b) = \begin{pmatrix} \rho^- \\ \rho u^- \\ \rho v^- \\ \rho w^- \\ \rho E^- \end{pmatrix}. \quad (2.134)$$

The Jacobian of the boundary condition conservative variable vector is the identity matrix

$$\frac{\partial Q_B(Q^-, Q_b)}{\partial Q^-} = I. \quad (2.135)$$

The boundary condition gradient vector is set to zero

$$\nabla Q_B(\nabla Q^-, \nabla Q_b) = 0, \quad (2.136)$$

in order to impose a zero gradient normal to the boundary. Hence, the Jacobian of the gradient vector is also zero.

2.2.4.7 Riemann Invariant Far-field

The Riemann invariant far-field boundary condition is a characteristics-based far-field boundary condition based on one-dimensional Riemann invariants of a steady isentropic inviscid flow.[120] The velocity magnitude normal to the boundary is compared with the local speed of sound to determine if the flow is subsonic or supersonic. For supersonic flows, i.e., $|\vec{V}^- \cdot \vec{n}| \geq c^-$, the supersonic inflow boundary condition given in Eq. 2.131 is imposed when $\vec{V}^- \cdot \vec{n} \leq 0$, and the supersonic outflow boundary condition given in Eq. 2.134 is imposed when $\vec{V}^- \cdot \vec{n} > 0$.

Constant Riemann invariants are imposed when the flow is subsonic normal to the boundary, i. e., $|\vec{V}^- \cdot \vec{n}| < c^-$. For an inflow boundary where $\vec{V}^- \cdot \vec{n} \leq 0$ the entropy, S , and tangential velocity vector, \vec{V}_τ , are obtained from the imposed boundary condition values,

$$\begin{cases} S = \frac{p_b}{\rho_b^\gamma} & \vec{V}^- \cdot \vec{n} \leq 0, \\ \vec{V}_\tau = \vec{V}_b - (\vec{V}_b \cdot \vec{n}) \vec{n} \end{cases} \quad (2.137)$$

and for an outflow boundary where $\vec{V}^- \cdot \vec{n} > 0$ the entropy and tangential velocity is obtained from interior values

$$\begin{cases} S = \frac{p^-}{(\rho^-)^\gamma} & \vec{V}^- \cdot \vec{n} > 0. \\ \vec{V}_\tau = \vec{V}^- - (\vec{V}^- \cdot \vec{n}) \vec{n} \end{cases} \quad (2.138)$$

The Riemann invariants R_p and R_m used to obtain the boundary speed of sound and normal velocity magnitude are given as

$$\begin{aligned} R_p &= (\vec{V}^- \cdot \vec{n}) + \frac{2c^-}{\gamma-1}, \\ R_m &= (\vec{V}_b \cdot \vec{n}) - \frac{2c_b}{\gamma-1}. \end{aligned} \quad (2.139)$$

The boundary speed of sound is

$$c_B = \frac{\gamma-1}{4} (R_p - R_m), \quad (2.140)$$

and the boundary normal velocity vector is

$$\vec{V}_n = \frac{1}{2} (R_p + R_m) \vec{n}. \quad (2.141)$$

The boundary condition density, velocity vector, and pressure are given by

$$\begin{aligned} \rho_B &= \left(\frac{c_B^2}{S\gamma} \right)^{\frac{1}{\gamma-1}} \\ \vec{V}_B &= \vec{V}_n + \vec{V}_\tau \\ p_B &= \rho_B^\gamma S \end{aligned} \quad (2.142)$$

The boundary condition vector given in Eq. 2.98 is obtained with the momentum and total energy equations given in Eqs. 2.101 and 2.102.

Using the chain rule, the Jacobians of the boundary condition density, velocity vector, and pressure with respect to the interior conservative variable vector are

$$\begin{aligned} \frac{\partial \rho_B}{\partial Q^-} &= \frac{\rho_B}{c_B^2(\gamma-1)} \left(\frac{\partial c_B^2}{\partial Q^-} - \frac{c_B^2}{S} \frac{\partial S}{\partial Q^-} \right), \\ \frac{\partial \vec{V}_B}{\partial Q^-} &= \frac{\partial \vec{V}_n}{\partial Q^-} + \frac{\partial \vec{V}_\tau}{\partial Q^-}, \\ \frac{\partial p_B}{\partial Q^-} &= \frac{\gamma}{\rho_B} \frac{\partial \rho_B}{\partial Q^-} \rho_B^\gamma S + \rho_B^\gamma \frac{\partial S}{\partial Q^-} = \frac{\gamma p_B}{\rho_B} \frac{\partial \rho_B}{\partial Q^-} + \rho_B^\gamma \frac{\partial S}{\partial Q^-}. \end{aligned} \quad (2.143)$$

The Jacobian of the boundary condition density requires the Jacobian of the boundary condition speed of sound squared,

$$\frac{\partial c_B^2}{\partial Q^-} = 2c_B \frac{\partial c_B}{\partial Q^-}, \quad (2.144)$$

which requires the Jacobian of the boundary condition speed of sound

$$\frac{\partial c_B}{\partial Q^-} = \frac{\gamma - 1}{4} \frac{\partial R_p}{\partial Q^-}. \quad (2.145)$$

Both the Jacobian of the boundary condition speed of sound in Eq. 2.145 and the boundary normal velocity vector Jacobian

$$\frac{\partial \vec{V}_n}{\partial Q^-} = \frac{1}{2} \frac{\partial R_p}{\partial Q^-} \vec{n}, \quad (2.146)$$

require the Jacobian of the Riemann invariant R_p , which is

$$\frac{\partial R_p}{\partial Q^-} = \frac{\partial \vec{V}^-}{\partial Q^-} \cdot \vec{n} + \frac{2}{\gamma - 1} \frac{\partial c^-}{\partial Q^-}. \quad (2.147)$$

The velocity vector Jacobian in Eq. 2.147 is given in Eq. 2.50 and the Jacobian of the speed of sound require in Eq. 2.147, which is

$$\frac{\partial c^-}{\partial Q^-} = \frac{1}{2c^-} \frac{\partial (c^-)^2}{\partial Q^-}, \quad (2.148)$$

requires the Jacobian of the speed of sound squared given as

$$\frac{\partial (c^-)^2}{\partial Q^-} \Rightarrow \begin{cases} \frac{\partial (c^-)^2}{\partial \rho^-} = \gamma \left(\frac{1}{\rho^-} \frac{\partial p^-}{\partial \rho^-} - \frac{p^-}{(\rho^-)^2} \right) \\ \frac{\partial (c^-)^2}{\partial (\rho \vec{V})^-} = \gamma \frac{1}{\rho^-} \frac{\partial p^-}{\partial (\rho \vec{V})^-} \\ \frac{\partial (c^-)^2}{\partial (\rho E)^-} = \gamma \frac{1}{\rho^-} \frac{\partial p^-}{\partial (\rho E)^-} \end{cases}. \quad (2.149)$$

The Jacobian of the speed of sound squared in Eq. 2.149 requires the Jacobian of the pressure given in Eq. 2.53. The Jacobian of the entropy is required in the boundary condition density and pressure Jacobians in Eq. 2.143, and the boundary condition velocity vector Jacobian in Eq. 2.143 requires the tangential velocity vector Jacobian. Both the boundary condition entropy and tangential velocity vector Jacobians are dependent on the flow direction on the boundary. For inflow where $\vec{V}^- \cdot \vec{n} \leq 0$ the boundary condition entropy and tangential velocity are independent of the interior conservative variable vector; hence

$$\begin{cases} \frac{\partial S}{\partial Q^-} = 0 & \vec{V}^- \cdot \vec{n} \leq 0. \\ \frac{\partial \vec{V}_t}{\partial Q^-} = 0 \end{cases} \quad (2.150)$$

For outflow where $\vec{V}^- \cdot \vec{n} > 0$ the Jacobians of the boundary condition entropy and tangential velocity vector are

$$\left\{ \begin{array}{l} \frac{\partial S}{\partial Q^-} = \begin{cases} \frac{\partial S}{\partial \rho^-} = \frac{\partial p^-}{\partial \rho^-} - \frac{\gamma p^-}{(\rho^-)^{\gamma-1}} \\ \frac{\partial S}{\partial (\rho \vec{V})^-} = \frac{1}{(\rho^-)^\gamma} \frac{\partial p^-}{\partial (\rho \vec{V})^-} \\ \frac{\partial S}{\partial (\rho E)^-} = \frac{1}{(\rho^-)^\gamma} \frac{\partial p^-}{\partial (\rho E)^-} \end{cases} \\ \frac{\partial \vec{V}_t}{\partial Q^-} = \frac{\partial \vec{V}^-}{\partial Q^-} - \left(\frac{\partial \vec{V}^-}{\partial Q^-} \cdot \vec{n} \right) \vec{n} \end{array} \right. \quad \vec{V}^- \cdot \vec{n} > 0, \quad (2.151)$$

which require the pressure and velocity Jacobians given in Eqs. 2.53 and 2.50.

2.2.5 Artificial Viscosity

Numerical calculations of orders of accuracy higher than one will exhibit oscillations in the solution in the vicinity of sharp gradients, e.g., shocks, unless limiting[121] or artificial viscosity[19, 27] schemes are included to mitigate the oscillations. The method chosen for this work is the partial differential equation based artificial viscosity formulation by Barter and Darmofal where the artificial viscosity coefficient, ε , is modeled with a Poisson equation. Details of the artificial viscosity formulation can be found in Refs. [96], [27], and [28]. The modified Euler equations to include the artificial viscosity are

$$\frac{\partial Q}{\partial t} + \nabla \cdot \vec{F}^a + \nabla \cdot \vec{F}^{av} + S_\varepsilon(Q) = 0 \quad (2.152)$$

where the conservative variables are $Q = \left[\rho, \rho u, \rho v, \rho w, \rho E, \varepsilon \right]^T$, \vec{F}^a are the inviscid fluxes give in Eq. 2.42 and the artificial viscosity flux

$$\vec{F}^{av}(Q, \nabla Q) = \left[F_x^{av}, F_y^{av}, F_z^{av} \right] = \left[\begin{pmatrix} -\tilde{\varepsilon} \frac{h_x}{h} \frac{\partial \rho}{\partial x} \\ -\tilde{\varepsilon} \frac{h_x}{h} \frac{\partial \rho u}{\partial x} \\ -\tilde{\varepsilon} \frac{h_x}{h} \frac{\partial \rho v}{\partial x} \\ -\tilde{\varepsilon} \frac{h_x}{h} \frac{\partial \rho H}{\partial x} \\ -\frac{c_1 c_2 N \lambda_{max}}{h_{min}} h_x^2 \frac{\partial \varepsilon}{\partial x} \end{pmatrix}, \begin{pmatrix} -\tilde{\varepsilon} \frac{h_y}{h} \frac{\partial \rho}{\partial y} \\ -\tilde{\varepsilon} \frac{h_y}{h} \frac{\partial \rho u}{\partial y} \\ -\tilde{\varepsilon} \frac{h_y}{h} \frac{\partial \rho v}{\partial y} \\ -\tilde{\varepsilon} \frac{h_y}{h} \frac{\partial \rho H}{\partial y} \\ -\frac{c_1 c_2 N \lambda_{max}}{h_{min}} h_y^2 \frac{\partial \varepsilon}{\partial y} \end{pmatrix}, \begin{pmatrix} -\tilde{\varepsilon} \frac{h_z}{h} \frac{\partial \rho}{\partial z} \\ -\tilde{\varepsilon} \frac{h_z}{h} \frac{\partial \rho u}{\partial z} \\ -\tilde{\varepsilon} \frac{h_z}{h} \frac{\partial \rho v}{\partial z} \\ -\tilde{\varepsilon} \frac{h_z}{h} \frac{\partial \rho H}{\partial z} \\ -\frac{c_1 c_2 N \lambda_{max}}{h_{min}} h_z^2 \frac{\partial \varepsilon}{\partial z} \end{pmatrix} \right] \quad (2.153)$$

includes the non-linear Poisson equation for ε and $S_\varepsilon(Q)$ is a source term for the Poisson equation. Note that

the gradient of the total enthalpy can be expressed in terms of the conservative variables and their gradients as

$$\nabla \rho H = \nabla \rho E + \frac{\partial p}{\partial \rho} \nabla \rho + \frac{\partial p}{\partial \rho u} \nabla \rho u + \frac{\partial p}{\partial \rho v} \nabla \rho v + \frac{\partial p}{\partial \rho w} \nabla \rho w + \frac{\partial p}{\partial \rho E} \nabla \rho E. \quad (2.154)$$

The artificial viscosity coefficient, $\tilde{\varepsilon}$, is a limited value of ε that is governed by the non-linear Poisson equation

$$\frac{\partial \varepsilon}{\partial t} + \nabla \cdot \left(\left[\frac{\eta}{\tau} \right] \nabla \varepsilon \right) + S_\varepsilon(Q) = 0 \quad (2.155)$$

where

$$\begin{aligned} S_\varepsilon(Q) &= -\frac{1}{\tau} \left(\frac{\bar{h}}{N} \lambda_{max} \tilde{s}_k - \varepsilon \right), \\ \tau &= \frac{h_{min}}{c_1 N \lambda_{max}}, \\ \left[\frac{\eta}{\tau} \right] &= -\frac{c_1 c_2 N \lambda_{max}}{h_{min}} \begin{bmatrix} h_x^2 & 0 & 0 \\ 0 & h_y^2 & 0 \\ 0 & 0 & h_z^2 \end{bmatrix}, \\ \lambda_{max} &= |\vec{V}| + c, \\ \vec{h} &= \begin{bmatrix} h_x & h_y & h_z \end{bmatrix}, \\ \bar{h} &= \frac{1}{3} (h_x + h_y + h_z), \\ h_{min} &= \min(h_x, h_y, h_z), \\ c_1 &= 3, \\ c_1 c_2 &= 15. \end{aligned} \quad (2.156)$$

Here, h_x , h_y , and h_z are the extents of the bounding box of a cell. The term \tilde{s}_k in the source term of Eq. 2.155 is a limited value of the shock sensor s_k expressed as

$$\tilde{s}_k = \begin{cases} 0 & s_k \leq s_0 - \kappa \\ \frac{1}{2}\varepsilon_0 \left(1 + \sin\left(\frac{1}{2}\frac{\pi(s_k - s_0)}{\kappa}\right)\right) & s_0 - \kappa < s_k \leq s_0 + \kappa \\ \varepsilon_0 & s_0 + \kappa < s_k \end{cases} \quad (2.157)$$

The value of s_k is given by a modified version of the resolution indicator[19] as

$$\begin{aligned} s_k &= \log_{10}(S_k), \\ S_k &= \frac{\int_{\Omega_e} (p_{111} - \bar{p})^2 d\Omega}{\int_{\Omega_e} p_{111}^2 d\Omega}, \end{aligned} \quad (2.158)$$

where $p_{111} = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} \sum_{k=0}^{N-1} p_{ijk} \psi_{ijk}$ is the linear polynomial expansion of pressure, and \bar{p} is the cell mean pressure value. The expression for obtaining $\tilde{\varepsilon}$ by limiting ε is

$$\begin{aligned} \tilde{\varepsilon} &= \begin{cases} 0 & \varepsilon \leq \tilde{\varepsilon}_{low} \\ \frac{1}{2}\tilde{\varepsilon}_{hi} \left(1 + \sin\left(\pi \left[\frac{\varepsilon - \tilde{\varepsilon}_{low}}{\tilde{\varepsilon}_{hi} - \tilde{\varepsilon}_{low}} - \frac{1}{2}\right]\right)\right) & \tilde{\varepsilon}_{low} < \varepsilon \leq \tilde{\varepsilon}_{hi} \\ \tilde{\varepsilon}_{hi} & \tilde{\varepsilon}_{hi} < \varepsilon \end{cases} \\ \tilde{\varepsilon}_{low} &= 0.01 \lambda_{max} \frac{\bar{h}}{N} \\ \tilde{\varepsilon}_{hi} &= \lambda_{max} \frac{\bar{h}}{N}. \end{aligned} \quad (2.159)$$

The non-linear Poisson equation in Eq. 2.155 is solved fully coupled with the fluid equations. The Jacobians for the artificial viscosity flux are

$$\frac{\partial \vec{F}_{x^i}^{av}(Q, \nabla Q)}{\partial Q} = \begin{pmatrix} -\frac{\partial \tilde{\varepsilon}}{\partial Q} \frac{h_{x^i}}{\bar{h}} \frac{\partial \rho}{\partial x^i} \\ -\frac{\partial \tilde{\varepsilon}}{\partial Q} \frac{h_{x^i}}{\bar{h}} \frac{\partial \rho u}{\partial x^i} \\ -\frac{\partial \tilde{\varepsilon}}{\partial Q} \frac{h_{x^i}}{\bar{h}} \frac{\partial \rho v}{\partial x^i} \\ -\frac{\partial \tilde{\varepsilon}}{\partial Q} \frac{h_{x^i}}{\bar{h}} \frac{\partial \rho H}{\partial x^i} + \tilde{\varepsilon} \frac{h_{x^i}}{\bar{h}} \frac{\partial^2 \rho H}{\partial Q \partial x^i} \\ -\frac{c_1 c_2 N}{h_{min}} \frac{\partial \lambda_{max}}{\partial Q} h_{x^i}^2 \frac{\partial \varepsilon}{\partial x^i} \end{pmatrix} \quad (2.160)$$

where the Jacobian of the limited artificial viscosity coefficient is

$$\frac{\partial \tilde{\epsilon}}{\partial Q} = \begin{cases} 0 & \epsilon \leq \tilde{\epsilon}_{low} \\ \frac{1}{2} \frac{\partial \tilde{\epsilon}_{hi}}{\partial Q} \left(1 + \sin \left(\pi \left[\frac{\epsilon - \tilde{\epsilon}_{low}}{\tilde{\epsilon}_{hi} - \tilde{\epsilon}_{low}} - \frac{1}{2} \right] \right) \right) + \frac{1}{2} \tilde{\epsilon}_{hi} \frac{\partial}{\partial Q} \left(\sin \left(\pi \left[\frac{\epsilon - \tilde{\epsilon}_{low}}{\tilde{\epsilon}_{hi} - \tilde{\epsilon}_{low}} - \frac{1}{2} \right] \right) \right) & \tilde{\epsilon}_{low} < \epsilon \leq \tilde{\epsilon}_{hi} \\ \frac{\partial \tilde{\epsilon}_{hi}}{\partial Q} & \tilde{\epsilon}_{hi} < \epsilon \end{cases} \quad (2.161)$$

The Jacobian of the sine function in Eq. 2.161 is

$$\begin{aligned} \frac{\partial}{\partial Q} \left(\sin \left(\pi \left[\frac{\epsilon - \tilde{\epsilon}_{low}}{\tilde{\epsilon}_{hi} - \tilde{\epsilon}_{low}} - \frac{1}{2} \right] \right) \right) &= \pi \cos \left(\pi \left[\frac{\epsilon - \tilde{\epsilon}_{low}}{\tilde{\epsilon}_{hi} - \tilde{\epsilon}_{low}} - \frac{1}{2} \right] \right) \frac{\partial}{\partial Q} \left(\frac{\epsilon - \tilde{\epsilon}_{low}}{\tilde{\epsilon}_{hi} - \tilde{\epsilon}_{low}} \right), \\ \frac{\partial}{\partial Q} \left(\frac{\epsilon - \tilde{\epsilon}_{low}}{\tilde{\epsilon}_{hi} - \tilde{\epsilon}_{low}} \right) &= \frac{\partial}{\partial \tilde{\epsilon}_{low}} \left(\frac{\epsilon - \tilde{\epsilon}_{low}}{\tilde{\epsilon}_{hi} - \tilde{\epsilon}_{low}} \right) \frac{\partial \tilde{\epsilon}_{low}}{\partial Q} + \frac{\partial}{\partial \tilde{\epsilon}_{hi}} \left(\frac{\epsilon - \tilde{\epsilon}_{low}}{\tilde{\epsilon}_{hi} - \tilde{\epsilon}_{low}} \right) \frac{\partial \tilde{\epsilon}_{hi}}{\partial Q} \\ &\quad + \frac{\partial}{\partial \epsilon} \left(\frac{\epsilon - \tilde{\epsilon}_{low}}{\tilde{\epsilon}_{hi} - \tilde{\epsilon}_{low}} \right) \frac{\partial \epsilon}{\partial Q}, \end{aligned} \quad (2.162)$$

where the Jacobians with respect to $\tilde{\epsilon}_{low}$, $\tilde{\epsilon}_{hi}$, and ϵ are

$$\begin{aligned} \frac{\partial}{\partial \tilde{\epsilon}_{low}} \left(\frac{\epsilon - \tilde{\epsilon}_{low}}{\tilde{\epsilon}_{hi} - \tilde{\epsilon}_{low}} \right) &= \frac{\epsilon - \tilde{\epsilon}_{low}}{(\tilde{\epsilon}_{hi} - \tilde{\epsilon}_{low})^2} - \frac{1}{\tilde{\epsilon}_{hi} - \tilde{\epsilon}_{low}}, \\ \frac{\partial}{\partial \tilde{\epsilon}_{hi}} \left(\frac{\epsilon - \tilde{\epsilon}_{low}}{\tilde{\epsilon}_{hi} - \tilde{\epsilon}_{low}} \right) &= -\frac{\epsilon - \tilde{\epsilon}_{low}}{(\tilde{\epsilon}_{hi} - \tilde{\epsilon}_{low})^2}, \\ \frac{\partial}{\partial \epsilon} \left(\frac{\epsilon - \tilde{\epsilon}_{low}}{\tilde{\epsilon}_{hi} - \tilde{\epsilon}_{low}} \right) &= \frac{1 - \tilde{\epsilon}_{low}}{\tilde{\epsilon}_{hi} - \tilde{\epsilon}_{low}}, \end{aligned} \quad (2.163)$$

and the Jacobians of $\tilde{\epsilon}_{low}$ and $\tilde{\epsilon}_{hi}$ with respect to the conservative variable vector are

$$\begin{aligned} \frac{\partial \tilde{\epsilon}_{hi}}{\partial Q} &= \frac{\partial \lambda_{max}}{\partial Q} \frac{\bar{h}}{N}, \\ \frac{\partial \tilde{\epsilon}_{low}}{\partial Q} &= 0.01 \frac{\partial \lambda_{max}}{\partial Q} \frac{\bar{h}}{N}. \end{aligned} \quad (2.164)$$

The Jacobian for the characteristic speed λ_{max} , which is

$$\frac{\partial \lambda_{max}}{\partial Q} = \frac{\partial |\vec{V}|}{\partial Q} + \frac{\partial c}{\partial Q}, \quad (2.165)$$

requires the Jacobian of the velocity magnitude,

$$\frac{\partial |\vec{V}|}{\partial Q} = \frac{1}{|\vec{V}|^2} \frac{1}{2} \frac{\partial (\vec{V})^2}{\partial Q}, \quad (2.166)$$

the Jacobian for the velocity magnitude squared in Eq. 2.119 and the Jacobian for the speed of sound in Eq. 2.148. Since Eq. 2.166 involves the inverse of the velocity magnitude, the limit of Eq. 2.166 as $|\vec{V}| \rightarrow 0$, i.e.,

$$\lim_{|\vec{V}| \rightarrow 0} \frac{\partial |\vec{V}|}{\partial Q} = \begin{cases} \lim_{|\vec{V}| \rightarrow 0} -\frac{1}{|\vec{V}|} \frac{(\vec{V})^2}{\rho} = 0 \\ \lim_{|\vec{V}| \rightarrow 0} \frac{1}{|\vec{V}|} \frac{\vec{V}}{\rho} = \frac{1}{\rho} \\ \frac{\partial |\vec{V}|}{\partial (\rho E)} = 0 \end{cases}, \quad (2.167)$$

is used when $|\vec{V}| = 0$. The Jacobian of the total enthalpy gradient

$$\frac{\partial \nabla \rho H}{\partial Q} = \frac{\partial p}{\partial Q \partial \rho} \nabla \rho + \frac{\partial p}{\partial Q \partial \rho u} \nabla \rho u + \frac{\partial p}{\partial Q \partial \rho v} \nabla \rho v + \frac{\partial p}{\partial Q \partial \rho w} \nabla \rho w + \frac{\partial p}{\partial Q \partial \rho E} \nabla \rho E, \quad (2.168)$$

requires the Hessian of the pressure given in Eq. 2.69. The Jacobian of the artificial viscosity flux with respect to the gradient of the conservative variable vector is

$$\frac{\partial \vec{F}_{x^i}^{av}(Q, \nabla Q)}{\partial \nabla Q} = \begin{pmatrix} -\tilde{\epsilon} \frac{h_{x^i}}{h} \\ -\tilde{\epsilon} \frac{h_{x^i}}{h} \\ -\tilde{\epsilon} \frac{h_{x^i}}{h} \\ -\tilde{\epsilon} \frac{h_{x^i}}{h} \frac{\partial^2 \rho H}{\partial \nabla Q \partial x^i} \\ -\frac{c_1 c_2 N \lambda_{max}}{h_{min}} h_{x^i}^2 \end{pmatrix}, \quad (2.169)$$

where the Jacobian of the total enthalpy gradient is

$$\frac{\partial \nabla \rho H}{\partial \nabla Q} \Rightarrow \begin{cases} \frac{\partial \nabla \rho H}{\partial \nabla \rho} = \frac{\partial p}{\partial \rho} \\ \frac{\partial \nabla \rho H}{\partial \nabla (\rho \vec{V})} = \frac{\partial p}{\partial (\rho \vec{V})} \\ \frac{\partial \nabla \rho H}{\partial \nabla (\rho E)} = 1 + \frac{\partial p}{\partial \rho E} \\ \frac{\partial \nabla \rho H}{\partial \nabla \epsilon} = 0 \end{cases}. \quad (2.170)$$

The Jacobian of the diffusive flux in the non-linear Poisson equation in Eq. 2.155 with respect to the conservative variable vector is

$$\frac{\partial}{\partial Q} \left(\left[\frac{\eta}{\tau} \right] \nabla \varepsilon \right) = \frac{\partial \lambda_{max}}{\partial Q} \frac{c_1 c_2 N}{h_{min}} \begin{bmatrix} h_x^2 & 0 & 0 \\ 0 & h_y^2 & 0 \\ 0 & 0 & h_z^2 \end{bmatrix} \nabla \varepsilon, \quad (2.171)$$

and the Jacobian with respect to the gradient of the conservative variable vector is

$$\frac{\partial}{\partial \nabla Q} \left(\left[\frac{\eta}{\tau} \right] \nabla \varepsilon \right) \begin{cases} \frac{\partial}{\partial \nabla \rho} \left(\left[\frac{\eta}{\tau} \right] \nabla \varepsilon \right) = 0 \\ \frac{\partial}{\partial \nabla (\rho V)} \left(\left[\frac{\eta}{\tau} \right] \nabla \varepsilon \right) = 0 \\ \frac{\partial}{\partial \nabla (\rho E)} \left(\left[\frac{\eta}{\tau} \right] \nabla \varepsilon \right) = 0 \\ \frac{\partial}{\partial \nabla \varepsilon} \left(\left[\frac{\eta}{\tau} \right] \nabla \varepsilon \right) = \left[\frac{\eta}{\tau} \right] \end{cases}. \quad (2.172)$$

The Jacobian of the source term $S_\varepsilon(Q)$ in Eq. 2.155 can be simplified if the source term is first expressed as

$$S_\varepsilon(Q) = -\frac{1}{\tau} \left(\frac{\bar{h}}{N} \lambda_{max} \tilde{s}_k - \varepsilon \right) = -\left(c_1 \frac{\bar{h}}{h_{min}} \lambda_{max}^2 \tilde{s}_k - \frac{c_1 N \lambda_{max}}{h_{min}} \varepsilon \right). \quad (2.173)$$

The Jacobian of the source term $S_\varepsilon(Q)$ as it is written in Eq. 2.173 with respect to the conservative variable vector is

$$\frac{\partial S_\varepsilon(Q)}{\partial Q} = -\left(c_1 \frac{\bar{h}}{h_{min}} \left(\frac{\partial \lambda_{max}^2}{\partial Q} \tilde{s}_k + \lambda_{max}^2 \frac{\partial \tilde{s}_k}{\partial Q} \right) - \frac{c_1 N}{h_{min}} \left(\frac{\partial \lambda_{max}}{\partial Q} \varepsilon + \lambda_{max} \frac{\partial \varepsilon}{\partial Q} \right) \right), \quad (2.174)$$

where the Jacobian of the characteristic speed is given in Eq. 2.165. The Jacobian of the limited shock sensor in Eq. 2.174 is

$$\frac{\partial \tilde{s}_k}{\partial Q} = \begin{cases} 0 & s_k \leq s_0 - \kappa \\ \frac{1}{2} \varepsilon_0 \cos \left(\frac{1}{2} \frac{\pi(s_k - s_0)}{\kappa} \right) \left(\frac{1}{2} \frac{\pi}{\kappa} \right) \frac{\partial s_k}{\partial Q} & s_0 - \kappa < s_k \leq s_0 + \kappa \\ 0 & s_0 + \kappa < s_k \end{cases}, \quad (2.175)$$

where the Jacobian of the shock sensor is

$$\begin{aligned}
\frac{\partial s_k}{\partial Q} &= \frac{\partial S_k}{\partial Q} \frac{1}{S_k \ln(10)}, \\
\frac{\partial S_k}{\partial Q} &= \frac{\int_{\Omega_e} 2(p_{111} - p_{000}) \left(\left(\frac{\partial p}{\partial Q} \right)_{111} - \left(\frac{\partial p}{\partial Q} \right)_{000} \right) d\Omega}{\int_{\Omega_e} p_{111}^2 d\Omega} \\
&\quad - \frac{\int_{\Omega_e} (p_{111} - p_{000})^2 d\Omega}{\int_{\Omega_e} p_{111}^4 d\Omega} \int_{\Omega_e} 2 \left(\frac{\partial p}{\partial Q} \right)_{111} p_{111} d\Omega.
\end{aligned} \tag{2.176}$$

2.2.6 Artificial Viscosity Boundary Conditions

2.2.6.1 Walls

A zero gradient Neumann boundary condition, $\partial \varepsilon / \partial \vec{n} = 0$, is imposed on walls for the non-linear Poisson equation. This is achieved with the boundary condition variable and boundary condition gradient

$$\begin{aligned}
\varepsilon_B(\varepsilon^-, \varepsilon_b) &= \varepsilon^-, \\
\nabla \varepsilon_B(\nabla \varepsilon^-, \nabla \varepsilon_b) &= 0,
\end{aligned} \tag{2.177}$$

which have the Jacobians with respect to the interior ε^- and gradient of ε^-

$$\begin{aligned}
\frac{\partial \varepsilon_B(\varepsilon^-, \varepsilon_b)}{\partial \varepsilon^-} &= 1, \\
\frac{\partial \nabla \varepsilon_B(\nabla \varepsilon^-, \nabla \varepsilon_b)}{\partial \nabla \varepsilon^-} &= 0.
\end{aligned} \tag{2.178}$$

2.2.6.2 Inflow/Outflow

A Robin type boundary condition is imposed on the non-linear Poisson equation for all boundary conditions that are not walls as suggested by Barter and Darmofal[96, 27]. For this boundary condition the boundary value and gradient are defined as

$$\begin{aligned}
\varepsilon_B(\varepsilon^-, \varepsilon_b) &= \varepsilon^-, \\
\nabla \varepsilon_B(\varepsilon^-, \nabla \varepsilon^-, \nabla \varepsilon_b) &= -\frac{\varepsilon^-}{10 \sqrt{\left(\vec{h} \cdot \vec{n} \right)^2}} \vec{n}.
\end{aligned} \tag{2.179}$$

The Jacobian of the boundary value with respect to the interior ε^- is and the gradient of ε^- are

$$\begin{aligned}\frac{\partial \varepsilon_B(\varepsilon^-, \varepsilon_b)}{\partial \varepsilon^-} &= 1, \\ \frac{\partial \nabla \varepsilon_B(\nabla \varepsilon^-, \nabla \varepsilon_b)}{\partial \nabla \varepsilon^-} &= 0.\end{aligned}\quad (2.180)$$

However, unlike other boundary conditions, the boundary gradient is a function of the interior ε^- . Hence, an additional Jacobian of the boundary gradient with respect to ε^- is required. This Jacobian is

$$\frac{\partial \nabla \varepsilon_B(\varepsilon^-, \nabla \varepsilon^-, \nabla \varepsilon_b)}{\partial \varepsilon^-} = -\frac{1}{10\sqrt{(\vec{h} \cdot \vec{n})^2}} \vec{n}. \quad (2.181)$$

2.3 Summary

The Discontinuous Galerkin discretization method used in this dissertation is presented, which is suitable for solving 1st and 2nd-order partial differential equations written in weak conservation form. Numerical solutions to partial differential equations are approximated with piecewise polynomials that are continuous within a cell, but the solution is discontinuous across cell boundaries. The numerical approximation is advanced toward a solution using Newton's method, which requires a linearization of the discrete partial differential equation. Hence, the fluxes of partial differential equations currently included in the code are presented along with the flux Jacobians, as well as the linearization of boundary conditions.

The code is structured as a general framework for numerically solving partial differential equations. Both scalar equations and systems of equations are currently included in the code. The scalar equations are used to verify that the many parts of the code are properly implemented. The system of partial differential equations that model fluid flow, i.e., the Navier-Stokes equations, included in the code are of primary interest to the author. The flux and flux Jacobians of the Navier-Stokes equations and the numerical upwinding scheme are presented, as well as multiple types of boundary conditions and their linearization. Finally, the scheme for limiting oscillations in the vicinity of shocks currently included in the code is outlined.

Chapter 3

Discontinuous Galerkin Solver Implementation

Details of the implementation of the discontinuous Galerkin method in the code are presented in this chapter. The code is divided into three distinct parts: Linear Algebra Libraries, Flux and Jacobian Integration, and Solution Advancement. The implementation is verified with a set of partial differential scalar equations, inviscid Euler equations, and the Navier-Stokes equations.

3.1 Linear Algebra Libraries

Many of the cell local operations required in a discontinuous Galerkin discretization can be reduced to dense linear algebra operations, i.e., matrix vector multiplications where the matrix has very few, if any, zero entries. Hence, it is important to have an efficient dense linear algebra library to minimize the computational cost of the code. In addition, an efficient sparse block matrix solver library is needed to advance the solution implicitly by solving the sparse block Jacobian matrix $\frac{\partial \mathcal{R}(Q)}{\partial Q}$. Two linear algebra libraries suitable for dense linear algebra operations and sparse block matrix operations were developed for the code. The libraries are invoked through an interface with a syntax that mimics the mathematical expressions used to document the algorithms. The syntax of the interface simplifies the implementation and legibility of the discontinuous Galerkin code. The interface is implemented using Template Expressions[122, 123, 124] to achieve clarity in the syntax without compromising computational efficiency.

3.1.1 Template Expressions

The code is written in C++, which is an object oriented programming language. The object oriented nature of the language provides great flexibility in the design of a modular code. In addition, operator overloading can be used to write code that resembles mathematical operations. However, a traditional use of operator overloading for classes generates temporary variables for each step in evaluating a mathematical expression. These temporary variables are the reason that linear algebra operations implemented in C++ using traditional operator overloading are inefficient. To give an example of this, consider the vector operation

$$\vec{x} \leftarrow a\vec{y} + b\vec{z} \quad (3.1)$$

where a and b are scalars and \vec{x} , \vec{y} , and \vec{z} are large vectors. The expression tree for Eq. 3.1 is illustrated in Fig. 3.1. The traditional operator overloading starts by parsing the leaves at the top of the tree and creates a temporary variable for each binary operation. The pseudo code using traditional operator overloading is given in Algorithm 3.1, where t_1 , t_2 , and t_3 are temporary vectors. Template Expressions, which were first introduced in the Blitz++ library[122], uses C++ templates to circumvent the need for temporary variables. The Blitz++ library also demonstrated that it is possible to write C++ code that is as computationally efficient as code written in FORTRAN. A summary of how Template Expressions are used in the code is presented here, and reader is referred to Ref. [124] for a more complete description of Template Expressions and other Template Metaprogramming techniques.

The principal behind Template Expressions is to use C++ templates to generate a custom data type during compilation that represents the entire expression tree to the right of the equal sign. There are two different flavors of Template Expressions. The first is a pure C++ implementation that was introduced in Blitz++, and the second is based on recursive function calls[123] that allows the use of Basic Linear Algebra Subprograms (BLAS) for each linear algebra operation. The latter is implemented here as optimized BLAS libraries are generally more efficient than native C++ code. The pseudo code for Template Expressions based on recursive function calls is given in Algorithm 3.2. The recursive function call algorithm avoids the use of temporary vectors by applying each operation in the expression tree to the vector \vec{x} .

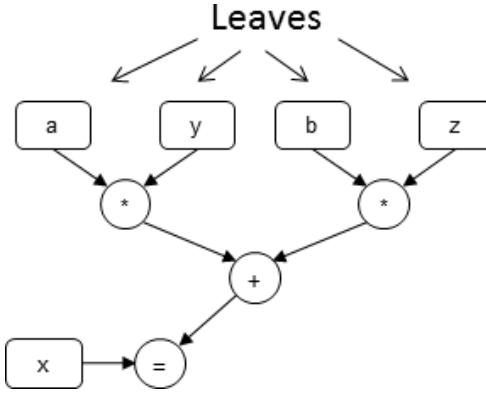


Figure 3.1: Expression Tree for The Linear Algebra Expression in Eq. 3.1

Algorithm 3.1 Pseudo Code The Linear Algebra Expression in Eq. 3.1 using Traditional Operator Overloading

```

for i to n
    t1[i] = a*y[i]

for i to n
    t2[i] = b*z[i]

for i to n
    t3[i] = t1[i] + t2[i]

for i to n
    x[i] = t3[i]

```

Algorithm 3.2 Pseudo Code The Linear Algebra Expression in Eq. 3.1 using Recursive Function Template Expressions

```

for i to n
    x[i] = a*y[i]

for i to n
    x[i] = x[i] + b*z[i]

```

Two different templated classes and respective generator functions are needed to represent the right hand side of the expression tree for Eq. 3.1. The first class represent the multiplication of a scalar with a vector is shown in Table 3.1. This class stores a reference to scalar and vector that are multiplied. The generator functions, given in Table. 3.2, are overloaded functions of the multiplication and division operators. These

generator functions accept any class that is derived from the class `VectorType` and a scalar. The `VectorType` is simply an empty templated class that is shown in Table 3.3. Using the `VectorType` as the base class for all data types involved in the vector Template Expressions prevents the compiler from attempting to invoke the generator functions with any other data types. The second class in the Template Expression represents summation of two vector types. The summation can be either addition or subtraction. This class is implemented in two steps. The `VectSumExpr` class represents the summation operation and stores reference to the two vector types that are summed as shown in Table 3.4. The summation operation is implemented in the two classes `VectorAddOp` and `VectorSubOp` shown in Table 3.5. One of these classes is given to `VectSumExpr` as the template argument `OpTag` using the generator functions shown in Table 3.6. The addition operator generates a `VectSumExpr` data type with `VectorAddOp` data type, and the subtraction operator generates a `VectSumExpr` with a `VectorSubOp` data type. The final class needed to complete the expression tree is the `Vector` class (See, Table 3.7) that represents a linear algebra vector. This class takes an integer for the size of the vector and allocates the memory array `_value`.

Table 3.1: Template Expression for Multiplication of a Scalar and a Vector

```
template <class VectType >
struct VectMulScalExpr
: public VectorType< VectMulScalExpr< VectType > >
{
    VectMulScalExpr(VectType const &Vect, double const Scal)
        : Vect(Vect), Scal(Scal) {}

    template<class VecType>
    inline void value(VecType& Res, const double sgn) const
    { Vect.value( Res, Scal*sgn); }

    template<class VecType>
    inline void plus (VecType& Res, const double sgn) const
    { Vect.plus( Res, Scal*sgn); }

protected:
    VectType const& Vect;
    double const Scal;
};
```

Table 3.2: Generator Functions for Multiplication of a Scalar and a Vector

```

template <class VectType>
inline VectMulScalExpr<VectType>
    operator * (VectorType<VectType> const& Vect, double const Scal)
{
    return VectMulScalExpr<VectType>(
        static_cast<VectType const&>(Vect), Scal);
}

template <class VectType>
inline VectMulScalExpr<VectType>
    operator * (double const Scal, VectorType<VectType> const& Vect)
{
    return VectMulScalExpr<VectType>(
        static_cast<VectType const&>(Vect), Scal);
}

template <class VectType>
inline VectMulScalExpr<VectType>
    operator / (VectorType<VectType> const& Vect, double const Scal)
{
    return VectMulScalExpr<VectType>( static_cast<VectType const&>(Vect)
        , 1./Scal);
}

```

Table 3.3: Class used to Identify Vector Datatypes

<pre> template <class Derived> struct VectorType { }; </pre>
--

Table 3.4: Templatized Class Representing a Binary Summation Operation in the Expression Tree

```

template <class L, template<class , class , class> class OpTag, class R >
struct VectSumExpr : public VectorType< VectExpression< L, OpTag, R > >
{
    VectSumExpr(L const &l , R const &r) : l(l), r(r) {}

    template<class VecType>
    inline void value(VecType& Vec, const double sgn) const
    { OpTag<L,R,T>::value(Vec, l, r, sgn); }

    template<class VecType>
    inline void plus (Vector<T>& Vec, const double sgn) const
    { OpTag<L,R,T>::plus (Vec, l, r, sgn); }

protected:
    L const& l;
    R const& r;
};

```

Table 3.5: Operator Tags for Addition and Subtraction

```

#define VECT_SUM_OPERATOR(name, neg) \
template<class VectorTypeL, class VectorTypeR, class T> \
struct name \
{ \
    \
    template<class VecType> \
    inline static void value(VecType& Vec, const VectorTypeL &VecL \
                           , const VectorTypeR &VecR, const T sgn) \
    { \
        VecL.value(Vec, sgn); \
        VecR.plus (Vec, neg sgn); \
    } \
    template<class VecType> \
    inline static void plus(VecType& Vec, const VectorTypeL &VecL \
                           , const VectorTypeR &VecR, const T sgn) \
    { \
        VecL.plus (Vec, sgn); \
        VecR.plus (Vec, neg sgn); \
    } \
}; \
VECT_SUM_OPERATOR(VectorAddOp, +) \
VECT_SUM_OPERATOR(VectorSubOp, -) \
#undef VECT_SUM_OPERATOR

```

Table 3.6: Generator Functions For Vector Summation

```
// Operator for addition and subtraction between two vectors
template <class L, class R>
inline VectSumExpr<L, VectorAddOp, R >
operator + (VectorType<L> const& l, VectorType<R> const& r)
{
    return VectSumExpr<L, VectorAddOp, R >( static_cast<L const&>(l)
                                                , static_cast<R const&>(r) );
}

template <class L, class R>
inline VectExpression<L, VectorSubOp, R >
operator - (VectorType<L> const& l, VectorType<R> const& r)
{
    return VectSumExpr<L, VectorSubOp, R >( static_cast<L const&>(l)
                                                , static_cast<R const&>(r));
}
```

The final C++ code that computes the expression in Eq. 3.1 is shown in Table 3.8. In this code, the expression tree is generated through the overloaded multiplication and addition operators in Tables 3.2 and 3.6. The multiplication operator in Table 3.2 is first invoked due to the multiplication operator precedence over the addition operator. The two multiplication operations will each create an instance of the class `VectMulScalExpr<Vector>`, which are in turn used in the arguments for the addition operator in 3.6. The addition operator will generate the class given in Table 3.9. This class is a complete representation of the right hand side of the expression tree and only contains references to the scalar and vector data. A traditional implementation would have used the overloaded operators to populate temporary vectors. The calculation occurs in the overloaded equal operator in the `Vector` class with the Template Expression scheme.

Table 3.7: Vector Class

```

struct Vector : public VectorType< Vector >
{
protected:
    const int size;
    double *_value;

public:
    Vector(const int size) : size(size), _value(new T[size]) {};
    ~Vector() { delete [] _value; }

    inline Vector& operator = (double const val)
    {
        for(register int i = 0; i < size; ++i)
            _value[i] = val;
    }

//=====
// Expression operators
    void value(Vector& Res, const T sgn) const
    {
        assert(Res.size == this->size);
        for(register int i = 0; i < size; ++i)
            Res._value[i] = sgn*_value[i];
    }

    void plus(Vector& Res, const T sgn) const
    {
        assert(Res.size == this->size);
        cblas_daxpy(size, sgn, value, 1, Res._value, 1)
    }

    template<class ExprType>
    inline Vector& operator = (VectorType<ExprType> const &ExprIn)
    {
        ExprType const& Expr = static_cast<ExprType const&>(ExprIn);
        Expr.value(*this, 1);
        return *this;
    }
};

```

The expression tree is parsed when the equal operator function on the x instance of Vector class calls the “value” function of the expression tree class with the x instance as the first argument, which is the “value” of the VectSumExpr class since the outermost class is the expression tree. That function will in

turn invoke the “value” function of the VectMulScalExpr instance that stores a reference to the variables a and y . The VectMulScalExpr then invokes the “value” function of the y Vector instance with the scalar a as the second argument. The “value” function on the Vector class then performs the calculation $x = ay$. The following function invoked is the “plus” function on the VectMulScalExpr instances stored in the VectSumExpr instance. This function invokes the “plus” function on the z instance of the Vector class with b as the scalar argument. The “plus” function on the vector class then performs the calculation $x = x + bz$. Hence, the Template Expression performs the linear algebra operation using the pseudo code in Algorithm 3.2 without instantiating temporary variables. The Template Expression strategy was used to develop both dense and block-sparse linear algebra libraries for the code.

Table 3.8: C++ Code for Computing the Expression in Eq. 3.1

```
Vector x(10), y(10), z(10);

y = 1; z = 2;

double a = 3; b = 2;

x = a*y + b*z;
```

Table 3.9: Template Expression Data Type Representing the Expression Tree in Eq. 3.1

```
VectSumExpr< VectMulScalExpr<Vector>
, VectorAddOp
, VectMulScalExpr<Vector> >
```

3.1.2 Dense Linear Algebra Library

The Template Expression strategy was used to develop a dense linear algebra library. The dense linear algebra library is also able to perform vector addition and vector-scalar multiplication as well as matrix-vector and matrix-matrix multiplication using template expressions. A dense linear algebra matrix solver based on an LU decomposition is also included in the template expressions. The library utilizes Basic Linear Algebra Subprograms (BLAS) routines when available to perform each calculation of the expression tree. A number of different BLAS libraries that are optimized for different compilers and/or hardware architectures are available. The Automatically Tuned Linear Algebra Software (ATLAS)[125, 126] has been commonly used with the dense linear algebra library to optimize its performance. Routines written in C++ that are

equivalent to the BLAS routines are included in the library in case a BLAS library is not available on the system where the code is compiled.

One limitation with Templates Expressions is that it is not possible to use the same vector x in a matrix-vector multiplication with the matrix M on both the left and right hand side of an expression, i.e.,

$$x = Mx \quad (3.2)$$

because the memory for the left hand side is used in the calculations on the right hand side. A run-time logical check (i.e. assertion) will halt the code in this situation. This calculation fundamentally requires the use of a temporary vector.

The block form of the Thomas algorithm shown in Algorithm 3.3 that solves the block tri-diagonal system

$$\begin{bmatrix} B_0 & C_0 \\ A_1 & B_1 & C_1 \\ A_2 & B_2 & C_2 \\ \ddots & \ddots & \ddots \\ A_{N-2} & B_{N-2} & C_{N-2} \\ A_{N-1} & B_{N-1} & C_{N-1} \\ A_N & B_N \end{bmatrix} \begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ \vdots \\ X_{N-2} \\ X_{N-1} \\ X_N \end{bmatrix} = \begin{bmatrix} D_0 \\ D_1 \\ D_2 \\ \vdots \\ D_{N-2} \\ D_{N-1} \\ D_N \end{bmatrix} \quad (3.3)$$

that is implemented using the dense linear algebra library is shown in Table 3.10. Note that the syntax of the dense linear algebra library in Table 3.10 closely resembles the syntax of Algorithm 3.3. This C++ code has the same execution time as an equivalent code written in the FORTRAN programming language.

Algorithm 3.3 Block Tri-diagonal Solver Based on the Thomas Algorithm

/* Perform the forward sweep */

$$(C_0, D_0) := B_0^{-1} (C_0, D_0)$$

for i = 1 to N-1

$$\alpha := B_i - A_i C_{i-1}$$

$$(C_i, D_i) := \alpha^{-1} (C_i, D_i - A_i D_{i-1})$$

/* Compute the last elements */

$$\alpha := B_N - A_N C_{N-1}$$

$$D_N := \alpha^{-1} (D_N - A_N D_{N-1})$$

/* Perform the back substitution */

$$X_N := D_N$$

for i = N-1 to 0

$$X_i := D_i - C_i X_{i+1}$$

Table 3.10: Block Tri-Diagonal Solver Based on the Thomas Algorithm Using Template Expressions

```

// Solves the block tri-diagonal system using the Thomas algorithm
// Fills solution into D. Warning: will modify C and B!!
template<int n, class T>
void BlockTridiagSolve( std::vector< TLA::TinyMatrix<n,n,T> >& A
                      , std::vector< TLA::TinyMatrix<n,n,T> >& B
                      , std::vector< TLA::TinyMatrix<n,n,T> >& C
                      , std::vector< TLA::TinyMatrix<n,1,T> >& D )
{
    int i;
    const int size = static_cast<int>(B.size());
    TLA::TinyMatrix<n,n,T> alpha;

    // Perform the forward sweep
    (C[0], D[0]) = !B[0]*(C[0], D[0]);

    for(i = 1; i < size - 1; i++)
    {
        alpha = (B[i] - A[i]*C[i-1]);
        (C[i], D[i]) = !alpha*(C[i], (D[i] - A[i]*D[i-1]));
    }

    // Compute the last element
    i = size - 1;
    alpha = (B[i] - A[i]*C[i-1]);
    D[i] = !alpha*(D[i] - A[i]*D[i-1]);

    // Now back substitute
    for(i = size - 2; i >= 0; i--)
        D[i] -= C[i]*D[i+1];
}

```

3.1.3 Block-Sparse Iterative Matrix Solver Library

The sparse block matrix solver library is built upon the dense linear algebra library. Each block in the sparse matrix is stored in a dense form using the dense linear algebra library. The sparse block matrix is either a block tri-, penta-, or hepta-diagonal matrix for one-, two-, and three-dimensional problems. The sparse matrix library uses a block tri-diagonal version of the Thomas algorithm[127] to solve block tri-diagonal matrices. For penta- and hepta-digonal matrices, the sparse matrix library relies on iterative solution procedures to solve the linear system. Template Expressions are used so that the syntax of the code directly

resembles pseudo code used to describe the algorithms. The primary solver used is the Flexible version of the General Minimum Residual (FGMRES)[128] iterative solver. However, the FGMRES solver may have a poor convergence rate without the aid of a preconditioner. Three different preconditioners are currently included in the library. In the order of increasing computational cost, they are: Lower Upper Symmetric Gauss-Seidel (LU-SGS)[121], Incomplete Lower Upper decomposition without any fill in (ILU(0))[128], and Incomplete Lower Upper decomposition with one level of fill in (ILU(1))[128]. Each preconditioner operates on the blocks in the sparse block matrix. Choosing the correct preconditioner is a compromise between computational cost and the effectiveness of the preconditioner to minimize the number of FGMRES iterations required to obtain a solution to the linear system. The relatively small time step required for time accurate calculations increases the diagonal dominance of the sparse block matrix. Hence, the relatively inexpensive LU-SGS preconditioner is adequate to converge the FMGRES solver generally in less than 20 iterations. Steady state problems lack the diagonal dominance that comes with using a small time step and the more expensive ILU(1) preconditioner tends to converge the FGMRES solver significantly faster for the problems considered in this dissertation than the LU-SGS preconditioner. Details of the preconditioners are give in Appendix C.

3.2 Flux and Flux Jacobian Integration

The flux and flux Jacobian integration part of the code is a general framework for evaluating the spatial integrals in Eq. 2.12. The integration framework does not require specific details of the partial differential equations. C++ templates are used to specialize the framework for a particular set of partial differential equations. Hence, the same integration code is used regardless of the partial differential equations. The template arguments contain subroutines that take cell local quantities to compute fluxes and flux Jacobians. As a result, all flux and flux Jacobian subroutines have the same set of arguments. A properties data type tailored for each partial differential equation is used to pass quantities that are not cell local to the flux and Jacobian subroutines. Non-cell local quantities include, for example, the ratio of specific heats, γ , reference Mach number, M_∞ , time step, Δt , and solution time to name a few.

The specific set of subroutines required by the framework depends on the partial differential equations. For equations with advective, subroutines that compute the advective flux and Jacobian

$$\vec{F}^a(Q), \quad \frac{\partial \vec{F}^a(Q)}{\partial Q}, \quad (3.4)$$

as well as the upwinding flux and the Jacobians of the upwinding flux with respect to both the interior and exterior variable vectors

$$\begin{aligned} & \phi(Q^+, Q^-) \\ & \frac{\partial \phi(Q^+, Q^-)}{\partial Q^+} \\ & \frac{\partial \phi(Q^+, Q^-)}{\partial Q^-} \end{aligned} \quad (3.5)$$

are required. Equations with diffusion require the diffusive flux and the Jacobians with respect to both the variable vector and the gradient of the variable vector

$$\begin{aligned} & \vec{F}^d(Q, \nabla Q), \\ & \frac{\partial \vec{F}^d(Q, \nabla Q)}{\partial Q}, \\ & \frac{\partial \vec{F}^d(Q, \nabla Q)}{\partial \nabla Q}. \end{aligned} \quad (3.6)$$

Partial differential equations with advection require boundary condition values and Jacobians with respect to the interior variable vector

$$\begin{aligned} & Q_B(Q^-, Q_b), \\ & \frac{\partial Q_B(Q^-, Q_b)}{\partial Q^-}, \end{aligned} \quad (3.7)$$

while equations with diffusion also require gradients of the dependent variables and their Jacobians with

respect to the interior gradient of the dependent variables at the boundary

$$\begin{aligned} & \nabla Q_B (\nabla Q^-, \nabla Q_b), \\ & \frac{\partial \nabla Q_B (\nabla Q^-, \nabla Q_b)}{\partial \nabla Q^-}. \end{aligned} \quad (3.8)$$

If the partial differential equations contain a source, a subroutine that evaluates the source and its Jacobian with respect to the dependent variables and its gradient

$$\begin{aligned} & S(Q, \nabla Q), \\ & \frac{\partial S(Q, \nabla Q)}{\partial Q}, \\ & \frac{\partial S(Q, \nabla Q)}{\partial \nabla Q}, \end{aligned} \quad (3.9)$$

are required. A few of the Jacobians associated with the diffusive fluxes are introduced by the solution procedure that are independent of the partial differential equation, namely

$$\frac{\partial \nabla Q}{\partial Q}, \frac{\partial \vec{r}}{\partial Q^+}, \frac{\partial \vec{r}}{\partial Q^-}, \frac{\partial \vec{r}_b(Q^-, Q_B)}{\partial Q^-}, \text{ and } \frac{\partial \vec{r}_b(Q^-, Q_B)}{\partial Q_B}, \quad (3.10)$$

and are computed by the framework regardless of the governing equations.

The framework sweeps through the cells of the computational grid and computes the volume integrals in Eq. 2.12 for each cell. A “compute once and apply twice” philosophy for cell boundary integrals is used to minimize the computational cost. The boundary flux is integrated twice on the boundary; once multiplied by the test function from the interior cell, and once multiplied by the test function from the exterior cell. The boundary condition fluxes are integrated after all the integrals interior to the computational domain are computed. The remainder of this section outlines the details of the evaluation and integration of the fluxes.

3.2.1 Cell Curvilinear Coordinates

Each cell in the grid is mapped to a normalized reference element in order to evaluate the integrals in Eq. 2.12. The coordinate transformation is a mapping of the Cartesian coordinates of a cell to curvilinear coordinates. A cell coordinate Jacobian matrix is then required to formulate the integrals in Eq. 2.12 in terms of the curvilinear coordinates using a substitution of variables.

3.2.1.1 Grid Polynomial Mapping

In one dimension, the coordinate mapping of a cell in Cartesian coordinates to the curvilinear coordinate, ξ , is

$$x(\xi) = \bar{x} + \frac{\Delta x}{2} \xi \quad (3.11)$$

where \bar{x} is the cell center and Δx is the cell width. For higher-dimensions, the code is formulated for structured meshes. These meshes can be generated using a traditional structured mesh generator. The Cartesian coordinates of a cell are represented with cell local polynomials $(x(\xi, \eta), y(\xi, \eta))$. The mapping of the Cartesian cell coordinates is formulated as a polynomial expansion using the same test functions, ψ , that are used in the DG discretization of the governing equations. Hence,

$$\begin{aligned} x(\xi, \eta) &= \sum_{i=0}^{N_g} \sum_{j=0}^{N_g} x_{ij} \psi_{ij}(\xi, \eta) \\ y(\xi, \eta) &= \sum_{i=0}^{N_g} \sum_{j=0}^{N_g} y_{ij} \psi_{ij}(\xi, \eta) \end{aligned} \quad (3.12)$$

where N_g is the order of the polynomial mapping, x_{ij} and y_{ij} are the coefficients of the expansion, and $\psi_{ij}(\xi, \eta) = P_i(\xi)P_j(\eta)$. The coefficients in the polynomial expansions are found by equating the expansions in Eq. 3.12 with cell nodal values that define a cell. For example, the following system of equations is solved to find the coefficients of $x(\xi, \eta)$ for the linear cell shown in Fig. 3.2a.

$$\begin{pmatrix} \psi_{00}(-1, -1) & \psi_{10}(-1, -1) & \psi_{01}(-1, -1) & \psi_{11}(-1, -1) \\ \psi_{00}(1, -1) & \psi_{10}(1, -1) & \psi_{01}(1, -1) & \psi_{11}(1, -1) \\ \psi_{00}(-1, 1) & \psi_{10}(-1, 1) & \psi_{01}(-1, 1) & \psi_{11}(-1, 1) \\ \psi_{00}(1, 1) & \psi_{10}(1, 1) & \psi_{01}(1, 1) & \psi_{11}(1, 1) \end{pmatrix} \begin{pmatrix} x_{00} \\ x_{10} \\ x_{01} \\ x_{11} \end{pmatrix} = \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} \quad (3.13)$$

The same process is repeated for the y coordinate to obtain the complete polynomial mapping of the cell. Additional nodes are required for a higher-order polynomial representation. For example, a quadratic cell requires nine nodal values as shown in Fig. 3.2b. Hence, assuming the same polynomial order of the mapping is used in each direction, a DG computational grid size of $m \times n$ cells requires $(N_g m + 1) \times (N_g n + 1)$ nodes.

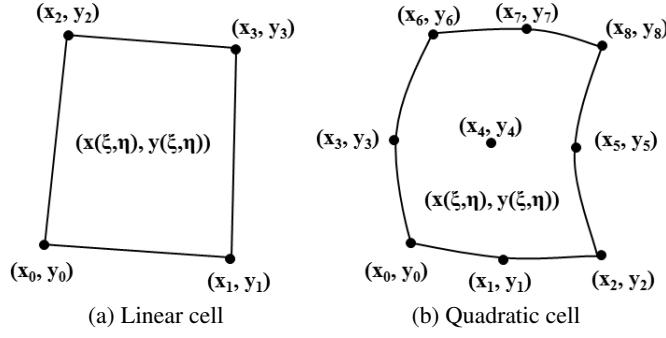


Figure 3.2: Nodal and modal representation of cells.

In three-dimensions the cell local representation, $(x(\xi, \eta, \zeta), y(\xi, \eta, \zeta), z(\xi, \eta, \zeta))$, of the cell coordinates are

$$\begin{aligned} x(\xi, \eta, \zeta) &= \sum_i^{N_g} \sum_j^{N_g} \sum_k^{N_g} x_{ijk} \psi_{ijk}(\xi, \eta, \zeta) \\ y(\xi, \eta, \zeta) &= \sum_i^{N_g} \sum_j^{N_g} \sum_k^{N_g} y_{ijk} \psi_{ijk}(\xi, \eta, \zeta) \\ z(\xi, \eta, \zeta) &= \sum_i^{N_g} \sum_j^{N_g} \sum_k^{N_g} z_{ijk} \psi_{ijk}(\xi, \eta, \zeta) \end{aligned} \quad (3.14)$$

where x_{ijk} , y_{ijk} and z_{ijk} are the coefficients of the expansion, and $\psi_{ijk}(\xi, \eta, \zeta) = P_i(\xi)P_j(\eta)P_k(\zeta)$. Eight nodes, shown in Fig. 3.3a, are used with a linear mapping, $N_g = 1$, and a quadratic mapping, $N_g = 2$, uses 27 nodes, as shown in Fig. 3.3b. A grid size of $l \times m \times n$ cells requires $(N_g l + 1) \times (N_g m + 1) \times (N_g n + 1)$ nodes.

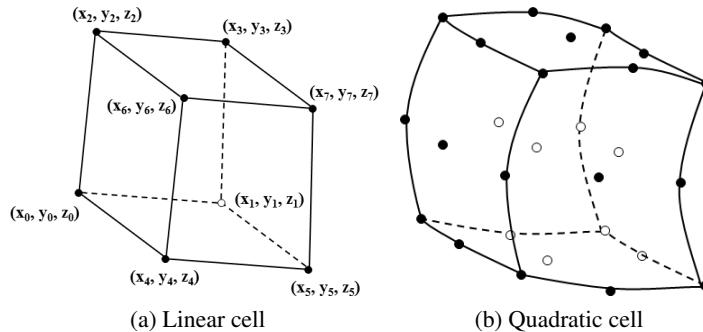


Figure 3.3: Nodes used to form 3D Cells

3.2.1.2 Cell Coordinate Jacobian Matrix

The cell coordinate Jacobian matrix, T , is required in order to express the Cartesian gradient, ∇_x , of a scalar quantity in terms of curvilinear gradients, ∇_ξ , i.e.

$$\nabla_x u = T^T \nabla_\xi u. \quad (3.15)$$

In addition, the Jacobian, J^{-1} , of the Jacobian matrix inverse, T^{-1} , is required to express the integrals in Eq. 2.12 in terms of cell coordinates using substitution of variables. Note that a constraint on the grid is that the mapping T^{-1} must exist. The cell Jacobian matrix relates an increment in Cartesian coordinates, $d\vec{x}$, to a cell curvilinear coordinate increment, $d\vec{\xi}$, as

$$d\vec{\xi} = T d\vec{x}, \quad (3.16)$$

by the chain rule. The mapping T is obtained from the inverse of T^{-1} , which maps an increment in cell curvilinear coordinates to Cartesian coordinates

$$d\vec{x} = T^{-1} d\vec{\xi}. \quad (3.17)$$

The mapping T^{-1} is computed directly using derivatives of the local cell mapping. The determinant of T^{-1} is the Jacobian, J^{-1} , required to express the integrals in terms of curvilinear coordinates. The coordinate Jacobian matrices for one-, two-, and three-dimensions given in the following assume that the grid does not deform in time. For deforming grids, the derivative of the Cartesian coordinates with respect to a curvilinear time would also need to be included in the Coordinate transformation.

1D Coordinate Jacobian Matrix: In one-dimension, Eq. 3.17 is

$$dx = x_\xi d\xi, \quad (3.18)$$

which implies that the Jacobian inverse matrix is

$$T^{-1} = J^{-1} = x_\xi = \frac{\Delta x}{2}. \quad (3.19)$$

The coordinate Jacobian matrix is thus

$$T = \xi_x = \frac{1}{x_\xi} = \frac{2}{\Delta x}. \quad (3.20)$$

2D Coordinate Jacobian Matrix: Equation 3.17 in two-dimensions is

$$\begin{aligned} dx &= x_\xi d\xi + x_\eta d\eta, \\ dy &= y_\xi d\xi + y_\eta d\eta, \end{aligned} \quad (3.21)$$

which yields the inverse coordinate Jacobian matrix and the inverse Jacobian as

$$\begin{aligned} T^{-1} &= \begin{pmatrix} x_\xi & x_\eta \\ y_\xi & y_\eta \end{pmatrix}, \\ J^{-1} = \det(T^{-1}) &= x_\xi y_\eta - x_\eta y_\xi. \end{aligned} \quad (3.22)$$

Hence, the coordinate Jacobian matrix is

$$T = J \begin{pmatrix} \xi_x & \xi_y \\ \eta_x & \eta_y \end{pmatrix} = J \begin{pmatrix} y_\eta & -x_\eta \\ -y_\xi & x_\xi \end{pmatrix} \equiv J\mathcal{T}. \quad (3.23)$$

3D Coordinate Jacobian Matrix: In three-dimensions, Eq. 3.17 is

$$\begin{aligned} dx &= x_\xi d\xi + x_\eta d\eta + x_\zeta d\zeta, \\ dy &= y_\xi d\xi + y_\eta d\eta + y_\zeta d\zeta, \\ dz &= z_\xi d\xi + z_\eta d\eta + z_\zeta d\zeta, \end{aligned} \quad (3.24)$$

which yields the inverse coordinate Jacobian matrix and the inverse Jacobian as

$$\begin{aligned}
T^{-1} &= \begin{pmatrix} x_\xi & x_\eta & x_\zeta \\ y_\xi & y_\eta & y_\zeta \\ z_\xi & z_\eta & z_\zeta \end{pmatrix}, \\
J^{-1} = \det(T^{-1}) &= (x_\xi y_\eta z_\zeta - x_\eta y_\xi z_\zeta - x_\xi y_\zeta z_\eta + x_\zeta y_\xi z_\eta + x_\eta y_\zeta z_\xi - x_\zeta y_\eta z_\xi) \\
&= (x_\xi (y_\eta z_\zeta - y_\zeta z_\eta) + x_\eta (y_\zeta z_\xi - y_\xi z_\zeta) + x_\zeta (y_\xi z_\eta - y_\eta z_\xi)) \\
&= (x_\xi \xi_x + x_\eta \eta_x + x_\zeta \zeta_x). \tag{3.25}
\end{aligned}$$

The coordinate Jacobian matrix is thus

$$T = J \begin{pmatrix} \xi_x & \xi_y & \xi_z \\ \eta_x & \eta_y & \eta_z \\ \zeta_x & \zeta_y & \zeta_z \end{pmatrix} = J \begin{pmatrix} y_\eta z_\zeta - y_\zeta z_\eta & x_\zeta z_\eta - x_\eta z_\zeta & x_\eta y_\zeta - x_\zeta y_\eta \\ y_\zeta z_\xi - y_\xi z_\zeta & x_\xi z_\zeta - x_\zeta z_\xi & x_\zeta y_\xi - x_\xi y_\zeta \\ y_\xi z_\eta - y_\eta z_\xi & x_\eta z_\xi - x_\xi z_\eta & x_\xi y_\eta - x_\eta y_\xi \end{pmatrix} \equiv J\mathcal{T}. \tag{3.26}$$

3.2.1.3 Curvilinear Polynomials

Let $\phi(\vec{\xi})$ be a general curvilinear polynomial defined over the interval $\vec{\xi} \in [-1, 1]$. This function must be related to a polynomial $\hat{\phi}(\vec{x})$ in Cartesian coordinates in order to express the integrals in Eq. 2.12 in curvilinear coordinates. The coordinate transformation is achieved by defining the Cartesian polynomial in terms of the curvilinear polynomial as

$$\hat{\phi}(\vec{x}) = \phi(\vec{\xi}(\vec{x})). \tag{3.27}$$

Then, the transformation from Cartesian to curvilinear coordinates of volume and boundary integrals will evaluate the Cartesian polynomial as a function of Curvilinear coordinates, which results in the original Curvilinear polynomial, i.e.,

$$\hat{\phi}(\vec{x}(\vec{\xi})) = \phi(\vec{\xi}) \tag{3.28}$$

This process is demonstrated here on a one-dimensional polynomial. Let the coordinate of a one-dimensional cell be defined by

$$x(\xi) = \bar{x} + \frac{\Delta x}{2}\xi, \quad (3.29)$$

where \bar{x} is the cell center and Δx is the cell width. The Curvilinear coordinate ξ in terms of the Cartesian coordinate x is

$$\xi(x) = \frac{2(x - \bar{x})}{\Delta x}. \quad (3.30)$$

Let the curvilinear polynomial be the linear polynomial

$$\phi(\xi) = \phi_0 + \xi\phi_1. \quad (3.31)$$

The polynomial in Cartesian coordinates is defined from the curvilinear polynomial as

$$\hat{\phi}(x) = \phi(\xi(x)) = \phi_0 + \frac{2(x - \bar{x})}{\Delta x}\phi_1 \quad (3.32)$$

Transforming the Cartesian polynomial into curvilinear coordinates now results in the original curvilinear polynomial, i.e.,

$$\hat{\phi}(x(\xi)) = \phi_0 + \frac{2(\bar{x} + \frac{\Delta x}{2}\xi - \bar{x})}{\Delta x}\phi_1 = \phi_0 + \xi\phi_1 = \phi(\xi) \quad (3.33)$$

3.2.1.4 Integrals in Curvilinear Coordinates

The volume integral in Eq. 2.12 that integrates the gradient of the test function, $\nabla_x \hat{\psi}(\vec{x})$, multiplied with a flux vector, is transformed into curvilinear coordinates using a substitution of variables as

$$\begin{aligned} \int_{\Omega} \nabla_x \hat{\psi}(\vec{x}) \cdot \hat{\phi}(\vec{x}) d\vec{x} &= \int_{\Omega} \nabla_x \hat{\psi}(\vec{x}(\vec{\xi})) \cdot \hat{\phi}(\vec{x}(\vec{\xi})) J^{-1} d\vec{\xi} \\ &= \int_{\Omega} J \mathcal{T}^T \nabla_{\xi} \psi(\vec{\xi}) \cdot \vec{\phi}(\vec{\xi}) J^{-1} d\vec{\xi} \\ &= \int_{\Omega} \mathcal{T}^T \nabla_{\xi} \psi(\vec{\xi}) \cdot \vec{\phi}(\vec{\xi}) d\vec{\xi}. \end{aligned} \quad (3.34)$$

Note that the integral in Eq. 3.34 is simplified by the cancellation in the Jacobians associated with the gradient and substitution of variables. The volume integral of a polynomial, $\hat{\phi}(\vec{x})$, multiplied with the test

function $\hat{\psi}(\vec{x})$, in source terms of Eq. 2.12 is expressed in curvilinear coordinates using substitution of variables as

$$\begin{aligned}\int_{\Omega} \hat{\psi}(\vec{x}) \hat{\phi}(\vec{x}) d\vec{x} &= \int_{\Omega} \hat{\psi}(\vec{x}(\vec{\xi})) \hat{\phi}(\vec{x}(\vec{\xi})) J^{-1} d\vec{\xi} \\ &= \int_{\Omega} \psi(\vec{\xi}) \phi(\vec{\xi}) J^{-1} d\vec{\xi}\end{aligned}\quad (3.35)$$

The normal vector of any cell face in the boundary integrals in Eq. 2.12 is given by

$$\vec{n}_{\xi_k} = \frac{\partial \vec{x}}{\partial \xi_i} \times \frac{\partial \vec{x}}{\partial \xi_j}, \quad (3.36)$$

where ξ_i and ξ_j are the curvilinear coordinates in the plane of the cell face and ξ_k is the curvilinear coordinate out of to the cell face. For three-dimensions, the normal vectors for a ξ constant, η constant, and ζ constant face are

$$\begin{aligned}\vec{n}_{\xi} &= \frac{\partial \vec{x}}{\partial \eta} \times \frac{\partial \vec{x}}{\partial \zeta} = \begin{bmatrix} y_{\eta} z_{\zeta} - y_{\zeta} z_{\eta}, & x_{\zeta} z_{\eta} - x_{\eta} z_{\zeta}, & x_{\eta} y_{\zeta} - x_{\zeta} y_{\eta} \end{bmatrix} = \begin{bmatrix} \xi_x, & \xi_y, & \xi_z \end{bmatrix}, \\ \vec{n}_{\eta} &= \frac{\partial \vec{x}}{\partial \zeta} \times \frac{\partial \vec{x}}{\partial \xi} = \begin{bmatrix} y_{\zeta} z_{\xi} - y_{\xi} z_{\zeta}, & x_{\xi} z_{\zeta} - x_{\zeta} z_{\xi}, & x_{\zeta} y_{\xi} - x_{\xi} y_{\zeta} \end{bmatrix} = \begin{bmatrix} \eta_x, & \eta_y, & \eta_z \end{bmatrix}, \\ \vec{n}_{\zeta} &= \frac{\partial \vec{x}}{\partial \xi} \times \frac{\partial \vec{x}}{\partial \eta} = \begin{bmatrix} y_{\xi} z_{\eta} - z_{\xi} y_{\eta}, & z_{\xi} x_{\eta} - x_{\xi} z_{\eta}, & x_{\xi} y_{\eta} - y_{\xi} x_{\eta} \end{bmatrix} = \begin{bmatrix} \zeta_x, & \zeta_y, & \zeta_z \end{bmatrix}.\end{aligned}\quad (3.37)$$

The normal vectors in two-dimensions are

$$\begin{aligned}\vec{n}_{\xi} &= \frac{\partial \vec{x}}{\partial \eta} = \begin{bmatrix} y_{\eta}, & -x_{\eta} \end{bmatrix} = \begin{bmatrix} \xi_x, & \xi_y \end{bmatrix}, \\ \vec{n}_{\eta} &= \frac{\partial \vec{x}}{\partial \xi} = \begin{bmatrix} -y_{\xi}, & x_{\xi} \end{bmatrix} = \begin{bmatrix} \eta_x, & \eta_y \end{bmatrix},\end{aligned}\quad (3.38)$$

on ξ and η constant faces, respectively. Normal vectors are obtained by taking the trace of the coordinate Jacobian matrix on the face of the cell. The normal vectors written in Eq. 3.38 are illustrated in Fig. 3.4. The normal vectors on the faces with $\xi = 1$ and $\eta = 1$ are outward facing, while the normal vectors on the faces with $\xi = -1$ and $\eta = -1$ are inward facing. Hence, the boundary integrals on the boundaries $\xi = -1$ and $\eta = -1$ must be negated to obtain outward facing normal vectors. The same is true in three-dimensions in Eq. 3.37. In one-dimension the normal vector is set to unity with the correct sign. Boundary integrals in

Eq. 2.12 are expressed in curvilinear coordinates as

$$\begin{aligned} \int_{\Gamma} \hat{\psi}(\vec{x}) \hat{\phi}(\vec{x}) \cdot \vec{n} d\Gamma &= \int_{-1}^1 \int_{-1}^1 \hat{\psi}(\vec{x}(\vec{\xi})) \hat{\phi}(\vec{x}(\vec{\xi})) \cdot \vec{n}_{\xi_k}(\vec{\xi}) d\xi_i d\xi_j \\ &= \int_{-1}^1 \int_{-1}^1 \psi(\vec{\xi}) \phi(\vec{\xi}) \cdot \vec{n}_{\xi_k}(\vec{\xi}) d\xi_i d\xi_j, \end{aligned} \quad (3.39)$$

where ξ_k is the curvilinear coordinate normal to the cell face.

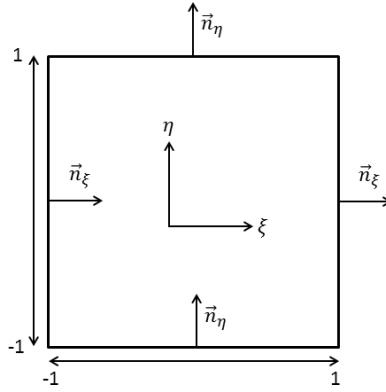


Figure 3.4: Illustration of the Two-Dimensional Unit Cell

3.2.2 Analytical Quadrature-free Integration

Traditionally, the spatial integrals of the DG scheme are evaluated by Gaussian Quadrature formulas due to their high accuracy and efficiency. In one dimension, a polynomial of degree $2N_{GQ}-1$ can be integrated exactly with N_{GQ} nodal values of the polynomial. In a DG formulation, a product of two N order polynomials can be integrated exactly with $N_{GQ} = N + 1$ nodal values. However, the quadrature formulas are only efficient if the information is readily available at the quadrature nodes. Unfortunately, no single set of quadrature nodes exists that can be used for the both volume and boundary integrals while maintaining the required accuracy of the integration.[29] One possible solution to this problem is to store the solution as a modal representation, and evaluate the polynomials at the quadrature nodes. However, this approach incurs a significant computational expense as it requires $N_{GQ}(N + 1)$ operations per node to evaluate a polynomial in higher dimensions. Hence, the volume integral would require $O((N + 1)^2)$ operations. The operation count can be reduced to $O(N + 1)$ by using a quadrature free formulation where the integrals are evaluated

analytically (at least for linear terms). Atkins and Shu[29], and Lockard and Atkins[30] describe such a formulation in which the integrals are computed and stored as matrices. They also developed special C language macros that unrolled the matrix-vector multiplication and operated only on non-zero elements of the matrices to further improve the algorithm. However, the matrix must exhibit a predictable pattern of non-zero elements to employ their technique. Lockard and Atkins[30] suggested a special ordering of the basis functions to simplify the pattern of non-zero elements. However, the cell Jacobian must also be a constant in order to compute a single matrix representative of the integration for all cells in a mesh; only linear triangular elements and quadrilateral elements aligned in a Cartesian coordinate system have Jacobians that are constant. If this technique of storing an integration were applied to general curved cells, matrices representing the integrals would need to be computed and stored for each mesh cell, a process that is likely to require significant storage. Furthermore, the development of C macros to unroll and optimize the matrix-vector multiplications would be much more complex.

The technique of computing a matrix representing the integrals and using C macros to unroll the matrix-vector multiplications is extended here. Rather than computing the integrals and storing the resulting matrices in memory during execution, the author used a custom pre-processor to compute the integrals prior to compile time. Similar to the C macro approach[30], the matrix-vector multiplications are expanded by the pre-processor and all zero entries removed. However, unlike the C macro approach, which must be re-designed for any given matrix-vector multiplication, the pre-processor code is able to automatically deduce zero entries in the matrix and remove them.

3.2.2.1 PyDG Preprocessor

The analytical integration of the product of polynomials is performed using a pre-processing tool (PyDG) that is written in the Python programming language, and relies on the symbolic manipulation package Sympy[129] and the code parsing package Blockit[130]. The PyDG pre-processor is developed to transform analytical expressions into C++ code.

The PyDG tool stores polynomial expansions as arrays of coefficients and polynomial products. These storage arrays are detailed below to avoid confusing them with standard vector and matrix notation. The polynomials are known to the compiler, but the coefficients are merely placeholders that are converted into

C++ code. For example, a 3rd-order expansion of the variable u can be viewed as

$$u(\xi) = \sum_{i=0}^2 u_i \psi_i(\xi) \Leftrightarrow \left\{ \begin{array}{lll} u[0] \psi_0 & u[1] \psi_1 & u[2] \psi_2 \end{array} \right\}, \quad (3.40)$$

where $u[i]$ represents an element of the array u using C++ notation. Similarly, the polynomial test functions are also stored as an array of polynomials, i.e.,

$$\psi_i(\xi), \quad i = 0, 1, 2 \Leftrightarrow \left\{ \begin{array}{lll} \psi_0 & \psi_1 & \psi_2 \end{array} \right\}. \quad (3.41)$$

Next, consider the multiplication $\psi_i u$, which is really three equations $\psi_0 u$, $\psi_1 u$, and $\psi_2 u$. This operation is performed by multiplying the expansion of u with each entry of the ψ array

$$\psi_i u = \left(\begin{array}{c} \psi_0 \left\{ \begin{array}{lll} u[0] \psi_0 & u[1] \psi_1 & u[2] \psi_2 \end{array} \right\} \\ \psi_1 \left\{ \begin{array}{lll} u[0] \psi_0 & u[1] \psi_1 & u[2] \psi_2 \end{array} \right\} \\ \psi_2 \left\{ \begin{array}{lll} u[0] \psi_0 & u[1] \psi_1 & u[2] \psi_2 \end{array} \right\} \end{array} \right) = \left(\begin{array}{c} \left\{ \begin{array}{lll} u[0] \psi_0^2 & u[1] \psi_0 \psi_1 & u[2] \psi_0 \psi_2 \end{array} \right\} \\ \left\{ \begin{array}{lll} u[0] \psi_1 \psi_0 & u[1] \psi_1^2 & u[2] \psi_1 \psi_2 \end{array} \right\} \\ \left\{ \begin{array}{lll} u[0] \psi_2 \psi_0 & u[1] \psi_2 \psi_1 & u[2] \psi_2^2 \end{array} \right\} \end{array} \right), \quad i = 0, 1, 2. \quad (3.42)$$

The polynomial products $\psi_i \psi_j$ are now expanded and evaluated using the symbolic manipulator, and integrated over the interval $\xi \in [-1, 1]$ to give

$$\int_{-1}^1 \psi_i u d\xi = \left(\begin{array}{c} \left\{ \begin{array}{lll} u[0]^2 & 0 & 0 \end{array} \right\} \\ \left\{ \begin{array}{lll} 0 & u[1] \frac{2}{3} & 0 \end{array} \right\} \\ \left\{ \begin{array}{lll} 0 & 0 & u[2] \frac{2}{5} \end{array} \right\} \end{array} \right), \quad i = 0, 1, 2. \quad (3.43)$$

Note that orthogonality of ψ_i leaves only the diagonal elements as non-zero. For simplicity, the subscript on ψ_i is dropped and is implied in future expressions involving weak formulations of an equation. The syntax for performing this product using the PyDG pre-processor is shown in Table 3.11.

Table 3.11: PyDG Code for Performing Polynomial Expansion Products

<pre> u = DGPolyVector('u' , poly=legendre , order=[2] , coord=[xi]) psi = DGPolyVector('' , poly=legendre , order=[2] , coord=[xi]) psiu = (psi*u).integrate((xi , -1, 1)) </pre>

The storage of the integral in Eq. 3.43 can be converted to either a matrix vector product or a vector depending on the need, i.e.

$$\int_{-1}^1 \psi_i u d\xi \Rightarrow \begin{pmatrix} 2 & 0 & 0 \\ 0 & \frac{2}{3} & 0 \\ 0 & 0 & \frac{2}{5} \end{pmatrix} \begin{pmatrix} u[0] \\ u[1] \\ u[2] \end{pmatrix} \text{ or } \begin{pmatrix} u[0] 2 \\ u[1] \frac{2}{3} \\ u[2] \frac{2}{5} \end{pmatrix}. \quad (3.44)$$

The syntax for these two operations are shown in Table 3.12. The integral is converted to a matrix by passing to the PyDG pre-processor the variable that is the vector in the matrix vector multiplication. The vector representation is achieved by summing the polynomial storage arrays horizontally in Eq. 3.43.

Table 3.12: PyDG Code for Converting an Integrated Polynomial Expression to a Linear Algebra Expression

Matrix	Vector
A = psiu . Matrix('u')	b = psiu . Vector()

The PyDG pre-processor is able to generate the C++ code required for a number of expressions used to compute the residual vector $\mathcal{R}(Q)$ and Jacobian matrix $\frac{\partial \mathcal{R}(Q)}{\partial Q}$ of the partial differential equations.

3.2.2.2 DGPoly++ C++ Polynomial Library

For efficiency, common operations, such as addition, subtraction, multiplication, and division of polynomials, have been generated using the PyDG pre-processor and collected into a C++ polynomial library named DGPoly++. This C++ library uses template expressions to allow polynomial expressions written in C++ code to be treated as if they were scalar quantities (with a few restrictions). The addition and subtraction operators are simple vector addition and subtraction operators. More care is required with the multiplication, division, and non-polynomial function operators. The library also contains operations for integrating fluxes and forming the block matrices in the Jacobian matrix $\frac{\partial \mathcal{R}(Q)}{\partial Q}$.

Hierarchical Polynomial Ordering: Basis functions in two- and three-dimensions are expressed as tensor products of one-dimensional basis functions, i.e.,

$$\begin{aligned} \sum_{i=0}^N \sum_{j=0}^N \psi_i(\xi) \psi_j(\eta), & \quad 2D \\ \sum_{i=0}^N \sum_{j=0}^N \sum_{k=0}^N \psi_i(\xi) \psi_j(\eta) \psi_k(\zeta). & \quad 3D \end{aligned} \quad (3.45)$$

The tensor product of the basis functions are often expressed in form of a table as shown in Fig. 3.5 for a two-dimensional tensor product with $N = 2$. This table also shows how the basis functions are nested as the order of the approximation increases. The DGPoly++ library takes advantage of this nested feature of the tensor product by ordering the modal coefficients in the order shown in Fig. 3.6. The mean value of the cell associated with ψ_{00} is always the first index in the u array. The linear approximation uses the first 4 coefficients of the array, and the quadratic approximation uses 9 coefficients. The u array is only as large as it needs to be to represent a given order of accuracy. However, this modal coefficient ordering simplifies the process of changing the order of accuracy.

$$N = 2 \quad \left\{ \quad N = 1 \quad \left\{ \quad N = 0 \{ \begin{array}{|c|c|c|} \hline \psi_{00} & \psi_{01} & \psi_{02} \\ \hline \psi_{10} & \psi_{11} & \psi_{12} \\ \hline \psi_{20} & \psi_{21} & \psi_{22} \\ \hline \end{array} \right\} \right\}$$

Figure 3.5: Table of Tensor Product of Basis Functions where $\psi_{ij} = \psi_i(\xi) \psi_j(\eta)$

$$\left. \begin{array}{l} u[0] \psi_{00} \\ u[1] \psi_{01} \\ u[2] \psi_{10} \\ u[3] \psi_{11} \\ u[4] \psi_{02} \\ u[5] \psi_{20} \\ u[6] \psi_{12} \\ u[7] \psi_{21} \\ u[8] \psi_{22} \end{array} \right\} N = 0 \quad \left. \begin{array}{l} \left. \begin{array}{l} u[0] \psi_{00} \\ u[1] \psi_{01} \\ u[2] \psi_{10} \\ u[3] \psi_{11} \end{array} \right\} N = 1 \end{array} \right\} N = 1 \quad \left. \begin{array}{l} \left. \begin{array}{l} u[0] \psi_{00} \\ u[1] \psi_{01} \\ u[2] \psi_{10} \\ u[3] \psi_{11} \\ u[4] \psi_{02} \end{array} \right\} N = 1 \end{array} \right\} N = 2$$

Figure 3.6: Modal Coefficient Ordering of Nested Orders of the Approximation

Polynomial Multiplication: The first assumption is that any product of two polynomial expansions is projected onto the same basis functions as the original expansions. This is achieved through an L^2 -projection, shown here for computing u^2

$$\int \psi u^2 d\Omega = \int \psi u u d\Omega, \quad u^2 = \sum_{i=0}^N u_i \psi_i, \quad u = \sum_{i=0}^N u_i \psi_i. \quad (3.46)$$

Using the same notation as the ψu product in Eq. 3.43, Eq. 3.46 can be written as a system of equations, shown here for $N = 2$

$$\begin{pmatrix} 2 & 0 & 0 \\ 0 & \frac{2}{3} & 0 \\ 0 & 0 & \frac{2}{5} \end{pmatrix} \begin{pmatrix} u[0] \\ u[1] \\ u[2] \end{pmatrix} = \begin{pmatrix} u[0]u[0]\frac{2}{3} + u[1]u[1]\frac{2}{3} + u[2]u[2]\frac{2}{5} \\ u[0]u[1]\frac{4}{3} + u[1]u[2]\frac{8}{15} \\ u[0]u[2]\frac{4}{5} + u[1]u[1]\frac{4}{15} + u[2]u[2]\frac{4}{35} \end{pmatrix} \quad (3.47)$$

which are solved to obtain the coefficients for u^2 . The matrix on the left hand side of Eq. 3.47 is inverted by the PyDG pre-processor at compile time as it only consists of known constants. The code for generating this multiplication operator is created using the PyDG pre-processor, and is shown in Table 3.13.

To call this function, the variables u and u^2 are declared as instances of a polynomial expansion data type, u is initialized, and the function is called using the overloaded multiplication operator. An example of such code is shown in Table 3.14.

The DGPoly++ library is restricted to computing products of only two polynomials at a time. In other words, it is not possible to compute u^3 with the syntax $u3 = u * u * u$. Instead, the calculation needs to be divided into two steps as shown in Table 3.15. While this syntax may seem inconvenient, it forces the developer to be aware of memory usage. In fact, the typical implementation of the triple product $u3 = u * u * u$ would compute a temporary variable for the first multiplication, and then use that temporary variable for the second multiplication, just like what is explicitly done in the correct DGPoly++ syntax of Table 3.15. When working with scalar quantities, a temporary variable does not have a large effect on computer resources. However, because we are working with polynomial expansions, the memory usage of a single expansion can be significant. The explicit nature of the DGPoly++ syntax forces awareness of memory usage, and also allows more memory efficient code to be generated. For example, Table 3.15 shows an alternative syntax that uses the allocated memory of $u3$ as temporary storage, and the allocation of $u2$ is not necessary.

Table 3.13: PyDG Code to Generate the Polynomial Product of Eq. 3.47

PyDG Code
<pre>template<class DGPoly_type, typename T> ProjectProduct_Square<DGPoly_type, T>::ProjectProduct_Square(T* __restrict u2, const T* __restrict u) { InlinePod u = DGPolyVector('u' , poly=legendre, order=[2], coord=[xi]) psi = DGPolyVector('' , poly=legendre, order=[2], coord=[xi]) psi2 = (psi*psi).integrate((xi, -1, 1)).Matrix('').inv() u2 = (psi*u*u).integrate((xi, -1, 1)).Vector() print VectorValue(psi2*u2, 'u2') end InlinePod }</pre>
Generated C++ Code
<pre>template<class DGPoly_type, typename T> ProjectProduct_Square<DGPoly_type, T>::ProjectProduct_Square(T* __restrict u2, const T* __restrict u) { const T _t0 = 2.*u[0]/3. + 4.*u[2]/15.; const T _t1 = 2.*u[0]/5. + 4.*u[2]/35.; const T _t2 = 2.*u[2]*u[2]/5. + 2.*u[1]*u[1]/3. + 2.*u[0]*u[0]; const T _t3 = _t0*u[1] + 2.*u[0]*u[1]/3. + 4.*u[1]*u[2]/15.; const T _t4 = _t1*u[2] + 2.*u[0]*u[2]/5. + 4.*u[1]*u[1]/15.; u2[0] = _t2/2.; u2[1] = 3.*_t3/2.; u2[2] = 5.*_t4/2.;</pre>

Table 3.14: C++ Code for Computing the Square of a Polynomial

```
// Define the datatypes for the template arguments
// for the polynomial expansion data type
typedef double T;
typedef DGPoly<legendre, Coordinate<xi>, Tensor<2>> DGPoly_type;

// Declare the polynomials
PolyExpansion<DGPoly_type, T> u, u2;

// Initialize u
u[0] = 1; u[1] = 2; u[2] = 3;

// Compute the Square
u2 = u*u;
```

Table 3.15: C++ Syntax for Computing a Polynomial Cubed

Incorrect syntax for a polynomial cube	Correct syntax for a polynomial cubed
<pre>// Declare the polynomials PolyExpansion<DGPoly_type, T> u, u3; // Compile time error! u3 = u*u*u;</pre>	<pre>// Declare the polynomials PolyExpansion<DGPoly_type, T> u, u2, u3; // Compute the cube u2 = u*u; u3 = u2*u; // Alternative syntax u3 = u*u; u3 = u3*u;</pre>

Non-polynomial Functions: Unfortunately, not all operations can be performed with pure polynomial operators. For example, non-integer exponent, exponential, min/max, and trigonometric functions cannot be easily integrated analytically. In these situations, the DGPoly++ library reverts to numerical Gaussian quadrature. Consider a generic non-polynomial function $f(u)$, this function is projected into the polynomial expansion f_p by

$$\int_{\Omega_e} \psi f_p d\Omega = \int_{\Omega_e} \psi f(u) d\Omega. \quad (3.48)$$

The left hand side of Eq. 3.48 is evaluated analytically, but the right hand side is evaluated using traditional numerical Gaussian quadrature.

The DGPoly++ library provides overloaded functions for performing these operations, as well as a mechanism for defining custom functions. A few examples of the syntax is shown in Table 3.16. The integer between chevrons in the function names is multiplied by the maximum order of the polynomial expansion to get the number of Gaussian quadrature points for the integration.

Table 3.16: C++ Code for Projecting Non-polynomial Functions

```
PolyExpansion<DGPoly_type , T> u , u2 , min_u , sin_u , exp_u ;  
  
// Initialize u2 ...  
  
u      = sqrt<2>(u2 );  
min_u = min<2>(u );  
sin_u = sin<2>(u );  
exp_u = exp<2>(u );  
// and more ...
```

Optimized Gauss Quadrature Integration: Gauss Quadrature integrates a one-dimensional polynomial of degree $2N_{GQ}-1$ exactly with N_{GQ} nodal values of the polynomial. However, Gauss quadrature can also be used to approximate the integration of a non-polynomial function. Gauss Quadrature integration is computed as a sum of Gauss Quadrature weights, w_i , multiplied with nodal values of the integrand evaluated at the Gauss Quadrature nodes, s_i . The one-dimensional integral of the right hand side of 3.48 using N_{GQ} Gauss Quadrature nodes is

$$\int_{-1}^1 \psi(\xi) f(u(\xi)) d\xi = \sum_{i=0}^{N_{GQ}} w_i \psi(s_i) f(u(s_i)). \quad (3.49)$$

Integrals in two-dimensions on quadrilateral elements are computed as double sum, namely

$$\int_{-1}^1 \int_{-1}^1 \psi(\xi, \eta) f(u(\xi, \eta)) d\xi d\eta = \sum_{i=0}^{\sqrt{N_{GQ}}} \sum_{j=0}^{\sqrt{N_{GQ}}} w_i w_j \psi(s_i, s_j) f(u(s_i, s_j)). \quad (3.50)$$

The Gauss Quadrature expressed in general here as the sum

$$\int_{\Omega} \psi(\vec{\xi}) f(u(\vec{\xi})) d\vec{\xi} = \sum_{i=0}^{N_{GQ}} W_i \psi(\vec{s}_i) f(u(\vec{s}_i)), \quad (3.51)$$

where W_i is either w_i , $w_i w_j$, or $w_i w_j w_k$ depending on the dimension of the polynomial.

A simplistic implementation of Gauss Quadrature integration would rely on a set of subroutines that compute the polynomial values $\psi(\vec{s}_i)$ and $u(\vec{s}_i)$ each time they are needed. However, this implementation would be computationally expensive as it would repeatedly recompute the value of the basis functions at the Gauss Quadrature nodes. Instead, an optimal evaluation of the polynomial u at the Gauss Quadrature nodes is a matrix-vector multiplication

$$u^{\vec{GQ}} = M_{GQ} \vec{u} \Rightarrow \begin{pmatrix} u_0^{GQ} \\ u_1^{GQ} \\ u_2^{GQ} \\ \vdots \\ u_{N_{GQ}}^{GQ} \end{pmatrix} = \begin{pmatrix} \psi_0(\vec{s}_0) & \psi_1(\vec{s}_0) & \cdots & \psi_{N_m}(\vec{s}_0) \\ \psi_0(\vec{s}_1) & \psi_1(\vec{s}_1) & \cdots & \psi_{N_m}(\vec{s}_1) \\ \psi_0(\vec{s}_2) & \psi_1(\vec{s}_2) & \cdots & \psi_{N_m}(\vec{s}_2) \\ \vdots & \vdots & \vdots & \vdots \\ \psi_0(\vec{s}_{N_{GQ}}) & \psi_1(\vec{s}_{N_{GQ}}) & \cdots & \psi_{N_m}(\vec{s}_{N_{GQ}}) \end{pmatrix} \begin{pmatrix} u_0 \\ u_1 \\ \vdots \\ u_{N_m} \end{pmatrix}, \quad (3.52)$$

where $u^{\vec{GQ}}$ is the Gauss Quadrature nodal values of u polynomial, \vec{u} is the modal coefficient of the u polynomial, and basis functions in the matrix M_{GQ} evaluated at the Gauss Quadrature nodes are precomputed either using a pre-processor or as part of an initialization process. Note that the matrix in Eq. 3.52 is not a square matrix. The implementation here uses the PyDG pre-procesor to compute the matrix M_{GQ} . The polynomial u is evaluated at the Gauss Quadrature nodes with an optimized matrix-vector multiplication subroutine in a BLAS library. The non-polynomial function is then evaluated at the Gauss Quadrature nodes, i.e. $f(u^{\vec{GQ}})$,

and multiplied by the matrix W_{GQ} , defined as

$$W_{GQ}f(u^{\vec{GQ}}) \Rightarrow \begin{pmatrix} W_0\psi_0(\vec{s}_0) & W_1\psi_0(\vec{s}_1) & W_2\psi_0(\vec{s}_2) & \cdots & W_{N_{GQ}}\psi_0(\vec{s}_{N_{GQ}}) \\ W_0\psi_1(\vec{s}_0) & W_1\psi_1(\vec{s}_1) & W_2\psi_1(\vec{s}_2) & \cdots & W_{N_{GQ}}\psi_1(\vec{s}_{N_{GQ}}) \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ W_0\psi_{N_m}(\vec{s}_0) & W_1\psi_{N_m}(\vec{s}_1) & W_2\psi_{N_m}(\vec{s}_2) & \cdots & W_{N_{GQ}}\psi_{N_m}(\vec{s}_{N_{GQ}}) \end{pmatrix} \begin{pmatrix} f(u_0^{GQ}) \\ f(u_1^{GQ}) \\ f(u_2^{GQ}) \\ \vdots \\ f(u_{N_{GQ}}^{GQ}) \end{pmatrix}, \quad (3.53)$$

to complete the integration in Eq. 3.51. Again, the matrix W_{GQ} is computed using the PyDG pre-processor and the matrix-vector multiplication in Eq. 3.53 is computed using optimized BLAS routines.

Polynomial Division: Two different approaches for computing the division of polynomials were considered. The first polynomial division approach is in accord with Lockard and Atkins[30]. They approximated polynomial division by first approximating the inverse of the denominator polynomial, and then multiplying the inverse with the numerator. Similar to the polynomial product, the inverse of a polynomial is expressed as a polynomial expansion of the same order as the original polynomial. To derive the equation, first consider the polynomial expansion u , and let ui represent the inverse of u . If ui were the true inverse of u , then the following equation would hold

$$ui * u = 1. \quad (3.54)$$

Eq. 3.54 is evaluated in a weak form by multiplying it with the test functions ψ and integrating

$$\int_{\Omega_e} \psi ui * u d\Omega = \int_{\Omega_e} \psi d\Omega. \quad (3.55)$$

After integration, Eq. 3.55 can be written as a system of equations

$$\begin{pmatrix} u[0]2 & u[1]\frac{2}{3} & u[2]\frac{2}{5} \\ u[1]\frac{2}{3} & u[0]\frac{2}{3} + u[2]\frac{4}{15} & u[1]\frac{4}{15} \\ u[2]\frac{2}{5} & u[1]\frac{4}{15} & u[0]\frac{2}{5} + u[2]\frac{4}{35} \end{pmatrix} \begin{pmatrix} ui[0] \\ ui[1] \\ ui[2] \end{pmatrix} = \begin{pmatrix} 2 \\ 0 \\ 0 \end{pmatrix}, \quad (3.56)$$

which is solved to obtain the coefficients for the approximate inverse ui . Unlike the polynomial product, the

system of equations in Eq. 3.56 consists of runtime information and is solved at runtime.

The second approach treats the polynomial division as a non-polynomial function in Eq. 3.48 with $f(u) = 1/u$. This method relies on Gauss Quadrature integration and does not require solving a system of linear equations, and is hence significantly more computationally efficient.

A formal analysis provided by Dr. Donald French¹ (See, Appendix B) shows that the error associated with the two approaches is $O(h^{N+1})$. Hence, the non-polynomial function approach is used here as it is significantly less computational resources.

The division operator has been overloaded for the polynomial expansion data type to call the function which computes this inverse. The division operator is restricted to scalar numerators. Restricting the numerator to be scalar forces the storage of the inverse and reuse of it as often as possible. The improper and proper syntax for computing a polynomial inverse is shown in Table 3.17.

Table 3.17: C++ Syntax for Computing a Polynomial Inverse

Incorrect syntax for a polynomial inverse	Correct syntax for a polynomial inverse
<pre>// Declare the polynomials PolyExpansion<DGPoly_type ,T> rho_u ; PolyExpansion<DGPoly_type ,T> rho_v ; PolyExpansion<DGPoly_type ,T> rho ; PolyExpansion<DGPoly_type ,T> rhoi ; PolyExpansion<DGPoly_type ,T> u , v ; // Initialize rho , rho_u , rho_v ... // Compile time error! u = rho_u/rho ; v = rho_v/rho ;</pre>	<pre>// Declare the polynomials PolyExpansion<DGPoly_type ,T> rho_u ; PolyExpansion<DGPoly_type ,T> rho_v ; PolyExpansion<DGPoly_type ,T> rho ; PolyExpansion<DGPoly_type ,T> rhoi ; PolyExpansion<DGPoly_type ,T> u , v ; // Initialize rho , rho_u , rho_v ... // Compute the inverse rhoi = 1/rho ; // Compute u and v u = rho_u*rhoi ; v = rho_v*rhoi ;</pre>

Polynomial Cartesian Derivatives: The Cartesian derivative of a cell polynomial is computed using the chain rule as outlined in Section 3.2.1.2. The metric terms in the matrix \mathcal{T} and the coordinate Jacobian J are computed during an initialization processes and stored for each cell in the computational grid. Product of

¹Private Communication

metric terms and the curvilinear derivative of the polynomial are computed using the polynomial multiplication outlined in Eqs. 3.46 and 3.47. A functor, which is a C++ class that mimics the syntax of a function, called “ProjectGrad” is used to compute the Cartesian derivative of a polynomial. The class stores a reference to the metric terms for a given cell and so that multiple derivatives can be computed. The Cartesian coordinate that the polynomial is differentiated with respect to is specified as a template argument of the class. As an example, the syntax for computing the x and y derivatives of a two-dimensional polynomial u is shown in Table 3.18.

Table 3.18: C++ Syntax for Computing the Cartesian Derivative of a Polynomial

```
// Declare the polynomials
PolyExpansion<DGPoly_type, T> u, u_x, u_y;

// Initialize u...

// Define the differentiation functors based on the cell (i,j,k)
Poly::ProjectGrad<X, Metric_type> ddx( Mesh(i,j,k) );
Poly::ProjectGrad<Y, Metric_type> ddy( Mesh(i,j,k) );

// Compute du/dx and du/dy
u_x = ddx( u );
u_y = ddy( u );
```

Flux Integration: The fluxes for the volume integrals in Eq. 2.12 are expressed as a polynomial expansion in the same set of basis functions as the conservative variables (See, Eq. 2.3), namely

$$\vec{\mathcal{F}}(\vec{\xi}) = \sum_{i=0}^{N_m} \vec{\mathcal{F}}_i \psi_i(\vec{\xi}). \quad (3.57)$$

The flux vector polynomial expansions in Eq. 3.57 is an L^2 -approximation of the exact polynomial expansion $\vec{F}(Q, \nabla Q)$, i.e. $\vec{\mathcal{F}} \approx \vec{F}(Q, \nabla Q)$. As the Legendre polynomials are orthogonal, the coefficients of the flux vector expansion are computed as

$$\vec{\mathcal{F}}_i = \frac{\int_{\Omega} \psi_i \vec{F}(Q, \nabla Q) d\vec{\xi}}{\int_{\Omega} \psi_i^2 d\vec{\xi}} \quad \forall i \in [0, N_m]. \quad (3.58)$$

The source term in Eq. 2.12 is also expressed as a polynomial expansion with the same L^2 -approximation. The volume integral of the gradient of the test function in Eq. 2.12 expressed in curvilinear coordinates in accord with Eq. 3.34 is

$$\int_{\Omega} \mathcal{T}^T \nabla_{\xi} \psi \cdot \vec{\mathcal{F}} d\vec{\xi}. \quad (3.59)$$

The integral in Eq. 3.59 is evaluated by recognizing that

$$\mathcal{T}^T \nabla_{\xi} \psi \cdot \vec{\mathcal{F}} = \nabla_{\xi} \psi \cdot \mathcal{T} \vec{\mathcal{F}}, \quad (3.60)$$

and defining $\vec{\mathcal{F}}_{\xi} = \begin{pmatrix} \mathcal{F}_{\xi}, & \mathcal{F}_{\eta}, & \mathcal{F}_{\zeta} \end{pmatrix}$ to be an L^2 -approximation to $\mathcal{T} \vec{\mathcal{F}}$, i.e.,

$$\int_{\Omega} \psi \vec{\mathcal{F}}_{\xi} d\vec{\xi} = \int_{\Omega} \psi \mathcal{T} \vec{\mathcal{F}} d\vec{\xi}. \quad (3.61)$$

The volume integral in Eq. 3.59 is thus

$$\int_{\Omega} \nabla_{\xi} \psi \cdot \vec{\mathcal{F}}_{\xi} d\vec{\xi}. \quad (3.62)$$

The syntax for computing the volume integral in Eq. 3.59 is given in Table 3.19. The DGPoly++ library computes the flux $\vec{\mathcal{F}}_{\xi}$ and subsequently evaluates the integral in Eq. 3.62. The PyDG used to generate the C++ that evaluates the integral

$$\int_{-1}^1 \int_{-1}^1 \frac{\partial \psi}{\partial \xi} \cdot \vec{\mathcal{F}}_{\xi} d\xi d\eta, \quad (3.63)$$

which is one term in Eq. 3.62, for $N = 2$ is shown in Table 3.20. Similar integration functions are generated for the remaining terms in the integral in Eq. 3.62.

Table 3.19: C++ Syntax for Computing Weak Volume Integral

```
FluxTensor_type F;
Poly :: GradTestFunction<Metric_type> GradPsi( Mesh(i,j,k) );
for( int iEq = 0; iEq < F.nEq; ++iEq )
    Internal_Integral[iEq] = Poly :: VolumeIntegral( GradPsi , F.Fx[iEq]
                                                    , F.Fy[iEq]
                                                    , F.Fz[iEq]);
```

Table 3.20: PyDG Code to Generate the Integral in Eq. 3.19

PyDG Code
<pre>template<class coord, class DGPoly_type, typename T> void VolumeIntegralGradPsi_d<coord, DGPoly_type, T>:: Integrate(T* __restrict Fint, const T* __restrict Fxi) { InlinePod Fxi = DGPolyVector('Fxi', poly=legendre, order=[2,2], coord=[xi, eta]) psi = DGPolyVector(' ', poly=legendre, order=[2,2], coord=[xi, eta]) VolumeInt = (ddxi(psi)*Fxi).integrate((xi,-1,1),(eta,-1,1)) print VolumeInt.VectorValue('Fint') end InlinePod }</pre>
Generated C++ Code
<pre>template<class coord, class DGPoly_type, typename T> void VolumeIntegralGradPsi_d<coord, DGPoly_type, T>:: Integrate(T* __restrict Fint, const T* __restrict F) { Fint[0] = 0; Fint[1] = sgn*8.0*F[0]; Fint[2] = 0; Fint[3] = sgn*2.6666666667*F[2]; Fint[4] = sgn*8.0*F[1]; Fint[5] = sgn*2.6666666667*F[3]; Fint[6] = 0; Fint[7] = sgn*1.6*F[6]; Fint[8] = sgn*1.6*F[7]; }</pre>

The fluxes for the boundary integrals are expressed as polynomial expansions in the coordinates of a cell face

$$\vec{\mathcal{F}}(\xi_i, \xi_j) = \sum_{m=0}^{N_{cb}} \vec{\mathcal{F}}_m \psi_m(\xi_i, \xi_j). \quad (3.64)$$

The coefficients of the fluxes are computed from the dependent variable vector and gradient of the dependent variable vector with the L^2 -approximation

$$\vec{\mathcal{F}}_m = \frac{\int_{-1}^1 \int_{-1}^1 \psi_m \vec{F} \left(Q|_{\xi_k=\pm 1}, \nabla Q|_{\xi_k=\pm 1} \right) d\xi_i d\xi_j}{\int_{-1}^1 \int_{-1}^1 \psi_m^2 d\xi_i d\xi_j} \quad \forall m \in [0, N_{cb}]. \quad (3.65)$$

The curvilinear form of the boundary integral of the flux

$$\int_{-1}^1 \int_{-1}^1 \psi|_{\xi_k=\pm 1} \vec{\mathcal{F}}(\xi_i, \xi_j) \cdot \vec{n}_{\xi_k} d\xi_i d\xi_j, \quad (3.66)$$

is simplified by letting

$$\int_{-1}^1 \int_{-1}^1 \psi \vec{\mathcal{F}}_n d\xi_i d\xi_j = \int_{-1}^1 \int_{-1}^1 \psi \vec{\mathcal{F}}(\xi_i, \xi_j) \cdot \vec{n}_{\xi_k} d\xi_i d\xi_j. \quad (3.67)$$

The curvilinear boundary integral in Eq. 3.66 is thus

$$\int_{-1}^1 \int_{-1}^1 \psi|_{\xi_k=\pm 1} \vec{\mathcal{F}}_n d\xi_i d\xi_j. \quad (3.68)$$

Table 3.21: C++ Syntax for Computing a Boundary Integral

```

Poly :: TestFunction psi;
FaceFluxVector Fn;

for( int iEq = 0; iEq < Fn.nEq; ++iEq )
{
    Internal_Integral[ iEq ] += 
        Poly :: Boundary<Face, side>:: Integral( psi, Fn[ iEq ] );
    External_Integral[ iEq ] += 
        Poly :: Boundary<Face, -side>:: Integral( psi, Fn[ iEq ] );
}

```

The syntax for computing the boundary integral in Eq. 3.68 is given in Table 3.21. The template arguments “Face” and “side” represent the coordinate that is constant on the cell face and the value of that coordinate respectively. The value of “side” is either 1 or -1 . The boundary integral for two neighboring cells are computed in Table 3.21. To demonstrate this, let “Face” be the ξ coordinate and let $\text{side}=-1$, the boundary integral for the internal cell is then

$$-\int_{-1}^1 \int_{-1}^1 \psi|_{\xi=-1} \vec{\mathcal{F}}_n d\eta d\zeta, \quad (3.69)$$

and the boundary integral for the external cell is

$$\int_{-1}^1 \int_{-1}^1 \psi|_{\xi=1} \vec{\mathcal{F}}_n d\eta d\xi. \quad (3.70)$$

The negative sign in Eq. 3.69 is required because the normal vector \vec{n}_ξ in Eq. 3.67 is inward facing on the cell boundary $\xi = -1$. The PyDG code used to generate the C++ code that evaluates the boundary integral in Eq. 3.69 with $N = 2$ is given in Table 3.22. Similar functions are defined for all cell boundaries.

Table 3.22: PyDG Code to Generate the Boundary Integral in Eq. 3.69

PyDG Code
<pre>template<class Face, int side, class DGPoly_type, typename T> void BoundaryIntegralValue<Face, side, DGPoly_type, T>:: Integrate(T* __restrict Fint, const T* __restrict Fn) { InlinePod Fn = DGPolyVector('Fn', poly=legendre, order=[2], coord=[eta]) psi = DGPolyVector(' ', poly=legendre, order=[2,2], coord=[xi, eta]) BndryInt = (psi*Fn).boundary(xi, -1).integrate((eta, -1, 1)) print BndryInt.VectorValue('Fint') end InlinePod }</pre>
Generated C++ Code
<pre>template<class Face, int side, class DGPoly_type, typename T> void BoundaryIntegralValue<Face, side, DGPoly_type, T>:: Integrate(T* __restrict Fint, const T* __restrict Fn) { const T _t0 = 4.0*Fn[0]; const T _t1 = 1.333333333333333*Fn[1]; const T _t2 = 0.8*Fn[2]; Fint[0] = -_t0; Fint[1] = _t0; Fint[2] = -_t1; Fint[3] = _t1; Fint[4] = -_t0; Fint[5] = -_t1; Fint[6] = -_t2; Fint[7] = _t2; Fint[8] = -_t2; }</pre>

Flux Linearization: The the block matrices that make up the Jacobian matrix $\frac{\partial \mathcal{R}}{\partial Q}$ are derived by first expressing the polynomial expansion of the dependent variable increment as a vector-vector product

$$\Delta Q = \sum_{i=0}^{N_m} \Delta Q_i \psi_i = \begin{pmatrix} \psi_0 & \psi_1 & \dots & \psi_{N_m} \end{pmatrix} \begin{pmatrix} \Delta Q_0 \\ \Delta Q_1 \\ \vdots \\ \Delta Q_{N_m} \end{pmatrix}. \quad (3.71)$$

In addition, the Jacobian of the flux is expressed as a polynomial expansion

$$\frac{\partial \vec{\mathcal{F}}}{\partial Q} (\vec{\xi}) = \sum_{i=0}^{N_m} \left(\frac{\partial \vec{\mathcal{F}}}{\partial Q} \right)_i \psi_i (\vec{\xi}). \quad (3.72)$$

The volume integral of the product of the Jacobian of the flux and the gradient of the test function is

$$\int_{\Omega} \mathcal{T}^T \nabla_{\xi} \psi^- \cdot \frac{\partial \vec{\mathcal{F}}}{\partial Q^-} \Delta Q^- d\vec{\xi}, \quad (3.73)$$

which using the relationship in Eq. 3.60 is written as

$$\int_{\Omega} \nabla_{\xi} \psi^- \cdot \mathcal{T} \frac{\partial \vec{\mathcal{F}}}{\partial Q^-} \Delta Q^- d\vec{\xi}. \quad (3.74)$$

Equation 3.74 is further simplified by introducing $\frac{\partial \vec{\mathcal{F}}_{\xi}}{\partial Q^-}$ defined as

$$\int_{\Omega} \psi \frac{\partial \vec{\mathcal{F}}_{\xi}}{\partial Q^-} d\vec{\xi} = \int_{\Omega} \psi \mathcal{T} \frac{\partial \vec{\mathcal{F}}}{\partial Q^-} d\vec{\xi}, \quad (3.75)$$

which results in

$$\int_{\Omega} \nabla_{\xi} \psi^- \cdot \frac{\partial \vec{\mathcal{F}}_{\xi}}{\partial Q^-} \Delta Q^- d\vec{\xi}. \quad (3.76)$$

Writing $\nabla_{\xi} \psi^-$ as a column vector and expressing ΔQ^- as the vector-vector product, Eq. 3.76 is written as

$$\int_{\Omega} \left(\begin{array}{c} \nabla_{\xi} \psi_0^- \cdot \frac{\partial \vec{\mathcal{F}}_{\xi}}{\partial Q^-} \left(\begin{array}{cccc} \psi_0^- & \psi_1^- & \dots & \psi_{N_m}^- \end{array} \right) \\ \nabla_{\xi} \psi_1^- \cdot \frac{\partial \vec{\mathcal{F}}_{\xi}}{\partial Q^-} \left(\begin{array}{cccc} \psi_0^- & \psi_1^- & \dots & \psi_{N_m}^- \end{array} \right) \\ \vdots \\ \nabla_{\xi} \psi_{N_m}^- \cdot \frac{\partial \vec{\mathcal{F}}_{\xi}}{\partial Q^-} \left(\begin{array}{cccc} \psi_0^- & \psi_1^- & \dots & \psi_{N_m}^- \end{array} \right) \end{array} \right) d\vec{\xi} \left(\begin{array}{c} \Delta Q_0^- \\ \Delta Q_1^- \\ \vdots \\ \Delta Q_{N_m}^- \end{array} \right), \quad (3.77)$$

which upon integration is one of the block matrices that make up the Jacobian matrix $\frac{\partial \mathcal{R}}{\partial Q}$.

The C++ code that computes the block matrix in Eq. 3.77 using the Jacobians of the Cartesian fluxes as expressed in Eq. 3.73 is shown in Table 3.23. The DGPoly++ library computes the intermediate Jacobian 3.75 and then computes the matrix in Eq. 3.77. The PyDG code that generates the C++ code to compute the integral of the derivative of the test function $\frac{\partial \psi}{\partial \xi}$, i.e.,

$$\int_{\Omega} \left(\begin{array}{c} \frac{\partial \psi_0^-}{\partial \xi} \frac{\partial \mathcal{F}_{\xi}}{\partial Q^-} \left(\begin{array}{cccc} \psi_0^- & \psi_1^- & \dots & \psi_{N_m}^- \end{array} \right) \\ \frac{\partial \psi_1^-}{\partial \xi} \frac{\partial \mathcal{F}_{\xi}}{\partial Q^-} \left(\begin{array}{cccc} \psi_0^- & \psi_1^- & \dots & \psi_{N_m}^- \end{array} \right) \\ \vdots \\ \frac{\partial \psi_{N_m}^-}{\partial \xi} \frac{\partial \mathcal{F}_{\xi}}{\partial Q^-} \left(\begin{array}{cccc} \psi_0^- & \psi_1^- & \dots & \psi_{N_m}^- \end{array} \right) \end{array} \right) d\vec{\xi}, \quad (3.78)$$

is given in Table 3.24.

Table 3.23: C++ Syntax for Computing the Jacobian of the Weak Volume Integral

```
JacobianTensor_type F_Q;
PolyExpMatrix_type M;
Poly::GradTestFunction<Metric_type> GradPsi( Mesh(i,j,k) );

for( int iEq = 0; iEq < nEq; ++iEq )
    for( int jVar = 0; jVar < nVar; ++jVar )
    {
        M = Poly::VolumeIntegral(GradPsi, F_Q.Fx(iEq, jVar)
                                , F_Q.Fy(iEq, jVar)
                                , F_Q.Fz(iEq, jVar));
        Diag(iEq, jVar) -= M;
    }
```

Table 3.24: PyDG Code to Generate the Integral in Eq. 3.78

PyDG Code
<pre>template<class coord, class DGPoly_type, typename T> void VolumeIntegralGradPsiMatrixValue_d<coord, DGPoly_type, T>:: Integrate(T* __restrict Fxi_Qint, const T* __restrict Fxi_Q) { InlinePod Fxi_Q=DGPolyVector('Fxi_Q',poly=legendre,order=[2,2],coord=[xi,eta]) dQ = DGPolyVector('dQ',poly=legendre,order=[2,2],coord=[xi,eta]) psi = DGPolyVector(' ',poly=legendre,order=[2,2],coord=[xi,eta]) VolumeInt = (ddxi(psi)*Fxi_Q*dQ).integrate((xi,-1,1),(eta,-1,1)) print VolumeInt.VectorValue('Fxi_Qint') end InlinePod }</pre>
Generated C++ Code
<pre>template<class coord, class DGPoly_type, typename T> void VolumeIntegralGradPsiMatrixValue_d<coord, DGPoly_type, T>:: Integrate(T* __restrict Fxi_Qint, const T* __restrict Fxi_Q) { const T _t0 = 8.0*Fxi_Q[0]; const T _t1 = 2.666666666667*Fxi_Q[1]; const T _t2 = 2.666666666667*Fxi_Q[2]; const T _t3 = 0.888888888889*Fxi_Q[3]; const T _t4 = 2.666666666667*Fxi_Q[0]; const T _t5 = 0.888888888889*Fxi_Q[1]; Fxi_Qint(0,0) = 0; Fxi_Qint(0,1) = 0; Fxi_Qint(0,2) = 0; Fxi_Qint(0,3) = 0; Fxi_Qint(1,0) = _t0; Fxi_Qint(1,1) = _t1; Fxi_Qint(1,2) = _t2; Fxi_Qint(1,3) = _t3; Fxi_Qint(2,0) = 0; Fxi_Qint(2,1) = 0; Fxi_Qint(2,2) = 0; Fxi_Qint(2,3) = 0; Fxi_Qint(3,0) = _t2; Fxi_Qint(3,1) = _t3; Fxi_Qint(3,2) = _t4; Fxi_Qint(3,3) = _t5; }</pre>

The boundary integral of the flux Jacobians with respect to the interior and exterior dependent variable vectors is

$$\int_{-1}^1 \int_{-1}^1 \psi^- \frac{1}{2} \left(\frac{\partial \vec{\mathcal{F}}}{\partial Q^+} \Delta Q^+ + \frac{\partial \vec{\mathcal{F}}}{\partial Q^-} \Delta Q^- \right) \cdot \vec{n}_{\xi_k} d\xi_i d\xi_j. \quad (3.79)$$

which can be simplified by defining

$$\begin{aligned} \frac{\partial \vec{\mathcal{F}}_n}{\partial Q^-} &\equiv \frac{\partial \vec{\mathcal{F}}}{\partial Q^-} \cdot \vec{n}_{\xi_k}, \\ \frac{\partial \vec{\mathcal{F}}_n}{\partial Q^+} &\equiv \frac{\partial \vec{\mathcal{F}}}{\partial Q^+} \cdot \vec{n}_{\xi_k}. \end{aligned} \quad (3.80)$$

Equation 3.79 therefore reduces to

$$\int_{-1}^1 \int_{-1}^1 \psi^- \frac{1}{2} \left(\frac{\partial \vec{\mathcal{F}}_n}{\partial Q^+} \Delta Q^+ + \frac{\partial \vec{\mathcal{F}}_n}{\partial Q^-} \Delta Q^- \right) d\xi_i d\xi_j, \quad (3.81)$$

which after replacing ΔQ^\pm with the vector-vector in Eq. 3.71 and ψ^- as a column vector becomes

$$\begin{aligned} \int_{-1}^1 \int_{-1}^1 \frac{1}{2} \left(\begin{array}{c} \psi_0^- \frac{\partial \vec{\mathcal{F}}_n}{\partial Q^-} \left(\begin{array}{cccc} \psi_0^- & \psi_1^- & \dots & \psi_{N_m}^- \end{array} \right) \\ \psi_1^- \frac{\partial \vec{\mathcal{F}}_n}{\partial Q^-} \left(\begin{array}{cccc} \psi_0^- & \psi_1^- & \dots & \psi_{N_m}^- \end{array} \right) \\ \vdots \\ \psi_{N_m}^- \frac{\partial \vec{\mathcal{F}}_n}{\partial Q^-} \left(\begin{array}{cccc} \psi_0^- & \psi_1^- & \dots & \psi_{N_m}^- \end{array} \right) \end{array} \right) d\xi_i d\xi_j \left(\begin{array}{c} \Delta Q_0^- \\ \Delta Q_1^- \\ \vdots \\ \Delta Q_{N_m}^- \end{array} \right) + \\ \int_{-1}^1 \int_{-1}^1 \frac{1}{2} \left(\begin{array}{c} \psi_0^- \frac{\partial \vec{\mathcal{F}}_n}{\partial Q^+} \left(\begin{array}{cccc} \psi_0^+ & \psi_1^+ & \dots & \psi_{N_m}^+ \end{array} \right) \\ \psi_1^- \frac{\partial \vec{\mathcal{F}}_n}{\partial Q^+} \left(\begin{array}{cccc} \psi_0^+ & \psi_1^+ & \dots & \psi_{N_m}^+ \end{array} \right) \\ \vdots \\ \psi_{N_m}^- \frac{\partial \vec{\mathcal{F}}_n}{\partial Q^+} \left(\begin{array}{cccc} \psi_0^+ & \psi_1^+ & \dots & \psi_{N_m}^+ \end{array} \right) \end{array} \right) d\xi_i d\xi_j \left(\begin{array}{c} \Delta Q_0^+ \\ \Delta Q_1^+ \\ \vdots \\ \Delta Q_{N_m}^+ \end{array} \right). \end{aligned} \quad (3.82)$$

Note that the Jacobian $\frac{\partial \vec{\mathcal{F}}_n}{\partial Q^\pm}$ is a polynomial expansion in the exterior cell which is multiplied by the test function of the interior cell. The C++ code that computes the matrices in Eq. 3.82 is shown in Table 3.25.

Similar to the boundary integral of the flux, the flux Jacobians are integrated while multiplied by the test function of the interior cell and then multiplied by the test function of the exterior cell. The PyDG code that generates the C++ code that computes the matrices in Eq. 3.82 on a $\xi = -1$ cell face with $N = 1$ is shown in Table 3.26.

Table 3.25: C++ Syntax for Computing a Boundary Integral

```

Poly :: TestFunction psi;
FaceJacobianVector Fn_QInt, Fn_QExt;

PolyExpMatrix_type MDiagL, MFaceL;
PolyExpMatrix_type MDiagR, MFaceR;

for( int iEq = 0; iEq < FaceFluxVector_type::nEq; ++iEq)
    for( int jVar = 0; jVar < FaceVariableVector_type::nvar; ++jVar)
    {
        (MDiagInt, MFaceInt) = Poly :: BoundaryAvg<Face, 1>
            :: Integral(psi, Fn_QInt(iEq, jVar)
                        , Fn_QExt(iEq, jVar));

        DiagInt(iEq, jVar) += MDiagInt;
        FaceInt(iEq, jVar) += MFaceInt;

        (MDiagExt, MFaceExt) = Poly :: BoundaryAvg<Face, -1>
            :: Integral(psi, Fn_QExt(iEq, jVar)
                        , Fn_QInt(iEq, jVar));

        DiagExt(iEq, jVar) += MDiagExt;
        FaceExt(iEq, jVar) += MFaceExt;
    }
}

```

Table 3.26: PyDG Code to Generate the Boundary Integral in Eq. 3.69

PyDG Code
<pre> template<class Face, int side, class DGPoly_type, typename T> void BoundaryAvgIntegralValue<Face, side, DGPoly_type, T>::Integrate(MatrixView_type& Internal, MatrixView_type& External, const T* __restrict Fn_QInt, const T* __restrict Fn_QExt) { InlinePod Fn_Q = DGPolyVector('Fn_Q', poly=legendre, order=[1], coord=[eta]) psi = DGPolyVector(' ', poly=legendre, order=[1,1], coord=[xi, eta]) side = -1 BndryInt = (psi*Avg(Fn_QInt*dQ)).boundary(xi, side) .integrate((eta, -1, 1)) BndryExt = (psi*Avg(Fn_QExt*dQ)).boundary(xi, -side) .integrate((eta, -1, 1)) print BndryInt.MatrixValue('Internal', 'dQ') print BndryExt.MatrixValue('External', 'dQ') end InlinePod } </pre>
Generated C++ Code
<pre> template<class Face, int side, class DGPoly_type, typename T> void BoundaryAvgIntegralValue<Face, side, DGPoly_type, T>::Integrate(MatrixView_type& Internal, MatrixView_type& External, const T* __restrict Fn_QInt, const T* __restrict Fn_QExt) { const T _t0 = -0.5*Fn_QInt[0]; const T _t5 = -0.5*Fn_QExt[0]; const T _t1 = 4.0*_t0; const T _t6 = 4.0*_t5; const T _t2 = -0.5*Fn_QInt[1]; const T _t7 = -0.5*Fn_QExt[1]; const T _t3 = 1.33333333333*_t2; const T _t8 = 1.33333333333*_t7; const T _t4 = 1.33333333333*_t0; const T _t9 = 1.33333333333*_t5; Internal(0,0) = _t1; External(0,0) = _t6; Internal(0,1) = -_t1; External(0,1) = _t6; Internal(0,2) = _t3; External(0,2) = _t8; Internal(0,3) = -_t3; External(0,3) = _t8; Internal(1,0) = -_t1; External(1,0) = -_t6; Internal(1,1) = _t1; External(1,1) = -_t6; Internal(1,2) = -_t3; External(1,2) = -_t8; Internal(1,3) = _t3; External(1,3) = -_t8; Internal(2,0) = _t3; External(2,0) = _t8; Internal(2,1) = -_t3; External(2,1) = _t8; Internal(2,2) = _t4; External(2,2) = _t9; Internal(2,3) = -_t4; External(2,3) = _t9; Internal(3,0) = -_t3; External(3,0) = -_t8; Internal(3,1) = _t3; External(3,1) = -_t8; Internal(3,2) = -_t4; External(3,2) = -_t9; Internal(3,3) = _t4; External(3,3) = -_t9; } </pre>

The flux Jacobians with respect to the gradient of the dependent variable vector require the Jacobian of the gradient vector with respect to the dependent variables. The gradient of the increment in the dependent variables, $\nabla(\Delta Q)$, is computed with the L^2 -approximation

$$\int_{\Omega} \psi \nabla(\Delta Q) d\vec{\xi} = \int_{\Omega} \psi \mathcal{T}^T \nabla_{\xi}(\Delta Q) d\vec{\xi}, \quad (3.83)$$

which is written in matrix form as

$$\begin{aligned} & \int_{\Omega} \begin{pmatrix} \psi_0^2 & & & \\ & \psi_1^2 & & \\ & & \ddots & \\ & & & \psi_{N_m}^2 \end{pmatrix} d\vec{\xi} \begin{pmatrix} \nabla(\Delta Q_0) \\ \nabla(\Delta Q_1) \\ \vdots \\ \nabla(\Delta Q_{N_m}) \end{pmatrix} = \\ & \int_{\Omega} \begin{pmatrix} \psi_0 \mathcal{T}^T \nabla_{\xi} \begin{pmatrix} \psi_0 & \psi_1 & \dots & \psi_{N_m} \end{pmatrix} \\ \psi_1 \mathcal{T}^T \nabla_{\xi} \begin{pmatrix} \psi_0 & \psi_1 & \dots & \psi_{N_m} \end{pmatrix} \\ \vdots \\ \psi_{N_m} \mathcal{T}^T \nabla_{\xi} \begin{pmatrix} \psi_0 & \psi_1 & \dots & \psi_{N_m} \end{pmatrix} \end{pmatrix} d\vec{\xi} \begin{pmatrix} \Delta Q_0 \\ \Delta Q_1 \\ \vdots \\ \Delta Q_{N_m} \end{pmatrix}. \end{aligned} \quad (3.84)$$

By defining the mass matrix to be

$$M = \int_{\Omega} \begin{pmatrix} \psi_0^2 & & & \\ & \psi_1^2 & & \\ & & \ddots & \\ & & & \psi_{N_m}^2 \end{pmatrix} d\vec{\xi}, \quad (3.85)$$

the polynomial expansion of the gradient of dependent variable vector increment is

$$\begin{aligned}
\nabla(\Delta Q) &= \\
\left(\begin{array}{cccc} \psi_0 & \psi_1 & \dots & \psi_{N_m} \end{array} \right) &\left(\begin{array}{c} \nabla(\Delta Q_0) \\ \nabla(\Delta Q_1) \\ \vdots \\ \nabla(\Delta Q_{N_m}) \end{array} \right) = \\
\left(\begin{array}{cccc} \psi_0 & \psi_1 & \dots & \psi_{N_m} \end{array} \right) M^{-1} \int_{\Omega} &\left(\begin{array}{c} \psi_0 \mathcal{T}^T \nabla_{\xi} \left(\begin{array}{cccc} \psi_0 & \psi_1 & \dots & \psi_{N_m} \end{array} \right) \\ \psi_1 \mathcal{T}^T \nabla_{\xi} \left(\begin{array}{cccc} \psi_0 & \psi_1 & \dots & \psi_{N_m} \end{array} \right) \\ \vdots \\ \psi_{N_m} \mathcal{T}^T \nabla_{\xi} \left(\begin{array}{cccc} \psi_0 & \psi_1 & \dots & \psi_{N_m} \end{array} \right) \end{array} \right) d\vec{\xi} \left(\begin{array}{c} \Delta Q_0 \\ \Delta Q_1 \\ \vdots \\ \Delta Q_{N_m} \end{array} \right) = \\
&\left(\begin{array}{cccc} \psi_0 & \psi_1 & \dots & \psi_{N_m} \end{array} \right) \frac{\partial \nabla Q}{\partial Q} \Delta \vec{Q} \quad (3.86)
\end{aligned}$$

Thus, the matrix in Eq. 3.86 is the Jacobian of the gradient vector with respect to the dependent variables.

The volume integral in Eq. 2.12 of the flux Jacobian with respect to the dependent variable gradient multiplied by the gradient of the test function,

$$\int_{\Omega} \mathcal{T}^T \nabla_{\xi} \psi^- \cdot \frac{\partial \vec{\mathcal{F}}}{\partial \nabla Q^-} \frac{\partial \nabla Q^-}{\partial Q^-} \Delta Q^- d\vec{\xi}, \quad (3.87)$$

is simplified by using the relationship in Eq. 3.60 and introducing

$$\frac{\partial \vec{\mathcal{F}}_{\xi}}{\partial \nabla Q^-} = \mathcal{T} \frac{\partial \vec{\mathcal{F}}}{\partial \nabla Q^-} \quad (3.88)$$

to

$$\int_{\Omega} \nabla_{\xi} \psi^- \cdot \frac{\partial \vec{\mathcal{F}}_{\xi}}{\partial \nabla Q^-} \frac{\partial \nabla Q^-}{\partial Q^-} \Delta Q^- d\vec{\xi}. \quad (3.89)$$

Equation 3.89 is written in matrix form

$$\int_{\Omega} \left(\begin{array}{c} \nabla_{\xi} \psi_0^- \cdot \frac{\partial \vec{\mathcal{F}}_{\xi}}{\partial \nabla Q} \left(\begin{array}{cccc} \psi_0^- & \psi_1^- & \dots & \psi_{N_m}^- \end{array} \right) \\ \nabla_{\xi} \psi_1^- \cdot \frac{\partial \vec{\mathcal{F}}_{\xi}}{\partial \nabla Q} \left(\begin{array}{cccc} \psi_0^- & \psi_1^- & \dots & \psi_{N_m}^- \end{array} \right) \\ \vdots \\ \nabla_{\xi} \psi_{N_m}^- \cdot \frac{\partial \vec{\mathcal{F}}_{\xi}}{\partial \nabla Q} \left(\begin{array}{cccc} \psi_0^- & \psi_1^- & \dots & \psi_{N_m}^- \end{array} \right) \end{array} \right) d\xi \frac{\partial \nabla Q^-}{\partial Q^-} \left(\begin{array}{c} \Delta Q_0^- \\ \Delta Q_1^- \\ \vdots \\ \Delta Q_{N_m}^- \end{array} \right) \quad (3.90)$$

by replacing $\frac{\partial \nabla Q^-}{\partial Q^-} \Delta Q^-$ with the polynomial expansion in Eq. 3.86 and writing $\nabla_{\xi} \psi^-$ as a column vector. Hence, the linearization matrix in the Jacobian $\frac{\partial \mathcal{R}}{\partial Q}$ is a product of the integral in Eq. 3.90 and the Jacobian matrix $\frac{\partial \nabla Q^-}{\partial Q^-}$.

The boundary integral in Eq. 2.12 of Jacobians with respect to the gradient of the dependent variable vector,

$$\int_{-1}^1 \int_{-1}^1 \psi^- \frac{1}{2} \left(\frac{\partial \vec{\mathcal{F}}}{\partial \nabla Q^+} \frac{\partial \nabla Q^+}{\partial Q^+} \Delta Q^+ + \frac{\partial \vec{\mathcal{F}}}{\partial \nabla Q^-} \frac{\partial \nabla Q^-}{\partial Q^-} \Delta Q^- \right) \cdot \vec{n}_{\xi_k} d\xi_i d\xi_j d\xi_j, \quad (3.91)$$

is simplified by defining

$$\begin{aligned} \frac{\partial \vec{\mathcal{F}}_n}{\partial \nabla Q^-} &\equiv \frac{\partial \vec{\mathcal{F}}}{\partial \nabla Q^-} \cdot \vec{n}_{\xi_k}, \\ \frac{\partial \vec{\mathcal{F}}_n}{\partial \nabla Q^+} &\equiv \frac{\partial \vec{\mathcal{F}}}{\partial \nabla Q^+} \cdot \vec{n}_{\xi_k}, \end{aligned} \quad (3.92)$$

to

$$\int_{-1}^1 \int_{-1}^1 \psi^- \frac{1}{2} \left(\frac{\partial \vec{\mathcal{F}}_n}{\partial \nabla Q^+} \frac{\partial \nabla Q^+}{\partial Q^+} \Delta Q^+ + \frac{\partial \vec{\mathcal{F}}_n}{\partial \nabla Q^-} \frac{\partial \nabla Q^-}{\partial Q^-} \Delta Q^- \right) d\xi_i d\xi_j d\xi_j. \quad (3.93)$$

By replacing $\frac{\partial \nabla Q^{\pm}}{\partial Q^{\pm}} \Delta Q^{\pm}$ with the polynomial expansion in Eq. 3.86 and writing ψ^- as a column vector, Eq. 3.93 becomes

$$\begin{aligned}
& \int_{-1}^1 \int_{-1}^1 \frac{1}{2} \left(\begin{array}{c} \psi_0^- \cdot \frac{\partial \vec{\mathcal{F}}_n}{\partial \nabla Q^-} \left(\begin{array}{cccc} \psi_0^- & \psi_1^- & \dots & \psi_{N_m}^- \end{array} \right) \\ \psi_1^- \cdot \frac{\partial \vec{\mathcal{F}}_n}{\partial \nabla Q^-} \left(\begin{array}{cccc} \psi_0^- & \psi_1^- & \dots & \psi_{N_m}^- \end{array} \right) \\ \vdots \\ \psi_{N_m}^- \cdot \frac{\partial \vec{\mathcal{F}}_n}{\partial \nabla Q^-} \left(\begin{array}{cccc} \psi_0^- & \psi_1^- & \dots & \psi_{N_m}^- \end{array} \right) \end{array} \right) d\xi_i d\xi_j \frac{\partial \nabla Q^-}{\partial Q^-} \left(\begin{array}{c} \Delta Q_0^- \\ \Delta Q_1^- \\ \vdots \\ \Delta Q_{N_m}^- \end{array} \right) + \\
& \int_{-1}^1 \int_{-1}^1 \frac{1}{2} \left(\begin{array}{c} \psi_0^- \cdot \frac{\partial \vec{\mathcal{F}}_n}{\partial \nabla Q^+} \left(\begin{array}{cccc} \psi_0^+ & \psi_1^+ & \dots & \psi_{N_m}^+ \end{array} \right) \\ \psi_1^- \cdot \frac{\partial \vec{\mathcal{F}}_n}{\partial \nabla Q^+} \left(\begin{array}{cccc} \psi_0^+ & \psi_1^+ & \dots & \psi_{N_m}^+ \end{array} \right) \\ \vdots \\ \psi_{N_m}^- \cdot \frac{\partial \vec{\mathcal{F}}_n}{\partial \nabla Q^+} \left(\begin{array}{cccc} \psi_0^+ & \psi_1^+ & \dots & \psi_{N_m}^+ \end{array} \right) \end{array} \right) d\xi_i d\xi_j \frac{\partial \nabla Q^+}{\partial Q^+} \left(\begin{array}{c} \Delta Q_0^+ \\ \Delta Q_1^+ \\ \vdots \\ \Delta Q_{N_m}^+ \end{array} \right). \tag{3.94}
\end{aligned}$$

The linearization matrix in the Jacobian $\frac{\partial \mathcal{R}}{\partial Q}$ is the matrix-matrix product of the integrals in Eq. 3.94 and the Jacobian of the dependent variable gradient vectors $\frac{\partial \nabla Q^\pm}{\partial Q^\pm}$.

3.3 Solution Advancement

Solution advancement for both steady state and time accurate calculations are included in the code. The solution advancement is kept distinctly separate from the spatial discretization in order to simplify the implementation of a variety of solution advancement algorithms without significant effort. The code currently includes Newton's method and a Quasi-Newton's method for steady state calculations. An explicit Runge-Kutta and an implicit Diagonally Implicit Runge-Kutta are included for time accurate calculations.

3.3.1 Convergence to Steady State

3.3.1.1 Newton's Method

Newton's method is a powerful method for solving non-linear systems of equations. Newton's method is an iterative method which written in delta form is

$$\frac{\partial \mathcal{R}(Q^n)}{\partial Q} \Delta Q^n = -\mathcal{R}(Q^n), \tag{3.95}$$

where the solution at iteration $n + 1$ is obtained as $Q^{n+1} = Q^n + \Delta Q^n$. The method can converge in one

iteration for linear problems, and can obtain quadratic convergence for non-linear problems. A certain number of assumptions must be satisfied to achieve quadratic convergence. The Jacobian matrix $\frac{\partial \mathcal{R}(Q)}{\partial Q}$ must be a complete linearization of $\mathcal{R}(Q)$, including boundary conditions. This condition is particularly important for systems of equations. A poor choice in the initial solution Q^0 can result in the loss of quadratic convergence, or even cause the solution to diverge. The linear system must also be solved to machine precision, a requirement which can demand a significant number of iterations of an iterative matrix solver. An approximate solve to the linear system results in an approximate update vector that is likely to degrade the convergence rate. The method might also diverge if the linearized system lacks diagonal dominance resulting in an ill-conditioned system with a high condition number. Hence, while Newton's method can be powerful under the right conditions, the method in general lacks robustness. Newton's method is included in the code, but is only applicable to simpler problems.

3.3.1.2 Quasi-Newton Method

A Quasi-Newton method is included that can obtain steady state solutions of Eq. 2.12 when the linear system is too stiff to obtain a solution with Newton's method. A complete linearization of Eq. 2.12, along with a psuedo-time term is used to form the system of linear equations on the entire domain Ω ,

$$\int_{\Omega_e} \psi \frac{\Delta Q^n}{\Delta t_e} d\Omega + \frac{\partial \mathcal{R}(Q^n)}{\partial Q} \Delta Q^n = -\mathcal{R}(Q^n), \quad (3.96)$$

that must be solved for each Quasi-Newton iteration, n . The addition of the pseudo-time term increases the diagonal dominance of the linearized system, and hence, reduces the condition number of the linearization matrix. The pseudo-time term is an approximation to the linearization of the time term and limits the step size of the Quasi-Newton method by adding a diagonal mass matrix to the linearization of $R(Q)$. The mass matrix is diagonal due to the orthogonality of the Legendre polynomials. Only the mean value of the coordinate transformation Jacobian is used in the integration of the pseudo-time term. The mean value is convenient to use as it is the first coefficient in a Legendre polynomial expansion. The solution at iteration $n+1$ is obtained with $Q^{n+1} = Q^n + \Delta Q^n$. The local time step, Δt_e , is computed for each cell using a *CFL* number

$$\Delta t_e = \frac{CFL^n h_e}{\bar{\lambda}_e}$$

where $h_e = \sqrt{\Omega_e}$, $\bar{\lambda}_e = |\bar{V}_e| + \bar{c}_e$ is the cell mean characteristic speed, and $|\bar{V}_e|, \bar{c}_e$ are the cell mean flow speed and speed of sound respectively. The *CFL* number is increased each Quasi-Newton iteration, as proposed by Orkwis and McRae[114], with the formula

$$CFL^n = CFL^0 \frac{\|\mathcal{R}(Q^0)\|}{\|\mathcal{R}(Q^n)\|} \quad (3.97)$$

where CFL^0 is the initial *CFL* number. The initial *CFL* number is typically set to a value between 1 and 10. Thus, the Quasi-Newton method will approach Newton's method as $CFL \rightarrow \infty$. Using the pseudo-time term effectively advances the solution until it is close enough to the final solution that Newton's method does not diverge.

3.3.2 Time Integration

3.3.2.1 Standard 4th-order Runge-Kutta

The explicit Runge-Kutta scheme is suitable for time integration of equations written as

$$\frac{\partial Q}{\partial t} = -\mathcal{R}(Q). \quad (3.98)$$

The solution at time level Q^{n+1} is advance from time level Q^n using the equation

$$Q^{n+1} = Q^n + \frac{1}{6} (dQ_1 + 2dQ_2 + 2dQ_3 + dQ_4)$$

where

$$\begin{aligned} dQ_1 &= -\Delta t \mathcal{R}(Q^n) \\ dQ_2 &= -\Delta t \mathcal{R}\left(Q^n + \frac{1}{2}dQ_1\right) \\ dQ_3 &= -\Delta t \mathcal{R}\left(Q^n + \frac{1}{2}dQ_2\right) \\ dQ_4 &= -\Delta t \mathcal{R}(Q^n + dQ_3). \end{aligned}$$

While this time integration is relatively simple to implement, stability restrictions often require a prohibitively small time step Δt in practice. This is particularly true for viscous flows.

3.3.2.2 Diagonally Implicit Runge-Kutta

The Diagonally Implicit Runge-Kutta (DIRK) scheme is used for implicit time integration of equations written as

$$\frac{\partial Q}{\partial t} = -\mathcal{R}(Q). \quad (3.99)$$

The method is called Diagonally Implicit because the Butcher tableau[131] has the same value on the diagonal, i.e,

$$\begin{array}{c|cc} c & A \\ \hline b^T & \end{array} = \begin{array}{c|ccc} \alpha & \alpha & 0 & 0 \\ \tau_2 & (\tau_2 - \alpha) & \alpha & 0 \\ 1 & b_1 & b_2 & \alpha \\ \hline & b_1 & b_2 & \alpha \end{array}.$$

An L-stable, three-stage, third-order accurate[132] DIRK scheme advances the solution in time with the equation

$$Q^{n+1} = Q^n + b_1 dQ_1 + b_2 dQ_2 + \alpha dQ_3, \quad (3.100)$$

where the advancement vectors are updated with the implicit equations

$$\begin{aligned} dQ_1 &= -\Delta t \mathcal{R}(Q^n + \alpha dQ_1), \\ dQ_2 &= -\Delta t \mathcal{R}(Q^n + (\tau_2 - \alpha) dQ_1 + \alpha dQ_2), \\ dQ_3 &= -\Delta t \mathcal{R}(Q^n + b_1 dQ_1 + b_2 dQ_2 + \alpha dQ_3), \end{aligned} \quad (3.101)$$

and the weights in the Butcher tableau are

$$\begin{aligned}
\alpha &= 0.435866521508459 \\
\tau_2 &= (1 + \alpha)/2 \\
b_1 &= -(6\alpha^2 - 16\alpha + 1)/4 \\
b_2 &= (6\alpha^2 - 20\alpha + 5)/4.
\end{aligned} \tag{3.102}$$

The value of α is obtained by finding the root of $x^3 - 3x^2 + \frac{3}{2}x - \frac{1}{6} = 0$ lying in $(\frac{1}{6}, \frac{1}{2})$. The implicit system of equations in Eq. 3.101 are solved on each stage using Newtons's method

$$\begin{aligned}
\left(I + \alpha \Delta t \frac{\partial \mathcal{R}}{\partial Q} \right) \Delta Q_i^k &= -dQ_i^k - \Delta t \mathcal{R} \left(Q_n + \sum_{j=1}^{i-1} a_{ij} dQ_j + \alpha dQ_i^k \right), \\
dQ_i^{k+1} &= \Delta Q_i^k + dQ_i^k.
\end{aligned} \tag{3.103}$$

3.4 Verification Test Cases

The problems presented in this section are used to verify that the code is properly implemented, i.e. the mathematical expressions in the code do not contain errors.[133] Verification is different from validation where solutions are compared with experimental data to confirm that the mathematical model is representative of physical phenomena. Computing the order of accuracy is a powerful technique for verification.[133] Any errors in the implementation of the mathematical method will, in general, result in a loss of the anticipated order of accuracy. The order of accuracy is obtained by computing a discretization error for a series of grids that retain the same point distribution function. When the error is plotted vs. a mesh spacing parameter, h , and the slope of the line is the order of accuracy. The mesh spacing parameter is taken as

$$\begin{aligned}
h &= \frac{1}{DOF}, \quad 1D, \\
h &= \frac{1}{\sqrt{DOF}}, \quad 2D.
\end{aligned} \tag{3.104}$$

where DOF is the number of degrees of freedom (i.e., number of unknowns per equation). Any finite element scheme that uses polynomial basis functions is expected to achieve an order of accuracy of $N + 1$.

3.4.1 Scalar Equations

The scalar equations in Section 2.2.1 are used to verify that the general framework of the code is properly implemented. The error used to compute the order of accuracy is an $L^2 - norm$ of the difference between the analytical, U , and numerical, u , solutions to the partial differential equation, i.e.

$$L^2 = \sqrt{\int_{\Omega} (U - u)^2 d\Omega}. \quad (3.105)$$

3.4.1.1 Linear Poisson Equation

Solutions obtained for the linear Poisson equation in Eq. 2.25 with $\mu(u) = 1$ are used to verify proper implementation of the BR2 discretization of diffusive fluxes. The method of manufactured solutions[133] is used to derive a source term so that the linear Poisson equation satisfies a chosen analytical solution. The source term is derived by choosing an analytical solution and differentiating the analytical solution in accord with diffusive flux in Eq. 2.25, i.e, given the analytical function $U(\vec{x})$, the source term, $S(\vec{x})$, is

$$\nabla \cdot \vec{F}^d(U(\vec{x}), \nabla U(\vec{x})) = -S(\vec{x}). \quad (3.106)$$

One-Dimension: The analytical solution is taken as[90]

$$U(x) = \sin(2\pi x) + 1 - x \quad x \in [0, 1], \quad (3.107)$$

which results in the source term

$$S(x) = -4\pi^2 \sin(2\pi x). \quad (3.108)$$

The boundary conditions are chosen as Dirichlet at $x = 0$ and Neumann at $x = 1$,

$$\begin{aligned} U(0) &= 1, \\ \frac{dU}{dx}(1) &= 2\pi. \end{aligned} \quad (3.109)$$

The L^2 -errors computed from solutions to the linear Poisson equation on a series of meshes with a uniform cell distribution and with increasing order of the approximation are shown in Fig. 3.7. The coarsest mesh consists of 5 cells. The finest mesh for $N = 1$ consists of 10,240 cells, and the finest mesh for the remaining orders of approximation consists of 640 cells. The slopes of the lines in Fig. 3.7 indicate that the scheme achieves the optimal convergence rate of $N + 1$. Solutions obtained on the coarsest mesh with 5 cells for each order are shown in Fig. 3.8. While the linear approximation, $N = 1$, captures the solution reasonably well, the higher-order approximations are nearly indistinguishable from the analytical solution. Inspection of the errors in Fig. 3.7, shows that the linear approximation requires 320 cells (640 DOF) to achieve the same error that the quartic solution $N = 4$ obtained with 5 cells (25 DOF).

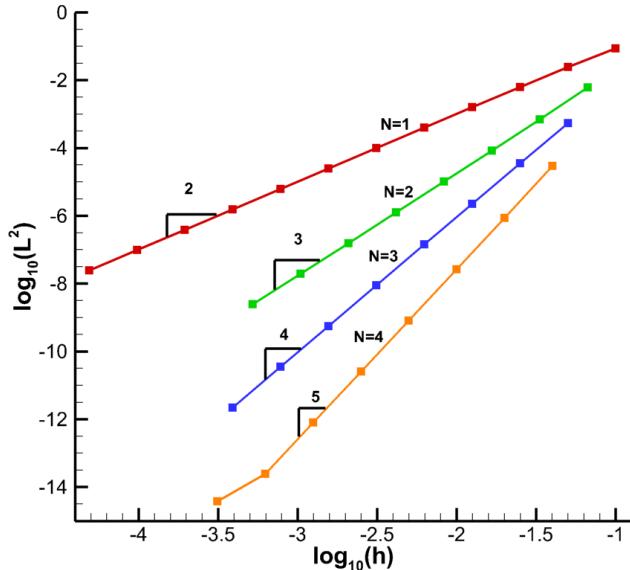


Figure 3.7: Convergence Rates for the One-dimensional Linear Poisson Equation

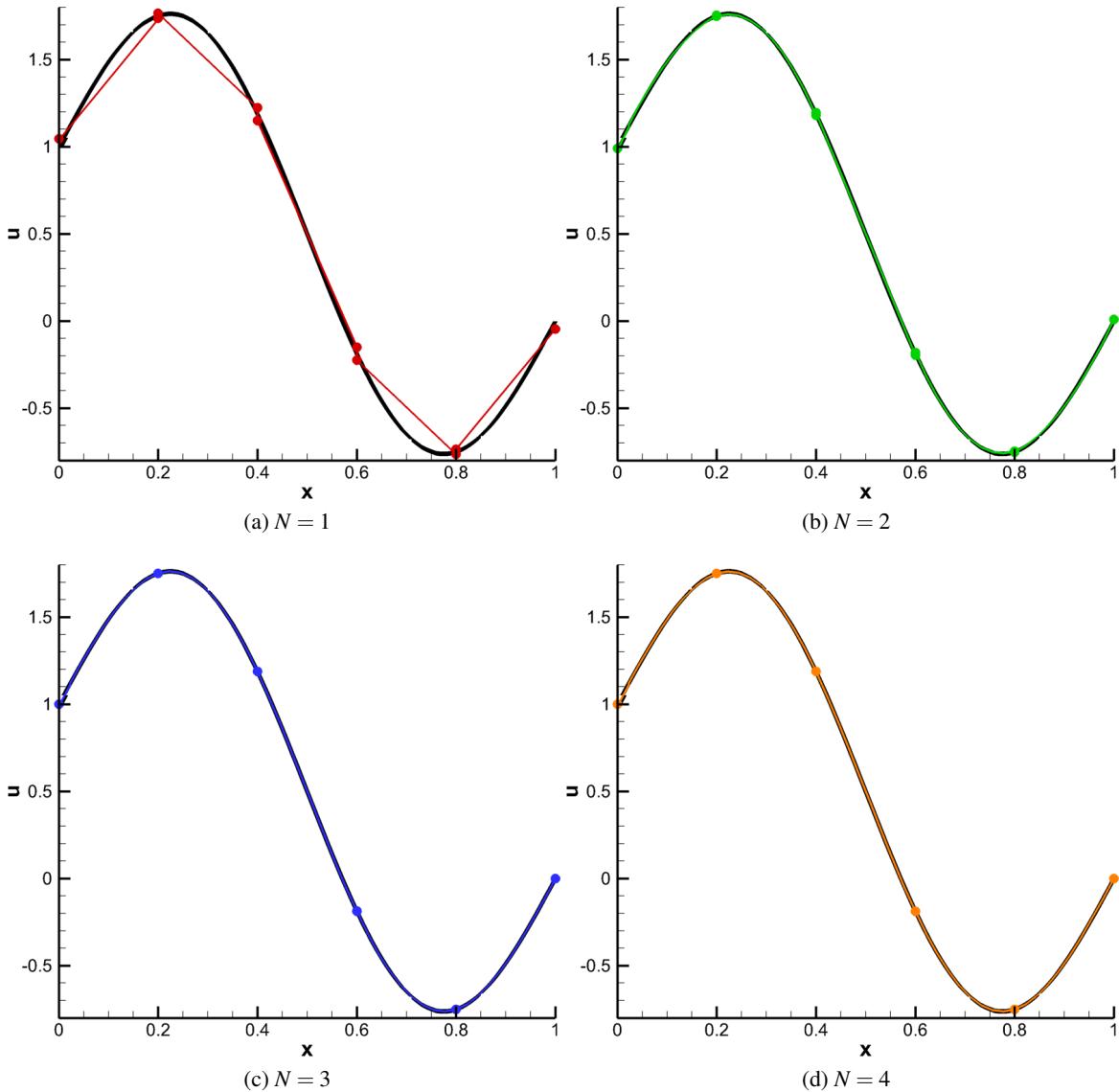


Figure 3.8: One-Dimensional Linear Poisson Equation Solutions with Five Cells (Black Line - Analytical Solution)

Two-Dimensions: The two dimensional analytical solution to the linear Poisson equation is taken as

$$U(x,y) = \sin(\pi x) \sin(\pi y), \quad (3.110)$$

which produces the source term

$$S(x,y) = -2\pi^2 \sin(\pi x) \sin(\pi y). \quad (3.111)$$

The computational domain is defined as $[0, 1] \times [0, 1]$ with the boundary conditions set as Dirichlet boundary conditions on all boundaries

$$\begin{aligned} U(0, y) &= U(1, y) = 0, \\ U(x, 0) &= U(x, 1) = 0. \end{aligned} \quad (3.112)$$

The L^2 -errors computed on a series of meshes ranging from 8×8 cells to 256×256 cells with an increasing order of the approximation is shown in Fig. 3.9. The slope of the lines in Fig. 3.9 show that the expected convergence rate of $N + 1$ is obtained. Note that that the errors on the coarsest mesh with 8×8 cells and $N = 4$ (64×64 DOF) is lower than the error on the finest mesh with 256×256 cells and $N = 1$ (512×512 DOF) as shown in Fig 3.9. Solutions on the coarsest mesh for the four different orders of approximation are shown in Fig. 3.10. Here it is evident that the higher-order approximations, $N \geq 2$, are better able to capture the curvature of the analytical solution compared with the linear approximation, $N = 1$.

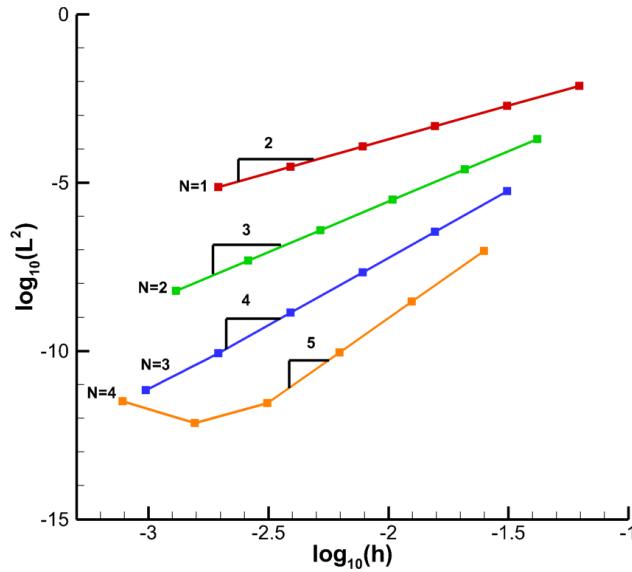


Figure 3.9: Convergence Rates for the Two-dimensional Linear Poisson Equation

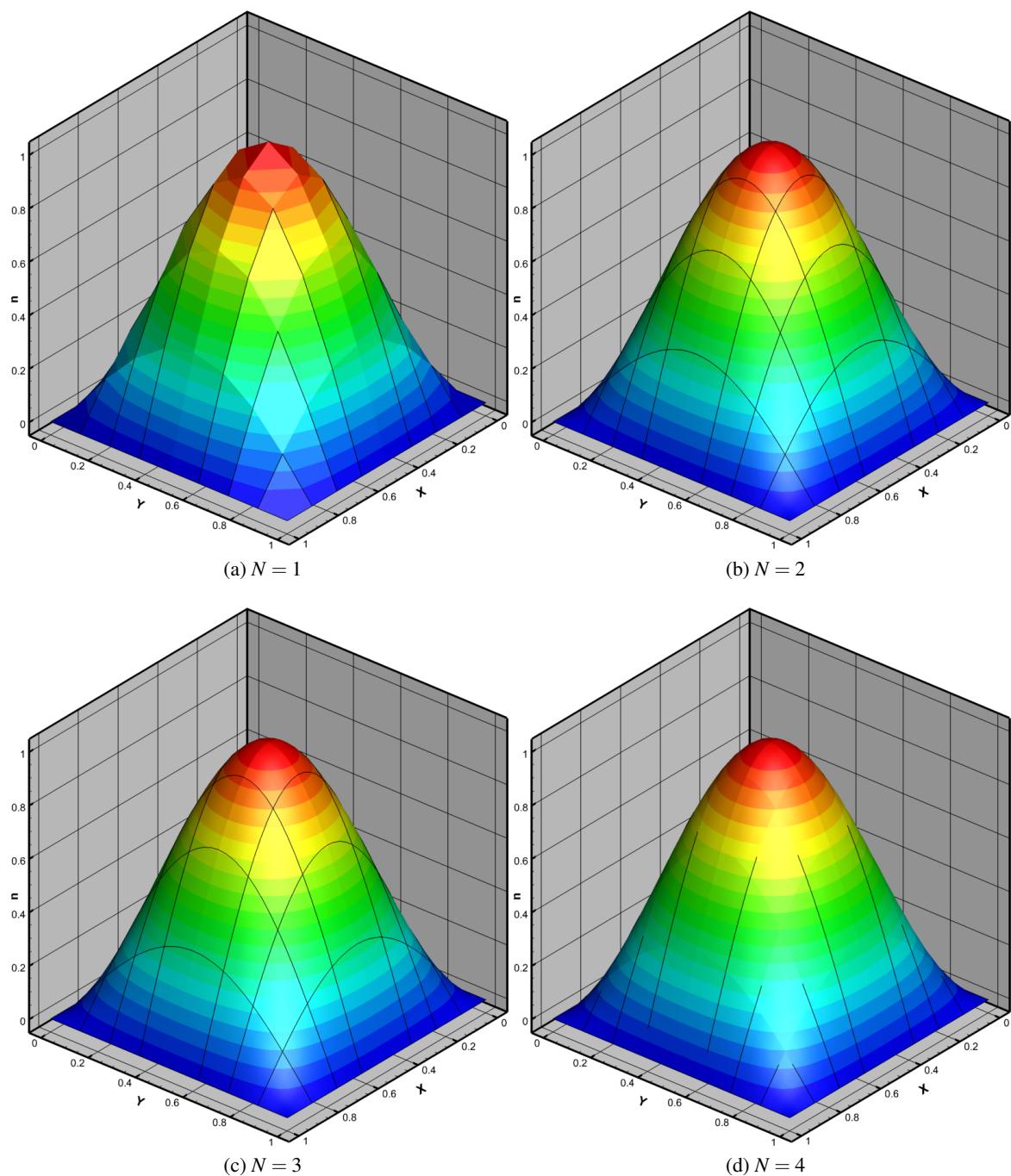


Figure 3.10: Two-Dimensional Linear Poisson Equation Solutions with 8×8 Cells

3.4.1.2 One-Dimensional Non-Linear Poisson Equation

A one-dimensional non-linear Poisson equation is used to verify the implementation of the BR2 scheme with a non-linear viscosity coefficient $\mu(u)$. [134] Using an exponential non-linear viscosity coefficient

$$\mu(u) = e^{\lambda u}, \quad (3.113)$$

where $\lambda \neq 0$ is an arbitrary constant, Eq. 2.25 with $S(x) = 0$ has an analytical solution

$$U(x) = \frac{1}{\lambda} \ln \left(\frac{A+Bx}{D} \right), \quad (3.114)$$

where A , B , and D are arbitrary coefficients. The boundary conditions are set as Dirichlet boundary conditions at $x = 0$ and $x = 1$;

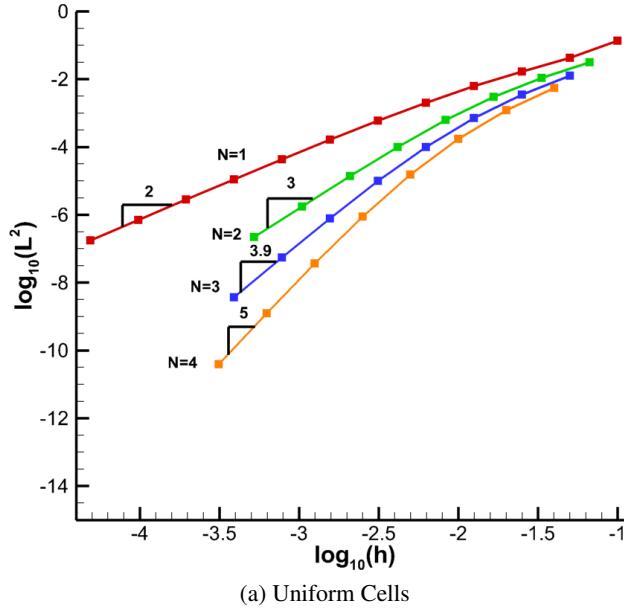
$$\begin{aligned} U(0) &= \frac{1}{\lambda} \ln \left(\frac{A}{D} \right), \\ U(1) &= \frac{1}{\lambda} \ln \left(\frac{A+B}{D} \right). \end{aligned} \quad (3.115)$$

Solutions to the non-linear Poisson equation are obtained with $A = 1$, $B = 50$, $D = 1$, and $\lambda = 1$. Two mesh distribution functions are used to compute L^2 -errors: a uniform distribution and a distribution with the clustering function

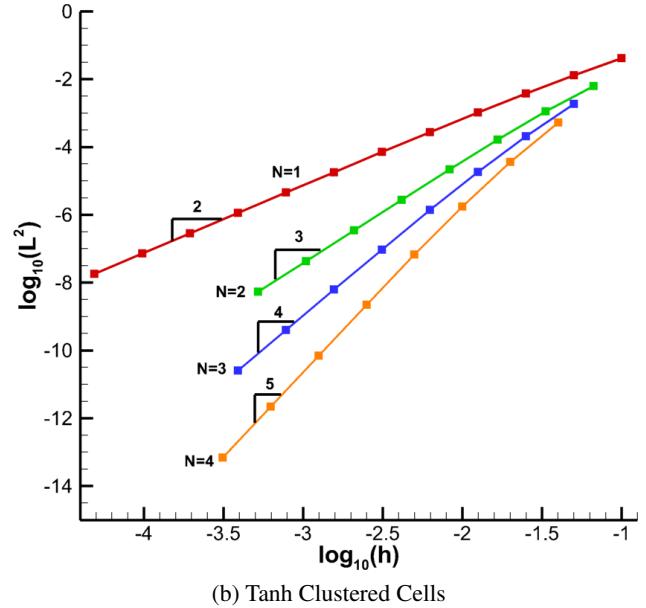
$$x(i) = 1 - \tanh \left[\frac{\pi}{2} \left(1 - \frac{i}{(i_{max} - 1)} \right) \right] / \tanh \left(\frac{\pi}{2} \right) \quad (3.116)$$

where i is an integer and i_{max} is the number of nodes used to define the cells of the mesh. The clustering function concentrates the cells toward $x = 0$ where the curvature of the analytical solution is the highest. The L^2 -errors computed on the two set of meshes with increasing cell count and increasing order of approximation are shown in Fig. 3.11. The coarsest meshes consist of 5 cells. The finest mesh consists of 10,240 cells for $N = 1$, and 640 cells for $N \geq 2$. The expected order of accuracy of $N + 1$ is obtained for both mesh distributions as shown by the slopes of the lines in Fig. 3.11. As expected, the L^2 -errors are lower for the clustered mesh distributions compared with the uniform mesh distributions. Solutions obtained using meshes with five uniform cells and five clustered cells are shown in Figs. 3.12 and 3.13, respectively. From these figures it is evident that the computed solutions on the clustered mesh are better able to resolve

the high curvature region of the solution near $x = 0$.



(a) Uniform Cells



(b) Tanh Clustered Cells

Figure 3.11: Convergence Rates for the One-dimensional Non-Linear Poisson Equation

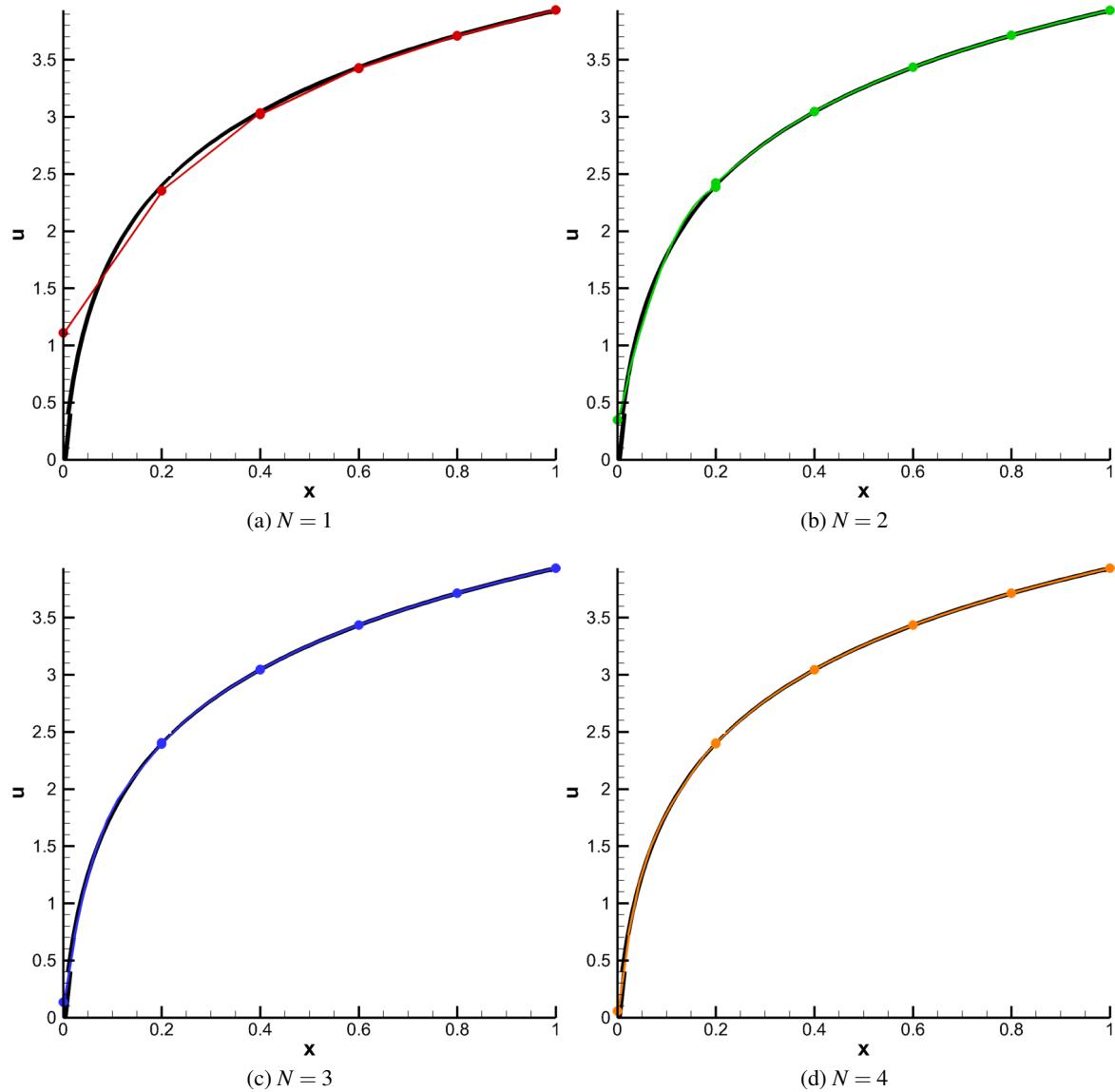


Figure 3.12: One-Dimensional Non-Linear Poisson Equation Solutions with Five Uniform Cells (Black Line - Analytical Solution)

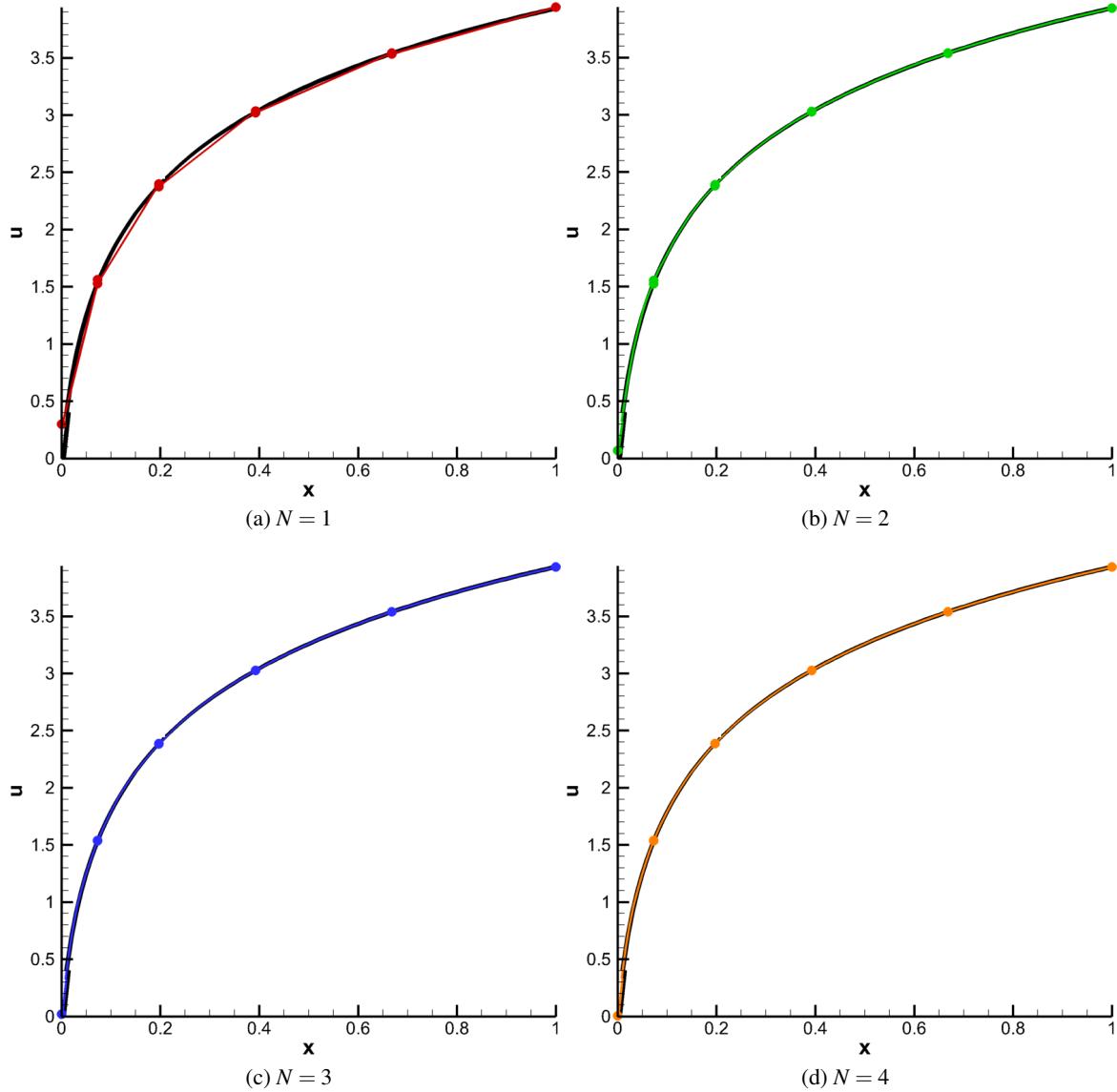


Figure 3.13: One-Dimensional Non-Linear Poisson Equation Solutions with Five Clustered Cells (Black Line - Analytical Solution)

3.4.1.3 Linear Advection Diffusion Equation

The linear advection diffusion equation is used to verify the implementation of the advection and upwind fluxes in the general framework of the code. A three-dimensional solution to the linear advection diffusion equation in Eq.2.30 [135, 115] is

$$U_{3D}(x, y, z) = \left(\frac{1 - \exp \left[(x-1) \frac{c_x}{\mu} \right]}{1 - \exp \left[-\frac{c_x}{\mu} \right]} \right) \left(\frac{1 - \exp \left[(y-1) \frac{c_y}{\mu} \right]}{1 - \exp \left[-\frac{c_y}{\mu} \right]} \right) \left(\frac{1 - \exp \left[(z-1) \frac{c_z}{\mu} \right]}{1 - \exp \left[-\frac{c_z}{\mu} \right]} \right), \quad (3.117)$$

with the Dirichlet boundary conditions

$$\begin{aligned} U_{3D}(x, 0, z) &= \left(\frac{1 - \exp \left[(x-1) \frac{c_x}{\mu} \right]}{1 - \exp \left[-\frac{c_x}{\mu} \right]} \right) \left(\frac{1 - \exp \left[(z-1) \frac{c_z}{\mu} \right]}{1 - \exp \left[-\frac{c_z}{\mu} \right]} \right), & U_{3D}(x, 1, z) &= 0, \\ U_{3D}(0, y, z) &= \left(\frac{1 - \exp \left[(y-1) \frac{c_y}{\mu} \right]}{1 - \exp \left[-\frac{c_y}{\mu} \right]} \right) \left(\frac{1 - \exp \left[(z-1) \frac{c_z}{\mu} \right]}{1 - \exp \left[-\frac{c_z}{\mu} \right]} \right), & U_{3D}(1, y, z) &= 0, \\ U_{3D}(x, y, 0) &= \left(\frac{1 - \exp \left[(x-1) \frac{c_x}{\mu} \right]}{1 - \exp \left[-\frac{c_x}{\mu} \right]} \right) \left(\frac{1 - \exp \left[(y-1) \frac{c_y}{\mu} \right]}{1 - \exp \left[-\frac{c_y}{\mu} \right]} \right), & U_{3D}(x, y, 1) &= 0. \end{aligned} \quad (3.118)$$

One-Dimension: A one-dimension solution to the linear advection diffusion is

$$U_{1D}(x) = U_{3D}(x, 0, 0) = \left(\frac{1 - \exp \left[(x-1) \frac{c_x}{\mu} \right]}{1 - \exp \left[-\frac{c_x}{\mu} \right]} \right) \quad (3.119)$$

with the Dirichlet boundary conditions

$$\begin{aligned} U_{1D}(0) &= 1, \\ U_{1D}(1) &= 0. \end{aligned} \quad (3.120)$$

Solutions are obtained with $c_x = 1$ and $\mu = 0.05$. Meshes with two different distribution functions are used to compute the L^2 -errors. The first distribution function is a uniform distribution and the second distribution function clusters the cells toward $x = 1$ (where the derivative of the solution is large) with the equation

$$x(i) = \tanh \left[\frac{\pi}{2} \frac{i}{(i_{max} - 1)} \right] / \tanh \left(\frac{\pi}{2} \right), \quad (3.121)$$

where i is an integer and i_{max} is the number of nodes used to define the cells of the mesh. L^2 -errors computed with an increasing cell count on the meshes with the two distribution functions and increasing order of approximation are shown in Fig. 3.14. The coarsest meshes consist of 5 cells. The finest mesh consist of 10,240 cells for $N = 1$, and 640 cells for $N \geq 2$. The slopes of the lines in Fig. 3.14 indicate that the expected order of accuracy of $N + 1$ is obtained with both mesh distributions. The non-linear reduction in error for $N = 4$ at the finest mesh resolutions in Fig. 3.14b is due to machine precision round off. Solutions computed on the mesh with five cells and a uniform cell distribution are shown in Fig. 3.15, and solutions computed on the mesh with five cells and the clustered cell distribution are shown in Fig. 3.16. The large derivative in the analytical solution at $x = 1$ is only resolved with one cell on the uniform mesh. As a result, a overshoot is observed in the numerical solutions for the lower orders of approximation $N \leq 2$. However, the higher orders of approximation $N \geq 3$ are able to capture the large derivative region reasonably well with a single cell. The clustered mesh has two cells within the region where the derivative of the analytical solution is large. Hence, only the solution with $N = 1$ exhibits an overshoot and the remaining orders of approximation capture the analytical solution well. The reduction in error on the clustered mesh relative to the uniform mesh is shown in the L^2 -errors in Fig. 3.14. While the slopes of the lines in Fig. 3.14 are the same, the lines for the clustered mesh are lower relative to the uniform mesh distribution. *This demonstrates that the magnitude of the error is dependent on the mesh distribution, but the order of accuracy is not.*

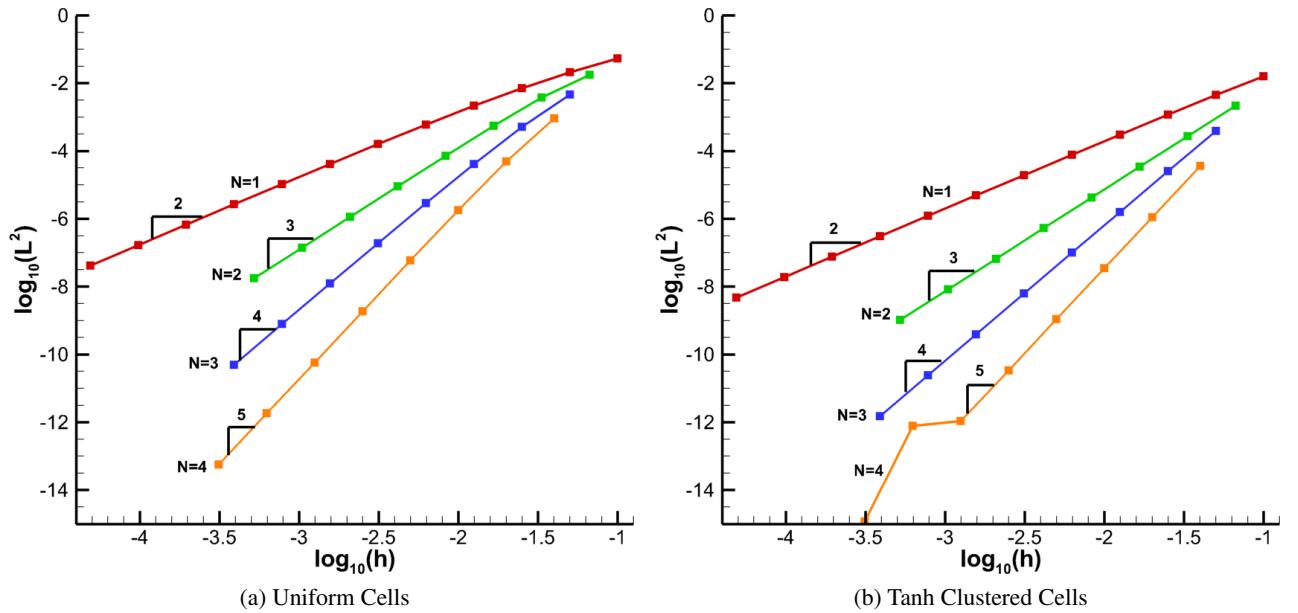


Figure 3.14: Convergence Rates for the One-dimensional Linear Advection Diffusion Equation

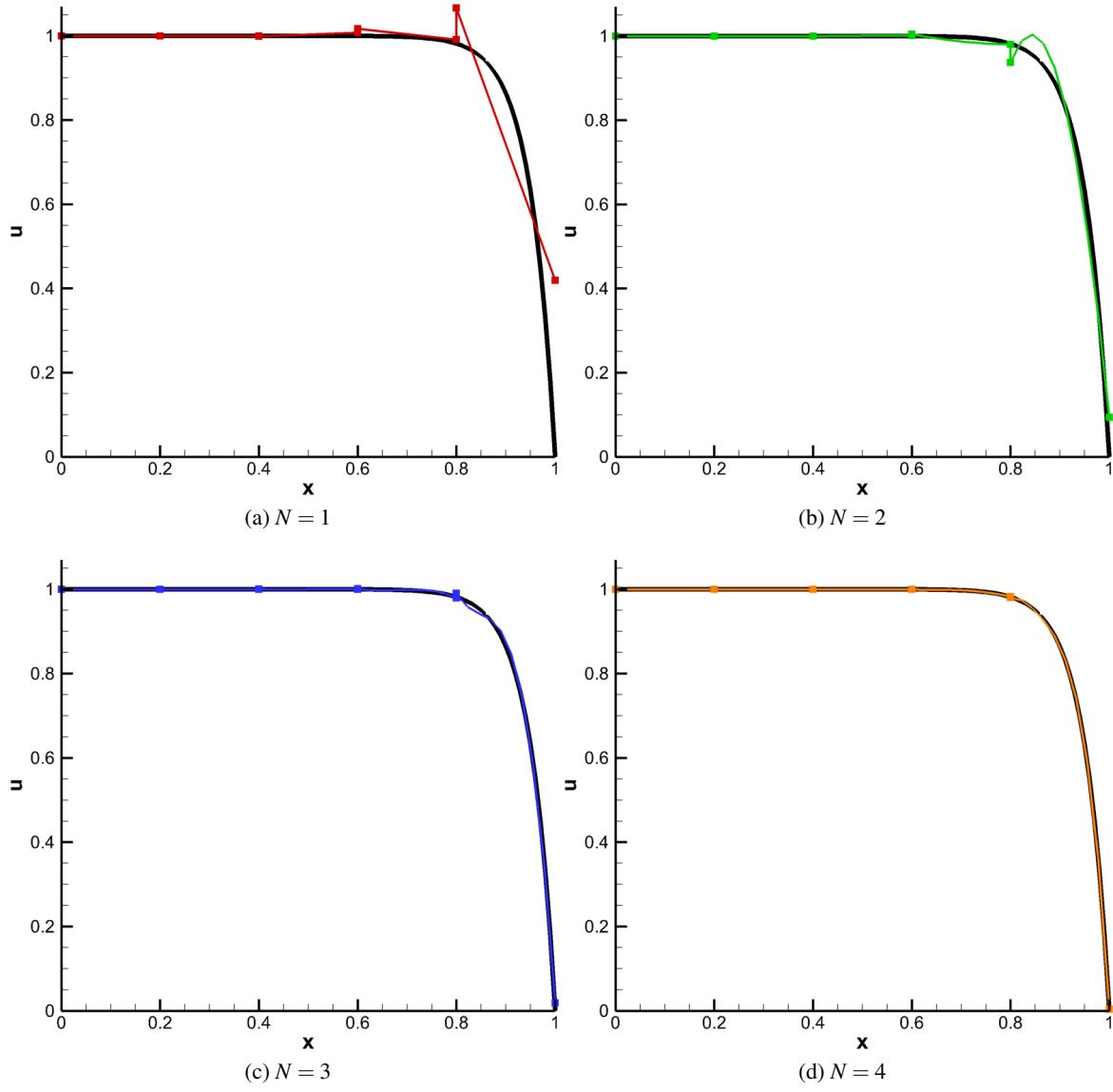


Figure 3.15: One-Dimensional Linear Advection Diffusion Equation Solutions with Five Uniform Cells
(Black Line - Analytical Solution)

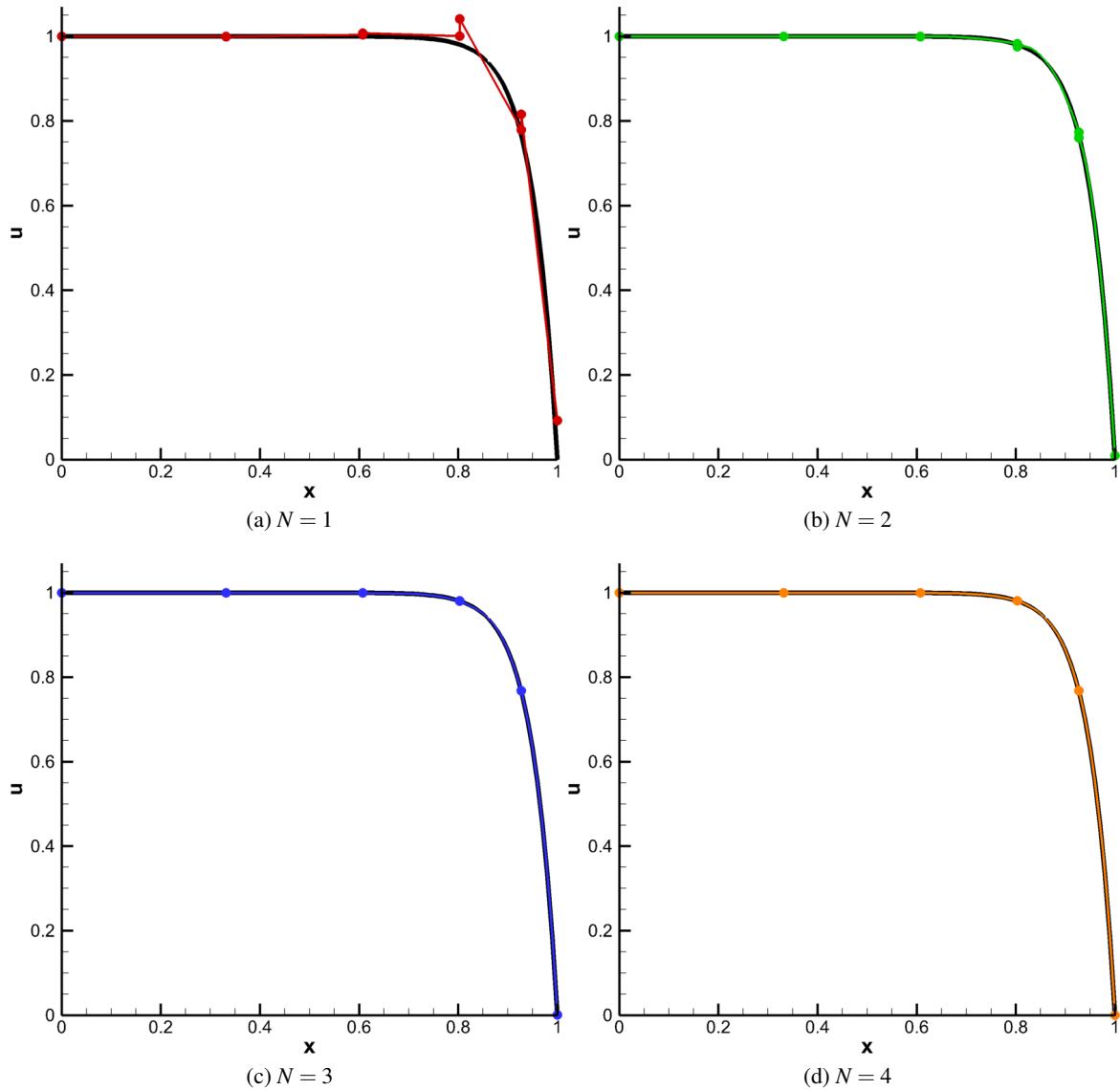


Figure 3.16: One-Dimensional Linear Advection Diffusion Equation Solutions with Five Clustered Cells
(Black Line - Analytical Solution)

Two-Dimensions: The two-dimensional solution to the linear advection diffusion equation (Eq. 2.30) is

$$U_{2D}(x, y) = U_{3D}(x, y, 0) = \left(\frac{1 - \exp \left[(x-1) \frac{c_x}{\mu} \right]}{1 - \exp \left[-\frac{c_x}{\mu} \right]} \right) \left(\frac{1 - \exp \left[(y-1) \frac{c_y}{\mu} \right]}{1 - \exp \left[-\frac{c_y}{\mu} \right]} \right), \quad (3.122)$$

with the Dirichlet boundary conditions

$$\begin{aligned}
U_{2D}(x, 0) &= \left(\frac{1 - \exp \left[(x-1) \frac{c_x}{\mu} \right]}{1 - \exp \left[-\frac{c_x}{\mu} \right]} \right), \quad U_{2D}(x, 1) = 0, \\
U_{2D}(0, y) &= \left(\frac{1 - \exp \left[(y-1) \frac{c_y}{\mu} \right]}{1 - \exp \left[-\frac{c_y}{\mu} \right]} \right), \quad U_{2D}(1, y) = 0.
\end{aligned} \tag{3.123}$$

Solutions to Eq. 2.30 are obtained with $\vec{c} = [c_x, c_y] = [1, 0.5]$ and $\mu = 0.05$. The L^2 -errors shown in Fig. 3.17 are computed using a series of uniform meshes ranging from 8×8 cells to 256×256 cells. Without loss in generality, only uniform meshes are considered here. The slope of the lines in Fig. 3.17 demonstrate that the expected order of accuracy of $N+1$ is obtained. Solutions with increasing orders of approximation computed using the coarsest 8×8 mesh are shown in Fig. 3.18. The higher-order approximations, $N \geq 2$, are better able to capture the high gradient region of the solution on the boundaries with $x = 1$ and $y = 1$ relative to the 2^{nd} -order approximation, $N = 1$.

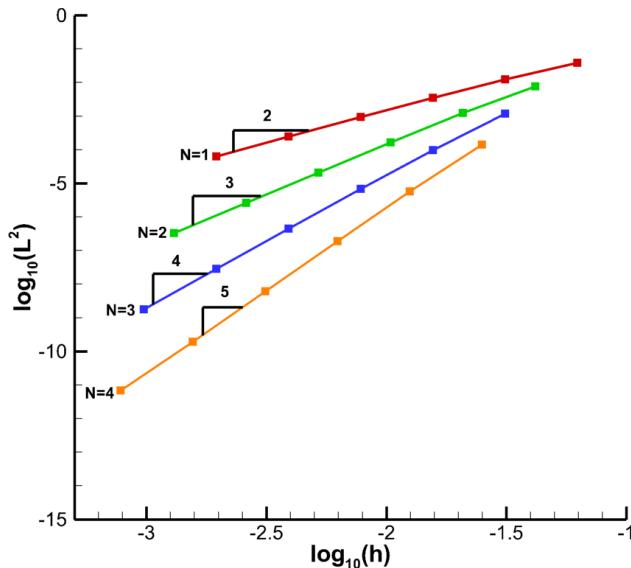


Figure 3.17: Convergence Rates for the Two-dimensional Linear Advection Diffusion Equation

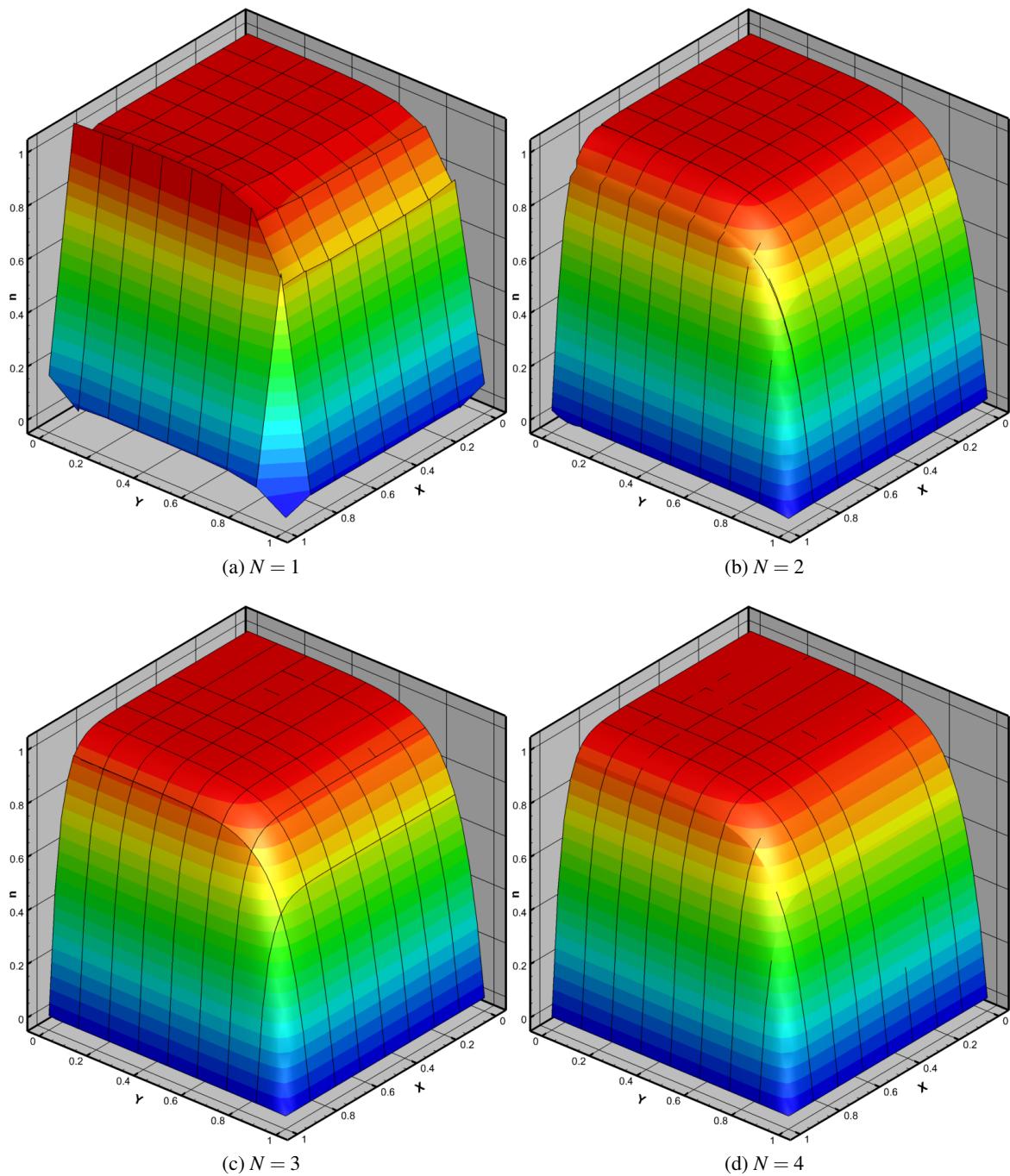


Figure 3.18: Two-Dimensional Linear Advection Diffusion Equation Solutions with 8×8 Uniform Cells

3.4.1.4 Burger's Equation

Burger's equation given in Eq. 2.33 is commonly used for testing a computational fluid dynamics code as it is representative of the Navier-Stokes equations. A commonly used solution to the one-dimensional Burger's equations is

$$U(x) = -\tanh\left(\frac{x}{2\mu}\right). \quad (3.124)$$

This one-dimensional solution can be extended to multiple dimensions as

$$U_{2D}(x, y) = U(\vec{c} \cdot \vec{x}) = -\tanh\left(\frac{\vec{c} \cdot \vec{x}}{2\mu}\right), \quad (3.125)$$

which is the one-dimensional equation along the direction vector \vec{c} . The derivation of the multidimensional solution is provided in Appendix D.

One-Dimension: Solutions to the one-dimensional Burger's equation using a series of uniform meshes are obtained with $\mu = 0.5$ and the Dirichlet boundary conditions

$$\begin{aligned} U(-4) &= -\tanh\left(\frac{-2}{\mu}\right), \\ U(4) &= -\tanh\left(\frac{2}{\mu}\right). \end{aligned} \quad (3.126)$$

The L^2 -error computed on a series of uniform meshes with increasing order of approximation is shown in Fig. 3.19. The smallest mesh size consists of 5 cells. The finest mesh consists of 10,240 cells for $N = 1$ and 640 cells for $N \geq 2$. The slopes of the lines in Fig. 3.19 indicate that the expected order of accuracy of $N + 1$ is obtained. Solutions with increasing order of accuracy computed using the coarsest mesh with 5 cells are shown in Fig. 3.20. Overshoots are observable in the solution with a linear approximation, $N = 1$, in the regions of high curvature. The higher orders of approximation, $N \geq 2$, capture the analytical solution reasonably well.

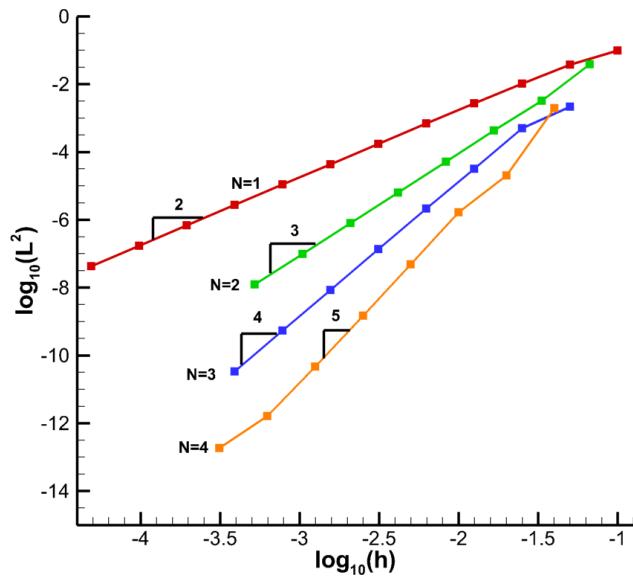


Figure 3.19: Convergence Rates for the One-dimensional Burger's Equation

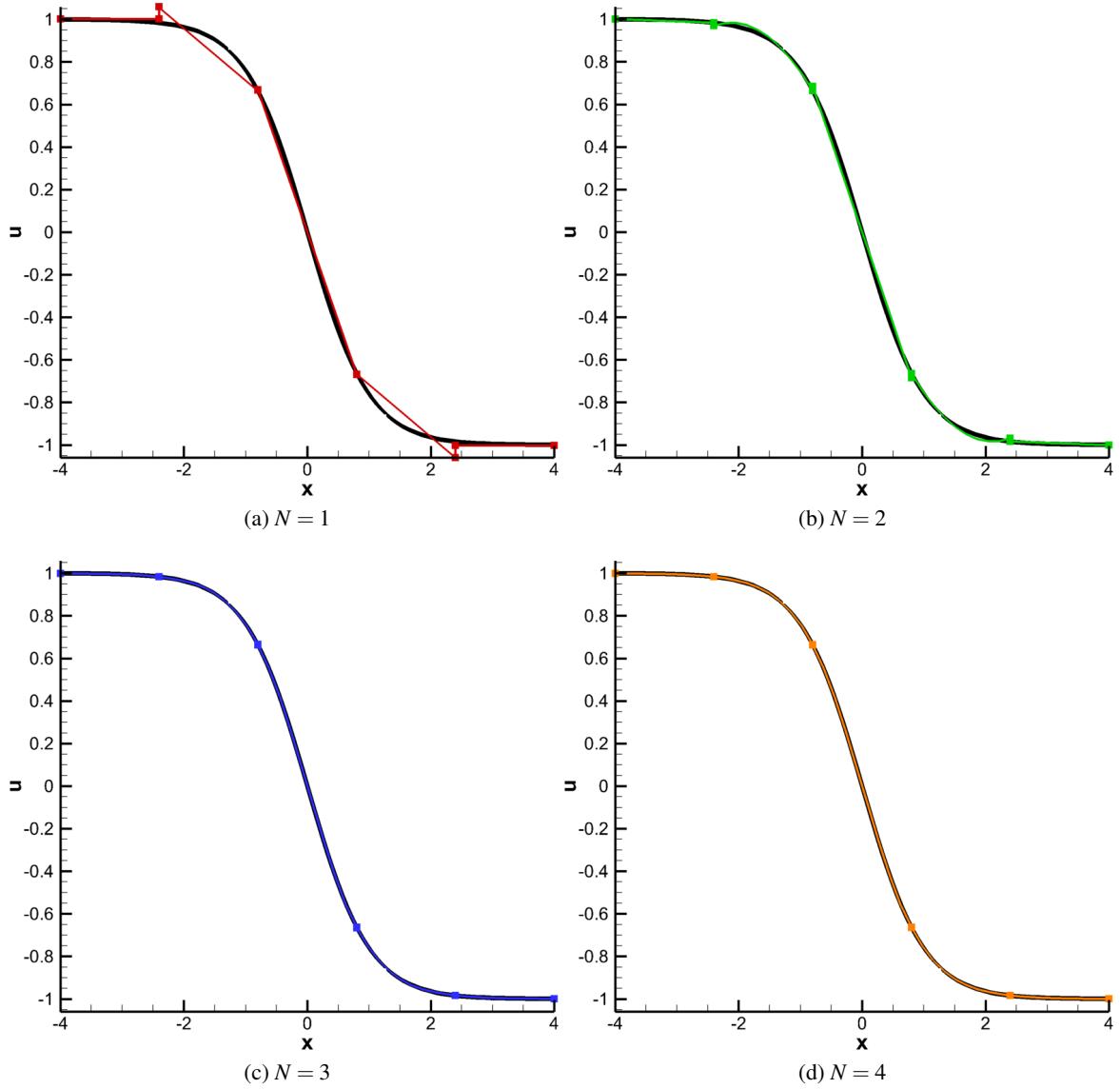


Figure 3.20: One-Dimensional Burger's Equation Solutions with Five Uniform Cells

Two-Dimensions: Two-dimensional solutions to the Burgers equation with $\vec{c} = \begin{bmatrix} 1, & 0.5 \end{bmatrix}$ and $\mu = 0.5$ with the boundary conditions

$$\begin{aligned} U_{2D}(-4, y) &= -\tanh\left(\frac{-c_x 4 + c_y y}{2\mu}\right), & U_{2D}(4, y) &= -\tanh\left(\frac{c_x 4 + c_y y}{2\mu}\right), \\ U_{2D}(x, -4) &= -\tanh\left(\frac{c_x x - c_y 4}{2\mu}\right), & U_{2D}(x, 4) &= -\tanh\left(\frac{c_x x + c_y 4}{2\mu}\right), \end{aligned} \quad (3.127)$$

are used to compute the L^2 -errors using uniform meshes with increasing cell count shown in Fig. 3.21. The mesh cell count ranges from 8×8 cells to 64×64 cells. The expected order of accuracy of $N + 1$ is obtained as indicated by the slopes of the lines in Fig. 3.21. Solutions computed using the coarsest mesh for all orders of accuracy are shown in Fig. 3.22. The higher order approximations, $N \geq 2$, are better able to resolve the high curvature region relative to the 2nd-order approximation, $N = 1$.

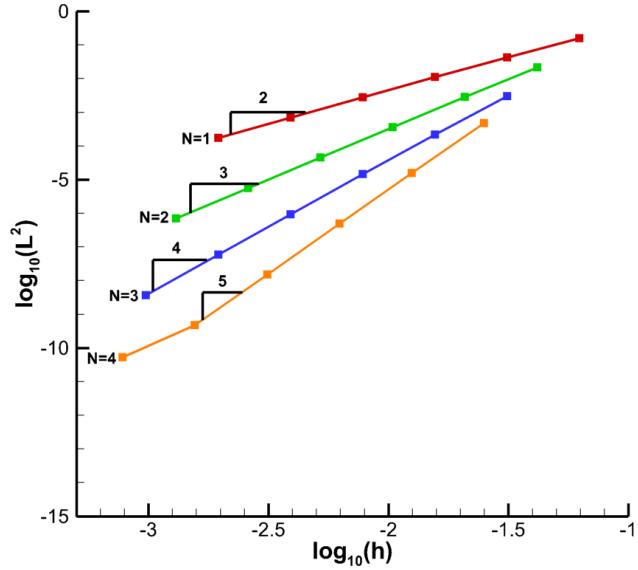


Figure 3.21: Convergence Rates for the Two-dimensional Burger's Equation (Black Line - Analytical Solution)

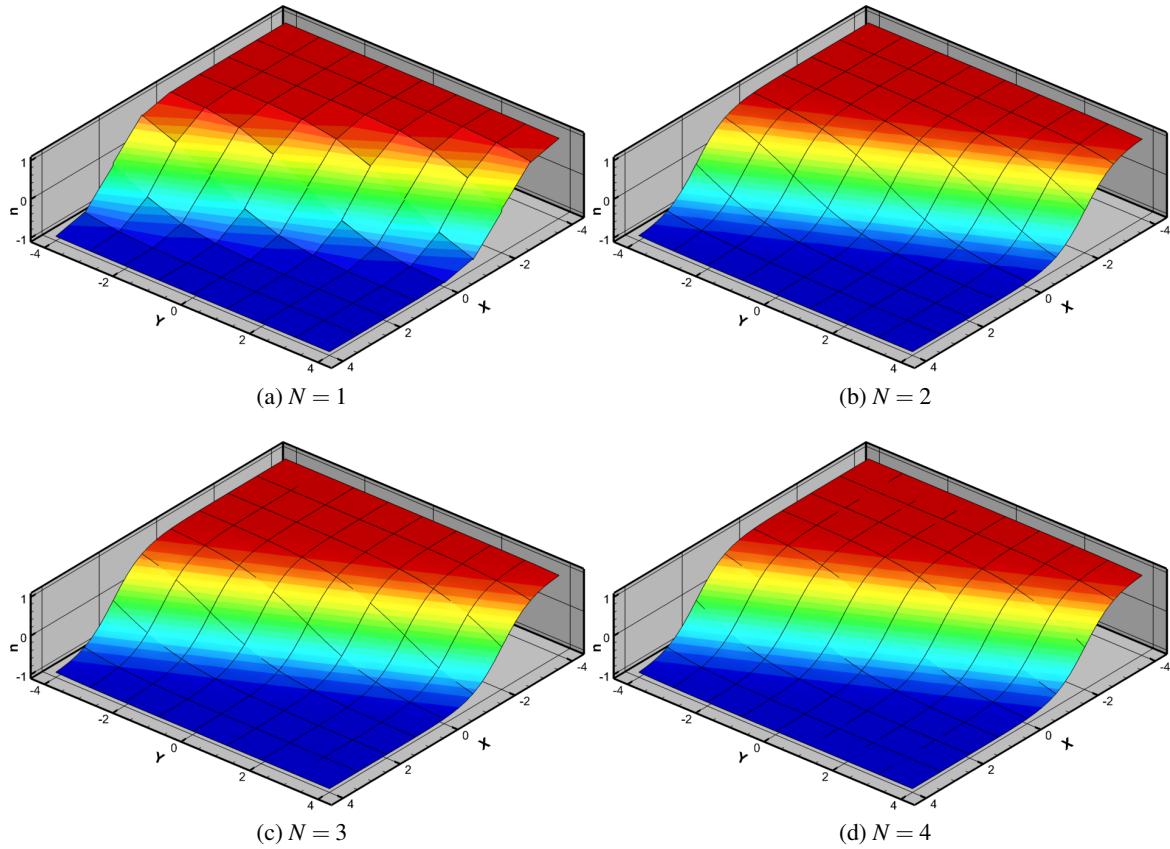


Figure 3.22: Two-Dimensional Burger's Equation Solutions with 8×8 Uniform Cells

3.4.2 Inviscid Flow

A set of inviscid flow problems are used to verify the implementation of the inviscid fluxes in the Navier-Stokes equations, i.e., the Euler equations. The first test case is a Gaussian smooth bump in a channel. This test case is used to verify the order of accuracy as well as compare the computational efficiency of the DG code relative to commercially available codes that utilize a finite volume discretization. The second problem is a smooth shock-free transonic flow problem known as the Ringleb flow. This is one of the few known analytical solutions to the compressible Euler equations. Finally, a shock problem with an oblique shock is used to verify the implementation of the artificial viscosity shock capturing scheme.

3.4.2.1 Gaussian Smooth Bump

Channel flow with a Gaussian smooth bump[45] is used to verify the that the code achieves the expected order of accuracy for an inviscid internal flow. The computational domain of the channel is defined in Fig. 3.23. Slip wall boundary conditions are imposed by enforcing $\vec{V} \cdot \vec{n} = 0$ on the upper and lower boundaries. The pressure for the slip wall boundary conditions is the pressure from the interior cell evaluated on the wall. The left inflow boundary specifies total pressure and temperature corresponding to $M_\infty = 0.5$, as well as a zero flow angle. A constant back pressure is applied to the right outflow boundary. A series of uniform quadrilateral grids with a quartic, $N_g = 4$, polynomial mapping are used to compare the entropy error, defined as

$$\text{Entropy Error} = \sqrt{\frac{\int \left(\frac{\frac{p}{\rho^\gamma} - \frac{p_\infty}{\rho_\infty^\gamma}}{\frac{p_\infty}{\rho_\infty^\gamma}} \right)^2 d\Omega}{\int d\Omega}}, \quad (3.128)$$

The entropy error is also computed on the same series of grids using Fidkowski et al.'s[118] XFLOW code, which also utilizes a discontinuous Galerkin discretization. A comparison of the entropy error for the two codes with increasing grid refinement and increasing order of approximation is shown in Fig. 3.24. The entropy error decreases with the expected order of accuracy of $N + 1$ and agrees well with values obtained with XFLOW. Contours of pressure coefficient computed using the coarsest grid with 6×2 cells with increasing order of accuracy are shown in Fig. 3.25. The grid with 6×2 is too coarse to capture any flow features with the piecewise constant approximation, i.e., $N = 0$. As the order of the approximation is increased in Fig. 3.25, the pressure coefficient contours approaches the fine grid pressure coefficient contours shown in Fig. 3.26. The pressure coefficient contours obtained with the 6×2 grid and $N = 4$ visually agrees reasonably well with the pressure contours computed with the fine grid shown in Fig. 3.26.

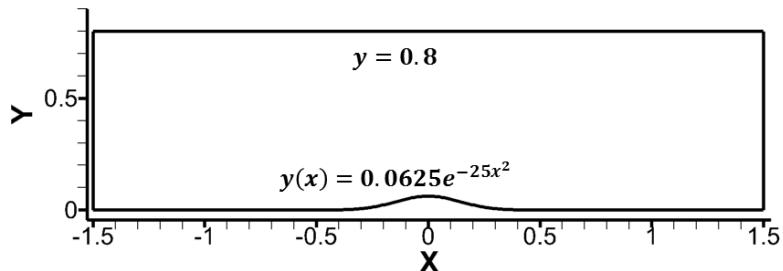


Figure 3.23: Smooth Bump Geometry

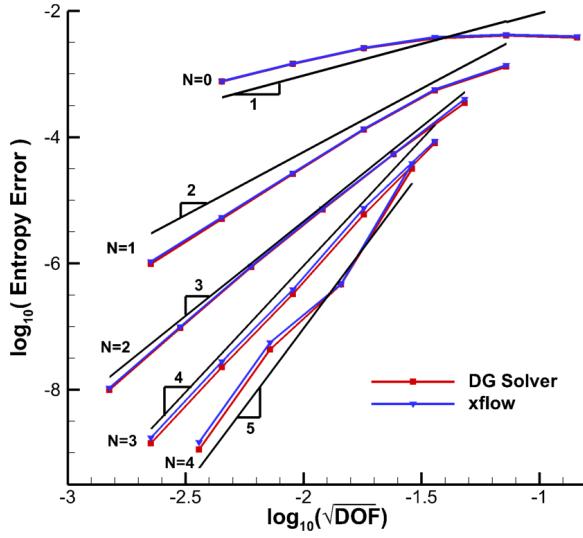


Figure 3.24: Smooth Bump Spatial Order of Accuracy Verification with XFLOW

The convergence history for all orders of approximation and each grid size is shown in Fig. 3.27. Due to the complete linearization, the code converges between 6 and 8 Quasi-Newton iterations. Furthermore, the convergence history is practically independent of both grid size and order of the approximation. However, each Quasi-Newton iteration with a complete linearization requires significantly more computational resources relative to a Quasi-Newton iteration with a partial linearization utilized in most codes based on a finite volume or finite difference discretization. For comparison with the DG code, a set of commercially available codes are used to compute the entropy error with a series of finite volume grids with five times as many cells in each direction as the DG grids. The degrees of freedom for the finite volume codes is approximately the same as the DG code with $N = 4$. A 2nd-order accurate approximation is used to compute all the solutions with the commercial codes. A solution is considered converged when the L^2 -norm of the residual vector has dropped by 10 orders of magnitude. The entropy error vs. grid resolution as well as entropy error vs. execution time for the different codes is shown in Fig. 3.28. Notably, only Star-CCM+ and OVERFLOW obtain the expected 2nd-order accuracy. The remaining codes have an order of accuracy of 1.5. This is likely an issue with the slip wall boundary condition as the entropy error occurs along the lower wall downstream of the bump.

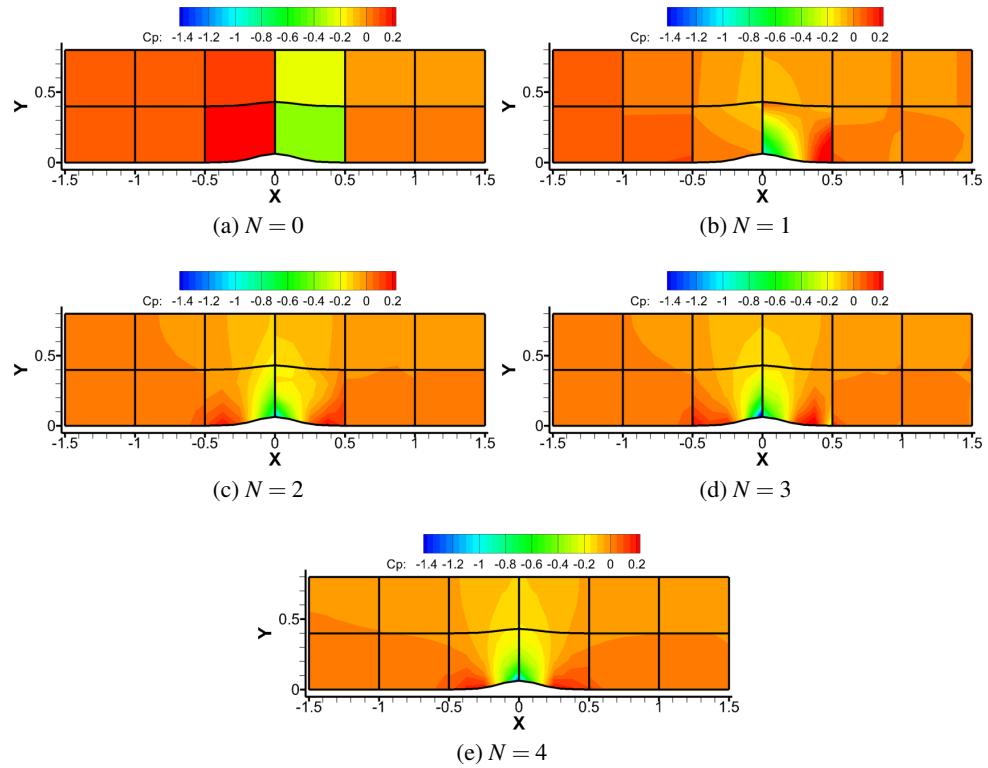


Figure 3.25: Gaussian Smooth Bump Solutions with 6×2 Grid

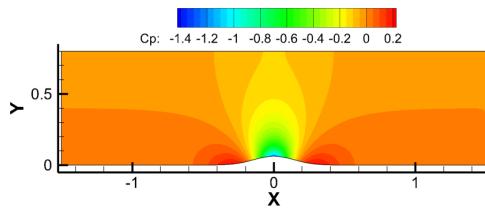


Figure 3.26: Gaussian Smooth Bump Solutions with 96×32 Mesh and $N = 4$

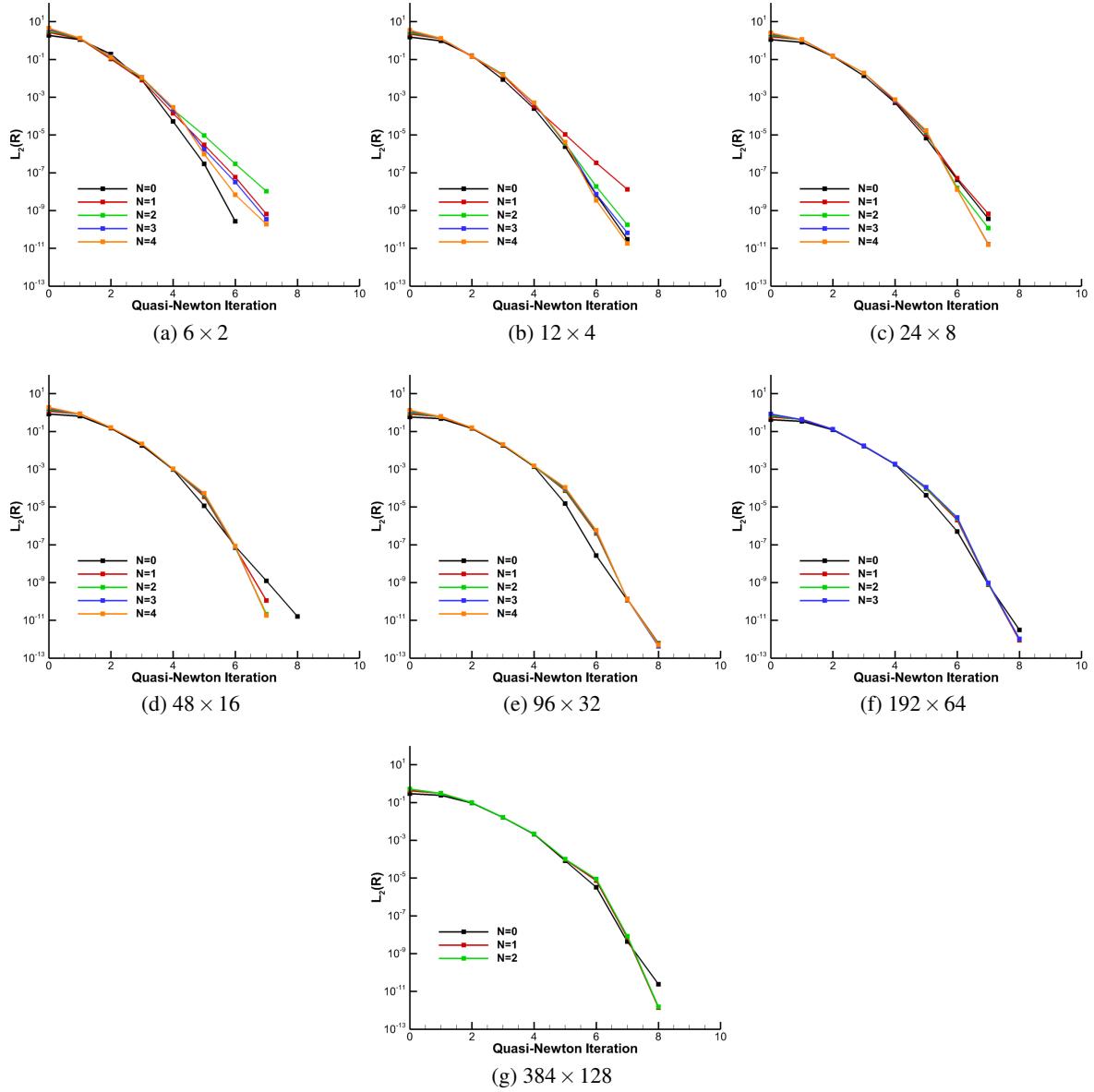


Figure 3.27: Gaussian Smooth Bump Quasi-Newton Convergence History for each Grid Size

The DG solver achieves a lower entropy error with less computational time with $N \geq 1$ relative to all the commercial solvers. There are two different ways of interpreting the results of the comparison of entropy error vs. execution time. First, if an error near machine zero is desired, the high-order of the DG code obtains an entropy error of the order 10^{-9} in approximately 2 min. However, such a small error is not always necessary for engineering type applications. Suppose that entropy error of an order 10^{-4} is desired. Using Fluent, this error level would require a grid with 192×64 and takes approximately 6.5 min to obtain

a converged solution. The DG code obtains the error level on the order of 10^{-4} in 1 sec using a grid with 24×8 cells and $N = 1$ (48×16 DOF). Alternatively, the same error level is obtained in 1 sec with the coarsest grid with 6×2 and $N = 3$ (24×8 DOF). Thus, regardless of the desired error level, the DG code obtains the desired error in significantly less time relative to the commercial codes.

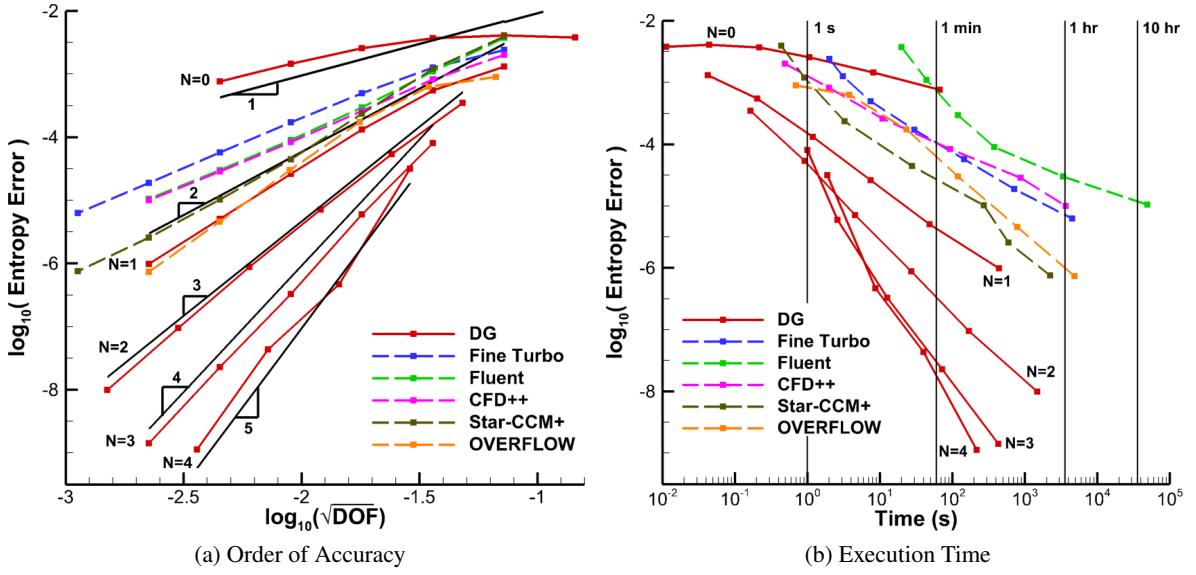


Figure 3.28: Gaussian Smooth Bump Spatial Order of Accuracy and Execution Time Comparison with Commercial Codes

3.4.2.2 Ringleb Flow

The transonic Ringleb flow problem[136] is one of the few known solutions to the steady compressible two-dimensional Euler equations. The solution is expressed in the hodograph $(V - \theta)$ plane where V is the velocity magnitude and θ is the flow angle with respect to the x -axis. The expression for the stream function, Ψ , is

$$\Psi = \frac{\sin(\theta)}{V}. \quad (3.129)$$

The expression for the streamlines in the $x - y$ plane are

$$\begin{aligned} x(V, k) &= \frac{1}{2} \frac{1}{\rho} \left(\frac{1}{V^2} - \frac{2}{k^2} \right) - \frac{J}{2}, \\ y(V, k) &= \pm \frac{1}{k\rho V} \sqrt{1 - \left(\frac{V}{k} \right)^2}, \end{aligned} \quad (3.130)$$

where

$$\begin{aligned} J(V) &= \frac{1}{a} + \frac{1}{3a^3} + \frac{1}{5a^5} - \frac{1}{2} \ln \left(\frac{1+a}{1-a} \right), \\ a(V) &= \sqrt{1 - \frac{\gamma-1}{2} V^2}, \\ \rho(V) &= a^{\frac{2}{\gamma-1}}, \\ k &= \frac{1}{\Psi}. \end{aligned} \quad (3.131)$$

The pressure is computed from the velocity magnitude as

$$p(V) = \frac{1}{\gamma} a^{\frac{2\gamma}{\gamma-1}}. \quad (3.132)$$

The analytical solution is used as the initial condition for the calculations. The velocity magnitude must be expressed in terms of the Cartesian coordinates using Eqs. 3.130 and 3.131 in order to compute the analytical solution on an arbitrary grid. The velocity magnitude is obtained from the recursive function

$$V(x, y) = \min \left(\sqrt{\frac{1}{2\rho(V)M(x, y, V)}}, \sqrt{\frac{2}{\gamma-1}} \right), \quad (3.133)$$

where

$$M(x, y, V) = \sqrt{\left(x + \frac{1}{2} J(V) \right)^2 + y^2}. \quad (3.134)$$

Both density and pressure is computed using the velocity magnitude computed using Eq. 3.133 as well as k , given by

$$k(x, V) = \sqrt{\frac{2}{2\rho(V) \left(x + \frac{1}{2} J(V) \right) + \frac{1}{V^2}}}, \quad (3.135)$$

which is required to compute the velocity components

$$\begin{aligned} u(x,y) &= \begin{cases} V(x,y) \cos(\theta(V,k)) & y > 0 \\ -V(x,y) \cos(\theta(V,k)) & y < 0 \end{cases}, \\ v(x,y) &= V(x,y) \sin(\theta(V,k)), \end{aligned} \quad (3.136)$$

where

$$\theta(V,k) = 2\pi - \sin^{-1}\left(\frac{V}{k}\right). \quad (3.137)$$

The series of grids, shown in Fig. 3.29 with increasing refinement used to compute the order of accuracy are generated using the streamline equations in Eq. 3.130. The value of k is linearly interpolated between $k_{min} = 0.7$ and $k_{max} = 1.2$. The velocity magnitude is interpolated between k and $V_{min} = 0.5$ with the function

$$V(k,i) = k + \left(\frac{i}{i_{max}-1}\right)^2 (V_{min} - k), \quad (3.138)$$

where i is an integer and i_{max} is the number of nodes. The solutions used to compute the entropy error are obtained with the analytical solution as the initial condition and the analytical solution is imposed on all boundaries of the computational domain. Entropy error (See, Eq. 3.128) computed from solutions with increasing order of accuracy with the series of grids is shown in Fig. 3.30. The coarsest grid consists of 6×2 cells and the finest grid consists of 96×32 cells. The 1st-order accurate solutions diverged using the three coarsest meshes. The slopes of the lines in Fig. 3.30 indicate that expected order of accuracy of $N+1$ is obtained for all orders of the approximation. The solutions on the coarsest 6×2 cell grid with increasing order of the approximation is shown in Fig. 3.31. The 2nd-order accurate solution, $N = 1$, exhibits some vertical asymmetry. This asymmetry vanishes as the order of the approximation is increased.

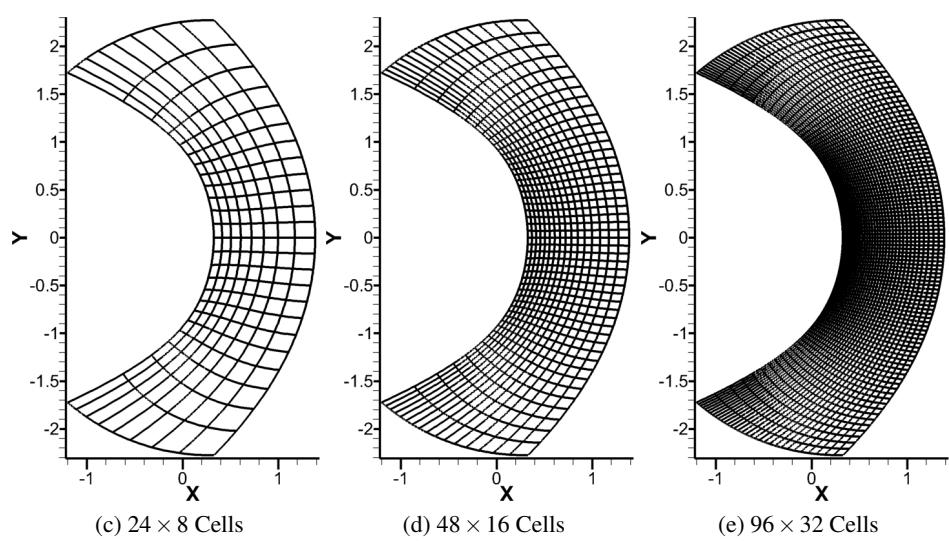
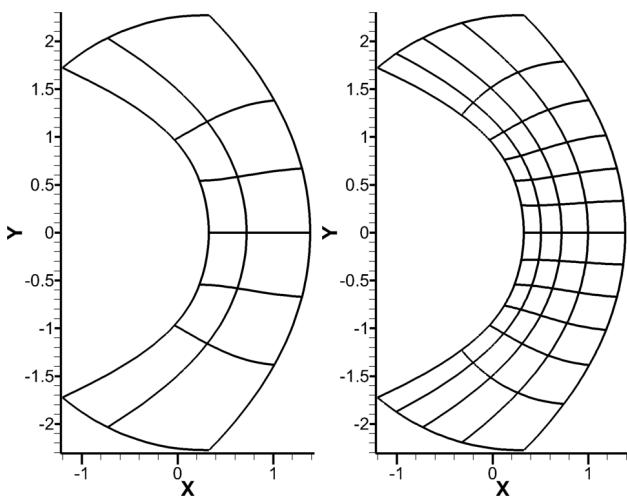


Figure 3.29: Grids for Ringleb Flow

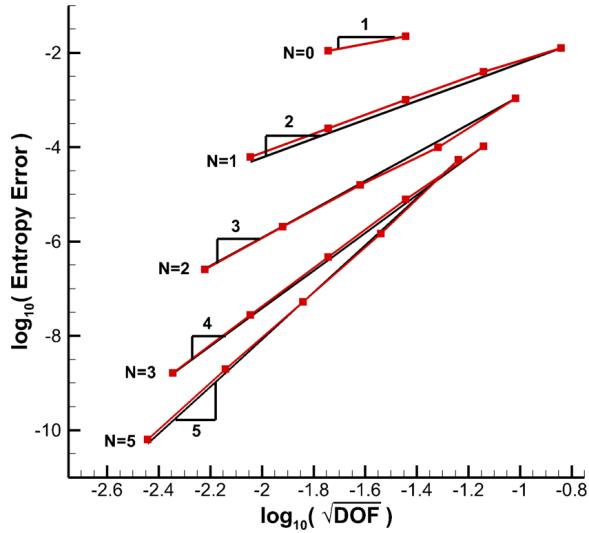


Figure 3.30: Ringleb Flow Spatial Order of Accuracy

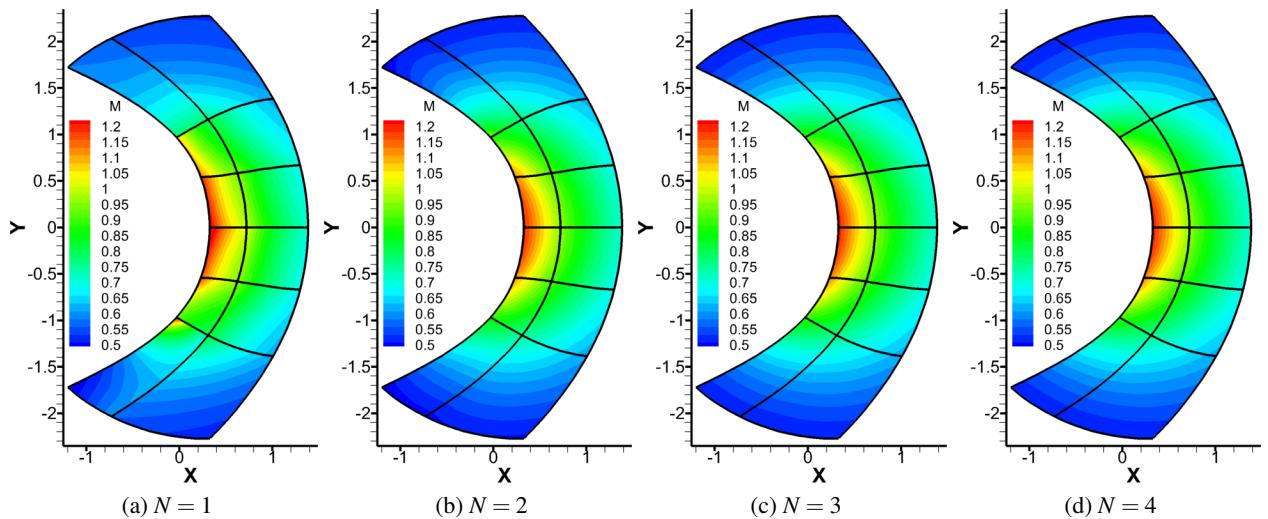


Figure 3.31: Mach Number Contours Computed for Ringleb Flow with 6×2 Grid

3.4.2.3 Oblique Shock

An oblique shock case is used to verify that the artificial viscosity outline in Section 2.2.5 is properly implemented. All the flow fields are computed on the uniformly spaced grid with 30×20 cells shown in Fig. 3.32. Supersonic flow is imposed on the left inflow boundary and the first three cells on the upper

boundary; shock conditions downstream of a 5° wedge are imposed on the remainder of the upper boundary. A slip wall condition is imposed on the lower wall, and all quantities are extrapolated on the right outflow boundary. These boundary conditions produce an oblique shock emanating from the upper boundary and reflects once off the lower wall. The parameters in the shock sensor limiter in Eq. 2.157 are $\kappa = 2.5$, $s_0 = 1$, and $\varepsilon_0 = 1$.

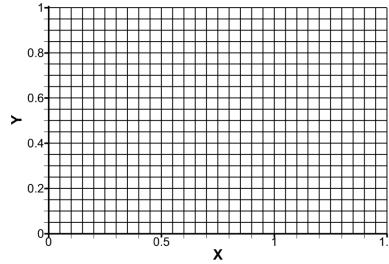


Figure 3.32: Oblique Shock Mesh

Surfaces of pressure coefficient computed both with and without artificial viscosity and with increasing order of approximation is shown in Fig. 3.33. The flow fields computed without artificial viscosity are colored by the pressure coefficient, and the flow fields computed with artificial viscosity are colored by the viscosity coefficient ε . In addition, the pressure coefficient obtained both with and without artificial viscosity on the lower wall is shown in Fig. 3.34. As expected, the piecewise constant approximation does not exhibit any overshoots. The flow fields computed with approximations $N \geq 1$ without artificial viscosity exhibit overshoots in the vicinity of the socks. The shock sharpens with increasing order of the approximation, but the magnitude of the overshoot does not change significantly. A small overshoot is still present in the $N = 1$ flow field with artificial viscosity. This could be remedied by modifying the parameters in the shock sensor limiter to increase the magnitude of the artificial viscosity. However, this is not necessary as the overshoot vanishes with increased order in the approximation. The increased order of the approximation also sharpens the shock and, per design, the magnitude of the artificial viscosity diminishes.

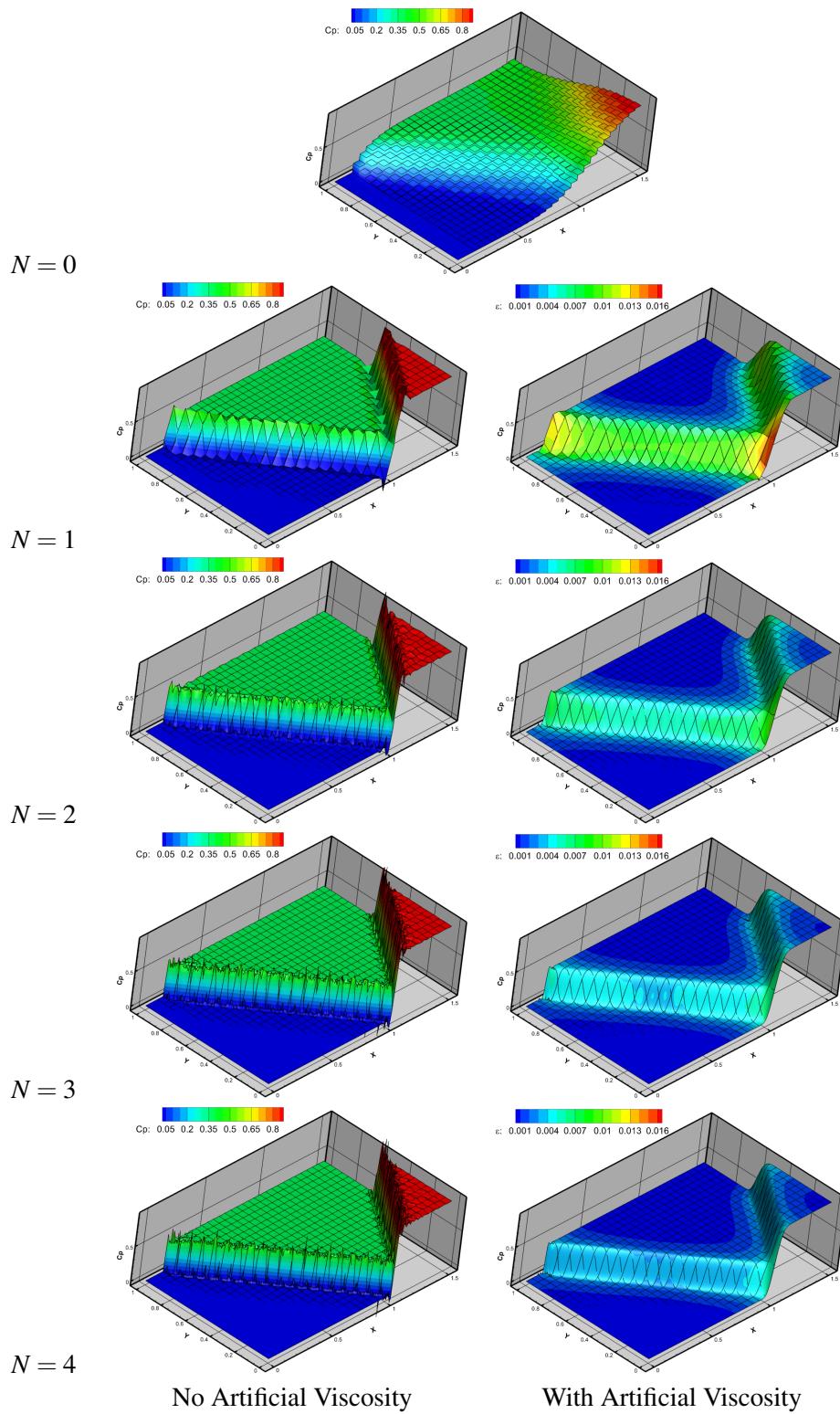


Figure 3.33: Oblique Shock Surfaces of Pressure Coefficient with and without Artificial Viscosity

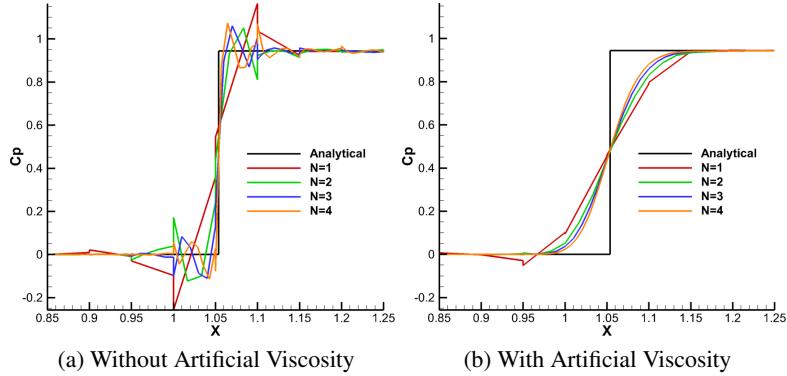


Figure 3.34: Oblique Shock Pressure Coefficient on the Lower Wall

The convergence history for each order of the approximation both with and without artificial viscosity is shown in Fig. 3.35. The calculations without artificial viscosity exhibit near quadratic convergence once the shock has developed in the solution. Including the artificial viscosity does increase the number of Quasi-Newton iterations for each order of the approximation, more so for the lower orders of the approximation. However, the convergence history still exhibits rapid convergence once the shocks have developed. This is a result of solving the artificial viscosity equation fully coupled in the linearization with the Euler equations.

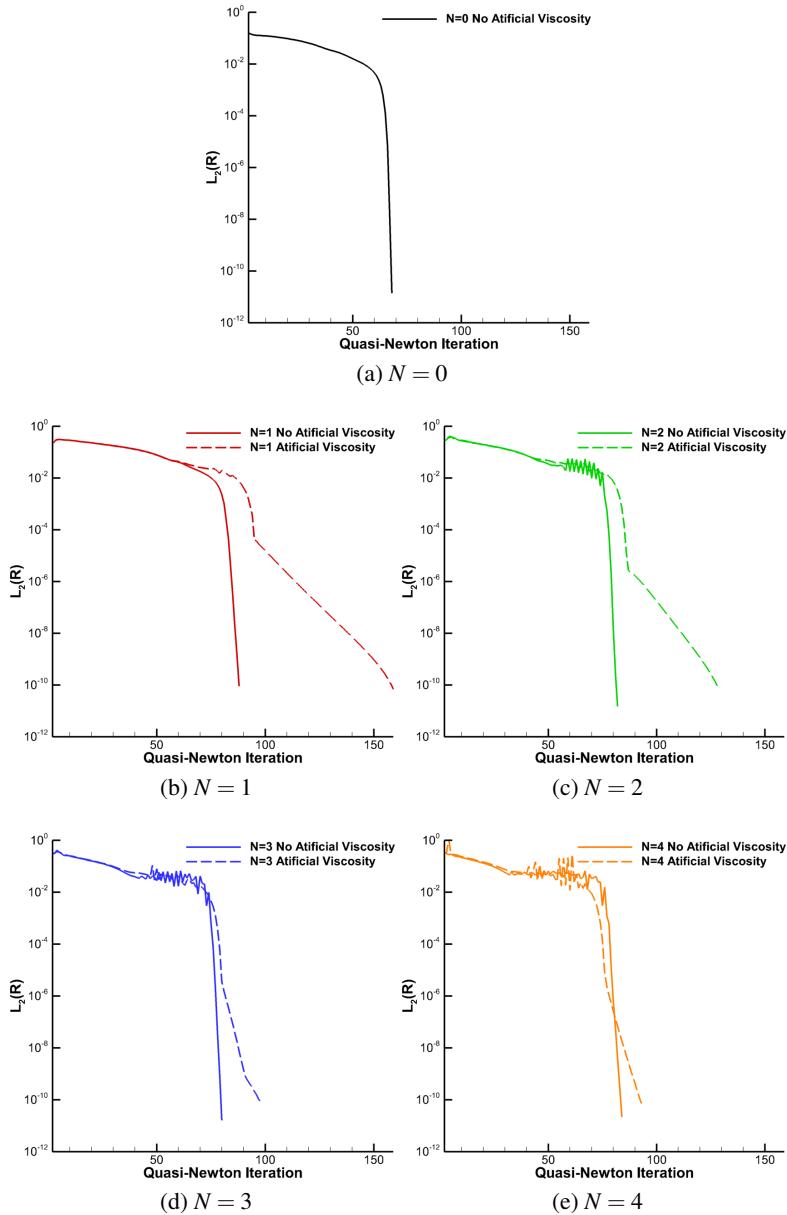


Figure 3.35: Oblique Shock Quasi-Newton Convergence History

3.4.3 Viscous Flow

While analytical solutions to the incompressible Navier-Stokes equations exist, no analytical solutions to the compressible Navier-Stokes equations exist to the knowledge of the author. Hence, a set of incompressible flow problems with increasing complexity are used to verify the implementation of the Navier-Stokes equations in the code. The first verification problem is a fully developed incompressible Channel flow, which

has an analytical solution. The second problem is a flat plate laminar boundary layer flow. The boundary computed with the present code is compared to the Blasius solution, which is a numerical similarity solution of the boundary layer equations. The last problem is a lid driven cavity flow. Flow fields computed with the present code are compared with flow fields computed by Ghia et al.[137] by solving the streamline-vorticity equations.

3.4.3.1 Channel Flow

Channel flow solutions were computed with a reference Mach number of $M_\infty = 0.05$ and Reynolds number of $Re = 10^3$ based on the channel height. The computational domain is shown in Fig. 3.36. The length of the channel is 80 channel heights, and the grid consisted of 50x10 linear cells. The flow is from left to right. Uniform velocity and density are imposed at the inlet and pressure is extrapolated. At the downstream exit, the velocity and density is extrapolated, and pressure is imposed. Adiabatic walls are imposed on the lower and upper surfaces of the channel.

Streamwise velocity at the outflow boundary for increasing order of the polynomial approximations is shown in Fig. 3.37. All solutions agree well with the analytical solution to fully developed channel flow given by

$$u_{\text{Channel}}(y) = -6(y^2 - y). \quad (3.139)$$

There is a slight overshoot in the peak streamwise velocity at the center of the channel. However, this is a result of comparing an incompressible analytical solution with the results of a compressible code. This overshoot increases with increasing reference Mach number.

The solution obtained with the linear polynomial approximation shown in Fig. 3.37a captures the analytical solution reasonably well near the walls, but it is not capable of capturing the curvature of the peak velocity. However, besides the overshoot in the peak velocity, the quadratic polynomial approximation in Fig. 3.37b captures the quadratic analytical solution well. Since the analytical solutions is quadratic, increasing the polynomial approximation to cubic and quadratic polynomial does not significantly improve the solution as shown in Figs. 3.37c and 3.37d.

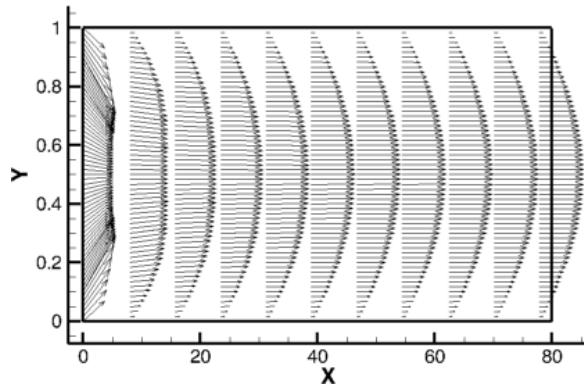


Figure 3.36: Computational domain for channel flow

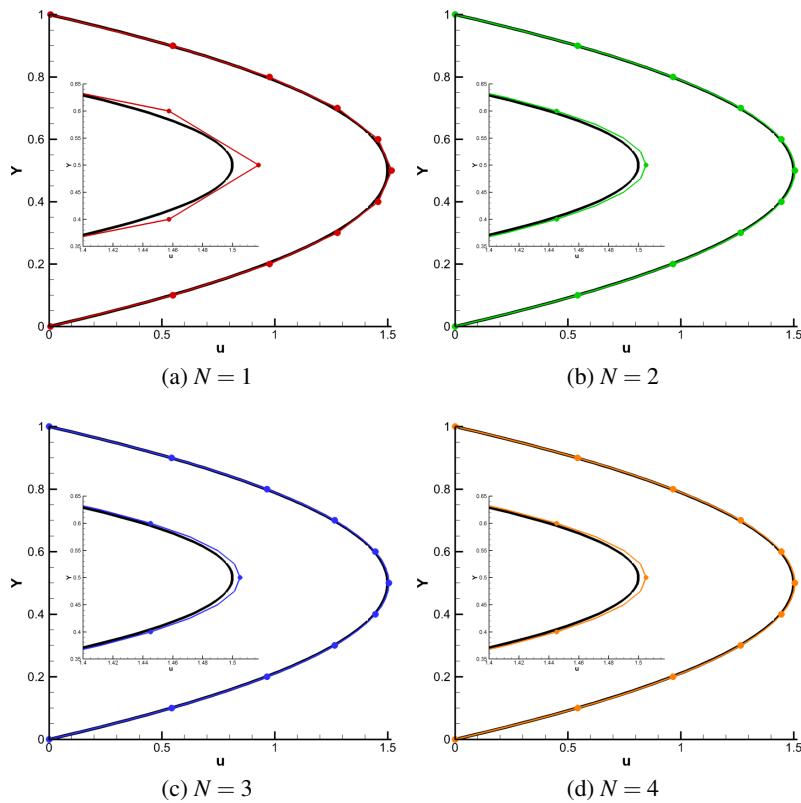


Figure 3.37: Fully Developed Channel Flow

3.4.3.2 Blasius Boundary Layer Flow

Laminar boundary layer flow computed for Reynolds numbers $Re = 10^6$ based on the length of the flat plate and $M_\infty = 0.05$ using the computational grids shown in Figs. 3.38. The grid consists of 38×19 cells, where the first cell on the wall has a height of 0.001. The flow is from left to right. Riemann invariants are imposed at the inlet and upper boundaries of the computational domain. On the outflow boundary pressure is imposed and the remaining quantities are extrapolated. On the lower surface of the domain, a slip wall boundary conditions is imposed for $x < 0$ and an adiabatic wall boundary conditions is applied for $x \geq 0$.

Solutions for $Re = 10^6$ are compared to the Blasius [138] boundary layer solution in Fig. 3.39. The boundary layer solutions are resolved with only 4 cells. The grid resolution is too coarse for the linear polynomial approximation in Fig. 3.39a to capture the boundary layer correctly. The u velocity component computed with the quadratic polynomial approximation agrees well with the Blasius solution, though discontinuities in the solution are still observable in the v velocity component as shown in Fig. 3.39b. Solutions computed with cubic and quartic polynomial approximations agree well with the Blasius solution as shown in Fig. 3.39c and 3.39d, though there is a slight over prediction in the magnitude of the v velocity component at the edge of the boundary layer.

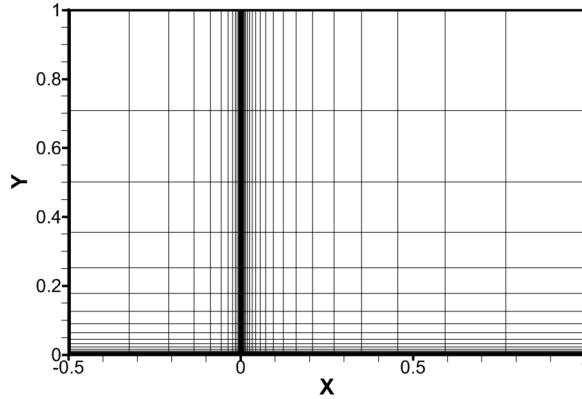


Figure 3.38: Computational Domain for Laminar Blasius Boundary Layer Flow

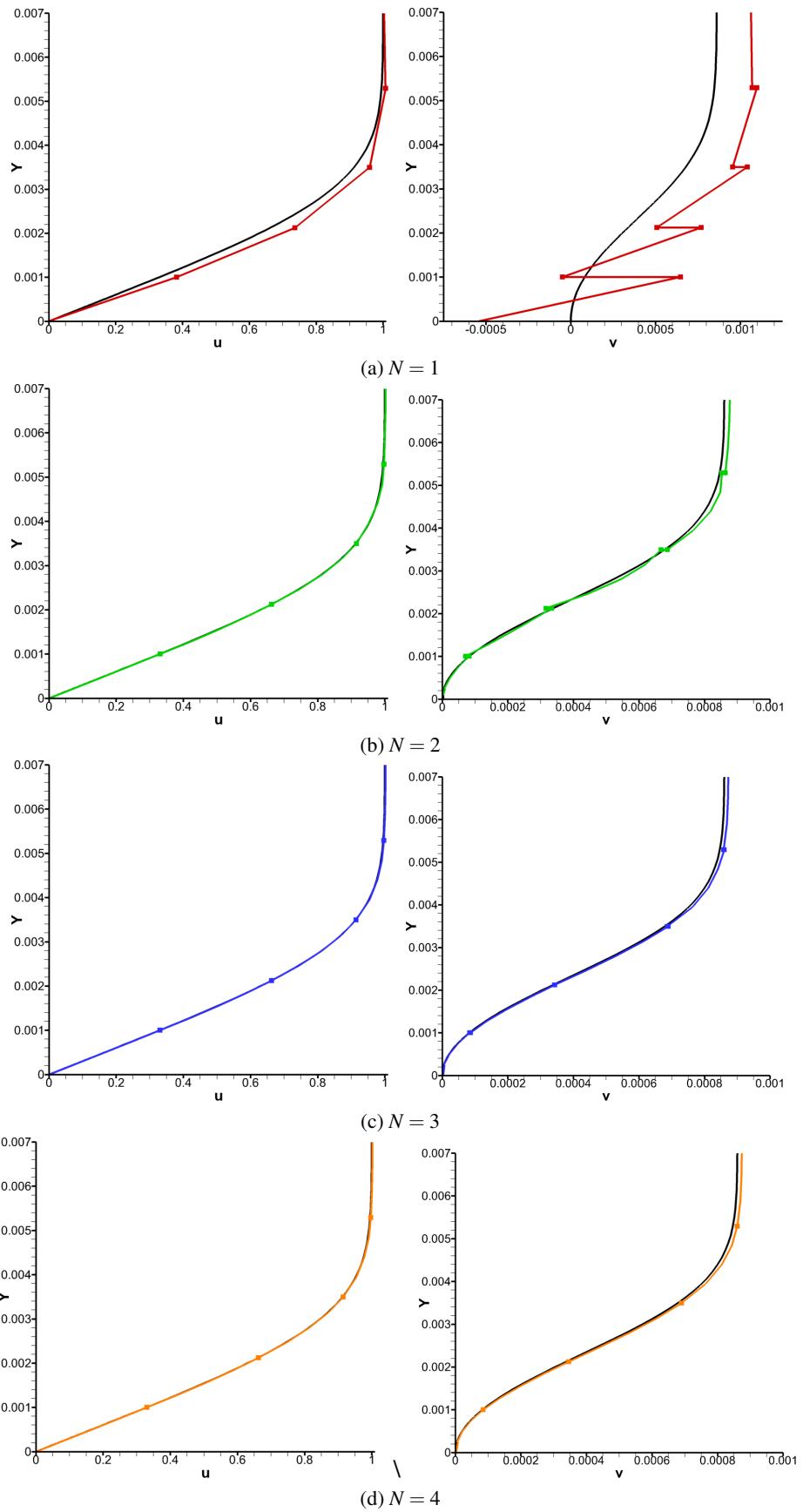


Figure 3.39: Blasius boundary layer flow $Re = 10^6$
155

3.4.3.3 Lid Driven Cavity Flow

A sketch of the computational domain for lid driven cavity flow [137] is shown in Fig. 3.40. Adiabatic wall boundary conditions are applied on the left, right, and lower surfaces of the square computational domain. Uniform tangential velocity and uniform density is imposed on the upper boundary, and pressure is extrapolated. Uniform grids consisting of 20x20, 30x30, and 40x40 cells were used to compute flows at $M_\infty = 0.05$ for $Re = 100$, $Re = 1000$, $Re = 3200$ respectively. The boundary conditions produce a dominant clockwise circulation in computational domain. Additional secondary vortices arise in the corners of the computational domain when the Reynolds number is increased.

Streamlines from calculations by Ghia et al.[137] are compared with solutions obtained with the DG code in Figs. 3.41, 3.42, and 3.43. The solutions from Ghia et al. used a finite-difference scheme and computational grid consisting of 129x129 nodes with a uniform distribution. For the three Reynolds numbers, the grids are too coarse for the linear polynomial approximation to completely capture the corner vortices. However, the streamlines for the corner vortices with $N \geq 2$ agree well with streamlines computed by Ghia et al. Velocity components extracted from the vertical and horizontal centerlines are compared with results from Ghia et al. in Fig. 3.44. The linear polynomial approximation underestimates the peak velocities as shown in Fig. 3.44 while the high-order approximations, $N \geq 2$, capture the peak velocities and agree well with the velocities by Ghia et al.

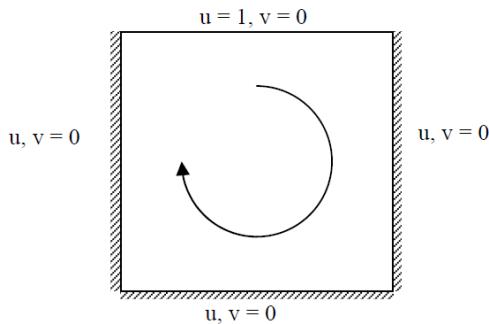
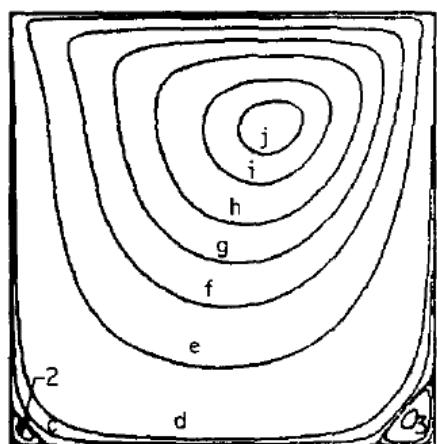
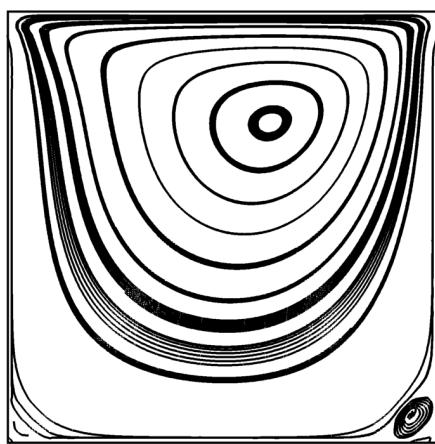


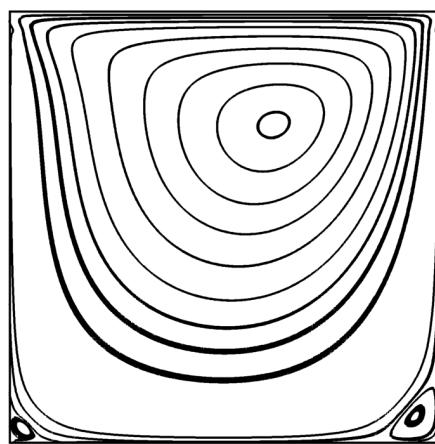
Figure 3.40: Lid Driven Cavity



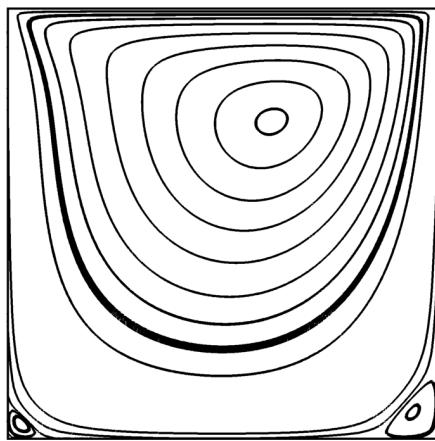
(a) Ghia et al.[137], 129x129 Nodes



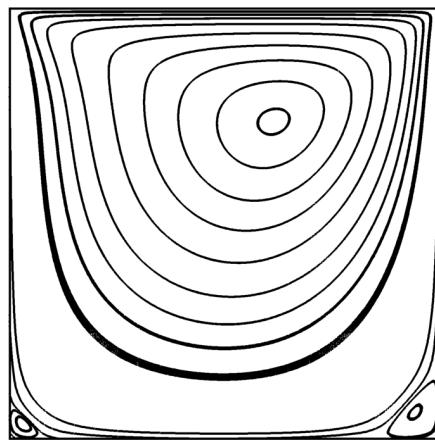
(b) $N = 1$



(c) $N = 2$

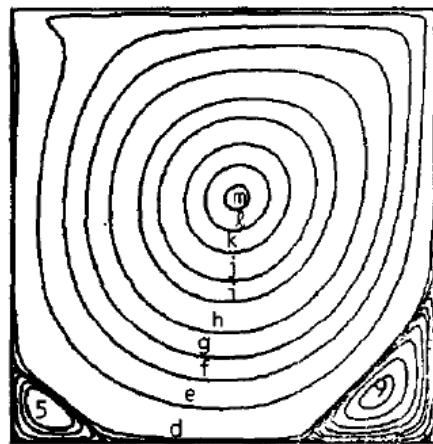


(d) $N = 3$

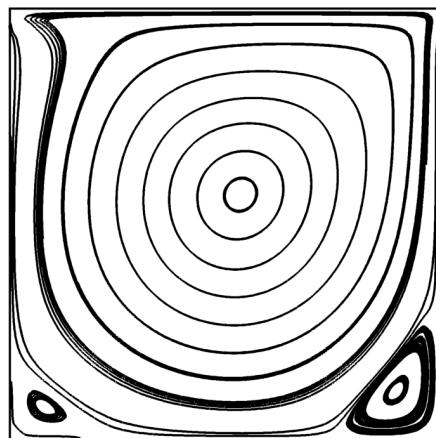


(e) $N = 4$

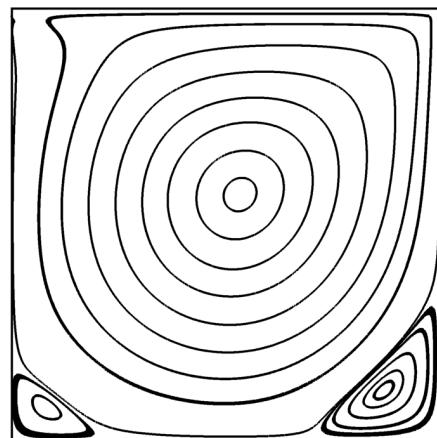
Figure 3.41: Streamline for the Lid Driven Cavity,(20×20) Cells, $Re = 100$



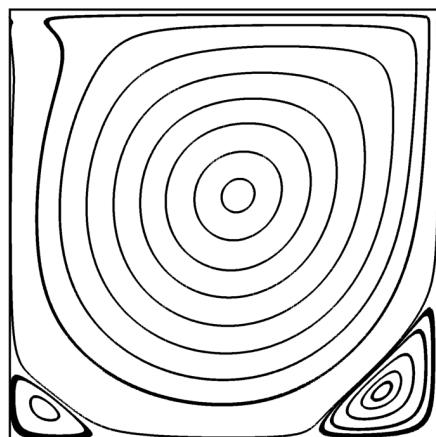
(a) Ghia et al.[137], 129x129 Nodes



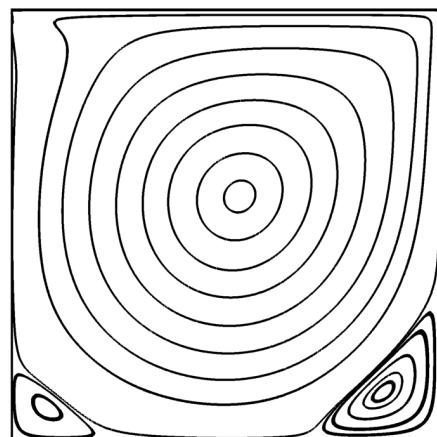
(b) $N = 1$



(c) $N = 2$

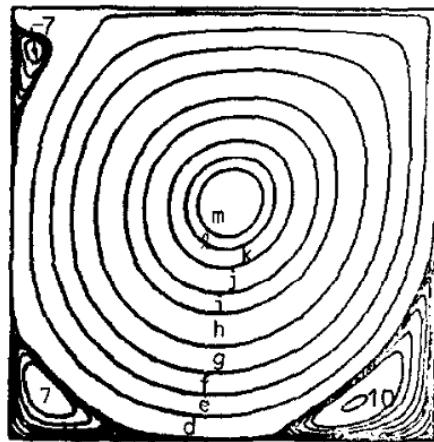


(d) $N = 3$

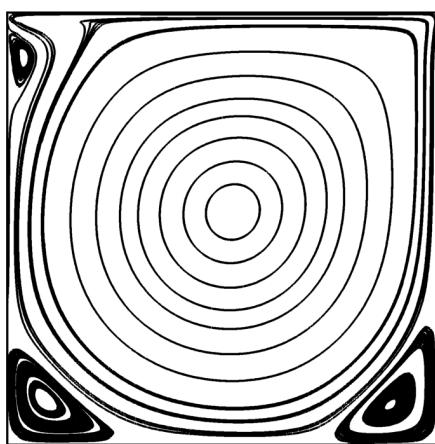


(e) $N = 4$

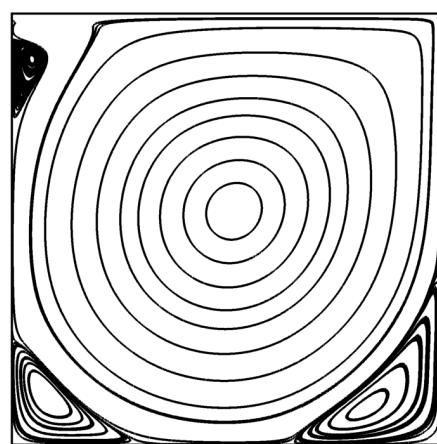
Figure 3.42: Streamline for the Lid Driven Cavity, (30×30) Cells, $Re = 1000$



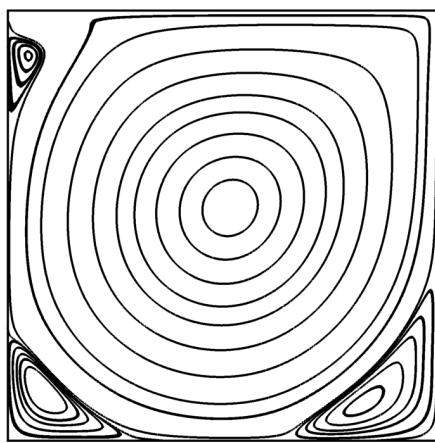
(a) Ghia et al.[137], 129x129 Nodes



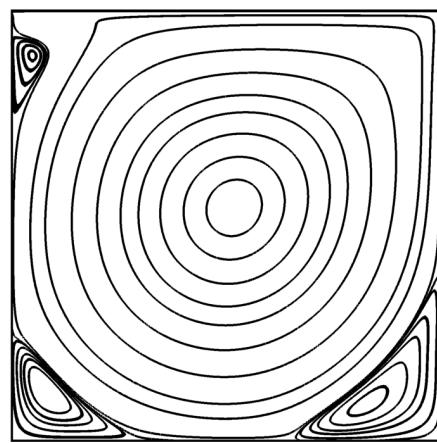
(b) $N = 1$



(c) $N = 2$



(d) $N = 3$



(e) $N = 4$

Figure 3.43: Streamline for the Lid Driven Cavity, (40×40) Cells, $Re = 3200$

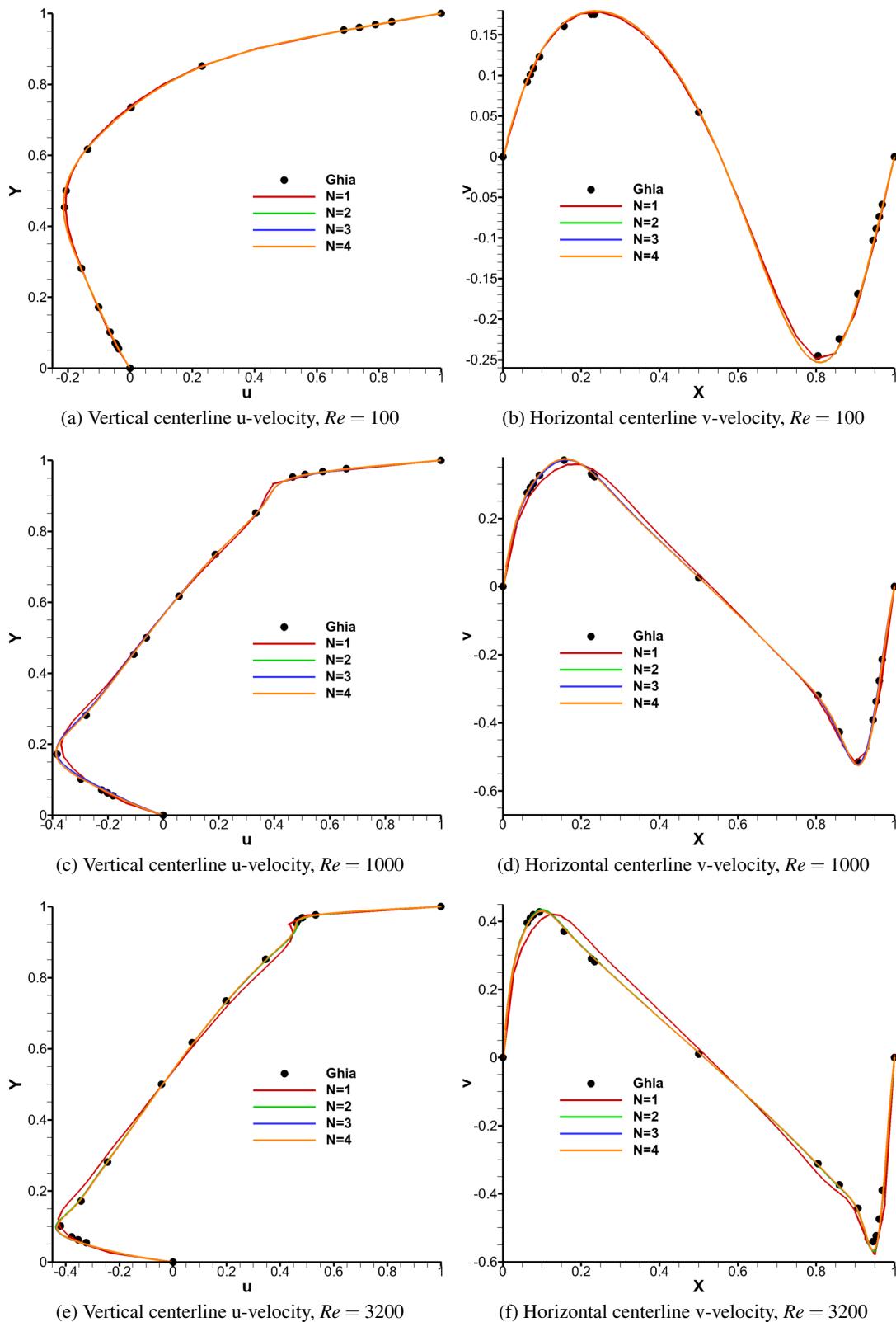


Figure 3.44: Lid driven cavity velocity components

3.5 Summary

Details of the implementation of the code based on the Discontinuous Galerkin discretization are presented, as well as coding strategies used promote computational efficiency. The code is written using the C++ programming language, which is often criticized for having a lower computational efficiency relative to the FORTRAN programming language. However, this is not an inherent feature of the C++ programming language, but rather an issue of efficient software engineering. The Template Expression strategy is used here to produce C++ with the same computational efficiency of a FORTRAN code, but with the extra benefits that come with an object oriented programming language such as C++. Using Template Expressions, block sparse matrix and dense matrix libraries were developed in C++ that have the equivalent computational efficiency of code written with FORTRAN. Template Expressions were also used to develop a polynomial library suitable for performing operations on polynomials, such as addition/subtraction, multiplication, division, differentiation, and integration. The polynomial library performs these mathematical operations using a Quadrature-free approach where integrals of the basis/test functions are evaluated analytically with a symbolic manipulator rather than using numerical integration such as Gauss Quadrature. The Quadrature-free integration combined with an orthogonal basis/test function is more computationally efficient than Gauss Quadrature for linear partial differential equations. This polynomial library is the backbone for the general framework of the code that computes the residual vector $\mathcal{R}(Q)$ and Jacobian matrix $\frac{\partial \mathcal{R}}{\partial Q}$. The general framework is designed to operate independent of the partial differential equation that is solved. Hence, a significant effort is not required to solve other partial differential equations written in conservation form.

The implementation of the code is verified by computing the expected order of accuracy from numerical solutions to a set of scalar partial differential equations and the Euler equations. Application to the compressible Navier-Stokes equations is qualitatively verified with comparisons to analytical and numerical solutions to the incompressible Navier-Stokes equations. The computational efficiency of the code is compared with a number of commercially available CFD codes by computing an inviscid channel flow with a Gaussian smooth bump on the lower wall. This test case demonstrated the benefit of the higher-order of accuracy of the discontinuous Galerkin scheme, as well as the benefit of using a complete linearization for the Quasi-Newton iterations. A formal study of the efficiency of the Quadrature-free integration for non-linear

partial differential equations is left as future work. Some preliminary comparisons of execution times with the XFLOW code indicate that the Quadrature-free integration technique is comparable in cost to traditional Gauss Quadrature integration for non-linear partial differential equations. However, any possible benefits in robustness by using the Quadrature-free integration needs to be studied.

Chapter 4

A Discontinuous Galerkin Chimera Scheme

The chapter provide details of the Discontinuous Galerkin Chimera scheme developed as part this dissertation. The scheme used to communicate the dependent variable vector from overset grids to artificial boundaries is presented, as well as the method for the linearization of the artificial boundary communication scheme. This is followed by a description of the algorithm developed to implement the artificial boundaries in the solver. A hole cutting scheme developed for the DG-Chimera based on a direct cut method that accounts for curved cells is also presented. The DG-Chimera scheme is verified by solving scalar partial differential equations with increasing complexity, followed by verification calculations of inviscid flow problems. The inviscid flow problems are also used to demonstrate that a mass flux error associated with the DG-Chimera scheme is consistent, i.e., it vanishes with grid refinement and/or increase in the order of the approximation. The use of curved cells with the discontinuous Galerkin discretization to represent curved geometry enables the DG-Chimera scheme to properly compute viscous flows when multiple overlapping grids are used to define the surface of the geometry. Finally, improved parallel performance obtained by including the linearization of the artificial boundary relative to a conventional explicit implementation is presented.

4.1 Artificial Boundaries

The inter-grid communication method is designed to maintain the interior discretization scheme on artificial boundaries. For an interior cell, the boundary integral of the inviscid terms from Eq. 2.12 is evaluated on all boundaries where the fluxes $\vec{F}(Q^-)$, $\vec{F}(Q^+)$, and the dissipation flux, $\phi(Q^+, Q^-)$, are evaluated using the

trace of the dependent variables taken from the cell interior and from neighboring cells as shown in Fig. 4.1. For an artificial boundary, the exterior conservative variables, Q^+ , must be provided by one, or multiple, cells from overlapping meshes. The two overlapping grids shown in Fig. 4.2 are used as an example to show how the exterior conservative variables are obtained for the boundary integral on the left boundary of the red cell.

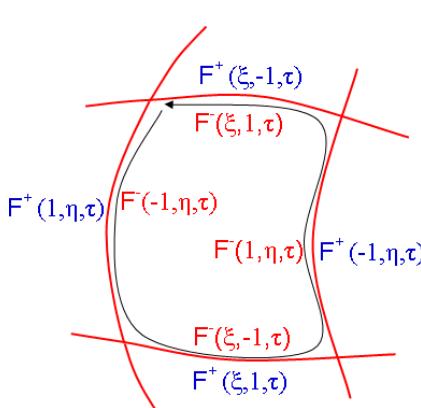


Figure 4.1: Interior Boundary Integration

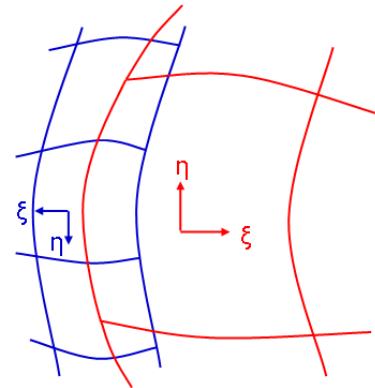


Figure 4.2: Overlapping grids

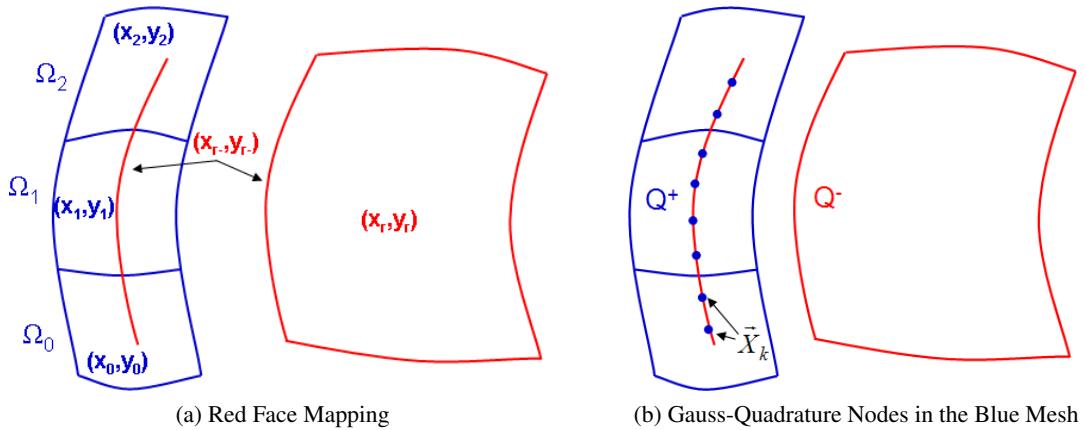


Figure 4.3: Obtaining Conservative Variables from the Blue Mesh

The left face of the red cell, defined by the coordinate mappings $x_r(-1, \eta)$ and $y_r(-1, \eta)$, $\eta \in [-1, 1]$, (See, Fig. 4.3a.), is seeded with Gauss-Quadrature (GQ) nodes to integrate the red mesh solution polynomial and obtain Q^+ from the blue cells. The polynomial mappings $x_r(-1, \eta)$ and $y_r(-1, \eta)$ are then used to obtain the Cartesian coordinate, $\vec{X}_k = (x_r(-1, s_k), y_r(-1, s_k))$, that corresponds to the Gauss-Quadrature node,

$s_k \in [-1, 1]$. The Cartesian coordinate, \vec{X}_k , is then used to obtain the corresponding cell local coordinates, $(\xi(\vec{X}_k), \eta(\vec{X}_k))$, in the cells of the blue mesh as shown in Fig. 4.3b. A Kd-tree [139] search algorithm is used to determine which GQ nodes are located within the bounding box of each blue cell Ω_i . GQ nodes that reside inside the bounding box of the cell Ω_i may or may not reside inside the cell Ω_i . To determine that the GQ node \vec{X}_k is located within a cell Ω_i , the local curvilinear coordinate location, $(\xi_i(\vec{X}_k), \eta_i(\vec{X}_k))$, corresponding to \vec{X}_k is found using Newton's method

$$\begin{pmatrix} \frac{\partial}{\partial \xi} x_i(\xi^n, \eta^n) & \frac{\partial}{\partial \eta} x_i(\xi^n, \eta^n) \\ \frac{\partial}{\partial \xi} y_i(\xi^n, \eta^n) & \frac{\partial}{\partial \eta} y_i(\xi^n, \eta^n) \end{pmatrix} \begin{pmatrix} \Delta \xi \\ \Delta \eta \end{pmatrix} = \begin{pmatrix} -(x_i(\xi^n, \eta^n) - x_r(-1, s_k)) \\ -(y_i(\xi^n, \eta^n) - y_r(-1, s_k)) \end{pmatrix}, \quad (4.1)$$

where

$$\begin{aligned} \xi^0 &= 0 \\ \eta^0 &= 0 \\ \xi^{n+1} &= \xi^n + \Delta \xi \\ \eta^{n+1} &= \eta^n + \Delta \eta. \end{aligned}$$

It is possible for the Newton method to diverge even though the node is located within the cell if either ξ^{n+1} or η^{n+1} exceed the valid range of $[-1, 1]$. Thus, both ξ^{n+1} or η^{n+1} are limited to the range $[-1, 1]$ after each iteration. The Newton solver is stopped when the L^2 -norm of the right hand side of Eq. 4.1 drops below a tolerance of 1e-10, or the Newton method reaches a maximum number of 20 iterations. The cell Ω_i is a donor cell for the coordinate \vec{X}_k if the L^2 -norm drops below the tolerance of 1e-10. If the L^2 -norm is above the required tolerance after 20 iterations, the coordinate \vec{X}_k is deemed to reside outside of the cell Ω_i , and Ω_i is discarded as a donor for the coordinate \vec{X}_k . The average nodal value is used if multiple donor cells exist for a given Gauss-Quadrature node. For stationary grids, this process of locating cell local curvilinear coordinates is performed once during an initialization stage.

The cell local curvilinear coordinates corresponding to $(\xi_i(\vec{X}_k), \eta_i(\vec{X}_k))$ are used to obtain nodal Q^+ values that correspond to \vec{X}_k . The coefficients for the modal representation of Q^+ are then obtained using the following inner product

$$Q^+ = \sum_{j=0}^N q_j^+ \psi_j \quad \text{where}$$

$$q_j^+ = \frac{\sum_{k=0}^{N_{GQ}} w_k \psi_j(s_k) Q_i\left(\xi_i(\vec{X}_k), \eta_i(\vec{X}_k)\right)}{\int_{-1}^1 \psi_j^2(s) ds} \quad \forall j \in [0, N]. \quad (4.2)$$

where w_k are the Gauss-Quadrature integration weights. The modal representation of Q^+ is then used to evaluate the inviscid flux term of Eq. 2.12. A numerical approximation is introduced in Eq. 4.2 by using a single set of Gauss-Quadrature nodes to integrate the nodal Q^+ values across cell boundaries without regard to possible discontinuities in the approximation across cell boundaries. These errors can be reduced, but not eliminated, by increasing the number of Gauss-Quadrature nodes. Numerical experiments, presented in the results section, indicate that increasing Gauss-Quadrature node count beyond $N_{GQ} = \lceil 3N/2 \rceil + 1$ does not significantly reduce the error.

The modal coefficients for the gradient ∇Q^+ is also obtained using Eq. 4.4a for the viscous terms in the artificial dissipation in Eq. 2.12. However, no cell exterior to the artificial boundary is available to perform the volume integral to obtain \vec{r}^+ . Thus, as an approximation, \vec{r}^- is used in place of \vec{r}^+ in Eq. 2.12. This is equivalent to assuming that an exterior cell exists of equal volume to the interior cell. This is also similar to how Dirichlet boundary conditions are imposed in the BR2 discretization.[13]

It is important to note that unlike standard overset methods the communication method for the DG-Chimera method has no requirement on the extent of overlap; only that the grids overlap or abut. In the special case where the face of the red cell is coincident with the face of one of the blue cells (4.3), the inter-grid communication for the advection flux reduces naturally to that of the interior scheme. Thus, mesh boundaries on a set of zonal meshes[140] will naturally use the interior scheme, as shown in Fig. 4.4a and 4.4b, for the advective fluxes. There is still an approximation in the diffusion terms as a result of using the BR2 discretization scheme. However, the results will show that this approximation is acceptable.

The method is also independent of the order of the geometric cell mapping of the two meshes. The communication scheme can connect two meshes consisting of linear cells, or quadratic and higher cells. Information can also be transferred between two meshes that do not use the same order of the geometric mapping as shown in Fig. 4.4c. However, gaps cannot exist between two meshes, i.e., they must overlap or abut. Two examples of using a zonal type interface that leads to gaps between meshes are shown in Fig.

4.5. As shown in Fig. 4.5a, two linear meshes that do not have coincident nodes along a curved artificial boundary may produce gaps between the meshes. The gap can be removed by adjusting the meshes so that the boundaries overlap. In Fig. 4.5b, the red mesh consists of cells with a quadratic geometric mapping and the blue mesh cells use a linear geometric mapping. In this case, even though the nodes on the common boundary are coincident, gaps are produced between the meshes as the faces of linearly mapped cells on the red mesh are secants to the curved boundary. Again, the gaps can be removed by overlapping the boundaries between the two meshes.

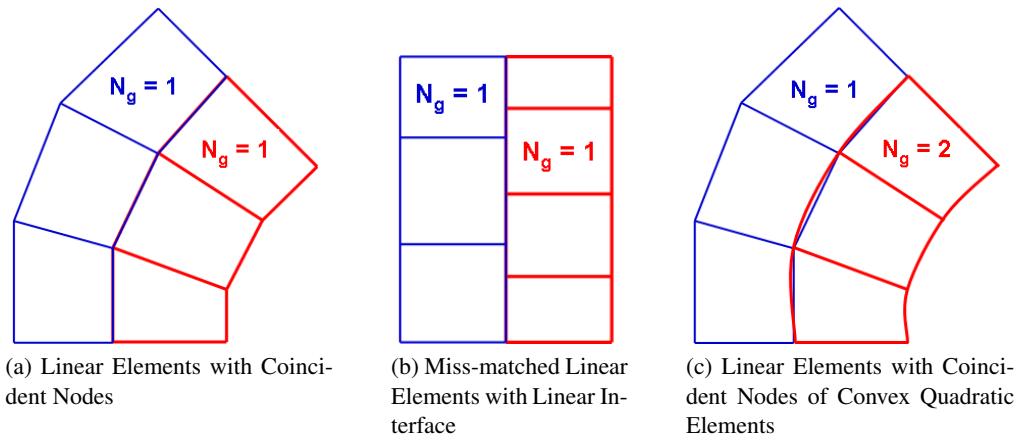


Figure 4.4: Sufficient Overlap for Zonal Type Interfaces

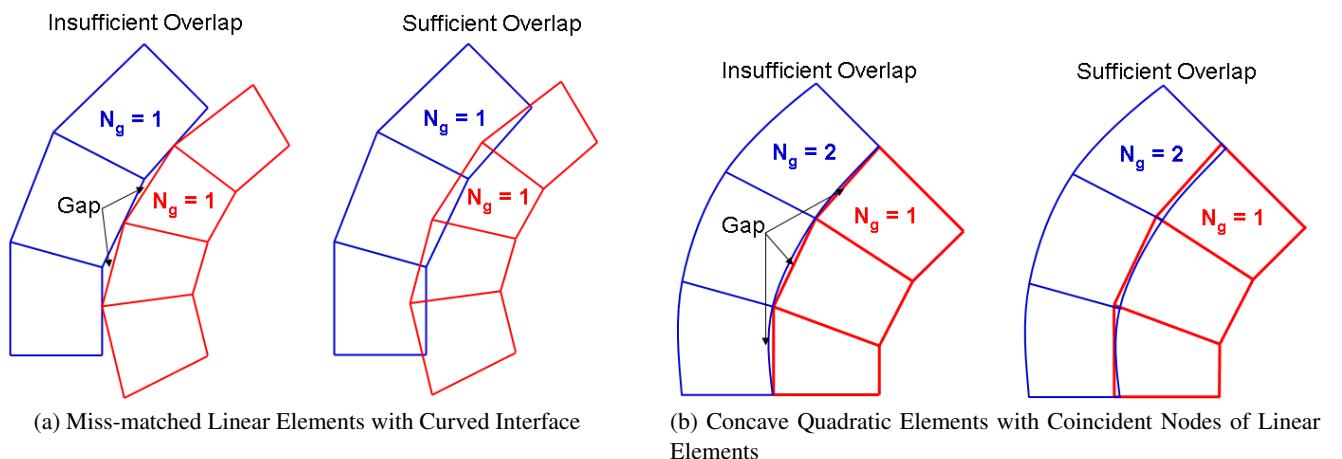


Figure 4.5: Insufficient Overlap for Zonal Type Interfaces and Corrections for Sufficient Overlap

The extension of the DG-Chimera method to three-dimensions is evident. A three-dimensional grid system corresponding to the grid system shown in Fig. 4.2. Similar to the two-dimensional case, Q^+ for the left cell of the red mesh must be obtained from the three cells in the blue grid.

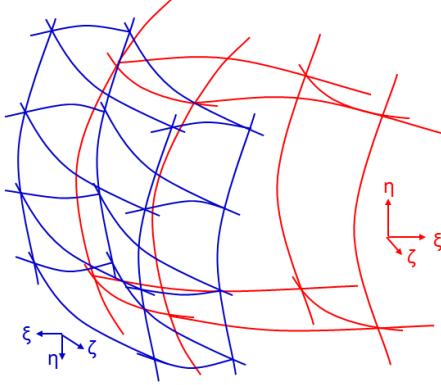


Figure 4.6: Overlapping grids

To obtain Q^+ from the blue cells, the left face of the red cell, defined by the coordinate mappings $x_g(-1, \eta, \zeta)$, $y_g(-1, \eta, \zeta)$, and $z_g(-1, \eta, \zeta)$, $\eta \in [-1, 1]$, $\zeta \in [-1, 1]$, (See, Fig. 4.7a.), is seeded with the appropriate number of Gauss-Quadrature nodes to integrate the red grid solution polynomial. The polynomial mappings $x_g(-1, \eta, \zeta)$, $y_g(-1, \eta, \zeta)$, and $z_g(-1, \eta, \zeta)$ are then used to obtain the Cartesian coordinate, $\vec{X}_{pq} = (x_g(-1, s_p, s_q), y_g(-1, s_p, s_q), z_g(-1, s_p, s_q))$, that corresponds to the Gauss-Quadrature nodes, s_p and s_q . The cell local coordinate, $(\xi(\vec{X}_{pq}), \eta(\vec{X}_{pq}), \zeta(\vec{X}_{pq}))$, in the cells of the blue grid that correspond to the Cartesian coordinate, \vec{X}_{pq} , as shown in Fig. 4.7b are located using a Kd-tree [139] search algorithm along with and Newton's method.

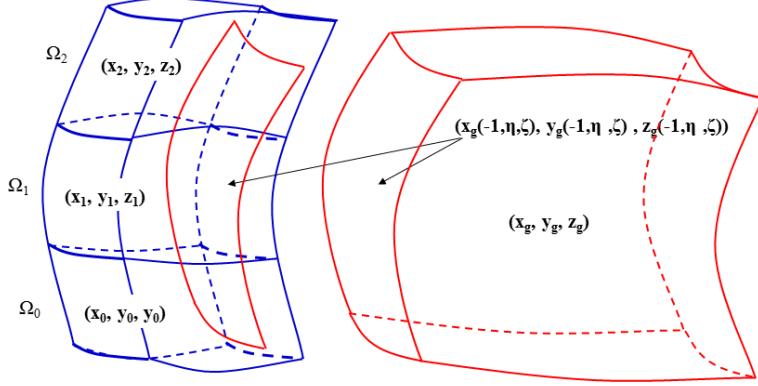
Similar to the two-dimensional algorithm, the Kd-tree [139] search algorithm is used to determine which Gauss-Quadrature (GQ) nodes are located within the bounding box of a given blue cell. The local curvilinear coordinate location, $(\xi_i(\vec{X}_{pq}), \eta_i(\vec{X}_{pq}), \zeta_i(\vec{X}_{pq}))$, of the blue cell, Ω_i , that corresponds to \vec{X}_{pq} is found using the blue cell local polynomial mappings $\begin{pmatrix} x_i & y_i & z_i \end{pmatrix}$ with a Newton method

$$\begin{pmatrix} \frac{\partial}{\partial \xi} x_i(\xi^n, \eta^n, \zeta^n) & \frac{\partial}{\partial \eta} x_i(\xi^n, \eta^n, \zeta^n) & \frac{\partial}{\partial \zeta} x_i(\xi^n, \eta^n, \zeta^n) \\ \frac{\partial}{\partial \xi} y_i(\xi^n, \eta^n, \zeta^n) & \frac{\partial}{\partial \eta} y_i(\xi^n, \eta^n, \zeta^n) & \frac{\partial}{\partial \zeta} y_i(\xi^n, \eta^n, \zeta^n) \\ \frac{\partial}{\partial \xi} z_i(\xi^n, \eta^n, \zeta^n) & \frac{\partial}{\partial \eta} z_i(\xi^n, \eta^n, \zeta^n) & \frac{\partial}{\partial \zeta} z_i(\xi^n, \eta^n, \zeta^n) \end{pmatrix} \begin{pmatrix} \Delta \xi \\ \Delta \eta \\ \Delta \zeta \end{pmatrix} = \begin{pmatrix} -(x_i(\xi^n, \eta^n, \zeta^n) - x_g(-1, s_p, s_q)) \\ -(y_i(\xi^n, \eta^n, \zeta^n) - y_g(-1, s_p, s_q)) \\ -(z_i(\xi^n, \eta^n, \zeta^n) - z_g(-1, s_p, s_q)) \end{pmatrix}, \quad (4.3)$$

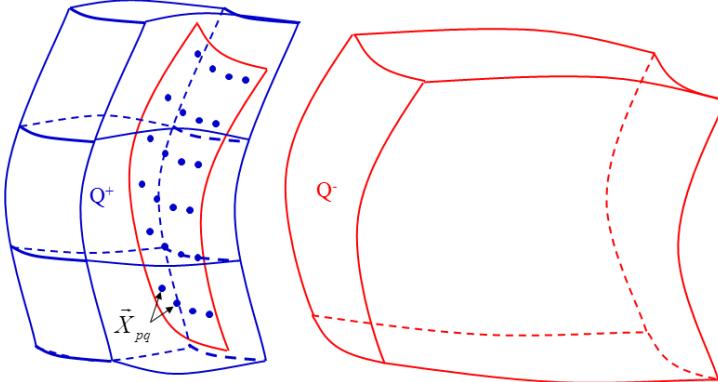
where

$$\begin{aligned} \xi^0 &= 0 \\ \eta^0 &= 0 \\ \zeta^0 &= 0 \\ \xi^{n+1} &= \xi^n + \Delta \xi \\ \eta^{n+1} &= \eta^n + \Delta \eta \\ \zeta^{n+1} &= \zeta^n + \Delta \zeta, \end{aligned} \quad (4.4)$$

to determine if the GQ node \vec{X}_{pq} is located within a blue cell. Again, ξ^{n+1} , η^{n+1} , and ζ^{n+1} are limited to the range $[-1, 1]$ after each iteration to prevent the Newton method from diverging. The Newton solver is stopped when the L^2 -norm of the right hand side of Eq. 4.3 drops below a given tolerance, or the Newton method reaches a maximum number of 20 iterations. If the L^2 -norm is above the required tolerance after 20 iterations, the coordinate \vec{X}_{pq} is deemed to reside outside of the cell Ω_i , and Ω_i is discarded as a donor.



(a) Gray Face Mapping



(b) Gauss-Quadrature Nodes in the Black Mesh

Figure 4.7: Obtaining Conservative Variables from the Black Mesh

The three-dimensional equation that corresponds to Eq. 4.2 to obtain the coefficients for the modal representation of Q^+ is

$$Q^+ = \sum_{j=0}^N \sum_{k=0}^N q_{jk}^+ \psi_{jk} \quad \text{where} \\ q_{jk}^+ = \frac{\sum_{p=0}^{N_{GQ}} \sum_{q=0}^{N_{GQ}} w_p w_q \psi_{jk}(s_p, s_q) Q_i \left(\xi_i(\vec{X}_{pq}), \eta_i(\vec{X}_{pq}), \zeta_i(\vec{X}_{pq}) \right)}{\int_{-1}^1 \int_{-1}^1 \psi_{jk}^2(\eta, \zeta) d\eta d\zeta} \quad \forall j \in [0, N], k \in [0, N]. \quad (4.5)$$

where w_p and w_q are the Gauss-Quadrature integration weights. The modal representation of Q^+ is then used to evaluate the inviscid flux term of Eq. 2.12.

4.1.1 Implicit Artificial Boundaries

Traditional implementations of a Chimera artificial boundary for an implicit Quasi-Newton solver will only update the boundary information as part of the right hand side evaluation. As a result, the scheme approaches an explicit scheme as the grids are sub-divided for parallel processing. Current trends in parallel computing are moving towards CPUs with greater quantities of cores. As a result, grids must be partitioned into smaller sizes to maintain parallel efficiency as the number of cores per CPU increases. However, increasing the number of processors degrades the parallel efficiency because the solution marching scheme becomes more explicit. To address this, an implicit artificial boundary scheme is presented that retains the original implicit scheme independent of the number of grid sub-divisions. The implicit scheme is retained through the use of parallel matrix-vector multiplications as part of an iterative Krylov space matrix solver.



Figure 4.8: Example Mesh for Implicit Chimera Artificial Boundary

Consider the Chimera mesh in Fig. 4.8. Equation 4.6 illustrates a traditional implicit scheme where the artificial boundaries are only used to update the right hand side vector.

$$\begin{bmatrix} A^1 & 0 \\ 0 & A^2 \end{bmatrix} \begin{bmatrix} \Delta Q^1 \\ \Delta Q^2 \end{bmatrix} = \begin{bmatrix} R^1(Q^1, Q^2) \\ R^2(Q^2, Q^1) \end{bmatrix} \Rightarrow$$

$$\begin{bmatrix} \begin{bmatrix} a_{1,1}^1 & a_{1,2}^1 & a_{1,3}^1 & 0 \\ a_{2,1}^1 & a_{2,2}^1 & 0 & a_{2,4}^1 \\ a_{2,1}^1 & 0 & a_{3,3}^1 & a_{3,4}^1 \\ 0 & a_{4,2}^1 & a_{4,3}^1 & a_{4,4}^1 \end{bmatrix} & 0 \\ 0 & \begin{bmatrix} a_{1,1}^2 & a_{1,2}^2 & a_{1,3}^2 & 0 \\ a_{2,1}^2 & a_{2,2}^2 & 0 & a_{2,4}^2 \\ a_{2,1}^2 & 0 & a_{3,3}^2 & a_{3,4}^2 \\ 0 & a_{4,2}^2 & a_{4,3}^2 & a_{4,4}^2 \end{bmatrix} \end{bmatrix} \begin{bmatrix} \Delta Q_1^1 \\ \Delta Q_2^1 \\ \Delta Q_3^1 \\ \Delta Q_4^1 \\ \Delta Q_1^2 \\ \Delta Q_2^2 \\ \Delta Q_3^2 \\ \Delta Q_4^2 \end{bmatrix} = \begin{bmatrix} R_1^1(Q_1^1, Q_2^1, Q_3^1) \\ R_2^1(Q_1^1, Q_2^1, Q_4^1, Q_1^2) \\ R_3^1(Q_1^1, Q_3^1, Q_4^1) \\ R_4^1(Q_3^1, Q_4^1, Q_2^1, Q_3^2) \\ R_1^2(Q_1^2, Q_2^2, Q_3^2, Q_2^1) \\ R_2^2(Q_1^2, Q_2^2, Q_4^2) \\ R_3^2(Q_1^2, Q_3^2, Q_4^2, Q_1^1) \\ R_4^2(Q_3^2, Q_4^2, Q_2^2) \end{bmatrix}$$
(4.6)

The super-scripts indicate grid number and the sub-script is the cell number. The R and ΔQ vectors for each grid are stacked to form a single vector for both grids. However, the two systems of equations

$$\begin{aligned}
A^1 \Delta Q^1 &= R^1(Q^1, Q^2) \\
A^2 \Delta Q^2 &= R^2(Q^2, Q^1)
\end{aligned}$$
(4.7)

are solved independently. The method does not account for the linearization terms $\frac{\partial R^1}{\partial Q^2}$ and $\frac{\partial R^2}{\partial Q^1}$. Without these terms, the quadratic convergence of the Newton solver cannot be realized. Equation 4.8 has the full linearization of the entire system of equations, including the linearization for the artificial boundaries designated by the C matrices.

$$\begin{bmatrix} A^1 & C^1 \\ C^2 & A^2 \end{bmatrix} \begin{bmatrix} \Delta Q^1 \\ \Delta Q^2 \end{bmatrix} = \begin{bmatrix} R^1(Q^1, Q^2) \\ R^2(Q^2, Q^1) \end{bmatrix} \implies$$

$$\begin{bmatrix} a_{1,1}^1 & a_{1,2}^1 & a_{1,3}^1 & 0 \\ a_{2,1}^1 & a_{2,2}^1 & 0 & a_{2,4}^1 \\ a_{2,1}^1 & 0 & a_{3,3}^1 & a_{3,4}^1 \\ 0 & a_{4,2}^1 & a_{4,3}^1 & a_{4,4}^1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 & 0 \\ c_{2,1}^1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & c_{4,3}^1 & 0 \end{bmatrix} \begin{bmatrix} \Delta Q_1^1 \\ \Delta Q_2^1 \\ \Delta Q_3^1 \\ \Delta Q_4^1 \end{bmatrix} = \begin{bmatrix} R_1^1(Q_1^1, Q_2^1, Q_3^1) \\ R_2^1(Q_1^1, Q_2^1, Q_4^1, Q_1^2) \\ R_3^1(Q_1^1, Q_3^1, Q_4^1) \\ R_4^1(Q_3^1, Q_4^1, Q_2^1, Q_3^2) \\ R_1^2(Q_1^2, Q_2^2, Q_3^2, Q_2^1) \\ R_2^2(Q_1^2, Q_2^2, Q_4^2) \\ R_3^2(Q_1^2, Q_3^2, Q_4^2, Q_4^1) \\ R_4^2(Q_3^2, Q_4^2, Q_2^2) \end{bmatrix}$$

(4.8)

A matrix-vector multiplication and dot product that spans all processors is required for the FGMRES algorithm to solve the system of equations in Eq. 4.8. Since the preconditioner is an approximation to A^{-1} , the C matrices are not included in the preconditioner. Hence, the complete C matrices are not necessary, only the sub-matrix-vector multiplications are required. The C matrices require the linearization of the interpolation operator I_h , namely

$$C^{n,m} \Delta Q^m = \frac{\partial R^n(I_h(Q^m))}{\partial Q^m} \Delta Q^m \quad (4.9)$$

which would have to be transferred from the processor with the donor grid to the processor with the receiver mesh. By using the chain rule the C matrix-vector multiplication can instead be written as

$$\frac{\partial R^n(I_h(Q^m))}{\partial Q^m} \Delta Q^m = \frac{\partial R^n(I_h)}{\partial I_h} \frac{\partial I_h(Q^m)}{\partial Q^m} \Delta Q^m = \frac{\partial R^n(Q)}{\partial Q} \frac{\partial I_h(Q^m)}{\partial Q^m} \Delta Q^m. \quad (4.10)$$

Because the interpolation operator, $I_h(Q^+)$, is a linear operator with respect to Q^+ , the following holds

$$\frac{\partial I_h(Q^m)}{\partial Q^m} \Delta Q^m = I_h(\Delta Q^m). \quad (4.11)$$

When the cell faces on the artificial boundary are coincident, the term $\frac{\partial I_h(Q^m)}{\partial Q^m}$ is the matrix that projects

the volume polynomial of a cell to a cell boundary. Using Eq. 4.11, the C matrix-vector multiplication is reduced to

$$C^{n,m} \Delta Q^m = \frac{\partial R^n(Q)}{\partial Q} I_h(\Delta Q^m), \quad (4.12)$$

where $\frac{\partial R^n(Q)}{\partial Q}$ is the Jacobian for of linearized interior scheme. Because the Jacobian for the interior scheme multiplies the interpolated values of ΔQ^m , only the interpolated values of the ΔQ^m need to be communicated across processors. All cell values associated with the interpolation operator would need to be transferred across processors if the $C^{n,m}$ were explicitly formulated and used as part of the matrix-vector multiplication. This communication must occur for each matrix-vector multiplication of Eq. 4.8 in the FGMRES algorithm. The communication and local matrix-vector multiplication are interlaced to minimize the impact of this communication on the parallel efficiency. The update algorithm is as follows:

1. Interpolate to GQ Nodes.
2. Perform a non-blocking send of interpolated information.
3. While information is communicated, perform the local matrix-vector multiplication.
4. Receive interpolated information and perform C matrix-vector multiplication.

The FGMRES algorithm requires a set of orthogonal vectors, v_n , that are computed in part by using the matrix-vector multiplication $w_{n+1} = Av_n$, where A is the matrix of equations solved by the FGMRES algorithm and w_{n+1} is a vector used to compute v_{n+1} . An example of the algorithm to perform this matrix-vector multiplication for two grids on separate processors is given in Table 4.1.

Table 4.1: Example of Implicit Chimera Artificial Boundary Algorithm

Step	Processor 1	Processor 2
1	Interpolate $I_h(v_n^1)$	Interpolate $I_h(v_n^2)$
2	Non-Blocking Send $I_h(v_n^1)$ to Process 2	Non-Blocking Send $I_h(v_n^2)$ to Process 1
3	Compute $w_{n+1}^1 = A^1 v_n^1$	Compute $w_{n+1}^2 = A^2 v_n^2$
4	Receive $I_h(v_n^2)$	Receive $I_h(v_n^1)$
5	Compute $w_{n+1}^1 = w_{n+1}^1 + C^1 I_h(v_n^2)$	Compute $w_{n+1}^2 = w_{n+1}^2 + C^2 I_h(v_n^1)$

In addition to the matrix-vector multiplication, the GMRES algorithm requires inner products of the v_n vectors. The inner product operations in parallel has a complexity of $O(n/p + \log(p))$, where n is the

degrees of freedom per-processor and p is the number of processors[141]. Thus, the computational time is dominated by the local inner products on each processor so long as $n/p > \log(p)$.

4.1.2 Implementation Strategy

The Chimera overset mesh shown in Fig. 4.9 is used as an example to detail the code structure for the artificial boundaries. Specifically, the artificial boundaries of the black grid are used to describe the details of the code structure. In this example, the left, upper, and right boundary of the black grid are artificial boundaries. A Boundary data type is used to store the Cartesian coordinates of the set of cell faces that represent a given artificial boundary. Three different instances of the Boundary data type, labeled Boundary 1, Boundary 2, and Boundary 3 are shown in Fig. 4.10. Two GQ nodes per face that must receive donor information from the red and blue grids are illustrated on the artificial boundaries in this example. The specific number of GQ nodes is dependent on the order of the approximation. Only grids with bounding boxes that intersect the bounding box of an artificial boundary are considered as possible donor grids. In this example, the bounding box of the red grid intersects the Boundary 1 and Boundary 2 bounding boxes, while the bounding box of the blue grid intersects the bounding boxes of Boundary 2 and Boundary 3.

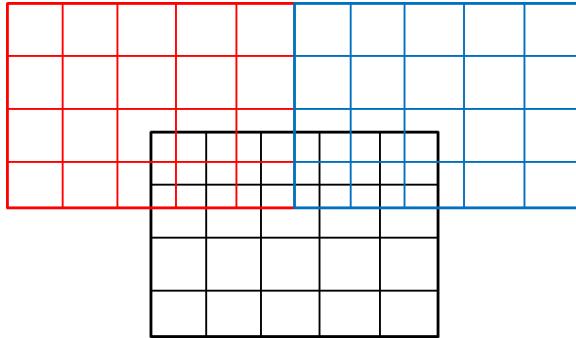


Figure 4.9: Example Chimera Mesh Used to Detail the Implementation of the Artificial Boundary Communication Scheme

An instance of a Donor Node Set data type is associated with each Boundary considered by a donor grid. In the example in Fig. 4.10, the red grid instantiates red Donor Node Set 1 and Donor Node Set 2 that correspond to Boundary 1 and Boundary 2 respectively, while the blue grid instantiates blue Donor Nodes Set 1 and Donor Nodes Set 2 that correspond to Boundary 2 and Boundary 3 respectively. A Donor Node

Set instantiates a Donor Node for each GQ node on the artificial boundary with which the Donor Node Set is associated. A Donor Node data type stores the index to a donor cell and the curvilinear cell coordinates of that donor cell that correspond to the Cartesian coordinates of a GQ node on an artificial boundary. A Donor Node is retained in the Donor Node Set if the search algorithm using the Kd-tree and Newton's method described in Section 4.1 successfully locates a donor cell for that Donor Node, otherwise the Donor Node is discarded. The red Donor Node Set 2 associated with Boundary 2 (See, Fig. 4.10) instantiates a Donor Node for each GQ node on Boundary 2, but only retains the left five GQ nodes that reside within the red grid. Similarly, the blue Donor Node Set 1 associated with Boundary 2 retains the right five GQ nodes on Boundary 2 that reside within the blue grid. All of the Donor Node Sets regardless of the artificial boundary with which they are associated for a given grid are stored in a Donor Collection data type. The red Donor Collection in the example shown in Fig. 4.10 stores the two Donor Node Sets associated with Boundary 1 and Boundary 2, as well as the artificial boundaries associated with the blue grid which are not illustrated in this example. The Donor Collection stores a single array where the interpolated information from the Donor Nodes is stored. The Donor Node Sets store the mapping information to map the interpolated values from Donor Nodes to the array in the Donor Collection.

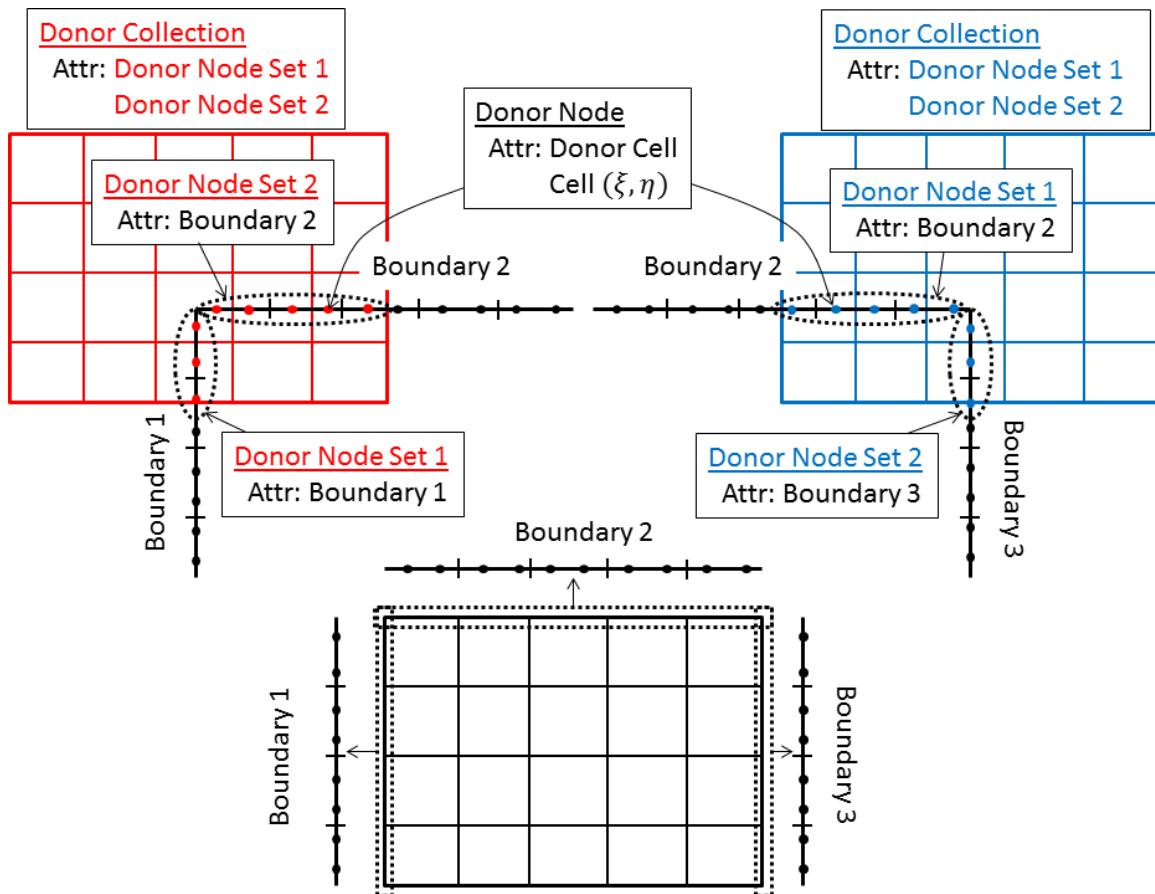


Figure 4.10: Details of the Donor Implementation

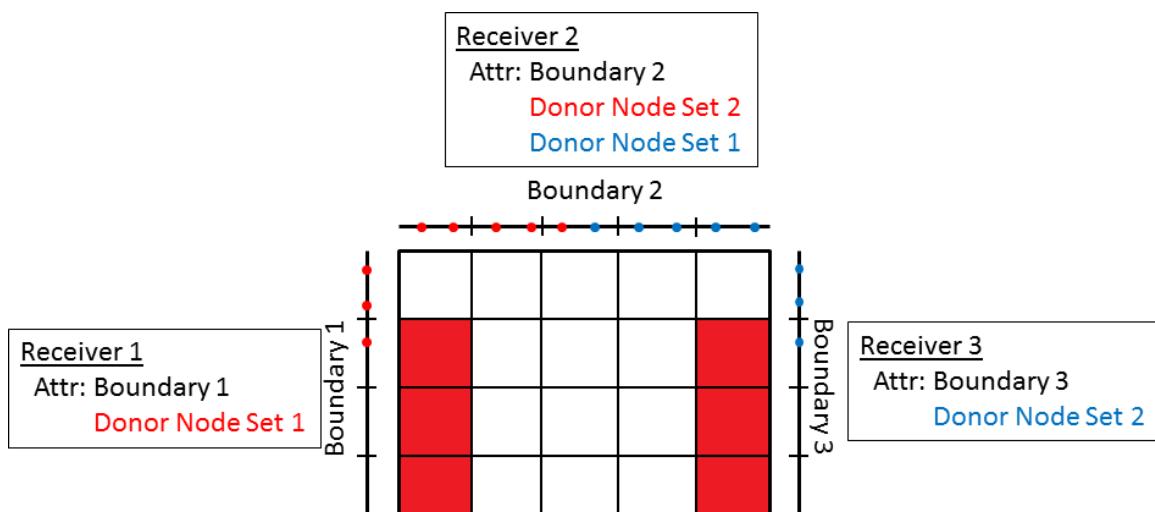


Figure 4.11: Details of the Receiver Implementation

A set of instances of a Receiver data type are also created for each Boundary of the black grid are illustrated as Receiver 1, Receiver 2, and Receiver 3 in Fig. 4.11. The Receiver data type evaluates the L^2 -projection in Eq. 4.2 and computes the flux integrals in Eq. 2.12 on the artificial boundaries. Each Receiver instance interrogates the Donor Collections of the red and blue grids and stores a reference to all the Donor Node Sets that are associated with the same artificial boundary as the Receiver instance. Hence, Receiver 1 references the red Donor Node Set 1, Receiver 2 references the red Donor Node Set 2 and the blue Donor Node Set 1, and Receiver 3 references the blue Donor Node Set 2. The mapping information in the Donor Node Sets are used by the Receiver instance to map the interpolated values from the array stored in the Donor Collections to the GQ nodes for each face on the artificial boundary. For Boundary 2, the left two cell faces receive interpolated values from the red grid for both GQ nodes, and the right most cell faces receive interpolated values from the blue grid. The cell face in the center of Boundary 2 receives interpolated values from both the red and blue grids. The average value of all donor grids is used as the value for the GQ node on the right hand side of Eq. 4.2 if multiple grids are overlapping such that a GQ node of an artificial boundary resides within multiple donor grids. Six cell in the black grid are colored red to indicate that they have GQ nodes without donor information since only part of Boundary 1 and Boundary 3 overlap the red and blue grids.

The coding strategy for the artificial boundaries also simplifies the implementation of periodic boundary conditions. The periodic boundary condition strategy is illustrated in Fig. 4.12. Here, the upper and lower boundaries are designated as periodic. Two instances of the Boundary data type that stores the Cartesian coordinates are created for the upper and lower boundaries of the grid. The Cartesian coordinates on the upper periodic boundary are then offset by a Δy so that it is co-incident with the lower boundary of the grid, and the lower periodic boundary is offset by the same Δy so that it is co-incident with the upper boundary of the grid. These two periodic boundaries then locate the appropriate GQ nodes in the grid, and the periodic boundary condition is established through the artificial boundary framework. Hence, periodic boundaries do not require any specialized code, and the grid coordinates are not required to be periodic because the periodic boundaries are artificial boundaries. This also means that periodic boundaries are properly established without any specialized logic even when the grid is decomposed for parallel execution.

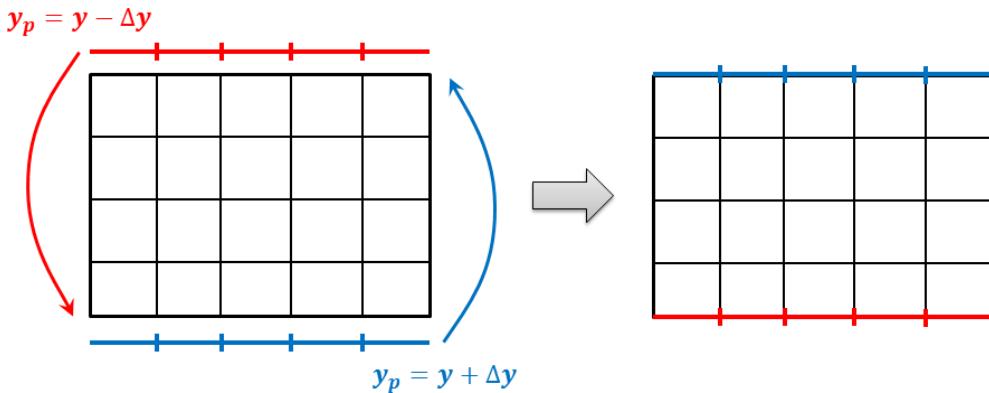


Figure 4.12: Schematic of the Periodic Boundary Condition Implementation

The scheme for distributed memory calculations builds upon the previously described inter-grid communication scheme. All inter processor communication relies on the Message Passing Interface (MPI) standard. The initialization process of the communication occurs in parallel on the distributed memory hardware. The grids in the Chimera overset mesh system are subdivided if the number of grids is less than the number of processors requested. The subdivision algorithm is currently a greedy type algorithm that recursively divides the largest dimension of the largest grid until the number of grids matches the number of processors. Grids are agglomerated on processors if the number of grids exceeds the number of processors. The solver relies on multi-threading when the multiple grids reside on a single process, where each grid is assigned a single thread. The partitioning algorithm is simplistic and could be improved. Once a processor is assigned grids from the partitioned system, the partitioned grid is read directly from the hard drive. Hence, a master node with enough memory to store the largest grid in the grid system is not required. Once all the processors have read the grid partitions, the bounding boxes of the artificial boundaries and grids are gathered on one processor where the intersections between artificial boundary bounding boxes and grid bounding boxes are used to determine which processors might need to establish communication. The intersection pattern is then distributed to all processors and the artificial boundaries are transferred between processors based on the intersection pattern. An instance of a Donor Collection Set data type is created on a given processor for each processor that might require interpolated values from the given processors. The communication pattern for the grids in the Chimera overset mesh shown in Fig. 4.9 with one grid per processors is shown in Fig. 4.13.

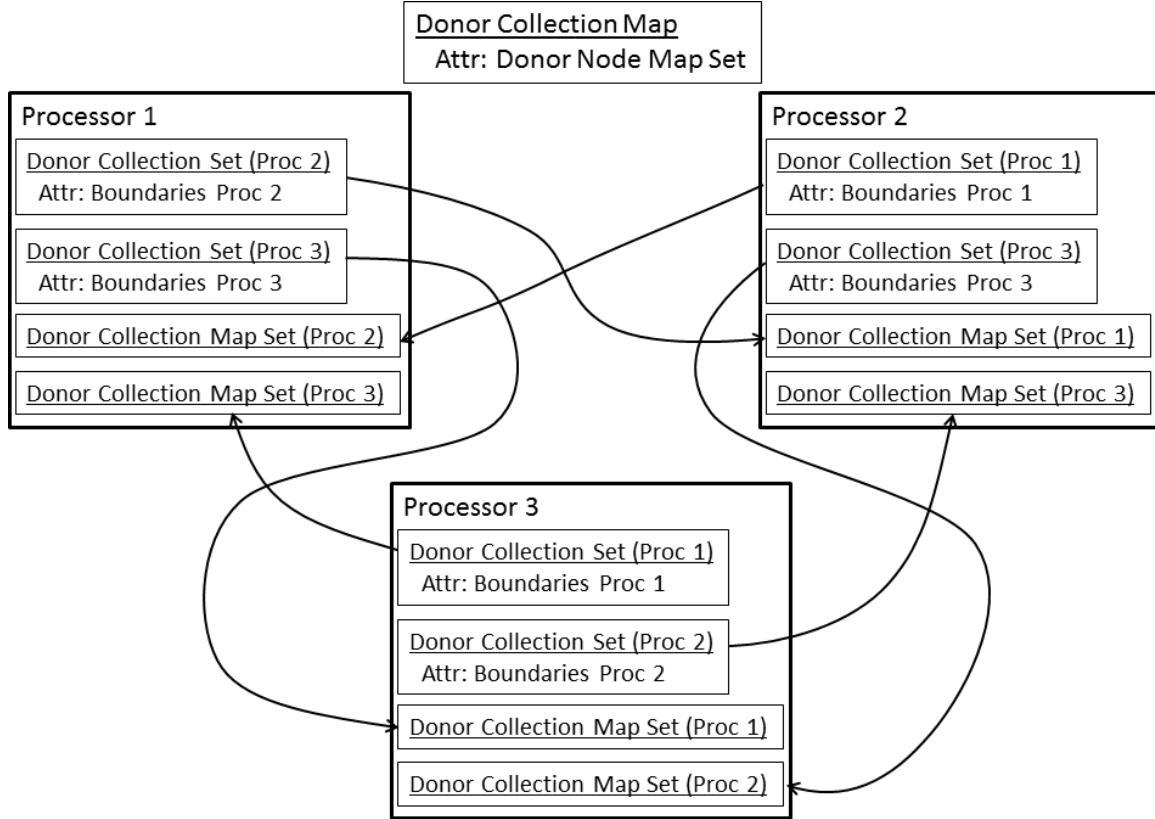


Figure 4.13: Schematic of the MPI Implementation

In this example each processor communicates with the other two processors. Using Processor 1 as an example, two Donor Collection Set instances are created that store the artificial boundaries from Processor 2 and Processor 3 labeled Donor Collection Set (Proc 2) and Donor Collection Set (Proc 3), respectively. Each Donor Collection Set stores a Donor Collection instance for each grid on the processor. In addition, the Donor Collection Set stores a single array that is populated by the Donor Collections. All of the mapping information of the Donor Collection Set is transferred back to the processor where the artificial boundaries originate, once the Donor Node Sets of each Donor Collection have located the donor cells. In the example shown in Fig. 4.13, the mapping information of the Donor Collection Set (Proc 2) on Processor 1 is transferred to Processor 2, and Donor Collection Set (Proc 3) is transferred to Processor 3. The transferred mapping information is stored in a Donor Collection Map Set. The Donor Collection Map Set stores a set of Donor Collection Maps as well as an array of equal size to the array stored in the Donor Collection Set to receive the interpolated information. Each Donor Collection Map contains a set of Donor Node Maps.

A Donor Node Map stores the same mapping information as is stored in a Donor Node Set, but lacks the Donor Nodes. Each Donor Collection Map is queried by each Receiver instance on the processor where the artificial boundaries originate and the Receivers store a reference to any Donor Node Map that is associated with the same Boundary as the Receiver. The Receivers now have the mapping information to map the interpolated values stored in the array of the Donor Collection Map Set to the artificial boundary.

4.2 Hole Cutting with Curved Cells

The hole cutting scheme presented here is based on a Direct Hole Cutting method.[142, 143, 144] The Direct Hole cutting method is a two step process. First, a set of cutting surfaces are used to define hole locations. Cells interior to the surfaces are designated as Hole cells and excluded from the computational domain, while cells exterior to the surface are flagged as Field cells and retained. In this step, the evaluation of cells as Hole or Field is restricted to cells that are immediately adjacent to the cutting surfaces. The second step is a fill process that flags remaining cells away from the cutting surfaces.

Cutting surfaces can be computational surfaces from the overset grid system or auxiliary surfaces provided solely to facilitate the hole cutting process. In addition, a cutting surface can be designated as a “Solid” or a “Field” cutting surface. A Solid cutting surface should be used when the cutting surface is extracted from a solid wall boundary of the overset grid system, whereas a Field cutting surface should be used when the cutting surface represents an artificial boundary. A circular grid consisting of a single layer of cells with cubic polynomial mappings, $N_g = 3$, and a rectangular background grid with linear polynomial mappings, $N_g = 1$, (shown in Fig. 4.14) are used to illustrate the difference between the types of cutting surfaces. Figure 4.14a shows the result of using the inner boundary of the circular mesh as a Solid cutting surface to flag cells in the background grid. Cells in the background grid that intersect the inner surface of the cylindrical grid are flagged as Hole cells. This guarantees that none of the cells in the background mesh that reside within the inner surface of the circular grid are flagged as Field cells. The final hole after the fill process is shown in Fig. 4.14b. Figure 4.14c shows the result of using the outer boundary of the circular grid to flag cells in the background grid. In this case, the cells in the background grid that intersect the cutting surface are flagged as Field cells. The final hole after the fill process is shown in Fig. 4.14d. For this example, the Field cutting surface maximizes the hole cut from the background grid while ensuring that the outer boundary of the circular grid is connected to the proper donor cells in the background mesh.

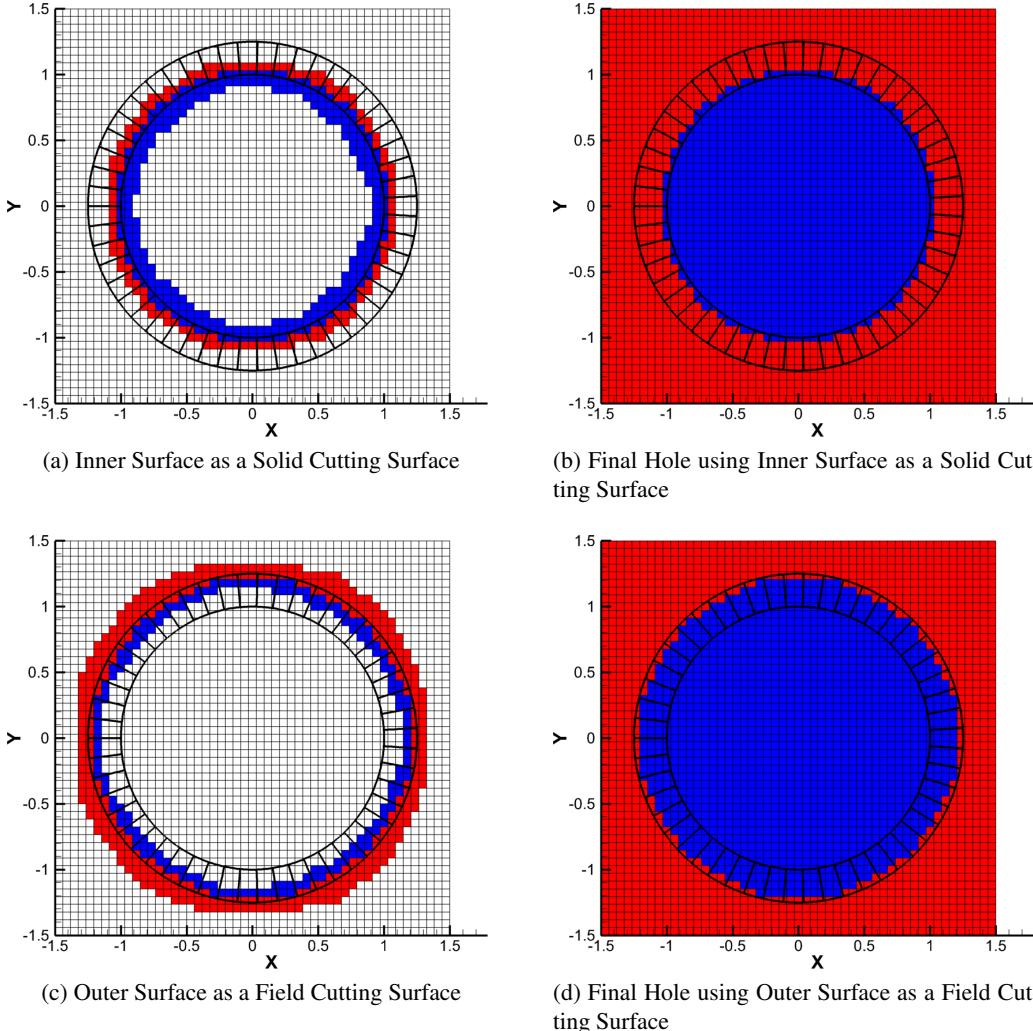


Figure 4.14: Cutting Surfaces Used to Flag Boundary Between Hole and Field Cells (Blue - Hole Cells, Red - Field Cells, White - Undetermined)

After the initial Hole and Field cells have been identified using the cutting surfaces, the remaining undetermined cells are evaluated and determined to be either Hole or Field cells through a fill process. The fill process sweeps symmetrically through the mesh and each cell encountered that is flagged as undetermined is given the designation of any neighboring cell that has been flagged as either Hole or Field. The symmetric sweeps ensure that the entire mesh is filled with the least computational effort. It is critical that the cutting surfaces are “water tight”, i.e., no connections between unassigned cells prior to the fill process; otherwise, the fill process can leak. A leak will result in regions of the mesh flagged as Hole cells that should be flagged as Field cells, or vice versa. An example of the fill process leaking from an open cutting surface is shown

in Fig. 4.15. Because the cutting surface used in Fig. 4.15a is open, the fill process leaks Hole cells into the domain that should be flagged as Field cells (Fig. 4.15b). Note that a cutting surface can take advantage of the domain boundary of the background grid to close the volume. For example, Fig. 4.16a illustrates how the grid around a half circle can be used to flag cells in the background grid. The final hole after the fill process, shown in Fig. 4.16b, has not leaked because the cutting surface intersects the boundary of the background grid.

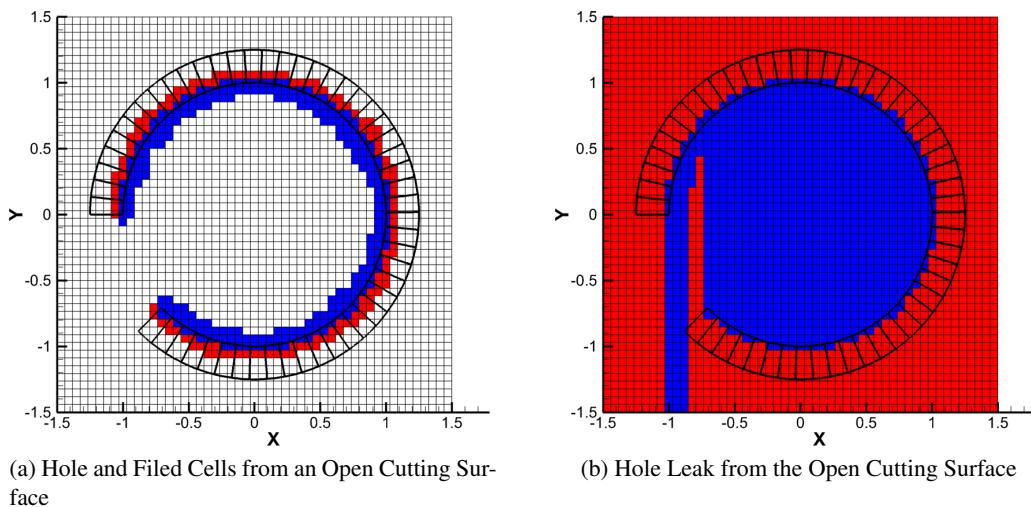


Figure 4.15: Open Cutting Surface Leading to a Leak in the Fill Process (Blue - Hole Cells, Red - Field Cells, White - Undetermined)

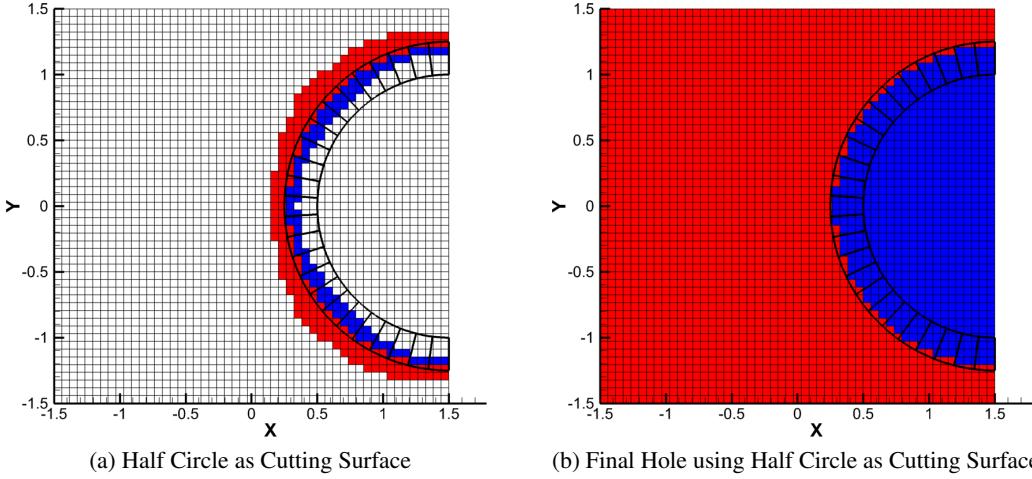


Figure 4.16: Half Circle using Background Grid Boundary to Close Cutting Volume (Blue - Hole Cells, Red - Field Cells, White - Undetermined)

4.2.1 Details of the Cutting Surface Flagging Process

The minimum distance between a cell and a parametric polynomial segment of the cutting surface is used to determine if the cell is a Hole or a Field cell. The Cartesian coordinates of an individual segment of a cutting surface, denoted as $(x_s(s), y_s(s))$ where $s \in [-1, 1]$ is the parameter, is a polynomial mapping similar to the polynomial mapping in Eq. 3.12 of a cell. The process of flagging cells in the background grid in the neighborhood of cutting surfaces is accomplished in two steps. The first step is to locate all cells in the background grid that are in the neighborhood of the cutting surface, and the second step is to assess whether the cells should be flagged as a Hole or Field cell.

The first step in locating cells in the neighborhood of the cutting surface uses a KD-Tree[139], which is populated with the bounding boxes of the cutting surface segments, to extract a set of segments with bounding boxes that intersect the bounding box of a given cell. A bounding box is defined by the minimum and maximum Cartesian extents of the segment or cell. The set of segments is then used in the second step to assess whether the cell should be flagged as a Hole or a Field cell.

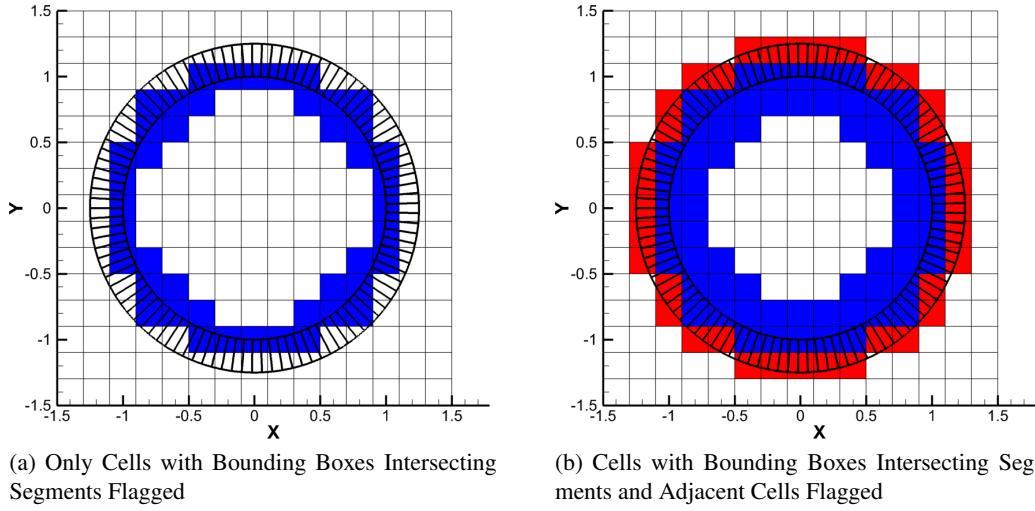


Figure 4.17: Cutting Surfaces with Small Segments in Size Relative to the Background Grid Cells (Blue - Hole Cells, Red - Field Cells, White - Undetermined)

Limiting the assessment to the cells with bounding boxes that intersect segment bounding boxes is not adequate when the cells are significantly larger in size relative to the segments. As demonstrated in Fig. 4.17a where the inner surface of the circular grid is used as a Solid cutting surface, all cells assessed intersect the cutting surface and are hence flagged as Hole cells. The result of the subsequent fill process would flag the entire background grid as Hole cells. To remedy this problem, the set of segments with bounding box intersections with a given cell are used to assess all cells adjacent to that cell. The result of assessing adjacent cells is shown in Fig. 4.17b. The subsequent fill process would, in this case, produce a proper hole.

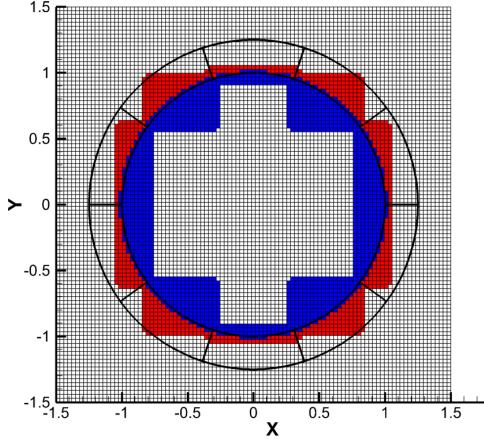


Figure 4.18: Cutting Surfaces with Segments Much Larger in Size Relative to the Background Grid Cells (Blue - Hole Cells, Red - Field Cells, White - Undetermined)

A small disadvantage of using intersecting bounding boxes to locate cells in the neighborhood of the cutting surfaces is that more cells than necessary may be evaluated when the segments are significantly larger than the cells in the background grid. As shown in Fig. 4.18, the size of the large segments relative to the background grid cells size has led to many cells away from the cutting surface to be flagged unnecessarily as Hole and Field cells. The bounding boxes of the segments are also distinguishable in the rectangular pattern of the flagged cells.

For the second step, a convention is needed to define which side of a segment is used to flag cells as Hole or Field cells. In this work, the field side of the segment is defined by the outward normal of the segment, and the hole side by the inward normal. The minimum distance vector, \vec{r} , between the cell and segment is used to determine if a cell is on the field side, the hole side, or if it intersects the segment. A positive dot product of the distance vector, \vec{r} , with the segment normal vector, \vec{n} , indicates a field cell, and a negative dot product a hole cell. A zero magnitude distance vector indicates that the segment intersects the cell.

The minimum distance vector is found by minimizing the function

$$d(\xi, \eta, s) = \sqrt{(x_c(\xi, \eta) - x_s(s))^2 + (y_c(\xi, \eta) - y_s(s))^2}. \quad (4.13)$$

Equation 4.13 is minimized numerically rather than analytically due to the possibly of different polynomial mappings of the cell and segment. Ideally Eq. 4.13 is minimized using a numerical procedure with quadratic convergence property such as Newton's method. The Jacobian and Hessian of Eq. 4.13 are easily

computable as the polynomial mappings are differentiable with respect to ξ , η , and s . However, when the segment intersects the cell there exists an infinite number of solutions with $d(\xi, \eta, s) = 0$. Under these circumstances, the solution to the system of equations is not unique and Newton's method fails to converge. Restricting the distance function to the boundaries of the cell produces a single solution with $d(\xi, \eta, s) = 0$ when the segment intersects the cell boundary. Unfortunately, in three-dimensions the segment is a plane and the cell is a volume. Restricting the search to the boundaries of the cell still has an infinite number of solutions when the plane segment intersects the cell. Furthermore, when the segment is smaller than the cell and is completely contained within the cell, restricting the distance equation to the boundary of the cell would preclude the method from determining that the segment intersects the cell. Gradient descent methods were also found to have difficulties in obtaining solutions with $d(\xi, \eta, s) = 0$ when the segment and cell intersect.

The gradient free Nelder-Mead minimization method[145] was found to be more robust than the gradient methods when the cell and segment intersect. The method uses a simplex of $M + 1$ vertices in an M -dimensional space. An example of the initial step of the Nelder-Mead method is shown in Fig. 4.19. The initial parametric values for the vertices, labeled P_0 through P_3 , are shown in Fig. 4.19a along with the corresponding Cartesian locations of the vertices on the segment and cell. The objective function, Eq. 4.13, is evaluated at all $M + 1$ vertices, and the vertices are sorted so that P_0 and P_3 have the minimum and maximum values of the objective function, respectively. The vertex with maximum value of the objective function, P_3 , is reflected through the parametric centroid of the remaining vertices, (P_0, P_1, P_2) , as shown in Fig. 4.19b. In this example, the reflected vertex would fall outside of the cell. To retain the reflected vertex inside the valid parametric domain, the reflected vertex is restricted to moving 80% of the distance to the boundary of the parametric domain as shown in Fig. 4.19c. If the reflected vertex were simply restricted to the boundary of the parametric domain, the method would break down whenever all the vertices are moved to a boundary of the cell. The method does not allow movement perpendicular to the boundary of the parametric domain in this situation as the centroid used to reflect P_3 is also located on the boundary of the parametric domain as shown in Fig. 4.21.

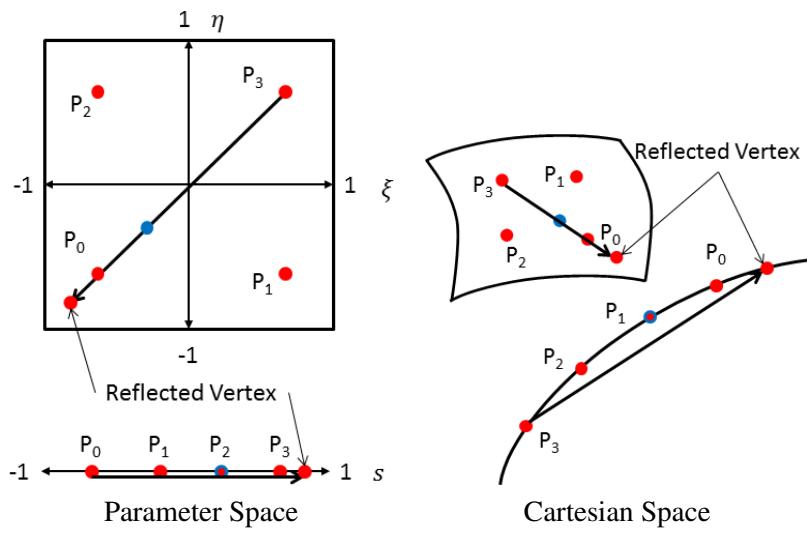
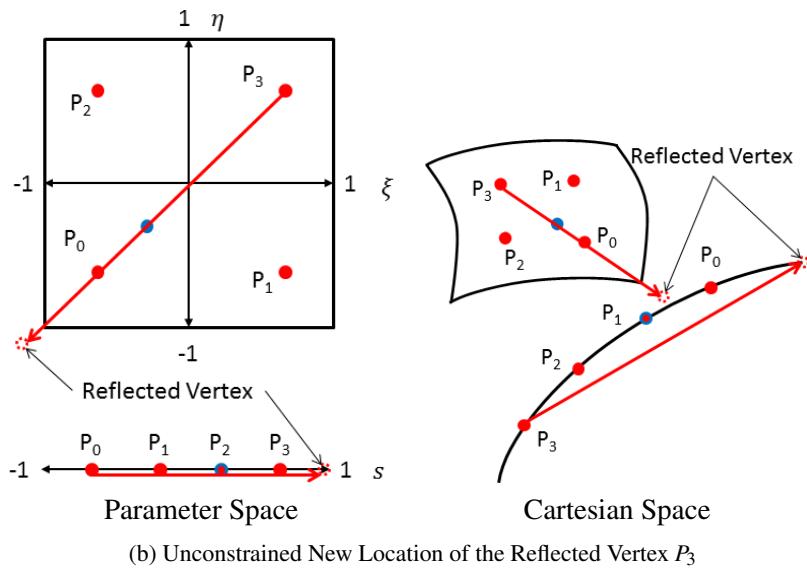
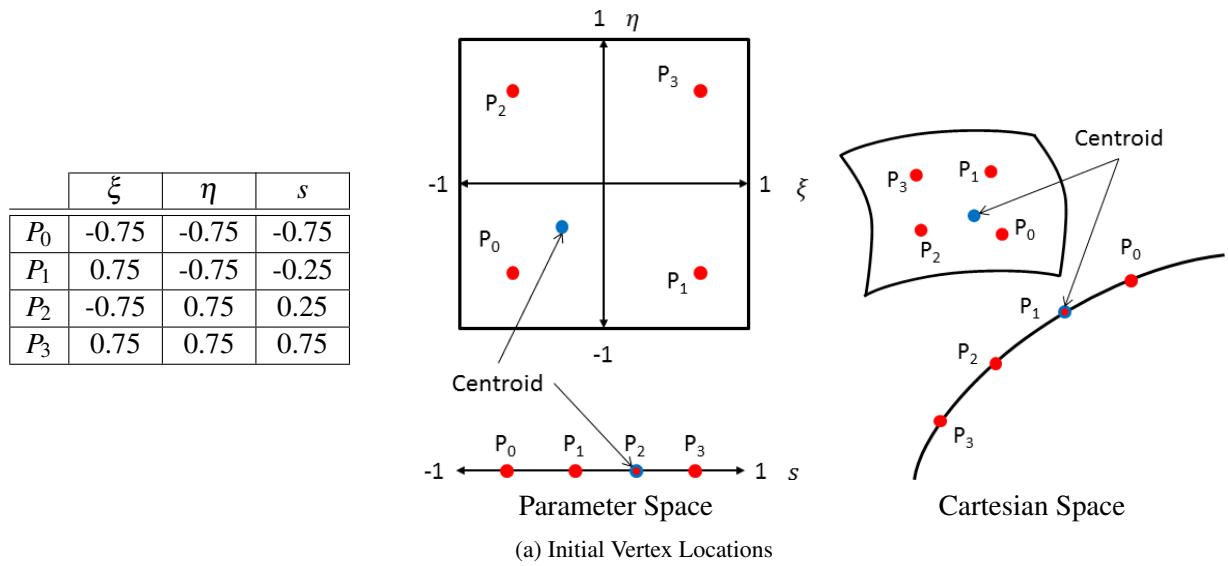


Figure 4.19: Nelder-Mead Method
188

If the objective function at the reflected vertex is smaller than the objective function of P_0 , the reflection distance is doubled; so long as it remains inside the parametric domain. If the vertex with twice the reflection distance still has the smallest objective function, it is retained and replaces P_3 . Otherwise, the original reflected vertex is retained and replaces P_3 . P_3 is also replaced with the reflected vertex when the objective function at the reflected vertex is smaller than the value at P_2 but greater than P_1 . If the objective function at the reflected vertex is greater than the value at P_3 , the reflection distance is halved, i.e., contracted, and replaces P_3 so long as the objective function decreases. If the contracted reflection fails to reduce the objective function, the vertices P_1 , P_2 , and P_3 are moved half the parametric distance to P_0 . The new set of vertices is again sorted based on an increasing value in the objective function and the method is repeated until $P_0 = P_3$ or $d(\xi, \eta, s) = 0$.

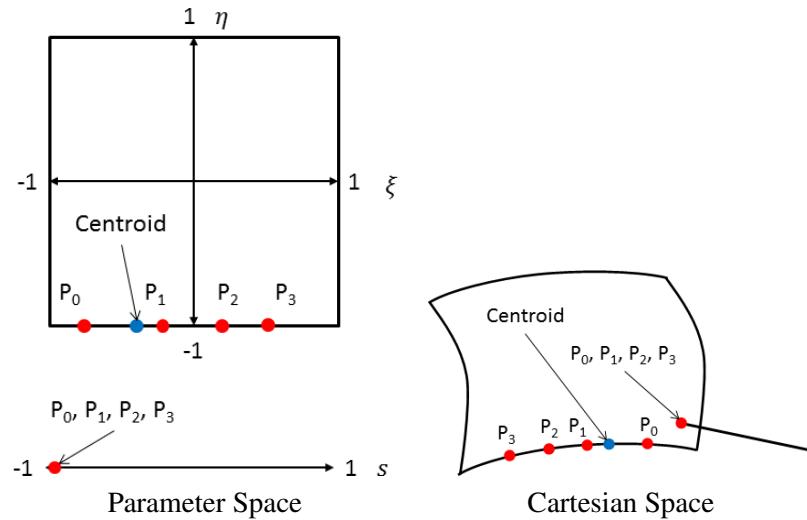


Figure 4.20: Nelder-Mead Degenerate Situation where Vertices Remain on Cell Boundary

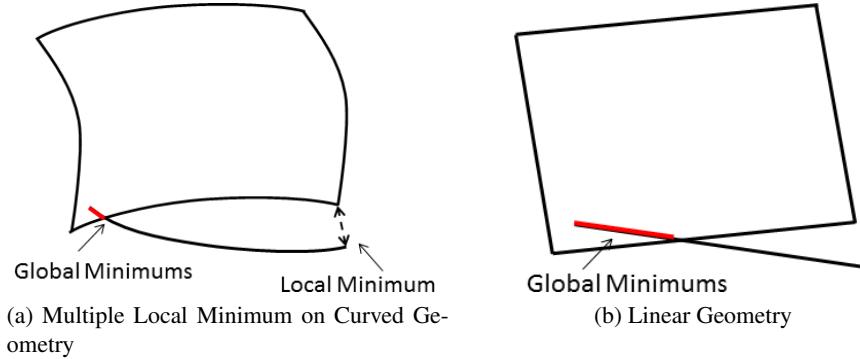


Figure 4.21: Nelder-Mead with Multiple Local Minimums

The Nelder-Mead method is a local-minimization method, and could converge to either the local or global minimums shown in Fig. 4.21a. If the local minimum is found, the cell would incorrectly be determined to not be intersecting the cell. An approach to avoid this situation is to first apply the minimum distance search using the linear component of the cell and segment as shown in Fig. 4.21b. The solution from the linear search is then used to initialize the Nelder-Mead method on the complete curved geometry. Unfortunately, this approach avoids the problem only in a limited number of situations. A robust method that always finds the global minimization has not yet been found.

It is possible for different cutting surface segments to flag a single cell as both a Hole and a Field cell. An example of this is shown in Fig. 4.22. In Fig. 4.22a, the segment with the normal vector \vec{n}_1 and the minimum distance \vec{r}_1 to the cell will flag the cell as a Field cell, but the segment with the normal vector \vec{n}_2 and minimum distance \vec{r}_2 will flag the cell as a Hole cell. To resolve this conflict, the segment with the shortest distance vector takes precedence. In this case, $|\vec{r}_1|$ is smaller than $|\vec{r}_2|$ and the cell is flagged as a Field cell. It is also possible for the two segments to share the same distance vector if the minimum distance occurs at the common end point of the two segments as shown in Fig. 4.22b. Similarly to the previous case, the segment with the normal vector \vec{n}_1 will flag the cell as a Field cell, and the segment with the normal vector \vec{n}_2 will flag the cell as a Hole cell. To resolve the conflict in this situation, the average of the normal vectors \vec{n}_1 and \vec{n}_2 is used to determine whether the cell should be a Hole or Field cell. In this example, the cell is flagged as a Field cell.

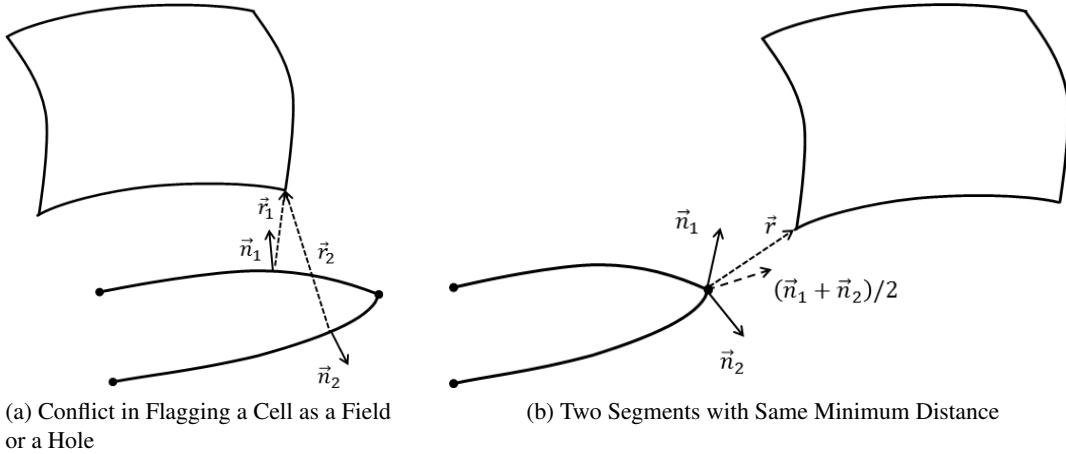


Figure 4.22: Resolving Contradictory Flagging of a Cell from Multiple Faces

4.2.2 Cutting Groups

The flagging process described in the previous section breaks down if multiple cutting surfaces overlap each other. This is demonstrated in Fig. 4.23 where two cylindrical grids overlap each other as well as a square background grid. The outer boundaries of the cylindrical grids are used to cut a hole in the background grid in order to minimize the overlapping regions. The result of the cutting surface flagging process when both cutting surfaces are used concurrently is shown in Fig. 4.23b. The cutting surfaces have flagged regions which should be Hole cells as Field cells because the two cutting surfaces overlap each other. As a result, the fill process produces incomplete holes with regions of Field cells as shown in Fig. 4.23b.

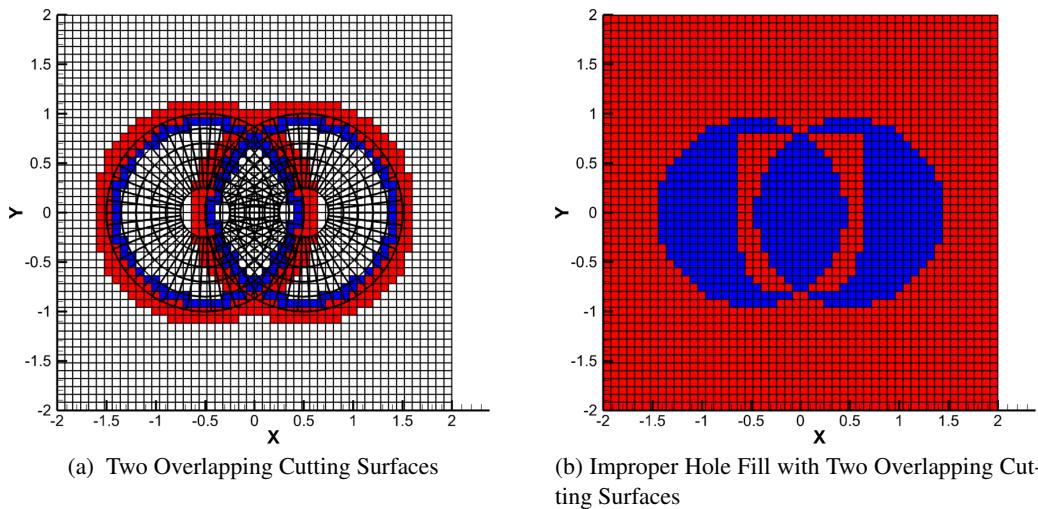


Figure 4.23: Example of Improper Hole Cutting with Two Overlapping Cutting Surfaces

To remedy this problem, the user has the ability to specify groups of cutting surfaces. Each group must contain a set of cutting surfaces that form a closed volume and the cutting surface groups must not overlap each other. Each group of cutting surfaces will perform the cutting surface flagging and fill process separately. This is demonstrated in Figs. 4.24a and 4.24b where holes have been cut using the left and right cylinder cutting surfaces respectively. Once all cutting surface groups have completed their hole cutting process, the union of all holes provides the final hole as shown in Fig. 4.24c. The final mesh with holes cut in the adjacent cylinders as well as the background mesh is shown in Fig. 4.25.

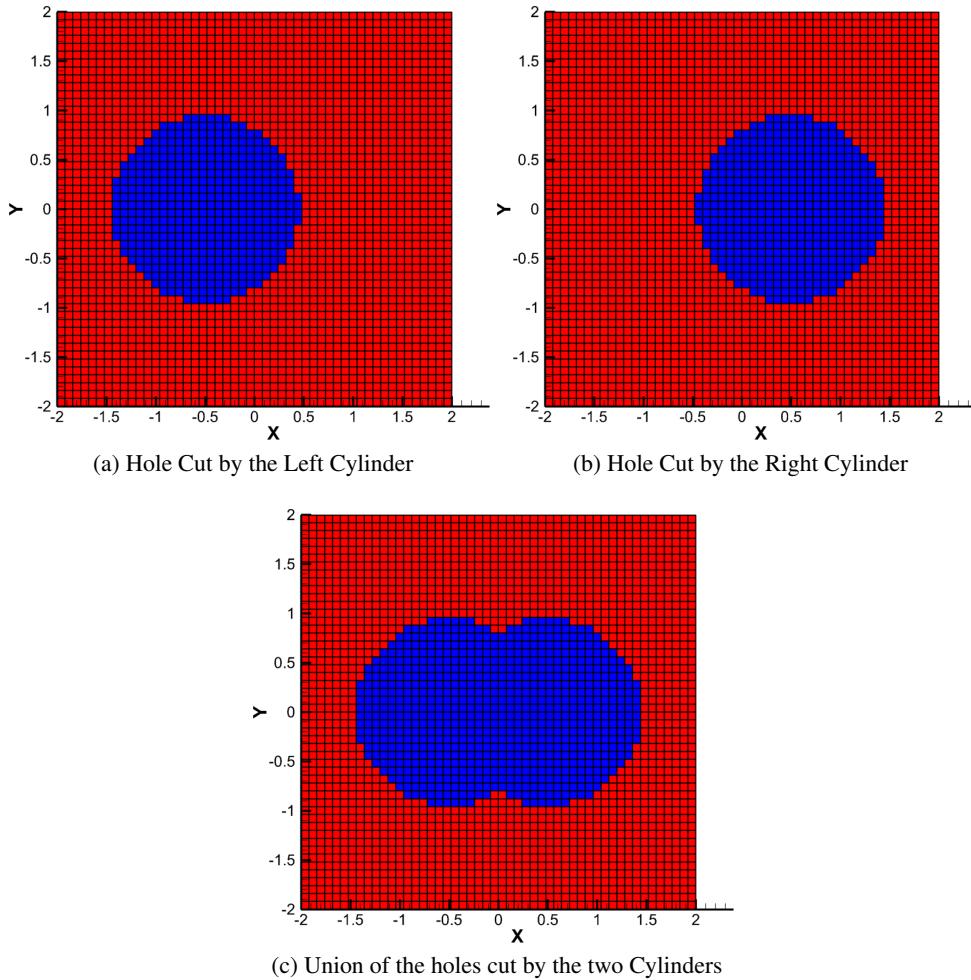


Figure 4.24: Examples of Normal Vectors used to Flag Field and Hole Nodes

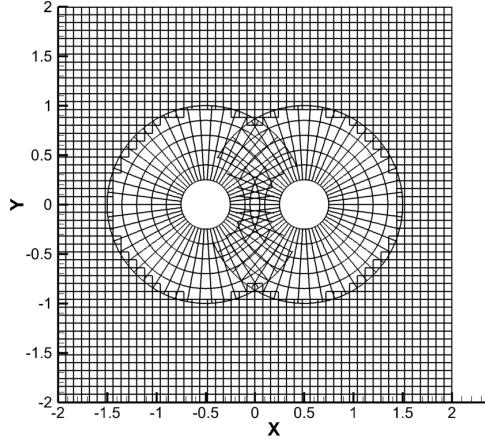


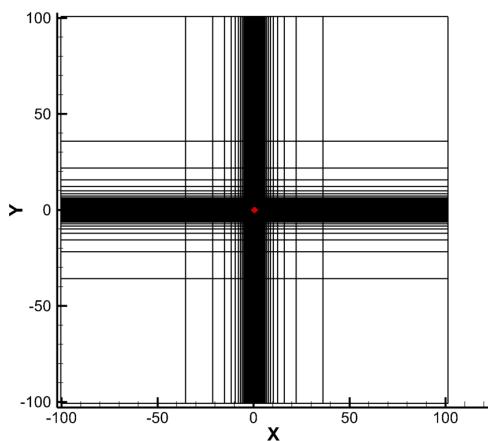
Figure 4.25: Final Holes in the Double Cylinder Mesh

4.2.3 Hole Cutting Examples

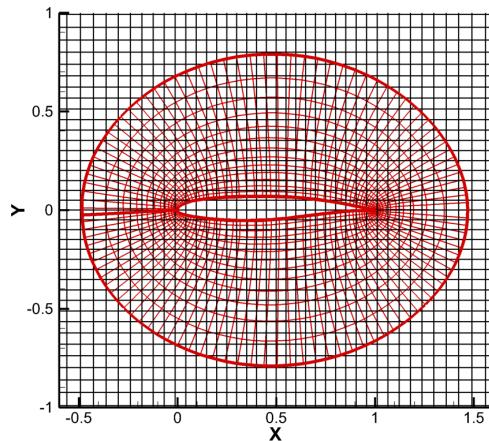
This section will demonstrate the hole cutting scheme on three geometric configurations, the SKF 1.1 airfoil[146], the SKF 1.1 airfoil with a flap[146], and a staggered tube bank similar to a heat exchanger[147, 148, 149, 150, 151]. The SKF 1.1 airfoil is an example of using a curved cutting boundary to cut a hole in a grid with linear cell mappings. The second example uses curved cutting boundaries to cut holes in grids with curved cells. The last example illustrates the use of cutting groups to create a hole from multiple overlapping cutting surfaces.

4.2.3.1 SKF 1.1 Airfoil

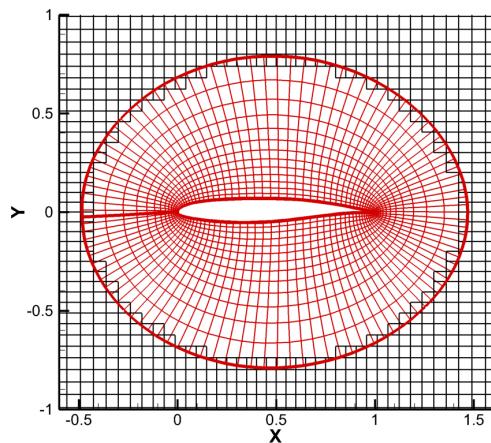
A Chimera overset mesh with a body fitted grid with a cubic polynomial mapping, $N_g = 3$, that defines the geometry of the SKF 1.1 airfoil[146] and a rectangular background with a linear polynomial mapping, $N_g = 1$, grid is shown in Fig. 4.26. The curved outer boundary of the airfoil grid is used to cut a hole in the linear background mesh as shown in Fig. 4.26c. Note the minimum overlapping region between the background grid and airfoil grid. For comparison, a finite volume Chimera grid for the SKF 1.1. airfoil is shown in Fig. 4.27, where the donor and receiver nodes are highlighted in blue and green for the airfoil grid and the background grid, respectively. A significant extent of overlap (relative to the DG-Chimera) is required to avoid a cyclic dependency between the receiver nodes on the two meshes.



(a) Chimera Overset Mesh



(b) Chimera Overset Mesh Closeup without Hole Cut



(c) Chimera Overset Mesh Closeup with Hole Cut

Figure 4.26: SKF 1.1 Airfoil

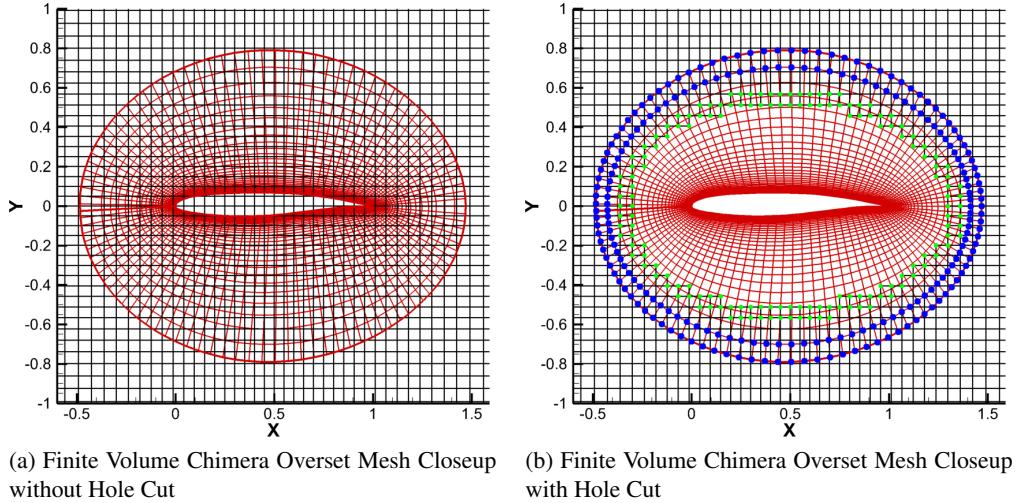
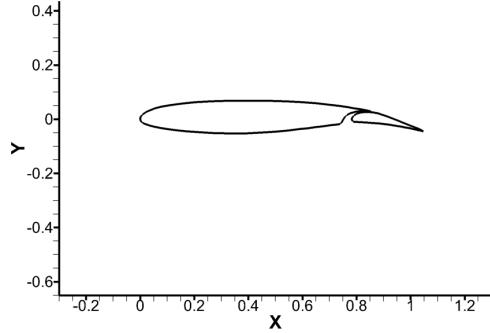


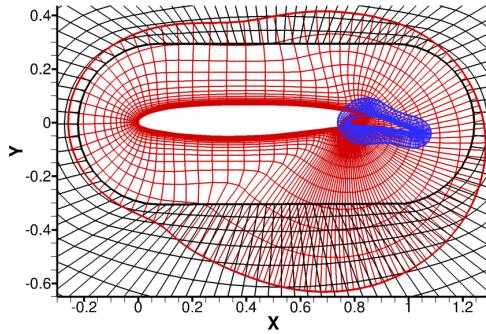
Figure 4.27: SKF 1.1 Airfoil Finite Volume Chimera Mesh

4.2.3.2 SKF 1.1 Airfoil with Flap

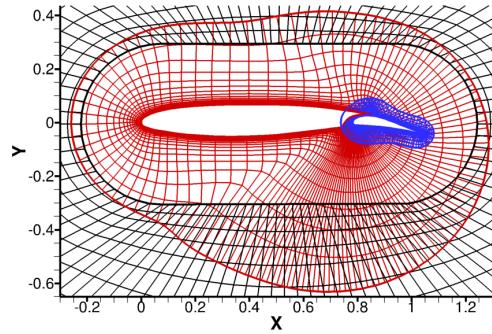
The geometry for the SKF 1.1 airfoil with the flap extended is shown in Fig. 4.28a (See, configuration 5 in Ref. [146]). A Chimera overset mesh consisting of 3 grids, one for the airfoil with $N_g = 3$, one for the flap with $N_g = 3$, and one which extends the farfield to 100 chords from the airfoil with $N_g = 1$, is shown in Fig. 4.28b without holes. The surface of the airfoil is used to cut a hole in the flap grid, and the surface of the flap is used to cut a hole in the airfoil grid. The final Chimera mesh with holes is shown in Fig. 4.28c. This is a situation where curved cutting surfaces are used to cut holes in grids consisting of curved elements. The hole cut by the flap is shown in Fig. 4.28d, and the hole cut by the airfoil is shown in Fig. 4.28e.



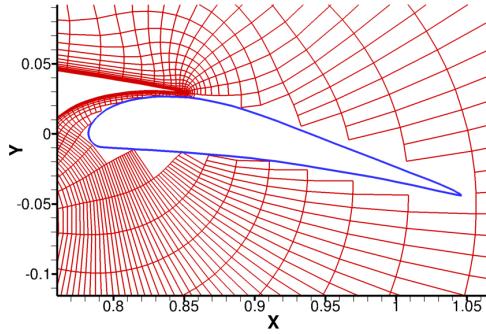
(a) SKF 1.1 Airfoil with Flap Geometry



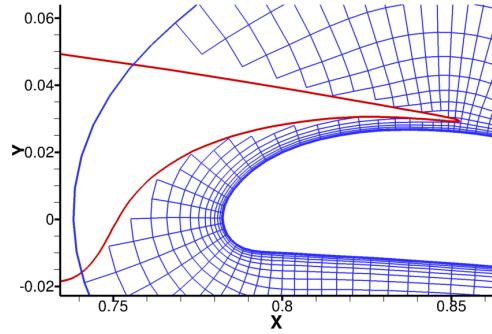
(b) SKF 1.1 Airfoil with Flap Overset Mesh without Holes



(c) SKF 1.1 Airfoil with Flap Overset Mesh with Hole Cuts



(d) Hole Cut from Airfoil Grid using the Flap Surface



(e) Hole Cut from Flap Grid using the Airfoil Surface

Figure 4.28: SKF 1.1 Airfoil with Flap Meshes

4.2.3.3 Staggered Tube Bank

The geometric layout of a generic staggered tube bank[147, 148, 149, 150, 151] is shown in Fig. 4.29a.

The Chimera overset mesh of the complete computational domain is shown in Fig. 4.29b. The tubes

are represented with cubic, $N_g = 3$, polynomial mappings and the background mesh uses linear, $N_g = 1$, polynomial mappings. A closeup of the Chimera overset region is shown in Fig. 4.29c. Both the inner and outer edges of the tubes are used to cut holes in the background mesh. Multiple cutting groups are needed to create the correct hole in the background mesh because the cutting boundaries overlap. The overset mesh before the hole is cut is shown in Fig. 4.30a. The hole cutting groups are shown in Figs. 4.30b through 4.30g. In each figure the cutting boundaries are indicated in green. The first cutting group is a Solid cutting group made up of the wall boundaries of the tubes. This group is used to cut both the background grid, as shown in Fig. 4.30b, as well as the tube grids, as shown in Fig. 4.30c. The subsequent cutting groups, denoted as Field 1 through Field 4 and shown in Figs. 4.30d through 4.30g, use the outer boundaries of the tube meshes to cut holes in the background mesh. The final hole is shown in Fig. 4.30h

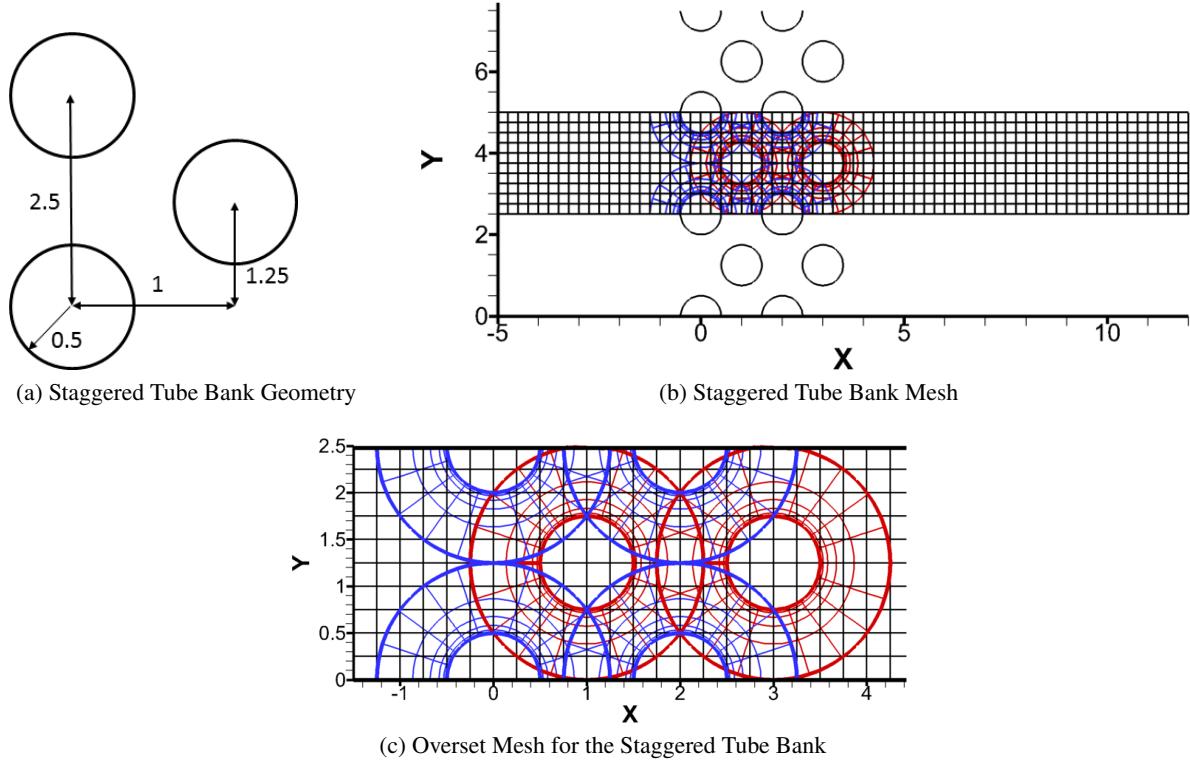
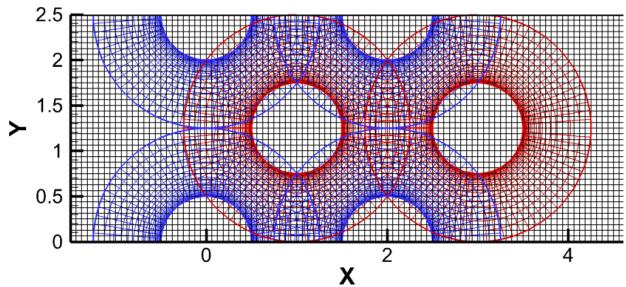
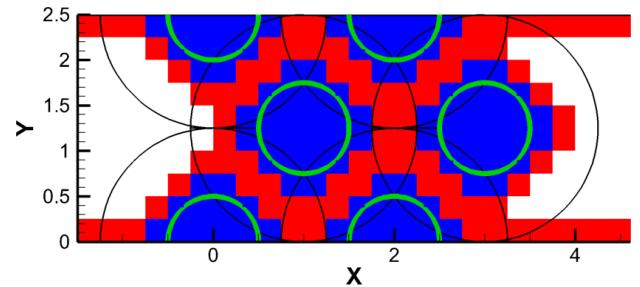


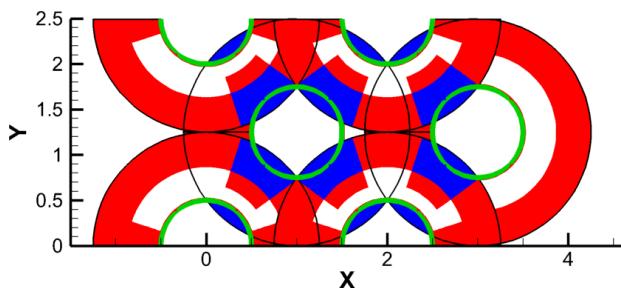
Figure 4.29: Staggered Tube Bank Overset Mesh



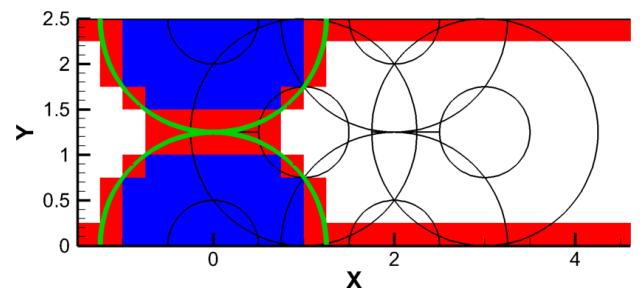
(a) Staggered Tube Mesh with 5 Points per Cell before Hole Cutting



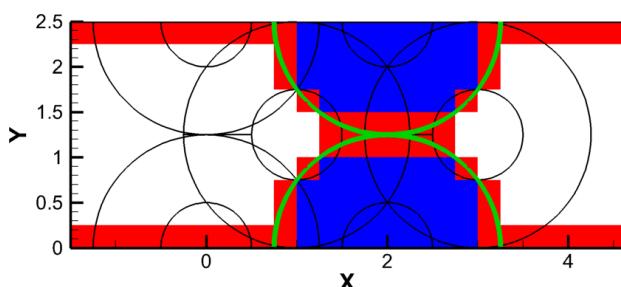
(b) Holes Cut in the Background Mesh using the Tube Wall Group



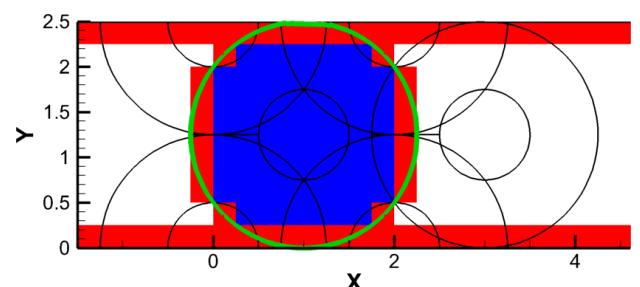
(c) Holes Cut in the Tube Meshes using the Tube Wall Group



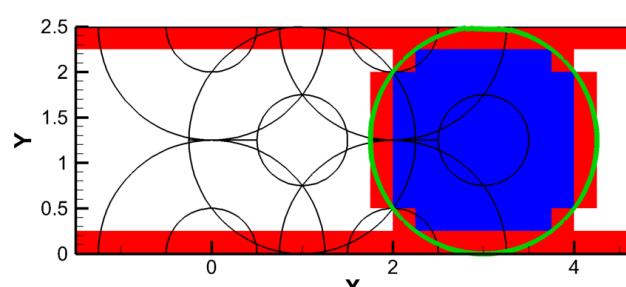
(d) Holes Cut in the Background Mesh using Field 1 Group



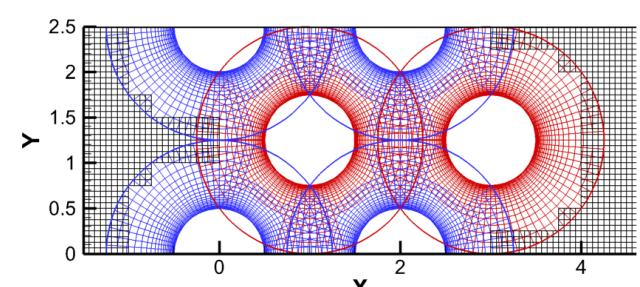
(e) Holes Cut in the Background Mesh using Field 2 Group



(f) Holes Cut in the Background Mesh using Field 3 Group



(g) Holes Cut in the Background Mesh using Field 4 Group



(h) Staggered Tube Mesh with 5 Points per Cell after Hole Cutting

Figure 4.30: Hole Cutting Groups for the Staggered Tube Bank (Green Edges are the Cutting Boundaries)

4.2.4 Summary

The small stencil of the DG-Chimera scheme allows for a simpler hole cutting process compared to high-order Finite Volume and Finite Difference schemes. The Direct Cut method[142, 143, 144] has been extended to account for the curvature of cutting surfaces. Cutting surfaces are organized into groups to resolve issues associated with multiple cutting surfaces overlapping each other. Each cutting group cuts an individual hole, and the final hole is the union of the holes cut by each cutting group. A numerical solution procedure was presented to find the minimum distance between a cutting surface and a cell. The minimum distance is used to flag the cell as a Field or a Hole cell. The numerical solution process works regardless of the orders of the geometric mapping polynomials for the cutting segment and cell, and is extensible to three-dimensions. The cutting method has been demonstrated on a set of meshes that involve mixed orders of the polynomial mappings amongst the grids.

Future work is required to resolve the issue of multiple local minimum solutions when searching for the minimum distance between a curved segment and a curved cell. In addition, a convenient set of cutting surfaces to produce holes is not always readily available in a complex configuration. Future work will include exploring similar techniques to those used in Pegasus5[42, 43] where cell quality is used to minimize overlapping regions.

4.3 Verification Test Cases

A set of verification problems are used to asses the proposed discontinuous Galerkin Chimera (DG-Chimera) scheme outlined in this chapter. A set of scalar partial differential equations are used to verify that DG-Chimera scheme retains the proper order of accuracy. The DG-Chimera implementation is also verified to obtain the correct order of accuracy for systems of partial differential equations using the inviscid Euler equations. The DG-Chimera method is also demonstrated to be suitable for subsonic, transonic, and supersonic internal and external flows. Finally the DG-Chimera scheme is applied to a set of viscous flow problems with multiple overlapping grids that conform to the geometry.

4.3.1 Scalar Equations

The set of two-dimensional scalar equation verification problems in Section 3.4.1 are used to verify the implementation of the DG-Chimera artificial boundary scheme. The order of accuracy is computed by

solving the linear Poisson equation, the linear advection and diffusion equation, and Burger's equation using a series of meshes with a zonal interface and three series of Chimera meshes. The zonal mesh consists of two grids with a vertical artificial boundary as shown for the coarsest grid in Fig. 4.31. The series of Zonal meshes are generated by doubling the number of cells in each direction five times. The coarsest Chimera meshes for the three series of Chimera meshes are shown in Fig. 4.32. The grid in the center of the meshes shown in Figs. 4.32a through 4.32c overlaps the background grids by 25%, 50%, and 75% of a cell width, respectively. The center grid is reduced in area such that the percentage of cell width overlap is maintained as the cell count increases. The L^2 -norm (given in Eq. 3.105) used to compute the errors to assess the order of accuracy does not account for the overlap in the Chimera meshes, and, hence, overlapping regions are integrated twice.

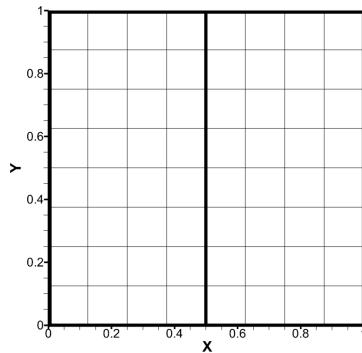


Figure 4.31: Coarsest Zonal Mesh for Scalar Equation Order of Accuracy Calculations

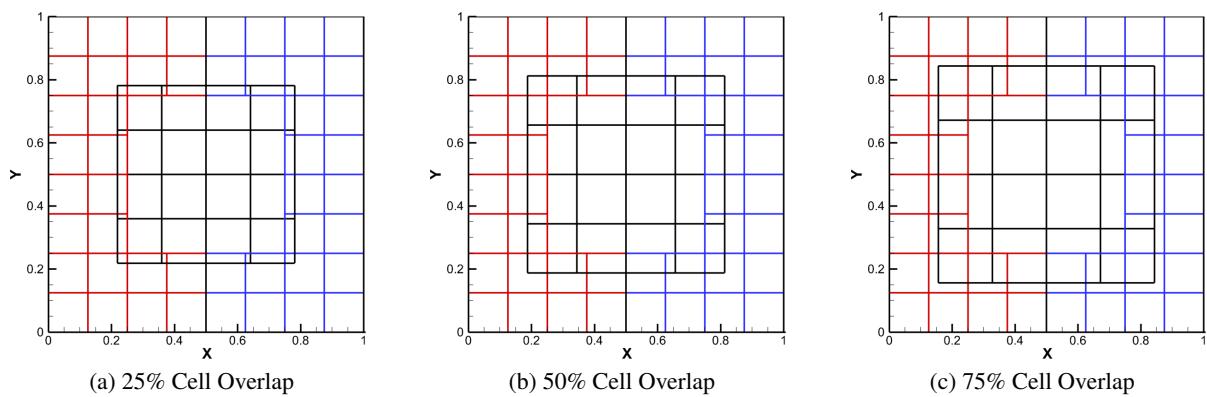


Figure 4.32: Coarsest Chimera Meshes for Scalar Equation Order of Accuracy Calculations

4.3.1.1 Linear Poisson Equation

Errors computed from solutions to the two-dimensional linear Poisson equation with the source term outlined in Section 3.4.1.1 using the Zonal and Chimera meshes with increasing cell refinement are shown in Fig. 4.33. The expected order of accuracy of $N + 1$ is obtained with the Zonal mesh as indicated by the slope of the lines. This implies that the artificial boundaries are properly implemented and the artificial boundary scheme reduces to the interior scheme for a Zonal boundary. The solutions computed on the coarsest Zonal mesh (See, Fig. 4.34) also agree well with solutions computed with a single grid (See, Fig. 3.10). However, the artificial boundaries introduce errors significant enough to drop the order of accuracy of the solutions computed using the Chimera meshes with $N = 1$ to first order accuracy (See, Fig. 4.33b). The correct order of accuracy is obtained for $N = 2$ and $N = 4$, but the order of accuracy for $N = 3$ is also degraded. This loss in order of accuracy is likely a result of the implementation of the BR2 scheme where artificial boundaries are treated similar to a Dirichlet boundary condition. Despite the degradation in the order of accuracy, the errors are consistent, i.e., they vanish with increased order of approximation and/or mesh refinement. Furthermore, no real trend between the different extent of overlap is discernible. Visually, the solutions shown in Fig. 4.35 that are computed using the coarsest Chimera mesh with 25% cell overlap agree well with the solutions computed using the single grid and Zonal mesh shown in Figs. 3.10 and 4.34, respectively.

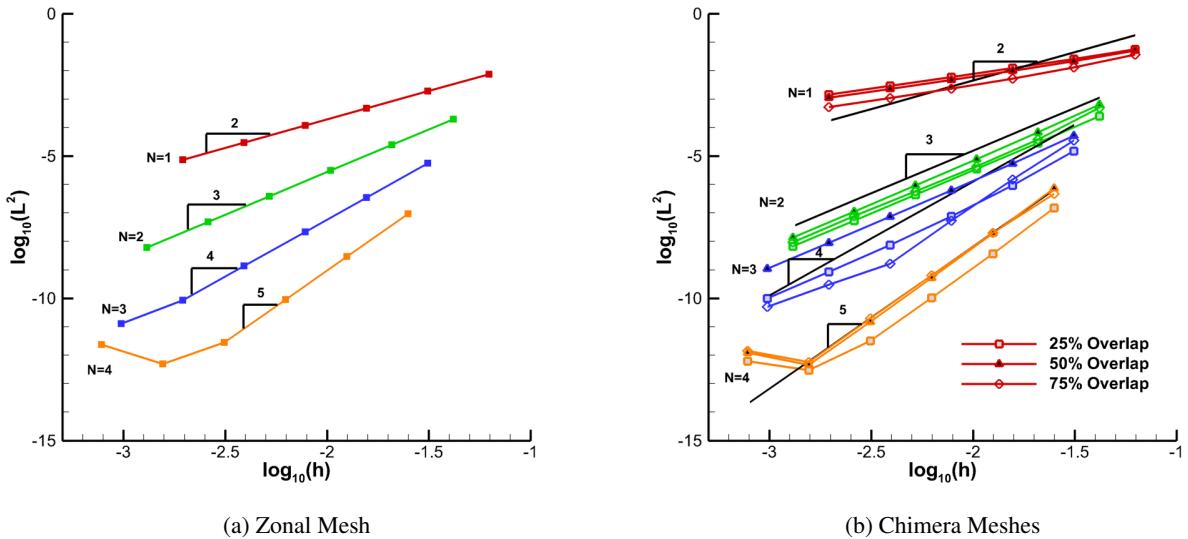


Figure 4.33: Convergence Rates for the Linear Poisson Equation using Zonal and Chimera Meshes

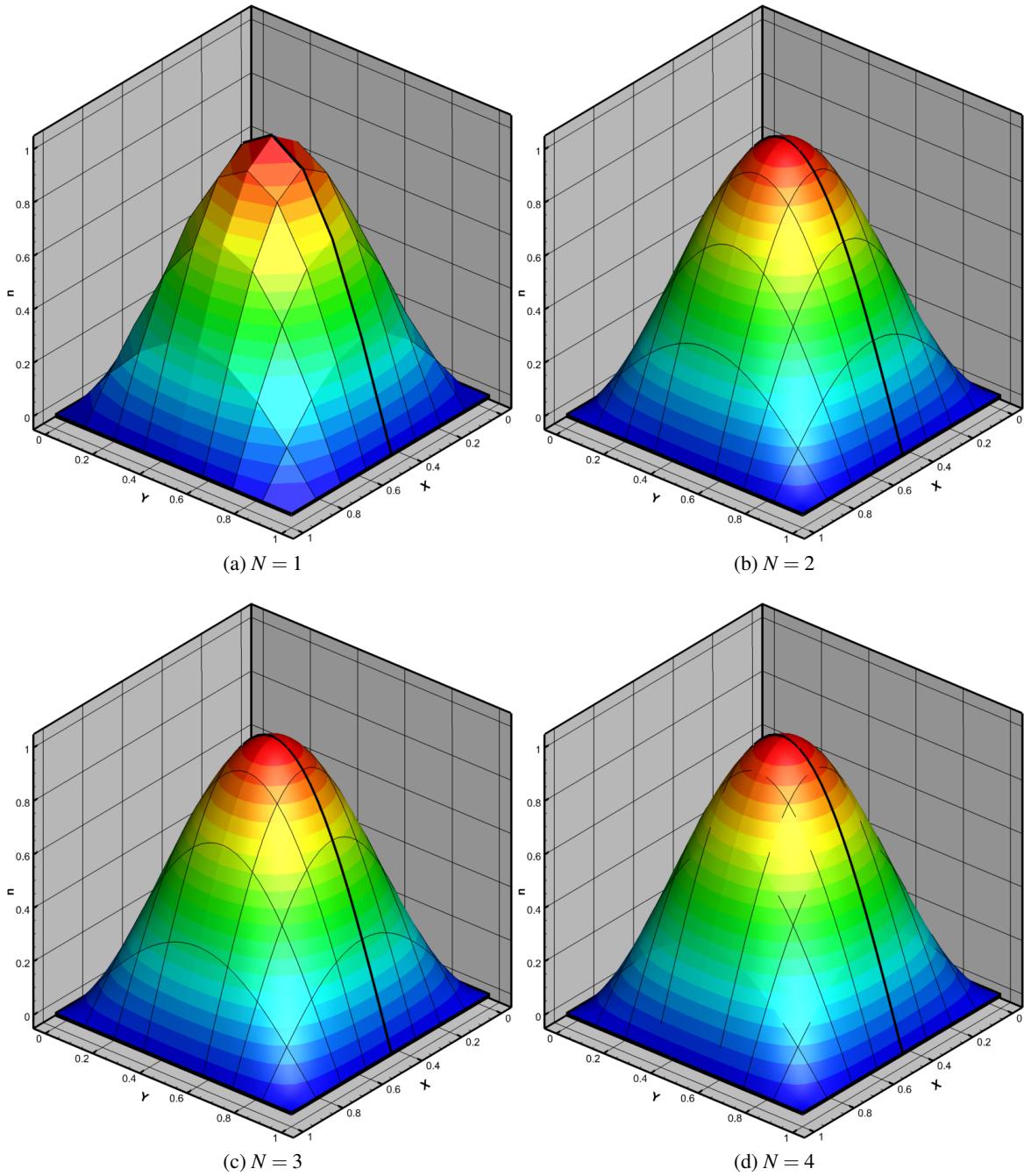


Figure 4.34: Two-Dimensional Linear Poisson Equation Solutions with Zonal Mesh with 8×8 Cells

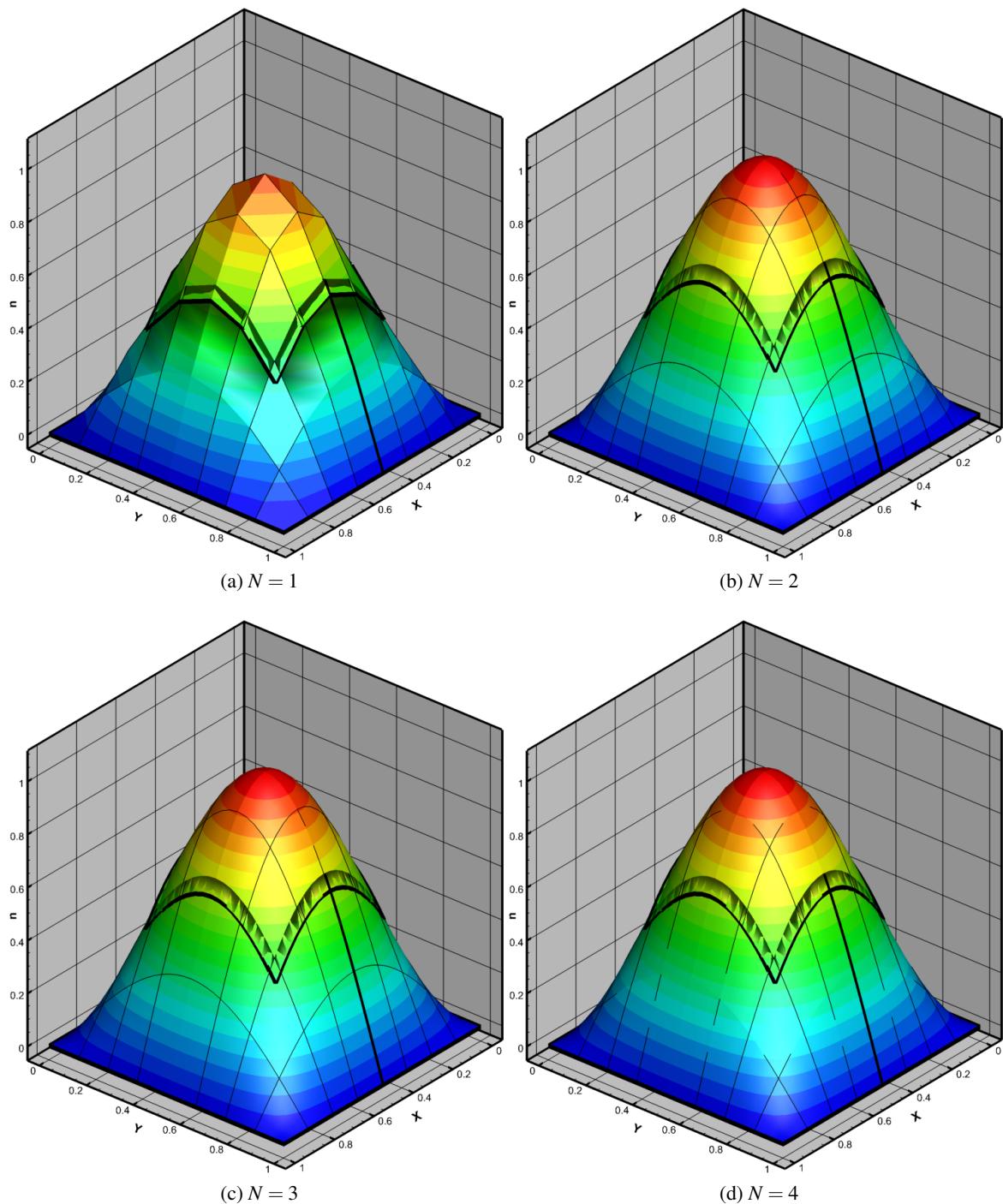


Figure 4.35: Two-Dimensional Linear Poisson Equation Solutions with Chimera Mesh with 25% Overlap and 8×8 Cells

4.3.1.2 Linear Advection Diffusion Equation

Numerical solutions to the two-dimensional linear advection diffusion equation with the boundary conditions and parameters given in Section 3.4.1.3 are computed using the Zonal and Chimera meshes. The error for increasing mesh resolution for the Zonal and Chimera meshes is shown in Fig. 4.36. The expected order of accuracy of $N + 1$ is achieved using the series of Zonal meshes as indicated by the slope of the lines shown in Fig. 4.36a. Unlike the solutions to the linear Poisson equation, the errors computed from the solutions obtained using the Chimera meshes exhibit the expected order of accuracy of $N + 1$; though there is a deficit in the order of accuracy for $N = 1$ with the finer mesh resolutions. Notably, little variation is observed in the errors computed from the solutions obtained using the Chimera meshes with the different extent of overlap. The solutions shown in Figs. 4.37 and 4.38 that are computed using the series of Zonal and Chimera meshes, respectively, agree well with the single grid solutions shown in Fig. 3.18.

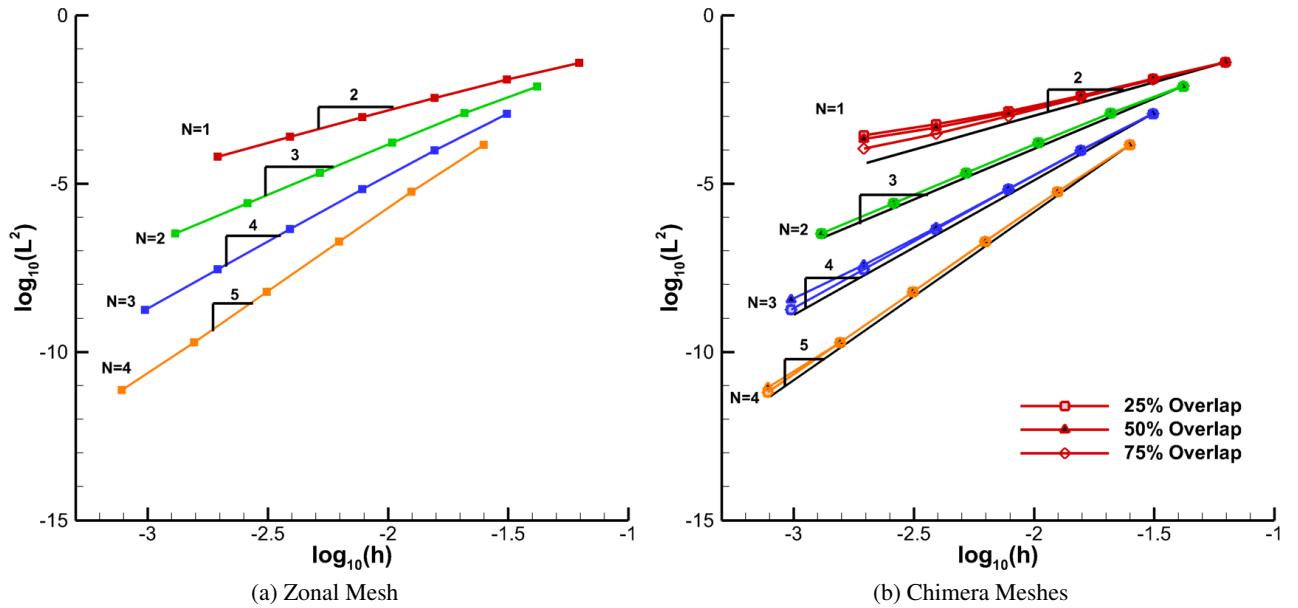


Figure 4.36: Convergence Rates for the Linear Advection Diffusion Equation using Zonal and Chimera Meshes

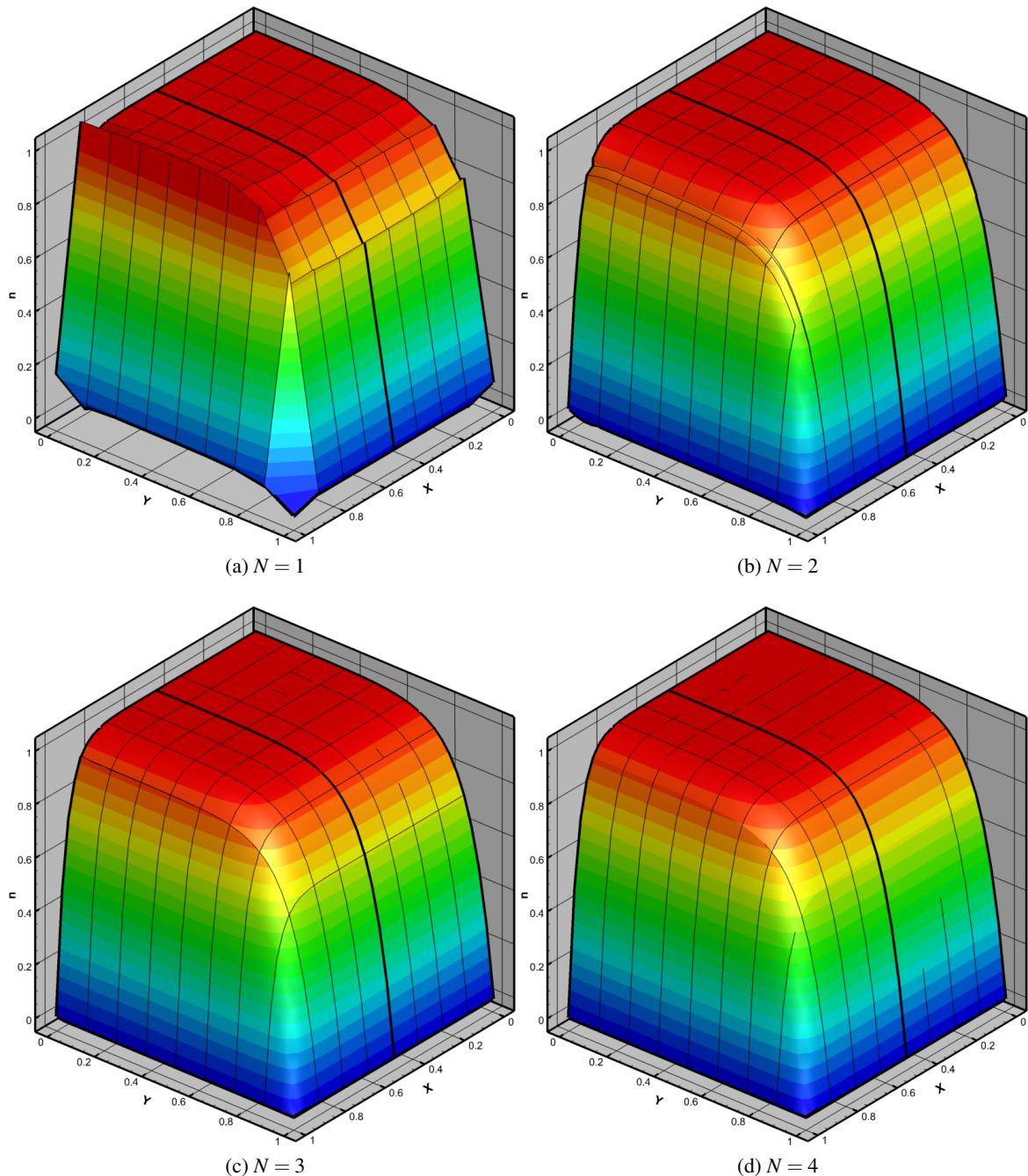


Figure 4.37: Two-Dimensional Linear Advection Diffusion Equation Solutions with a Zonal Mesh with 8×8 Cells

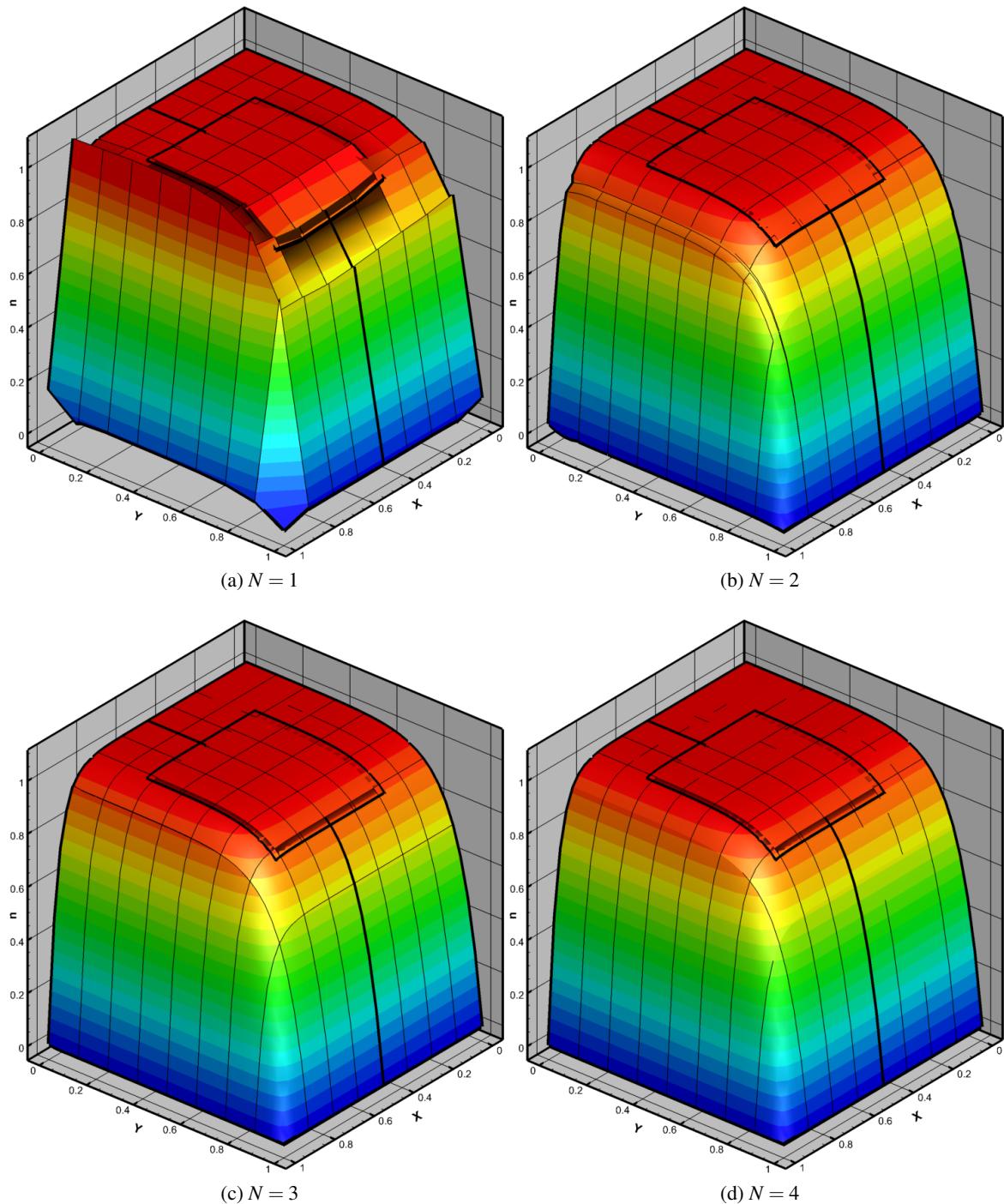


Figure 4.38: Two-Dimensional Linear Advection Diffusion Equation Solutions with a Chimera Mesh with 25% Overlap and 8×8 Cells

4.3.1.3 Burger's Equation

The errors computed from numerical solutions to Burger's equation with the parameters outlined in Section 3.4.1.4 using the Zonal and Chimera meshes are shown in Fig. 4.39. The expected order of accuracy of $N + 1$ is achieved using the Zonal mesh as well as the Chimera meshes as indicated by the slope of the lines in Fig. 4.39. Similar to the linear advection diffusion calculations, there is a deficit in the order of accuracy for $N = 1$ and $N = 3$ with the Chimera meshes at the finer mesh resolutions. Only a small difference in the error with $N = 2$ and $N = 4$ is observable between the three Chimera meshes with differing extents in the overlap. There is a larger disparity in the errors for $N = 1$ and $N = 4$ between the Chimera meshes with differing extents in overlap. The solutions computed with the Chimera mesh with 75% overlap are the lowest, which is consistent with the errors computed from solutions to the linear Poisson equations shown in Fig. 4.33b. Hence, it is likely that the spread in the errors for $N = 1$ and $N = 3$ shown in Fig. 4.39b are a result of the discretization of the diffusive terms in Burger's equation. The solutions shown in Figs. 4.40 and 4.41 visually agree well with the single grid solutions shown in Fig. 3.22.

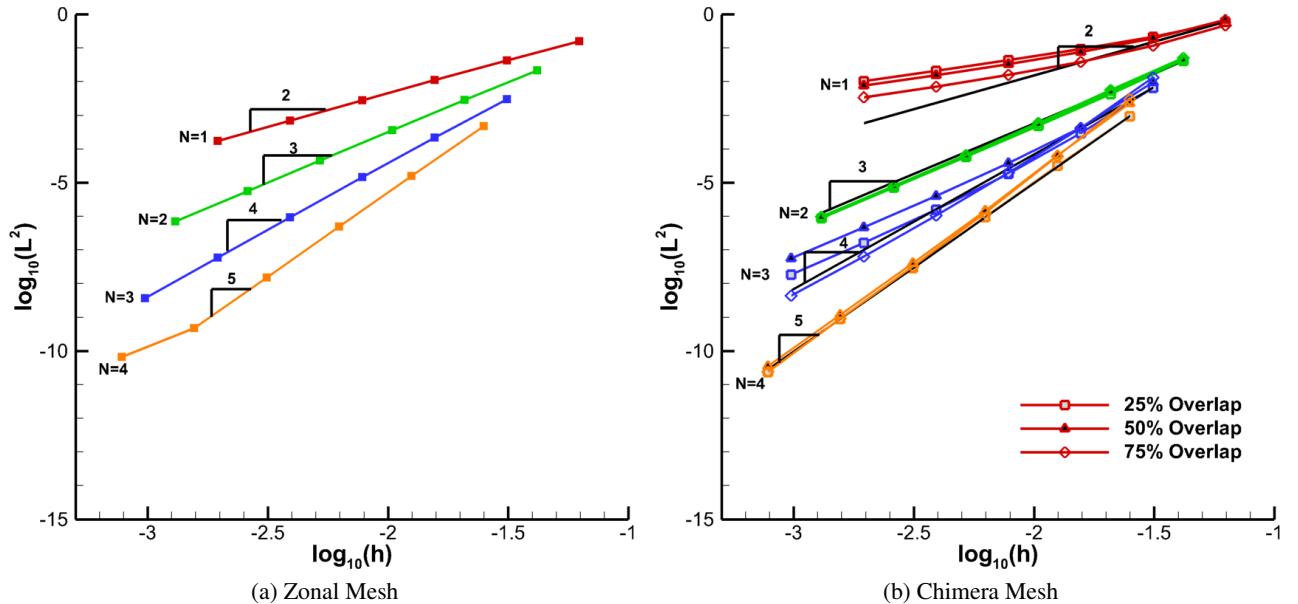


Figure 4.39: Convergence Rates for Burger's Equation using Zonal and Chimera Meshes

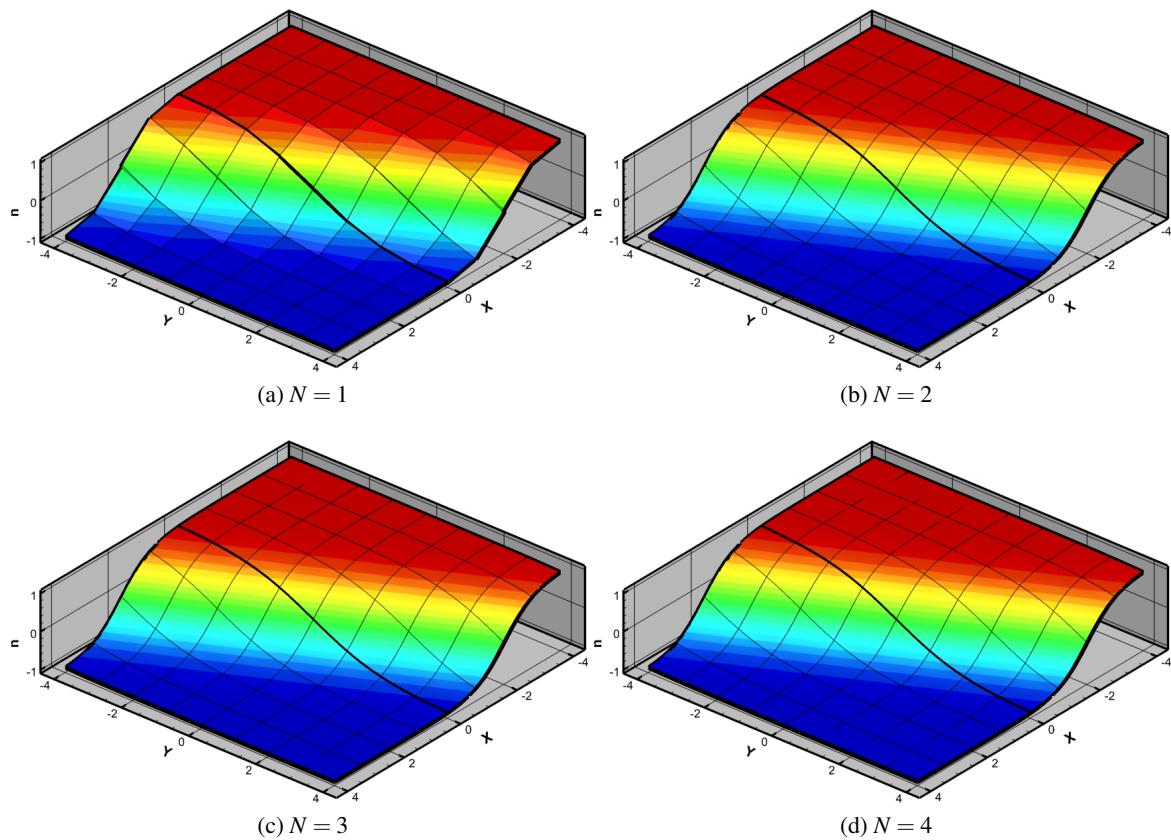


Figure 4.40: Two-Dimensional Burger's Equation Solutions with 8×8 Cells

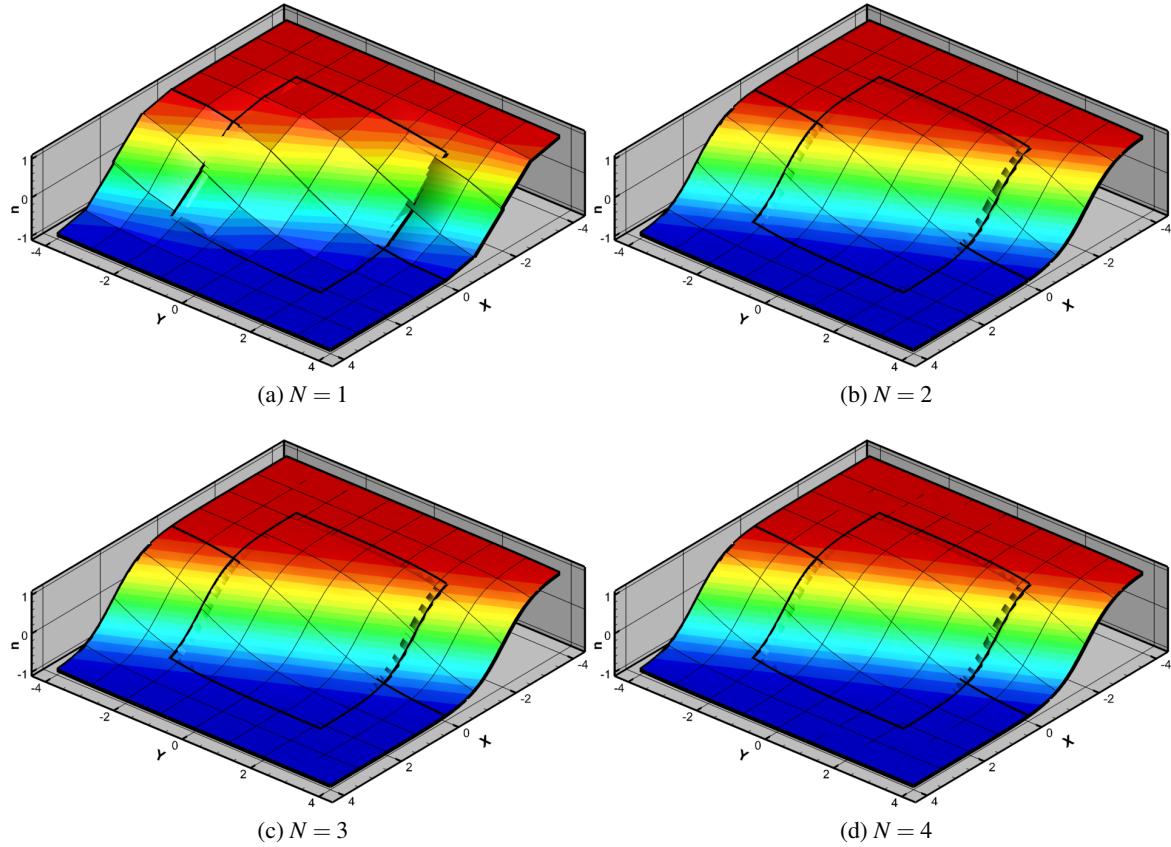


Figure 4.41: Two-Dimensional Burger's Equation Solutions with Chimera Mesh with 25% Overlap and 8×8 Cells

4.3.2 Inviscid Flow

This section presents results of applying the DG-Chimera scheme to a selection of inviscid flow problems: a subsonic channel flow with a Gaussian smooth bump, a turbomachinery cascade in subsonic flow, a transonic channel flow with a 10% circular arc, a supersonic normal shock in a diffuser, the SKF 1.1 airfoil[146] in subsonic and transonic flow, and a circular cylinder in supersonic flow. These flow problems demonstrate that the DG-Chimera scheme is applicable for both internal and external flow problems ranging from subsonic to supersonic flows. The channel flow with a Gaussian smooth bump is used to characterize the numerical approximations associated with the artificial boundaries. A small error in mass flux is observed similar to traditional finite volume and finite difference Chimera schemes that rely on a discrete interpolation of the conservative variables[152, 153, 154, 155, 156]. However, calculations of inviscid channel flow with

a smooth bump are used here to demonstrate that the DG-Chimera scheme maintains the proper order of accuracy despite these errors. This has also been demonstrated for finite volume Chimera schemes based on discrete interpolation[157]. The mass flux errors are also consistent, i.e., they go to zero with mesh refinement and/or increase in the order of the polynomial approximation. The turbomachinery cascade blade demonstrates that the artificial boundaries do not introduce significant errors when applied to internal flows with a high degree of turning. The transonic channel flow with a 10% bump and super-sonic diffuser flow with a shock demonstrate that the DG-Chimera scheme is able to capture shocks that cross the artificial boundaries in an internal flow. Flow fields computed using the SKF 1.1 airfoil demonstrate that the DG-Chimera scheme is applicable to external sub-sonic and transonic flows. The Mach 2 cylinder is used to demonstrate that the artificial boundaries do not introduce significant errors when a strong shock crosses the artificial boundaries. These flow problems were selected specifically because they can be meshed with a single grid. The single grid solution is then used as a reference for a comparison with a Chimera mesh with comparable grid resolution. The results demonstrate that the flow fields computed using the Chimera meshes are nearly identical to the flow fields computed using the single grids for $N \geq 1$. Two additional flow problems are presented: the SKF 1.1 airfoil with a flap and an isentropic convecting vortex. The SKF 1.1 airfoil with a flap is a more complex geometry that cannot be meshed with a single structured mesh and is more representative of the traditional use of Chimera meshes. The convecting vortex flow problem demonstrates that the high-order discretization is better able to maintain the vortex pressure deficit relative to a lower-order discretization with a given number of degrees of freedom. Finally, the DG-Chimera method is applied to a three-dimensional inviscid flow problem.

Cubic and quartic polynomial expansions are used for grid cells that represent the geometry. The use of curved elements to represent geometry was shown to be necessary by Bassi and Rebay[14] for high-order of accuracy.

4.3.2.1 Gaussian Smooth Bump

Channel flow with a Gaussian smooth bump[45] is used to verify the solver order of accuracy with Chimera artificial boundaries, as well as to assess errors introduced by using GQ integration in Eq. 4.2 that spans multiple donor cells. The computational domain of the channel is defined in Fig. 4.42. Slip wall boundary conditions are imposed by enforcing $\vec{V} \cdot \vec{n} = 0$ on the upper and lower boundaries. The pressure for the slip wall boundary conditions is the pressure from the interior cell evaluated on the wall. The left inflow

boundary specifies total pressure and temperatures corresponding to $M_\infty = 0.5$, as well as a zero flow angle. A constant back pressure is applied to the right outflow boundary.

The computational domain is divided in two, an upstream and downstream domain with an interface at $x = 0$. These grids are used to assess the number of GQ nodes required to evaluate the integral in Eq. 4.2. Both the upstream and downstream domains are meshed with uniform quadrilateral grids with a quartic, $N_g = 4$, polynomial mapping. These are Zonal meshes[140] since the upstream and downstream grids do not overlap. The downstream grids have between 2 through 8 times as many cells in the vertical direction as the upstream grid. The meshes are labeled, 2Y through 8Y to denote the ratio of cells between the downstream and upstream grids and are illustrated for the coarsest upstream grid in Fig. 4.43. The entropy error, defined as

$$\text{Entropy Error} = \sqrt{\frac{\int \left(\frac{\frac{p}{\rho^\gamma} - \frac{p_\infty}{\rho_\infty^\gamma}}{\frac{p_\infty}{\rho_\infty^\gamma}} \right)^2 d\Omega}{\int d\Omega}}, \quad (4.14)$$

computed using the zonal meshes for increasing mesh resolution and order of approximation are shown with three different GQ node counts in Fig. 4.44. The entropy error shifts horizontally to the left as the vertical cell count ratio increases due to the increase in degrees of freedom. However, despite the sudden change in cell size, the correct order of accuracy of $N + 1$ in the entropy error is maintained regardless of the ratio in vertical cell count or the number of GQ nodes. The effect of the numerical approximations of the Zonal interface are characterized by the difference in mass flux between the inflow and outflow boundaries.

The mass flux at both boundaries is computed using the same fluxes used to impose the boundary conditions. The mass flux errors for the Zonal meshes are shown in Fig. 4.45. These results are used to assess the appropriate number of GQ nodes required to minimize the mass flux errors. Each plot shows the mass flux error for a given polynomial approximation, N , and GQ node count, N_{GQ} , with increasing cell refinement for the seven vertical cell count ratios. The order of the polynomial approximation increases down the rows of plots, and the GQ node count increases across the columns. In general, the mass flux error decreases with increase in the polynomial approximation and/or increase in GQ nodes. This behavior demonstrates that DG-Chimera scheme is consistent. There is less of a reduction in the mass flux error when going from $N_{GQ} = \lceil 3N/2 \rceil + 1$ to $N_{GQ} = 2N + 1$ compared to increasing the GQ node count from $N_{GQ} = N + 1$ to $N_{GQ} = \lceil 3N/2 \rceil + 1$. Hence, a GQ node count of $N_{GQ} = \lceil 3N/2 \rceil + 1$ is deemed adequate based on this study.

No real trend is observed between the different vertical cell count ratios.

For the next comparison, the computational domain is divided into three parts: upstream and downstream domains, and a domain centered at $x = 0$. The meshes for the upstream and downstream domains have the same cell count, but the center domain has twice the number of cells in the vertical direction. Both Zonal and Chimera meshes are used to grid the three domains. The center grid in the Zonal mesh does not overlap the upstream and downstream domains, whereas the center grid in the Chimera mesh overlaps the upstream and downstream grids by a half cell width as shown in Fig. 4.46. The entropy errors computed from a single grid, the Zonal mesh, and the Chimera mesh are shown in Fig. 4.47. The entropy errors computed using the Zonal and Chimera meshes are comparable to those computed on the single grid. Most importantly, the proper $N + 1$ order of accuracy is observed on all three meshes. The mass flux error computed using both the Zonal and Chimera meshes also tends towards zero for increased grid resolution and increased order of approximation as shown in Fig. 4.48.

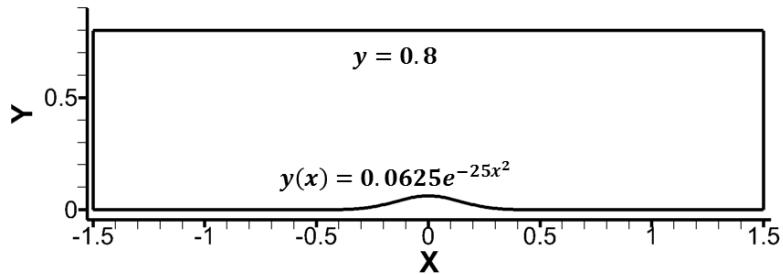


Figure 4.42: Smooth Bump Geometry

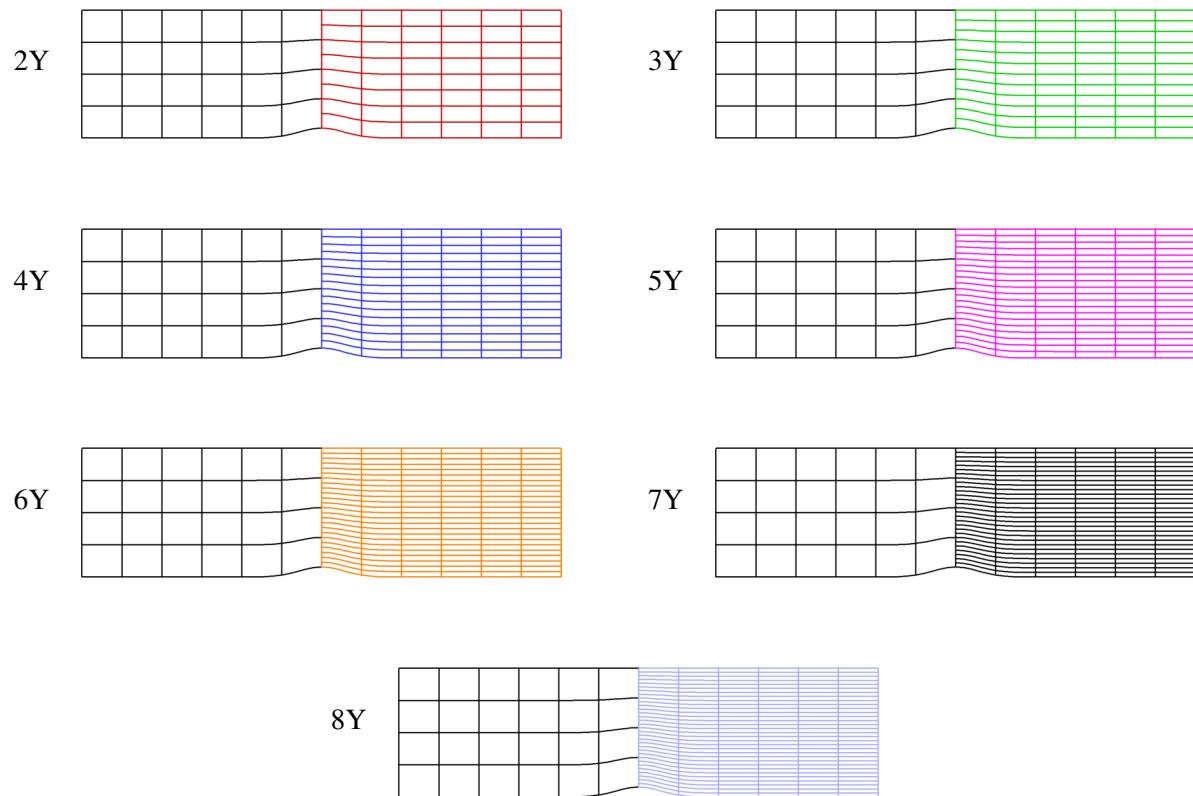


Figure 4.43: Smooth Bump Zonal Meshes

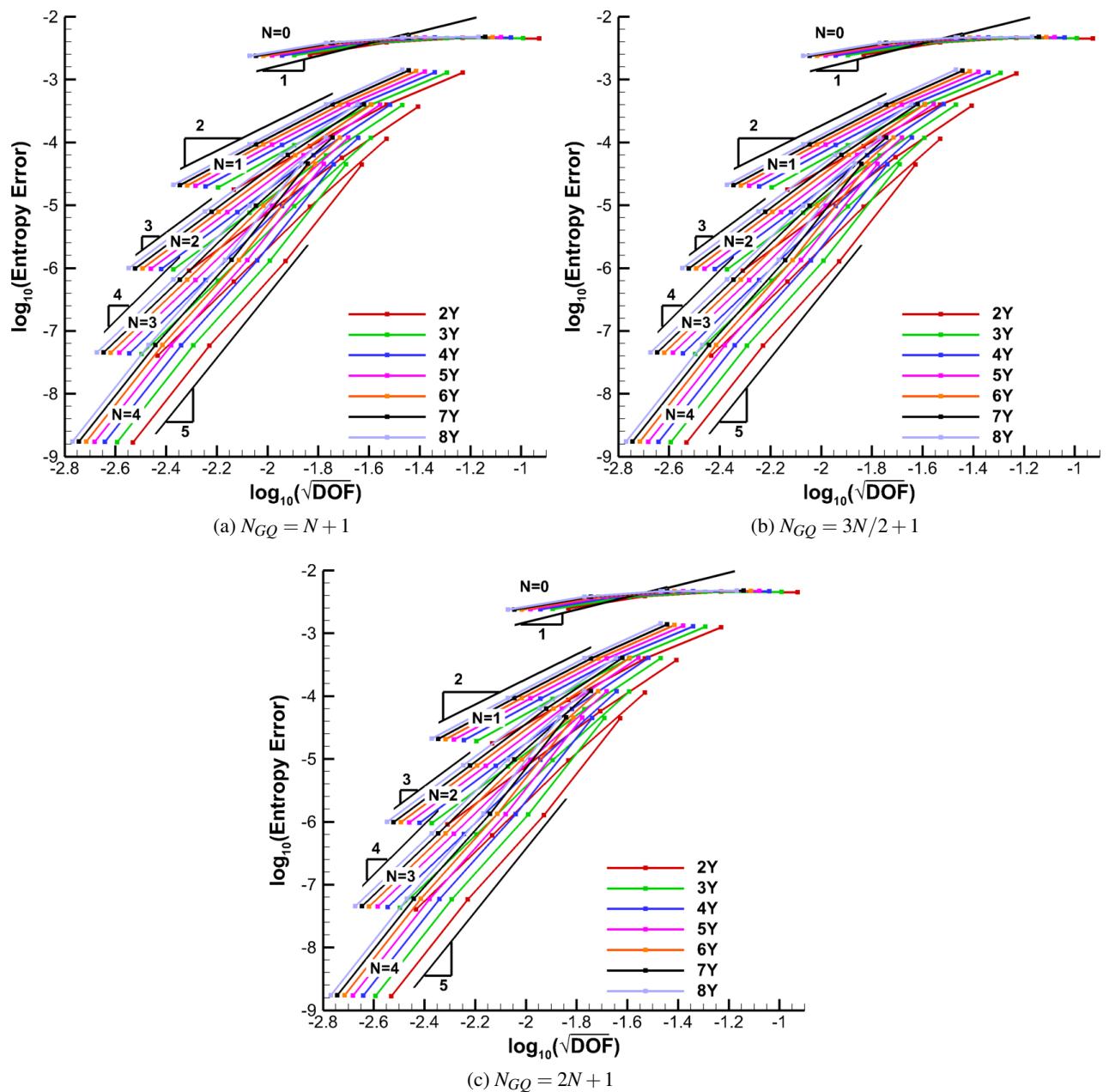


Figure 4.44: Observed Order of Accuracy using the Zonal Meshes with Different Number of Quadrature Nodes

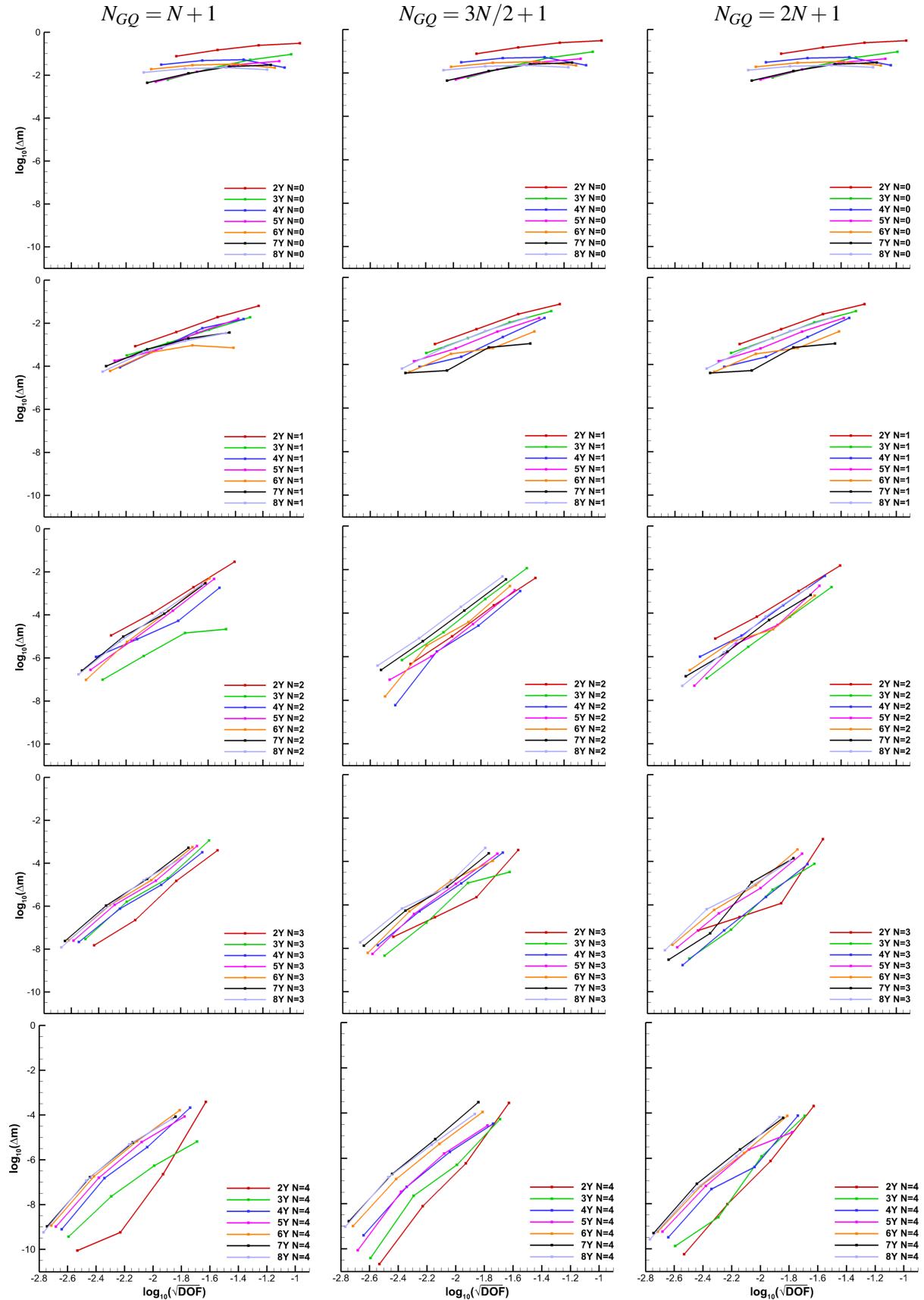
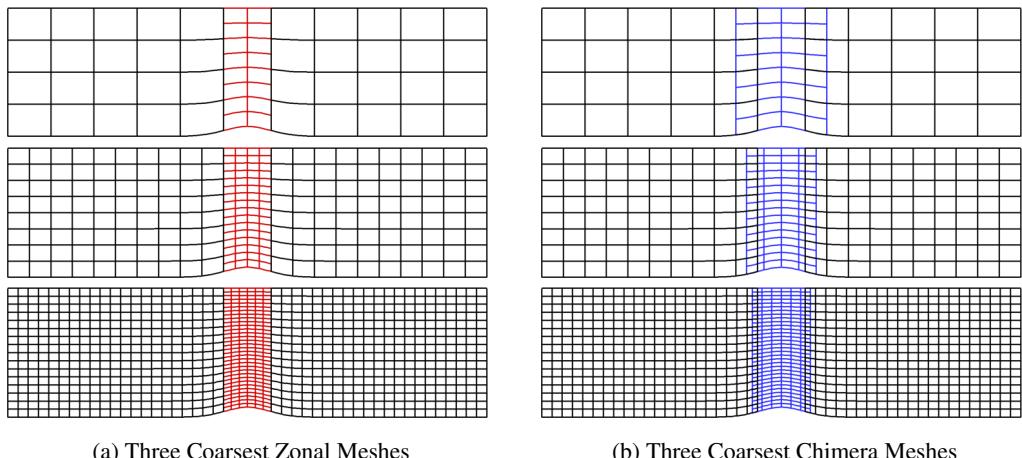


Figure 4.45: Smooth Bump Zonal Mesh Mass Flux Error



(a) Three Coarsest Zonal Meshes

(b) Three Coarsest Chimera Meshes

Figure 4.46: Smooth Bump with 3 Domains

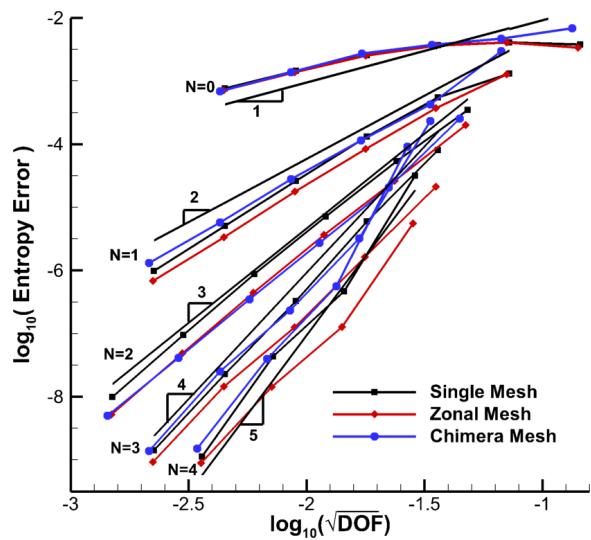


Figure 4.47: Smooth Bump Spatial Order of Accuracy with 3 Domains

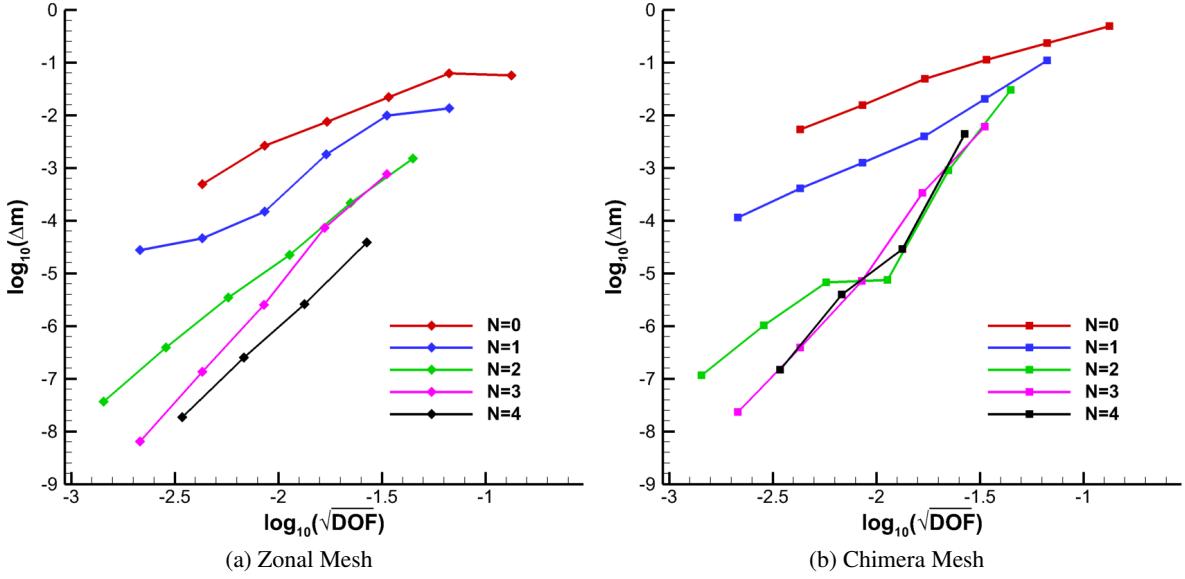


Figure 4.48: Smooth Bump with 3 Domains Mass Flux Error

4.3.2.2 Turbomachinery Cascade

A turbomachinery cascade is used to demonstrate the DG-Chimera scheme on an internal subsonic flow problem with a high degree of turning. The cascade blade geometry has a 35° leading edge metal angle and a 100° turning angle. The maximum thickness to chord ratio is 16.35% and the blade-to-blade spacing to axial chord ratio is 0.898, which yields a Zweifel[158, 159] loading coefficient of 1.06. The blade shape is defined by a quartic B-Spline thickness distribution and a cubic B-Spline meanline curvature distribution that is integrated twice to give the meanline. A zonal mesh with coincident nodes on all interfaces, shown in Fig. 4.49a, is used to compute a reference flow field. The Chimera mesh, shown in Fig. 4.49b, uses the grid from the zonal mesh that defines the surface of the cascade blade and a single background grid with the same point distribution along the upper and lower boundaries as the zonal mesh. The region of the background grid that would otherwise reside inside the cascade rotor has been excluded from the computational domain using the hole-cutting procedure. Note that the grid representing the blade only has two cells normal to the wall and the cell size differs significantly near the trailing edge as shown in Fig. 4.49c. A periodic boundary is used to connect the upper and lower boundaries of the computational domain. Total pressure and temperature corresponding to $M_\infty = 0.25$ and a ratio of velocity components $v/u = 0.5$ is enforced

at the inflow boundary to ensure the proper flow angle. A fixed static back pressure with a fixed outlet to inlet static pressure ratio of $P_{out}/P_{in} = 0.8853$ is imposed at the outflow boundary. The Cartesian force coefficients, mass averaged outflow angle, β , difference in mass flux between the inlet and exit boundaries, surface pressure coefficient, and pressure coefficient contours are given in Fig. 4.50 with increasing order of approximation. The Cartesian force coefficients and mass averaged outflow angle computed using the two meshes agree well for $N \geq 1$. The mass flux error for the zonal mesh are machine zero, and the mass flux error tends to decrease for the Chimera mesh with increase in the order of approximation. The mass flux error does increase slightly when the order of approximation increases from $N = 2$ to $N = 3$. This type of behavior is also observed in the mass flux errors computed on the Chimera grid for the inviscid channel flow with a smooth bump, (See, Fig. 4.48b), were the mass flux error with $N = 2$ is sometimes lower than with $N = 3$. Surface pressure coefficient and pressure coefficient contours computed using the two meshes agree well for $N = 1$, and are indistinguishable for $N \geq 2$.

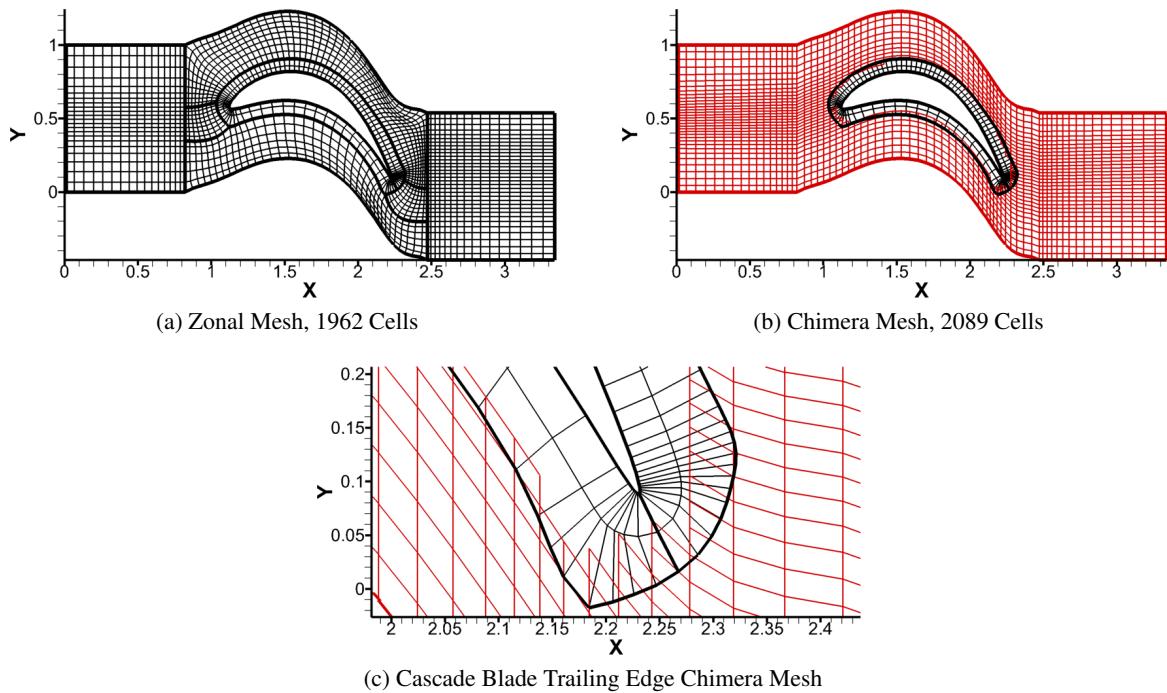


Figure 4.49: Turbomachinery Cascade Blade Meshes

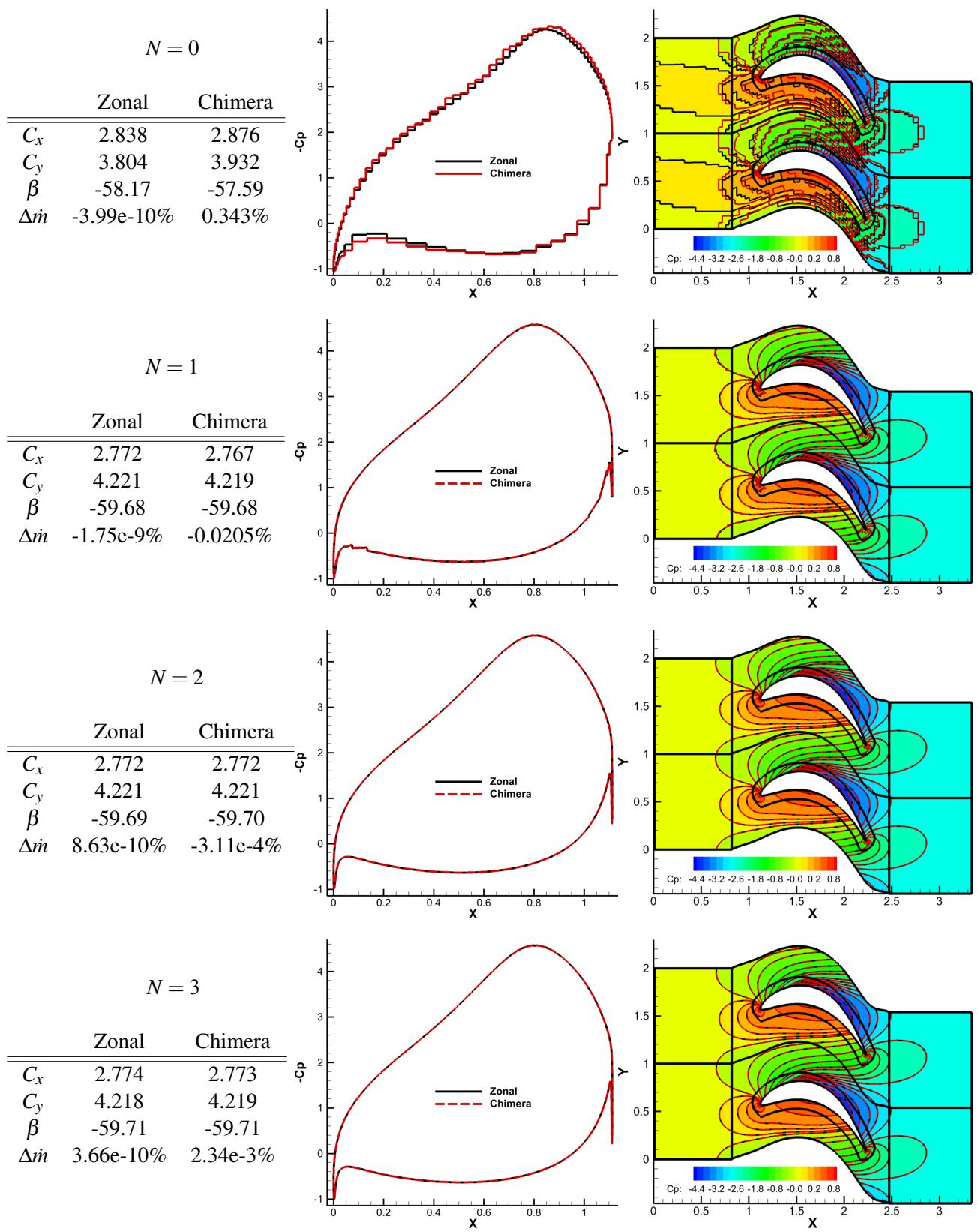


Figure 4.50: Turbomachinery Cascade Blade, ($M_\infty = 0.25$)

4.3.2.3 Channel Flow with 10% Circular Arc

This case demonstrates the DG-Chimera scheme on an internal transonic flow with a shock[160, 153, 156].

A single grid and a Chimera overset mesh for a channel flow with a circular arc on the lower wall are shown in Fig. 4.51. The circular arc has a unit length and extends 10% of the channel height. Total pressure and temperature boundary conditions are imposed at the inflow. A fixed back pressure condition is imposed on the right exit boundary. Zero mass flux through the upper and lower boundaries is enforced with a slip wall boundary condition. An inflow Mach number of $M_\infty = 0.675$ is chosen to produce a transonic shock on the downstream portion of the arc. The difference in mass flux, the lower surface pressure coefficient, and pressure coefficient contours are shown in Fig. 4.52 for increasing order of the approximating polynomial. The mass flux error tends to decrease with increased order of the polynomial approximation, though more gradually than the sub-sonic flows. In addition, for $N \geq 1$, the surface pressure coefficient and pressure coefficient contours agree well between the single grid and the Chimera mesh. Differences in the surface pressure can be attributed to the different topology of the two meshes. For the single grid, the grid lines align with the shock, whereas the grid lines on the surface of the arc for the Chimera mesh do not.

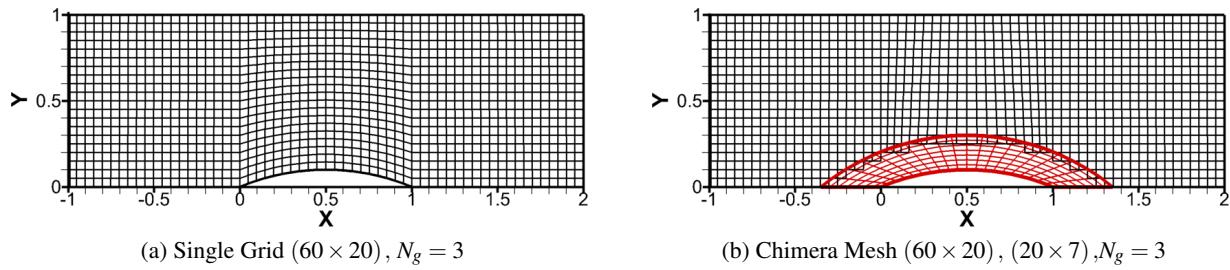


Figure 4.51: Channel with 10% Circular Arc Meshes

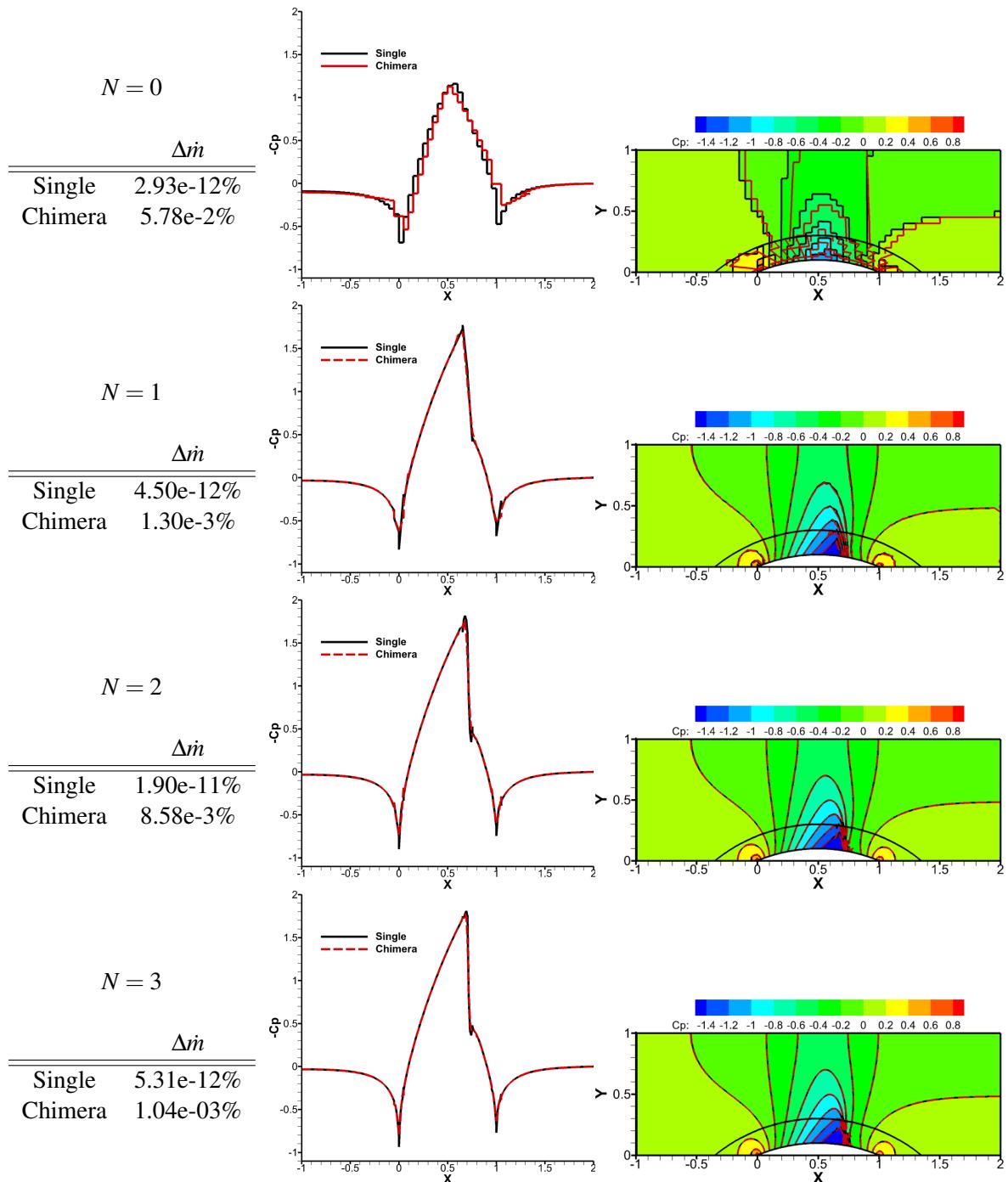


Figure 4.52: Channel with 10% Circular Arc, ($M_\infty = 0.675$)

4.3.2.4 Normal Shock in a Diffuser

A diffuser test case is used to demonstrate the DG-Chimera scheme on an internal supersonic flow with a normal shock. The single grid and Chimera overset mesh with linear cell mappings for the diffuser are

shown in Fig. 4.53. The diffuser expands at a 10° angle on the upper and lower surfaces. The grids at the inflow and outflow boundaries in the Chimera mesh are used to integrate the mass flux over a non-overlapping boundary. A supersonic inflow with $M_\infty = 1.1$ is imposed on the inflow boundary. A fixed back pressure computed from the normal shock equations[161] for a normal shock with an upstream Mach number of 1.1 is imposed on the outflow boundary. The mass flux error, surface pressure coefficient from the lower wall, and pressure coefficient contours are shown in Fig. 4.54. The mass flux error between the inflow and outflow is machine zero for the single grid, and tends to decrease as the order of the polynomial approximation increases. Aside from $N = 0$, the surface pressure and pressure contours agree well between the flow fields computed on the single grid and the Chimera mesh.

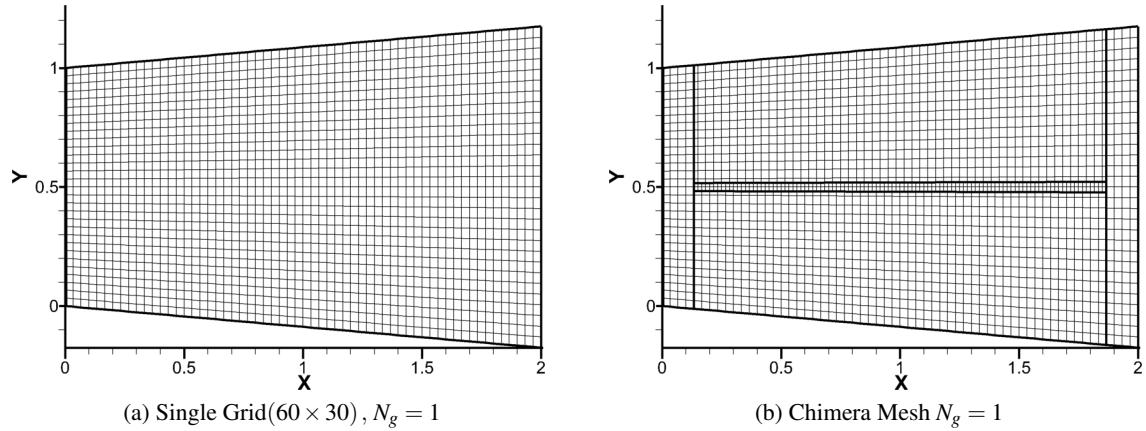


Figure 4.53: Diffuser Meshes

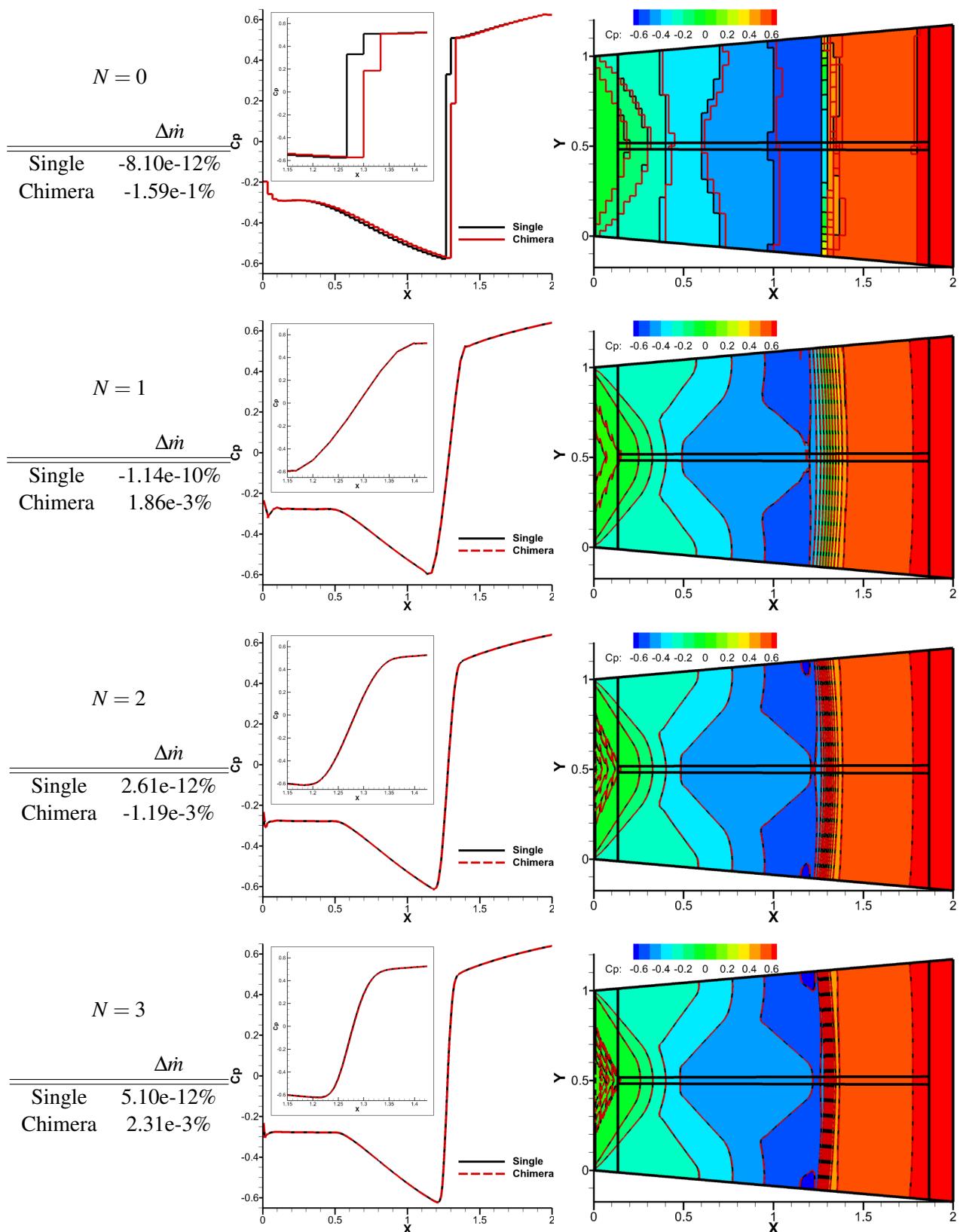


Figure 4.54: Normal Shock Pressure Coefficient

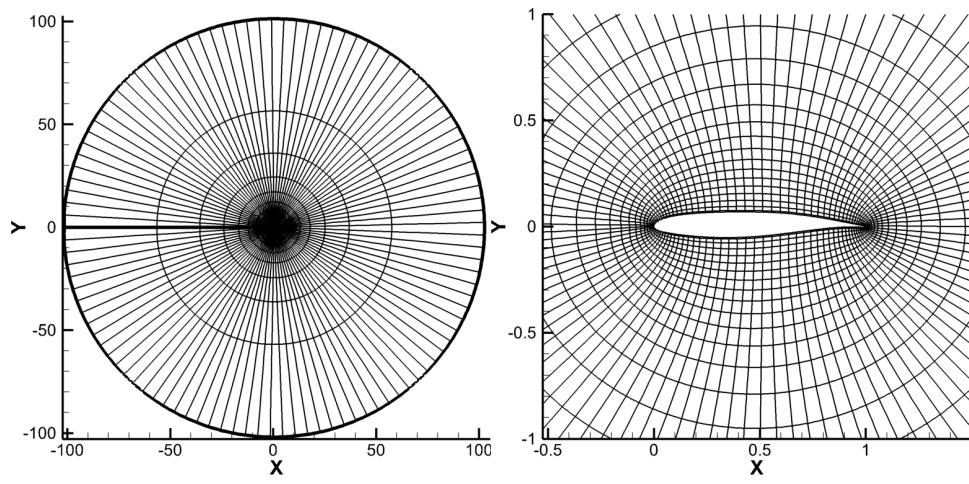
4.3.2.5 SKF 1.1 Airfoil

The SKF 1.1 airfoil is used to demonstrate the DG-Chimera scheme on both external subsonic flow as well as external transonic flow. The three meshes used to compute the flow about the SKF 1.1 airfoil[146] are shown in Fig. 4.55. The first mesh in Fig. 4.55a is a single O-grid with a cubic cell mapping. The second mesh (Fig. 4.55b) is a Chimera overset mesh that uses an O-grid with a cubic cell mapping to represent the airfoil, and a second O-grid with a linear cell mapping to establish the farfield boundary 100 chords away from the airfoil. The third mesh shown in Fig. 4.55c is also a Chimera grid. It uses the same grid to represent the airfoil as the O-grid Chimera mesh and a rectangular background grid with a hole for the airfoil that uses a linear cell mapping. The farfield boundary is located 100 chords away from the airfoil.

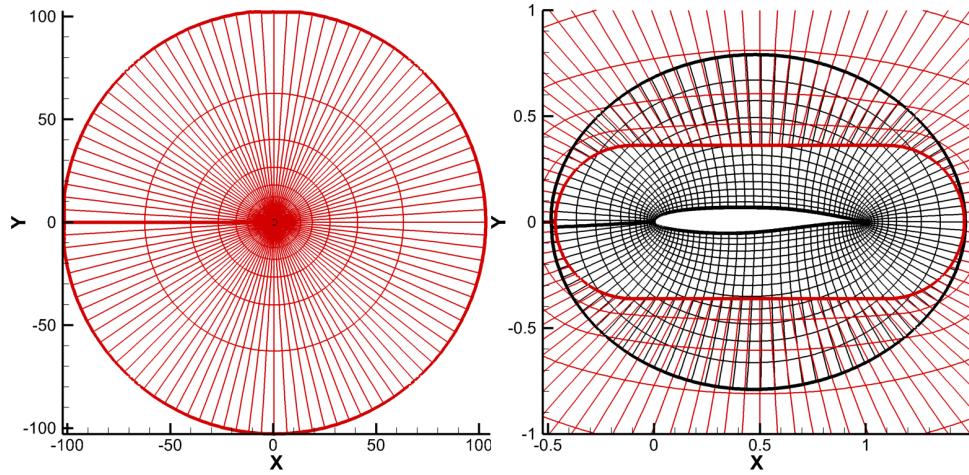
For all three meshes, a slip wall boundary condition is imposed on the surface of the airfoil and a Riemann invariant condition with an angle of attack $\alpha = 2.5^\circ$ is imposed at the farfield boundary. Two different flow fields are computed with the three meshes. The first flow field is subsonic with $M_\infty = 0.4$ and the second is transonic with a shock on the upper surface of the airfoil with $M_\infty = 0.76$. Figures 4.56 and 4.57 show lift and drag, mass flux error, surface pressure coefficient, and pressure coefficient contours for the subsonic and transonic flow fields respectively. Lift and drag computed using the two Chimera meshes also agrees well with the values computed using the single grid for $N \geq 1$. The tabulated mass flux error is the mass flux integral over the farfield boundary because this is an external flow. The mass flux error for the subsonic flow field on the single grid is machine zero for all orders of the polynomial approximation, and generally decreases as the order of the polynomial approximation increases for the two Chimera meshes. Similarly the surface pressure coefficient and pressure coefficient contours agree well between the Chimera meshes and the single grid for $N \geq 1$. Notably, for $N = 3$, the stagnation pressure at the trailing edge of the airfoil nearly reaches the value of the stagnation pressure at the leading edge.

For the transonic solution, the lift and drag coefficients computed using the Chimera meshes and the single grid again agree well for $N \geq 1$. The mass flux error is near machine zero for the single grid. The increase in the mass flux error relative to the subsonic solution is a result of using the approximation in the lifting operator in the discretization of the artificial viscosity on the artificial boundaries as described in section 4.1. The mass flux error generally decreases for the Chimera meshes as the order of the polynomial approximation increases. However, the mass flux error does not decrease as rapidly compared to the subsonic flow field. While the surface pressure coefficient and pressure coefficient contours agree well between

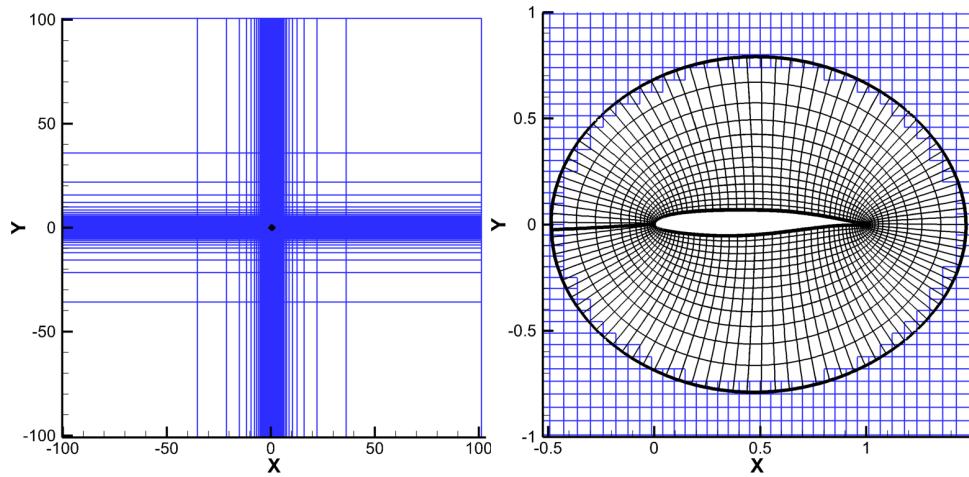
the flow fields computed on the three meshes for $N \geq 1$, small differences can be observed in the shock region. These differences can be attributed to the difference in cell size of the background grids in the two Chimera meshes and the artificial viscosity. The artificial viscosity is different on the three meshes as it is a direct function of the cell size. Away from the shock the surface pressure coefficient and pressure coefficient contours are indistinguishable between the meshes for $N \geq 2$.



(a) Single Grid (105×30), $N_g = 3$



(b) Chimera O-Grid Mesh (105×14), $N_g = 3$, (104×16), $N_g = 1$



(c) Chimera R-Grid Mesh (105×14), $N_g = 3$, (104×104), $N_g = 1$

Figure 4.55: SKF 1.1 Airfoil Meshes

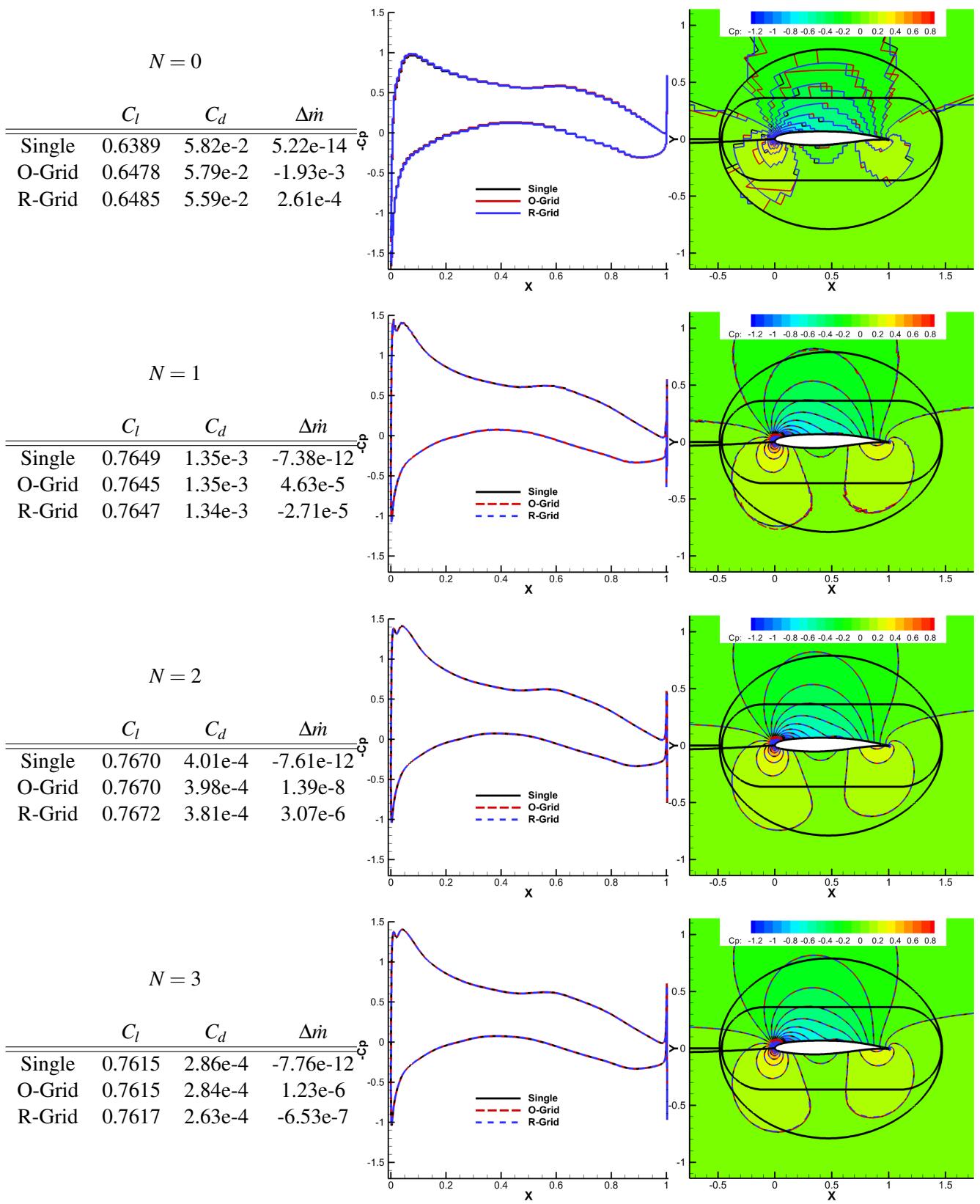


Figure 4.56: SKF 1.1 Airfoil, ($M_\infty = 0.4$, $\alpha = 2.5^\circ$)

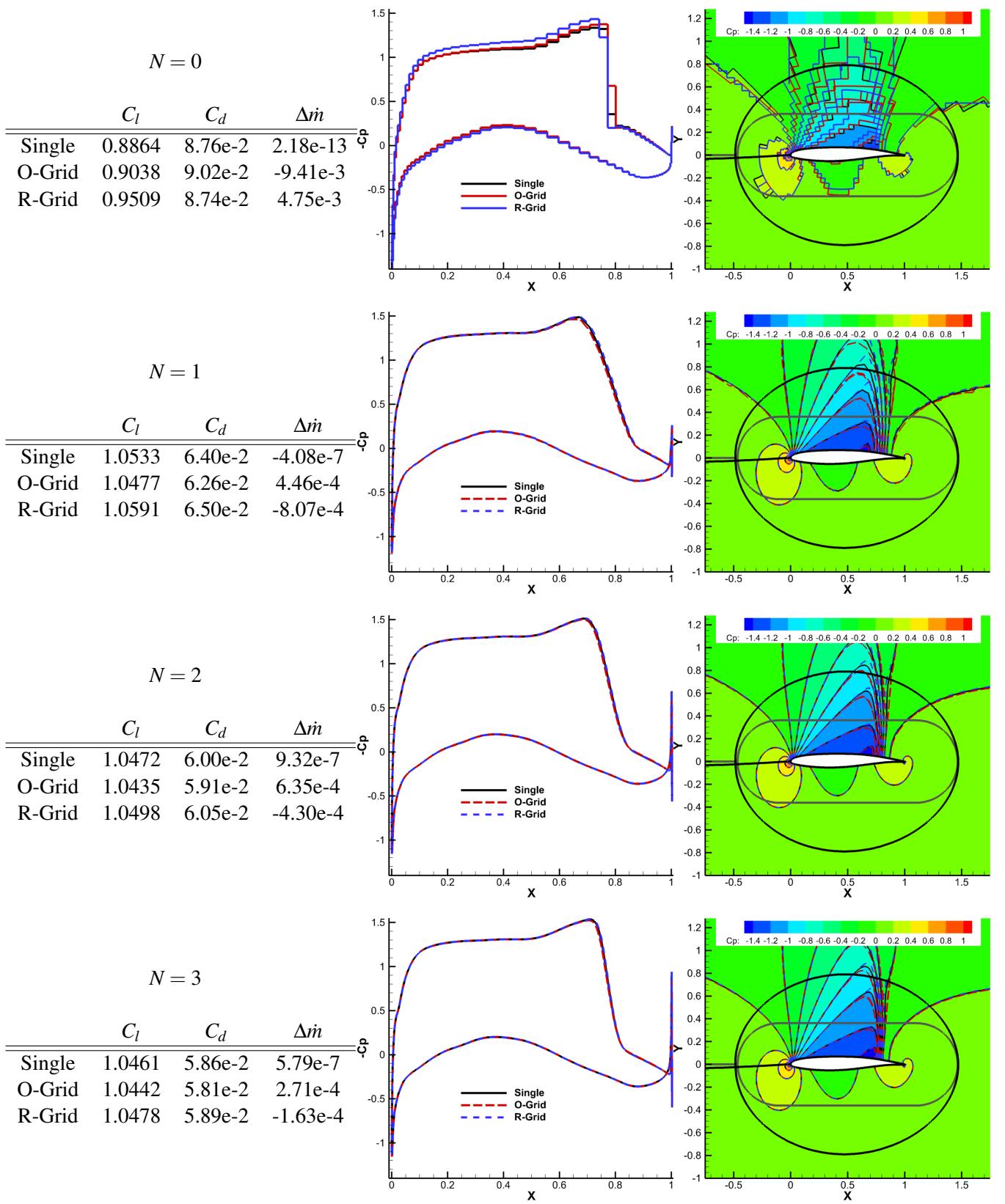


Figure 4.57: SKF 1.1 Airfoil, ($M_\infty = 0.76$, $\alpha = 2.5^\circ$)

4.3.2.6 Supersonic Circular Cylinder

This case is used to demonstrate the DG-Chimera scheme on an external supersonic flow. The three meshes used to compute the flow about a circular cylinder at $M_\infty = 2.0$ are shown in Fig. 4.58. The mesh in Fig. 4.58a is a single grid consisting of cells with a cubic polynomial mapping. The second mesh shown in Fig. 4.58b uses a grid with 13 cells normal to the surface and a cubic polynomial mapping to represent the surface of the cylinder. Two C-grids consisting of linearly mapped cells are used for the farfield. The set of cells on the outflow plane from the single grid are also retained from the single grid in order to form a boundary on the computational domain without overlapping cells. The third mesh (Fig. 4.58c) is constructed from the second but replaces the grid furthest away from the cylinder with a rectangular grid with a hole. A slip wall boundary condition is imposed on the surface of the cylinder, and freestream values are imposed on all conservative variables on the farfield boundary. All conservative variables are extrapolated on the $x = 0$ boundary.

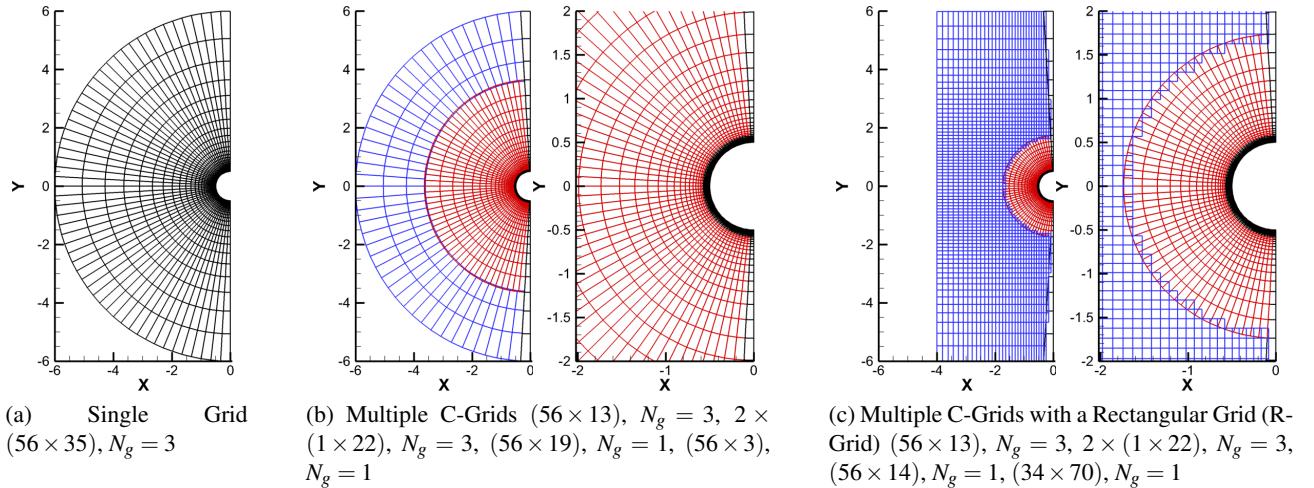


Figure 4.58: Supersonic Inviscid Cylinder Meshes, ($M_\infty = 2.0$)

Lift and drag coefficients, mass flux error, as well as surface pressures and contours of pressure coefficient for increasing order of approximation are shown in Fig. 4.59. As the flow is symmetric, the lift coefficient is zero for all computed flow fields. The drag coefficient also agrees well between the flow fields computed using Chimera meshes and the single grid. The mass flux error is computed as the integral over the inflow and outflow boundaries. The mass flux error is machine zero for the single grid calculations.

The mass flux error initially increases from $N = 0$ to $N = 1$ for the Chimera meshes. The mass flux error decreases as the order of the approximation is further increased. Again, the mass flux error does not decrease as rapidly relative to the subsonic flows. The surface pressure coefficients agree well for $N = 0$, and the pressure coefficient contours computed using the single grid and C-Grid also agree well for all orders of approximation. A slight difference between the surface pressure coefficient computed with the R-Grid relative to the other two meshes for $N \geq 1$ is noted. These differences are primarily a result of differing grid resolution in the R-Grid when compared to the other two meshes. Most importantly, for both Chimera meshes, the shock is able to seamlessly pass through the artificial boundary.

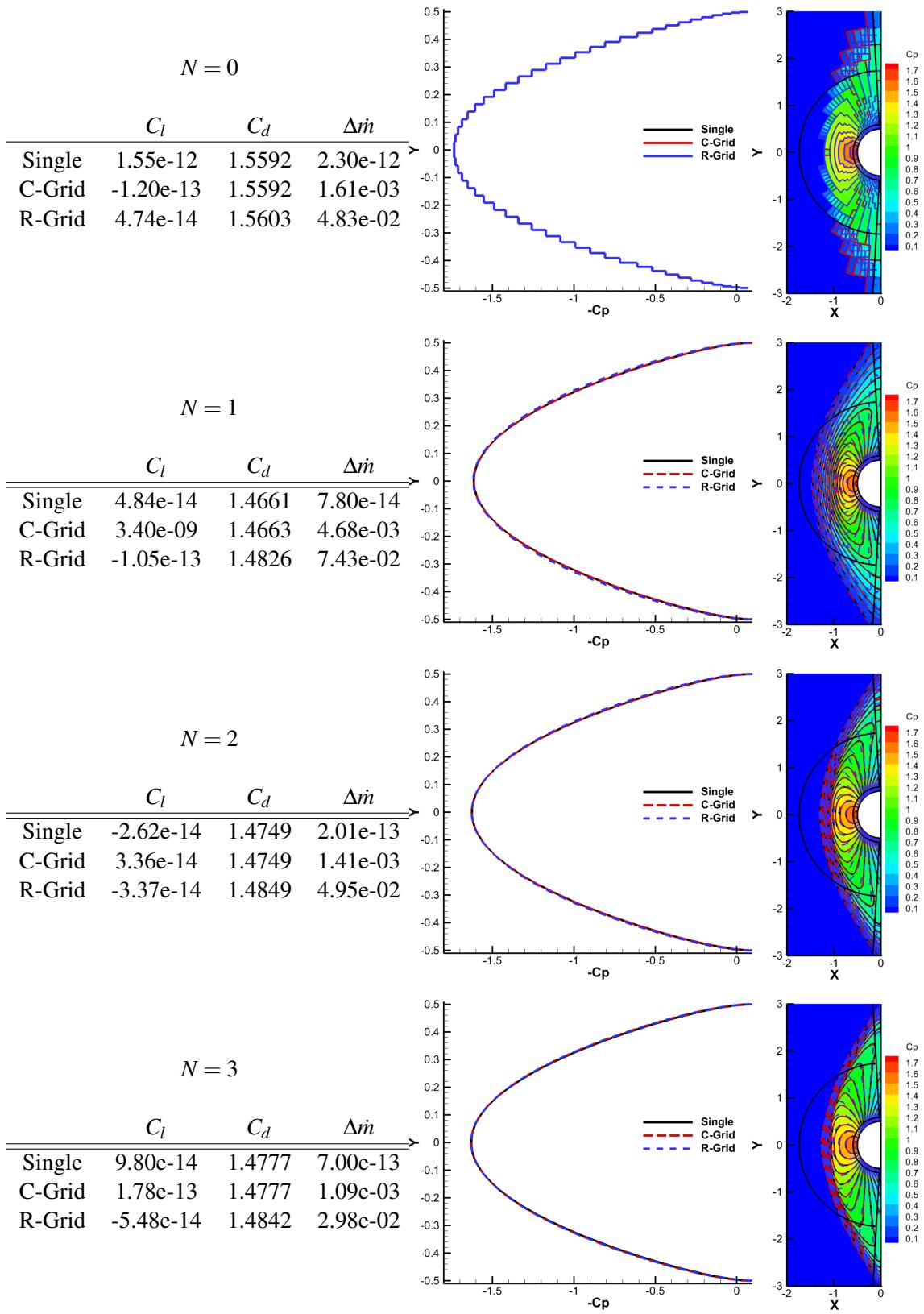
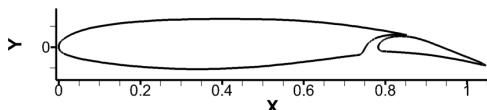


Figure 4.59: Circular Cylinder Pressure Coefficient, ($M_\infty = 2$)

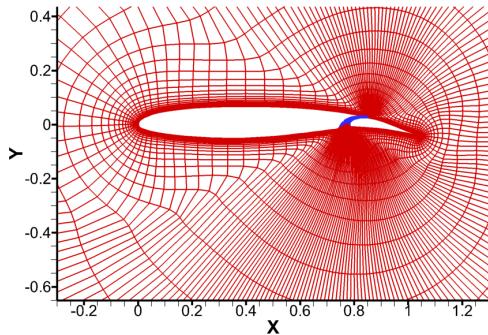
4.3.2.7 SKF 1.1 Airfoil with Flap

The geometry for the SKF 1.1 airfoil with the flap extended is shown in Fig. 4.60a (See, configuration 5 in Ref. [146]). This geometry is used to demonstrate a traditional use of Chimera grids to mesh complex configurations.[38] A zonal mesh[140] consisting of two grids using a cubic polynomial mapping, $N_g = 3$. without any overlapping regions is shown in Figs. 4.60b and 4.60c. The mesh consists of a grid that wraps around both the airfoil and the flap, and a second grid that spans the gap between the airfoil and flap as shown in Fig. 4.60c. The farfield boundary is located 100 chords away from the airfoil. A Chimera overset mesh consisting of 3 grids, one for the airfoil with $N_g = 3$, one for the flap with $N_g = 3$, and one which extends the farfield to 100 chords from the airfoil with $N_g = 1$, is shown in Fig. 4.60d. The surface of the airfoil is used to cut a hole in the flap grid, and the surface of the flap is used to cut a hole in the airfoil grid. The hole cut by the flap is shown in Fig. 4.60e, and the hole cut by the airfoil is shown in Fig. 4.60f. The hole cutting results in artificial boundaries with significant disparity in cell sizes between donor and receiver cells as shown in Fig. 4.60g.

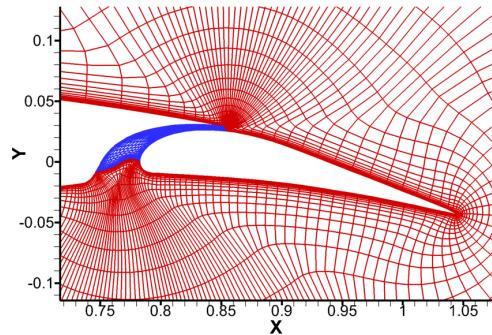
The inviscid flow field is computed about the SKF 1.1 airfoil with the flap using the zonal and Chimera meshes. The farfield boundary is imposed using a Riemann invariant boundary condition with $M_\infty = 0.2$ and $\alpha = 3^\circ$, and the airfoil surface boundary condition is a slip wall boundary condition. Lift, drag, mass flux error, surface pressure coefficient, and pressure coefficient contours for the two meshes are shown with increasing order of the approximation polynomial in Fig. 4.61. Lift and drag computed using the two meshes agree well for $N \geq 1$. The mass flux error is computed as the integral of the farfield boundary. The zonal mesh has a machine zero mass flux error for all orders of approximation, and the mass flux error for the Chimera mesh decreases with an increase in the order of approximation. The surface pressure and pressure contours computed using the two meshes agrees well for $N \geq 1$. Hence, the Chimera mesh is able to obtain solutions of similar quality of the zonal mesh.



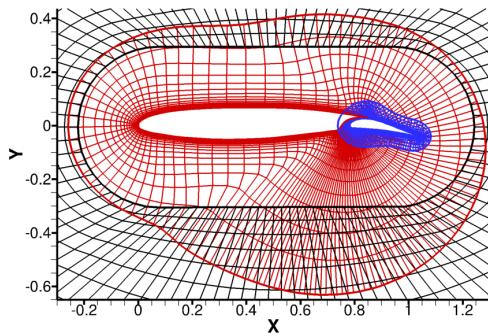
(a) SKF 1.1 Airfoil with Flap Geometry



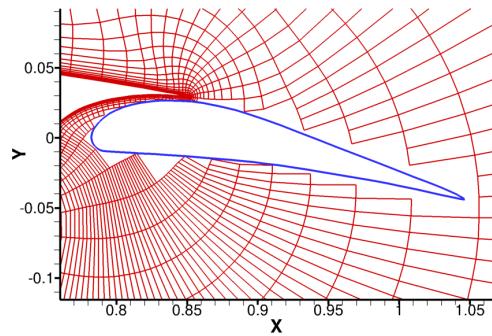
(b) SKF 1.1 Airfoil with Flap Zonal Mesh



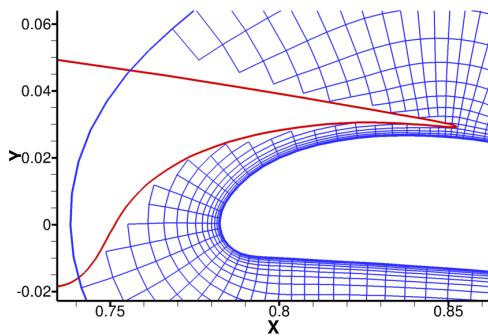
(c) Closeup of Zonal Mesh around the Flap



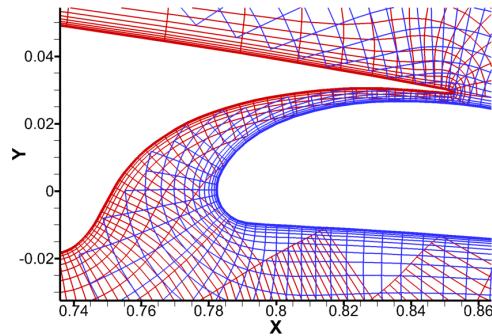
(d) SKF 1.1 Airfoil with Flap Overset Mesh with Hole Cuts



(e) Hole Cut from Airfoil Grid using the Flap Surface



(f) Hole Cut from Flap Grid using the Airfoil Surface

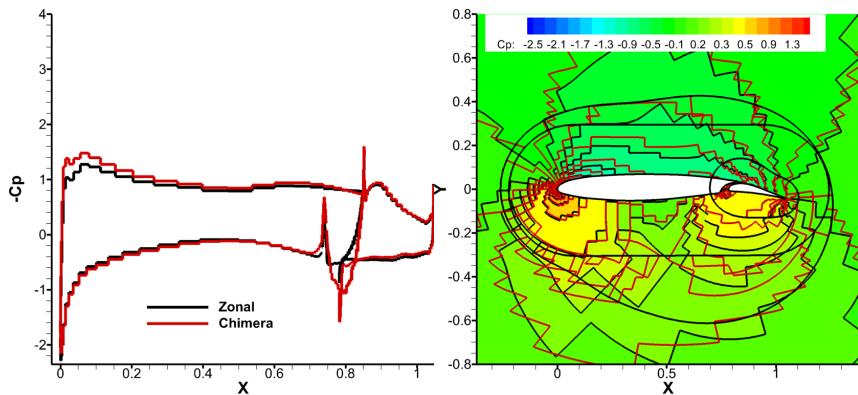


(g) Closeup of Chimera Mesh around the Flap

Figure 4.60: SKF 1.1 Airfoil with Flap Meshes

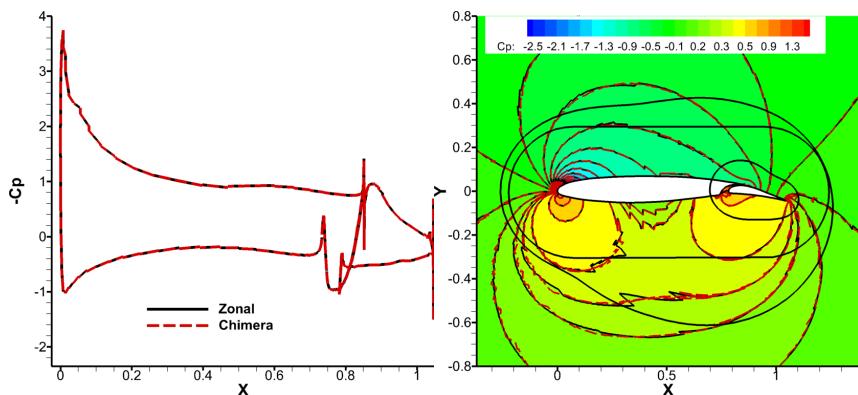
$N = 0$

	Zonal	Chimera
C_l	1.2441	1.3365
C_d	0.156	0.166
Δm	5.44e-13	-3.07e-3



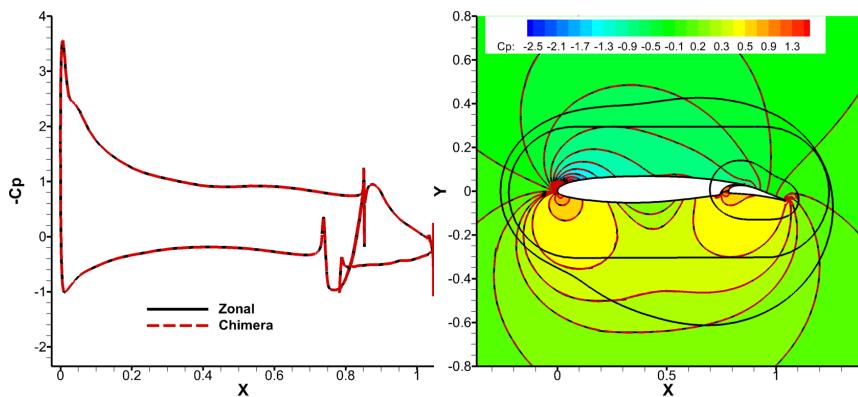
$N = 1$

	Zonal	Chimera
C_l	1.4830	1.4836
C_d	1.34e-3	1.30e-3
Δm	2.12e-13	1.69e-4



$N = 2$

	Zonal	Chimera
C_l	1.4663	1.4661
C_d	7.37e-4	5.01e-4
Δm	-9.22e-13	-4.00e-5



$N = 3$

	Zonal	Chimera
C_l	1.4769	1.4801
C_d	7.29e-4	5.96e-4
Δm	-1.86e-12	-1.21e-5

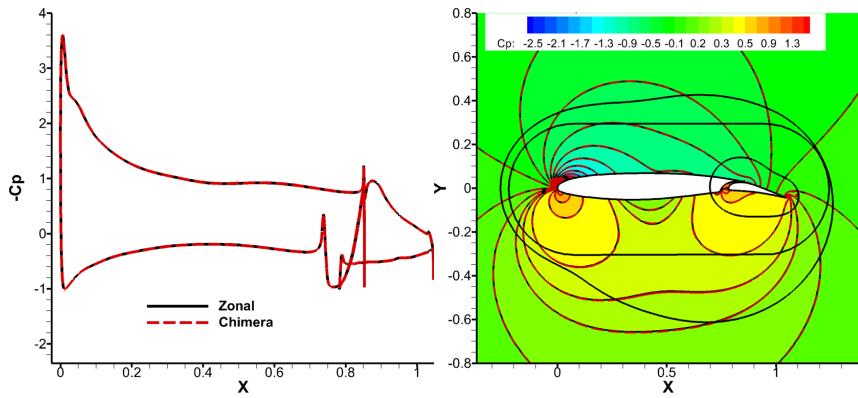


Figure 4.61: SKF 1.1 Airfoil with Flap Pressure Coefficient, ($M_\infty = 0.2$, $\alpha = 3^\circ$)

4.3.2.8 Isentropic Convecting Vortex

The DG-Chimera scheme is applied to an inviscid problem consisting of a compressible vortex convecting in a rectangular domain with periodic conditions imposed on all sides.[162, 163, 164] This flow problem demonstrates the benefits of high-order accurate schemes with low dissipation. The vortex is initially positioned at $(x_0, y_0) = (0.05, 0.05)$ and convects with the free-stream for 12 characteristic time units where it returns to its starting position. The analytical solution is given by

$$\begin{aligned} f(x, y, t) &= \left(1 - ((x - x_0) - |V_\infty| \cos(\theta)t)^2 - ((y - y_0) - |V_\infty| \sin(\theta)t)^2\right) / r_c^2 \\ u_v(x, y, t) &= |V_\infty| \left(\cos(\theta) - \frac{\varepsilon((y - y_0) - |V_\infty| \sin(\theta)t)}{2\pi r_c} \exp\left(\frac{f(x, y, t)}{2}\right) \right) \\ v_v(x, y, t) &= |V_\infty| \left(\sin(\theta) + \frac{\varepsilon((x - x_0) - |V_\infty| \cos(\theta)t)}{2\pi r_c} \exp\left(\frac{f(x, y, t)}{2}\right) \right) \\ \rho_v(x, y, t) &= \rho_\infty \left(1 - \frac{\varepsilon^2(\gamma - 1) M_\infty^2}{8\pi^2} \exp(f(x, y, t)) \right)^{\frac{1}{\gamma-1}} \\ p_v(x, y, t) &= p_\infty \left(1 - \frac{\varepsilon^2(\gamma - 1) M_\infty^2}{8\pi^2} \exp(f(x, y, t)) \right)^{\frac{\gamma}{\gamma-1}} \end{aligned} \quad (4.15)$$

where θ is the flow angle, ε is a measure of the strength of the vortex, and r_c is a measure of the size of the vortex. Solutions are obtained using $M_\infty = 0.5$, $\varepsilon = 1$, $r_c = 0.005$, and $\theta = 0$. The vortex is advanced in time with a time step of $\Delta t = 0.005$ using the unsteady Euler equations in Eq. 2.41 that are discretized with a 3rd-order accurate three stage Diagonally Implicit Runge-Kutta (DIRK)[132] scheme. The implicit system of equations associated with each stage of the DIRK scheme is solved with a Newton's method that is converged until the L^2 -norm of the residual vector drops below a tolerance of 5×10^{-10} .

The initial vortex location on the four meshes used to convect the vortex are shown in Fig. 4.62. The meshes consist of a background grid and a wavy grid that cuts a hole in the background grid. The wavy grid is formed by perturbing the coordinates from a uniform square grid using the formula

$$\begin{aligned} x_w &= x + L_s 0.04 \sin(2\pi(y - y_s) / L_s) \\ y_w &= y + L_s 0.04 \sin(2\pi(x - x_s) / L_s), \end{aligned} \quad (4.16)$$

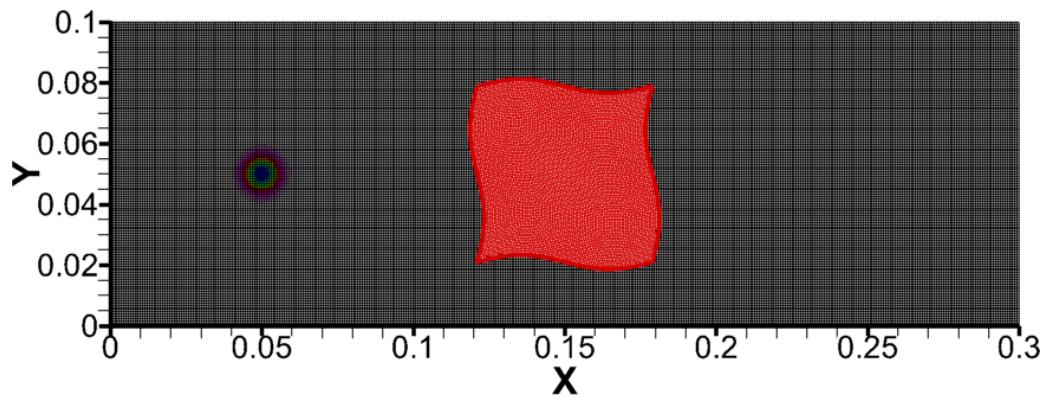
where $L_s = 0.058$ is the height and width of the square and (x_s, y_s) is the lower left hand corner of the square.

The cell count in each mesh decreases as the order of the polynomial approximation is increased such that the total number of degrees of freedom remains a constant 52,272 in the background grid and 7,056 in the wavy grid. The vortex is also convected on the background grid without the presence of the wave grid to assess the influence of the artificial boundaries.

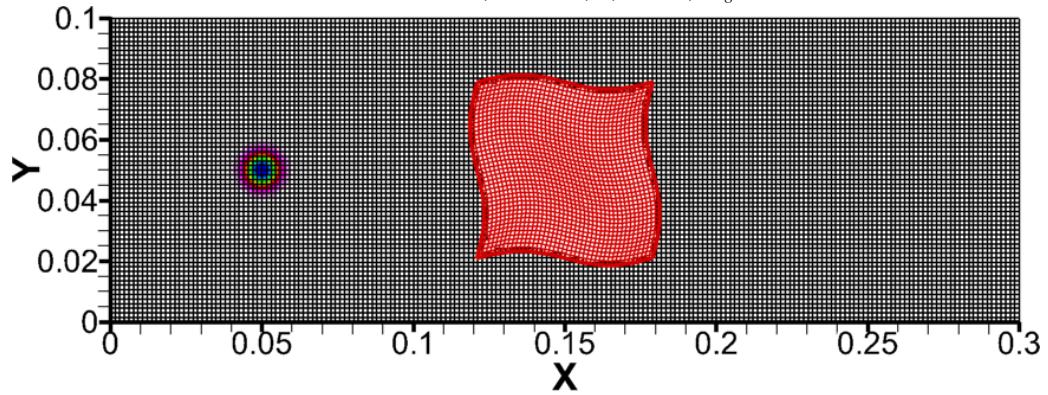
The pressure coefficient along the horizontal centerline and pressure coefficient contours at the initial time and final time of $t=12$ is shown in Fig. 4.63. The entropy rise defined as

$$\text{Entropy Rise} = \frac{\frac{p}{\rho^\gamma} - \frac{p_\infty}{\rho_\infty^\gamma}}{\frac{p_\infty}{\rho_\infty^\gamma}}, \quad (4.17)$$

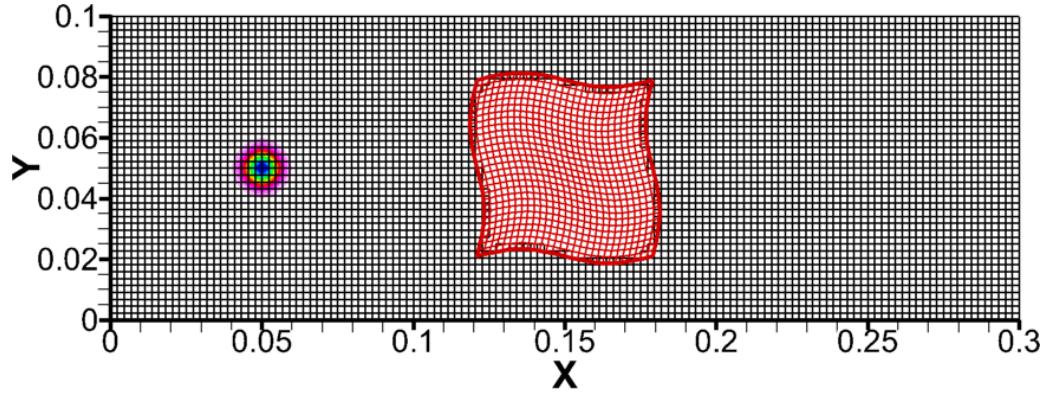
along the horizontal centerline is also shown in 4.63. The vortex for the 1st-order accurate, $N = 0$, solution dissipates within the first characteristic time. As a result, the solution at $t = 12$ is nearly a uniform stream solution. The 2nd-order accurate solution, $N = 1$, has maintained the vortex, though it has dissipated significantly and is asymmetric on the horizontal centerline. It is difficult to discern the vortex in the pressure coefficient contours. The 3rd-order accurate solution, $N = 2$, has preserved the pressure deficit associated with the vortex well. The magnitude of the pressure deficit is small. The pressure coefficient contours at $t = 12$ also agree well with the initial condition. The 4th-order solution, $N = 3$, at $t = 12$ also agrees well with the initial condition in the horizontal centerline pressure and the pressure contours. As expected, the entropy error decreases as the order of the polynomial approximation increases. Notably, the solutions on the single grid and the Chimera mesh agree well in both pressure coefficient and entropy error for $N \geq 1$. This indicates that the artificial boundaries do not introduce a significant error and the vortex is able to convect at the correct speed across the wavy grid in the Chimera mesh. These results also demonstrate that the low dissipation associated with high-order discretization is able to maintain the vortex over a longer period of time for a given number of degrees of freedom.



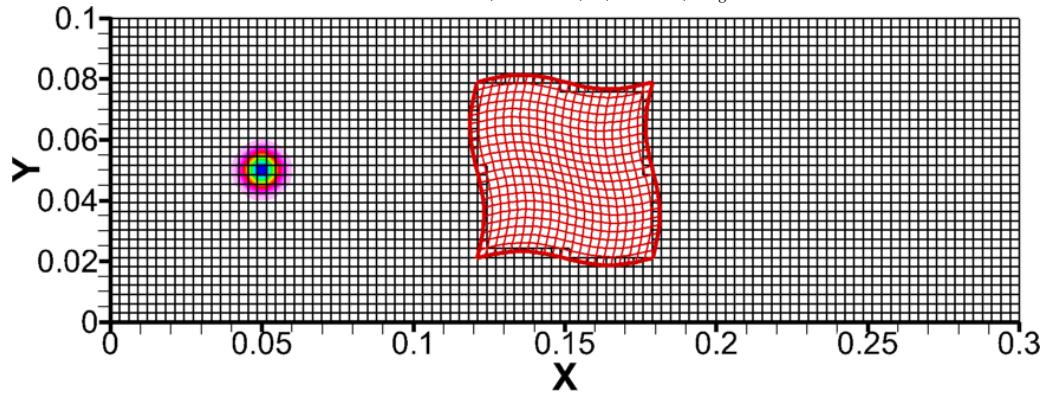
(a) Mesh for $N = 0$, (396×132) , (84×84) , $N_g = 1$



(b) Mesh for $N = 1$, (198×66) , (42×42) , $N_g = 1$



(c) Mesh for $N = 2$, (132×44) , (28×28) , $N_g = 1$



(d) Mesh for $N = 3$, (99×33) , (21×21) , $N_g = 1$

Figure 4.62: Isentropic Convecting Vortex Meshes, ($M_\infty = 0.5$)

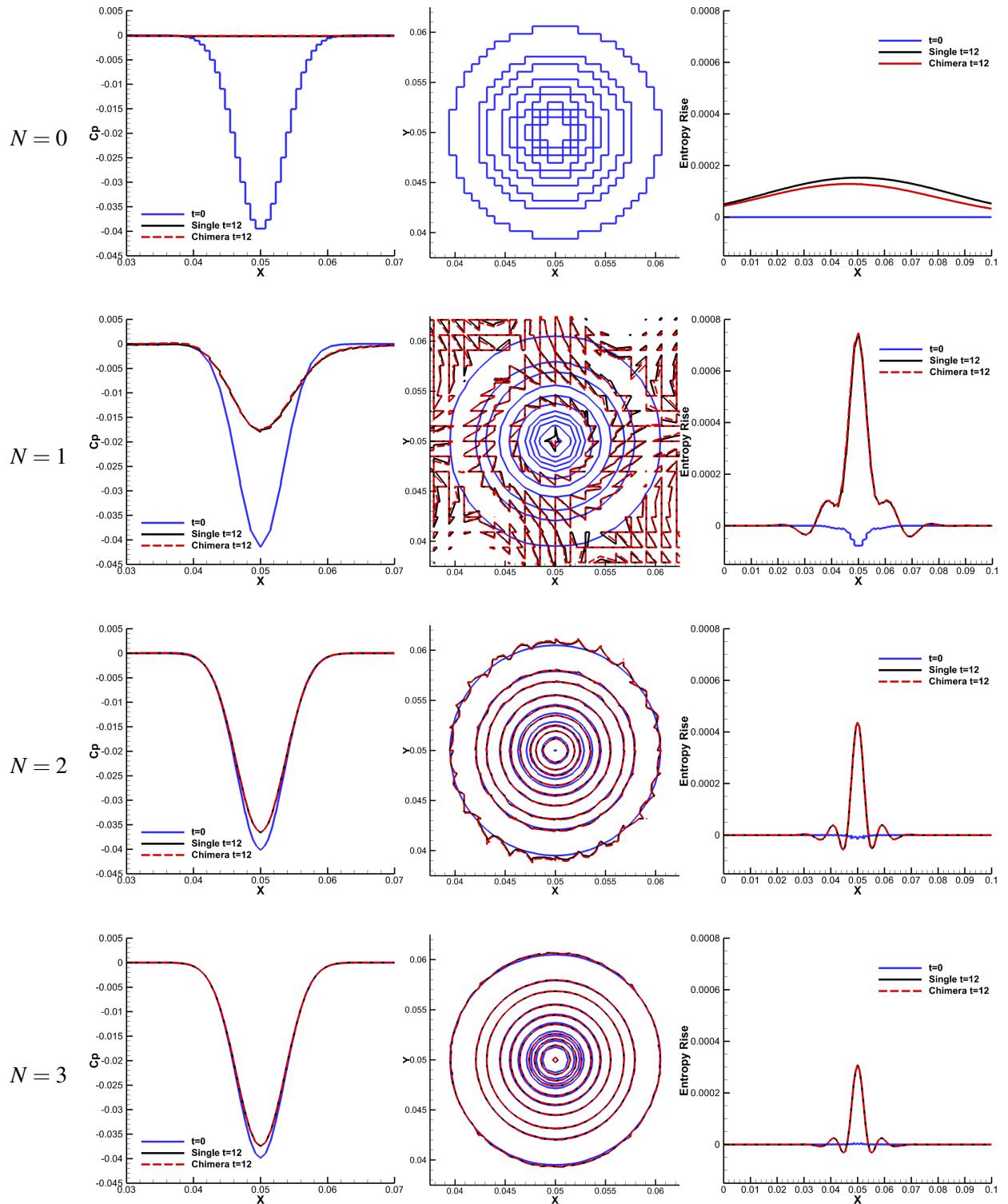


Figure 4.63: Convecting Isentropic Vortex after 12 Characteristic Times, ($M_\infty = 0.5$)

4.3.2.9 Analytical 3-D Body of Revolution

This case will test the solver for computing an external flow over a high-order curved geometry in 3D[45].

The geometry is a streamline body based on a 10 percent thick airfoil which is extended to 3D through a surface of revolution. The half model surface coordinates are given by

$$\begin{cases} 16(x - \frac{1}{4})^2 + 400z^2 = 1, & x \in [0, \frac{1}{3}], y \in [0, \frac{1}{100}] \\ z = \frac{1}{10\sqrt{2}}(1-x), & x \in (\frac{1}{3}, 1], z > 0, y \in [0, \frac{1}{100}] \\ z = -\frac{1}{10\sqrt{2}}(1-x), & x \in (\frac{1}{3}, 1], z < 0, y \in [0, \frac{1}{100}] \\ 16(x - \frac{1}{4})^2 + 400(z^2 + (y - \frac{1}{100})^2) = 1, & x \in [0, \frac{1}{3}], y > \frac{1}{100} \\ 200(z^2 + (y - \frac{1}{100})^2) - (1-x)^2 = 0, & x \in (\frac{1}{3}, 1], y > \frac{1}{100} \end{cases} \quad (4.18)$$

which is illustrated in Fig. 4.64. The reference area is 0.1. A zonal mesh and a Chimera mesh are shown in Fig. 4.65. The zonal mesh consist of two grids; one on the upper surface and one on the lower surface of the body. The Chimera mesh is created by retaining the portion of the zonal mesh in the vicinity of the body and adding a rectangular background grid. The hole in the background grid is cut manually. A slip wall boundary condition is used to impose the surface of the body, and a Riemann Invariant boundary condition is imposed on the farfield boundary that is located 100 chords away upstream and surrounding the body and 300 chords downstream of the body.

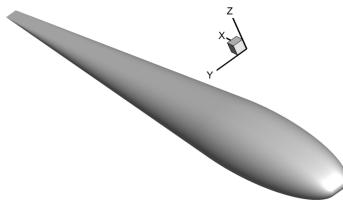


Figure 4.64: Analytical 3D Body

The flow field surrounding the body is computed with $\alpha = 1^\circ$ and $M_\infty = 0.5$. Lift and drag coefficient and mass flux error along with surface pressure coefficient and contours of pressure coefficient on the symmetry plane computed using both the Zonal and Chimera meshes with increasing order of approximation are shown in Fig. 4.66. Similar to the two-dimensional cases considered here, the lift and drag coefficients computed

using the zonal and Chimera meshes agree well. In addition, the max flux error, which is computed as the integral over all the boundaries of the computational domain, are small for the zonal and Chimera meshes. Furthermore, the mass flux error decreases as the order of approximation increases. Visually, the surface pressure coefficient and the pressure coefficient contours also agree well between the flow fields computed with the zonal and Chimera meshes.

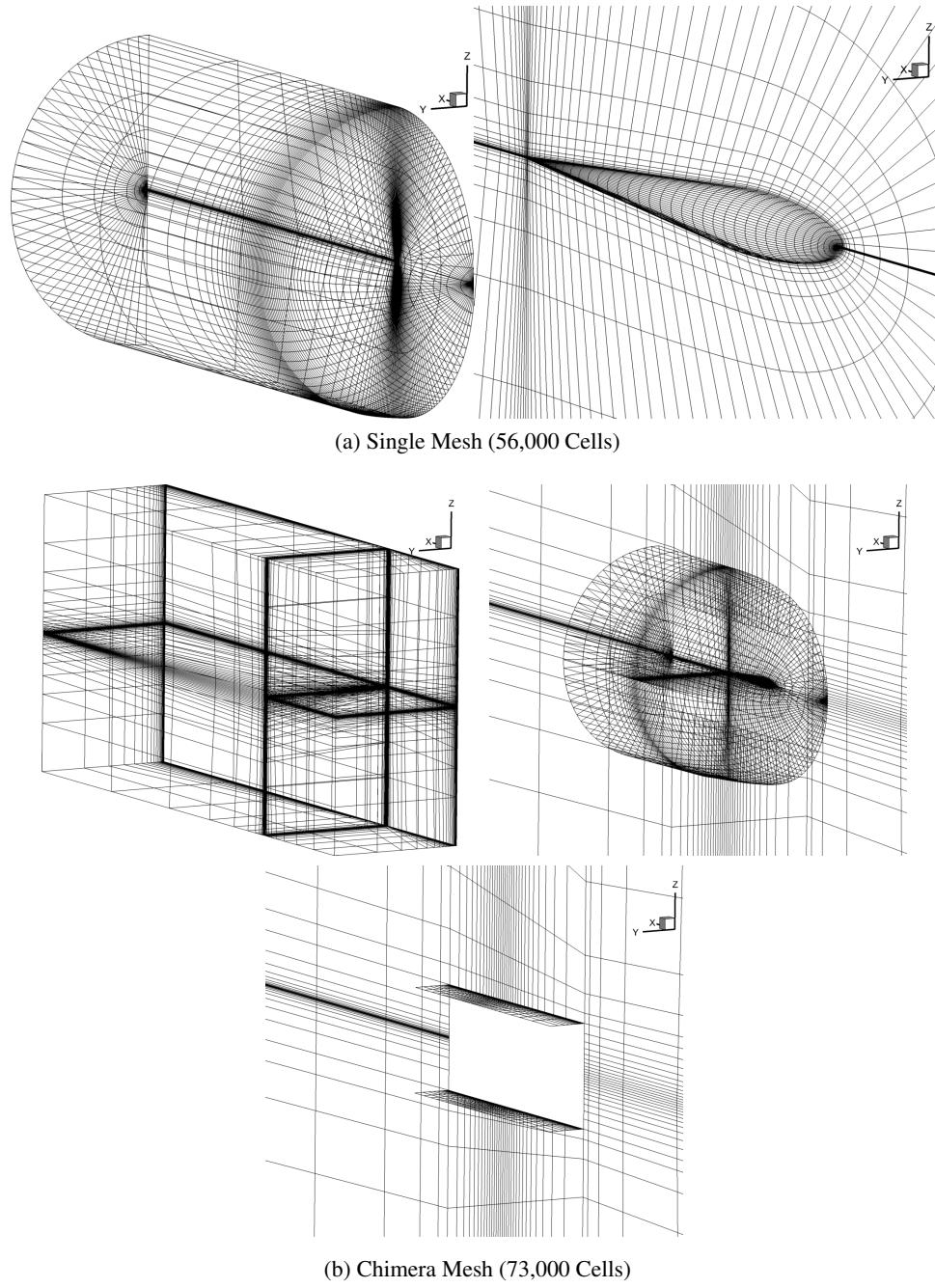


Figure 4.65: Analytical 3-D Body Meshes

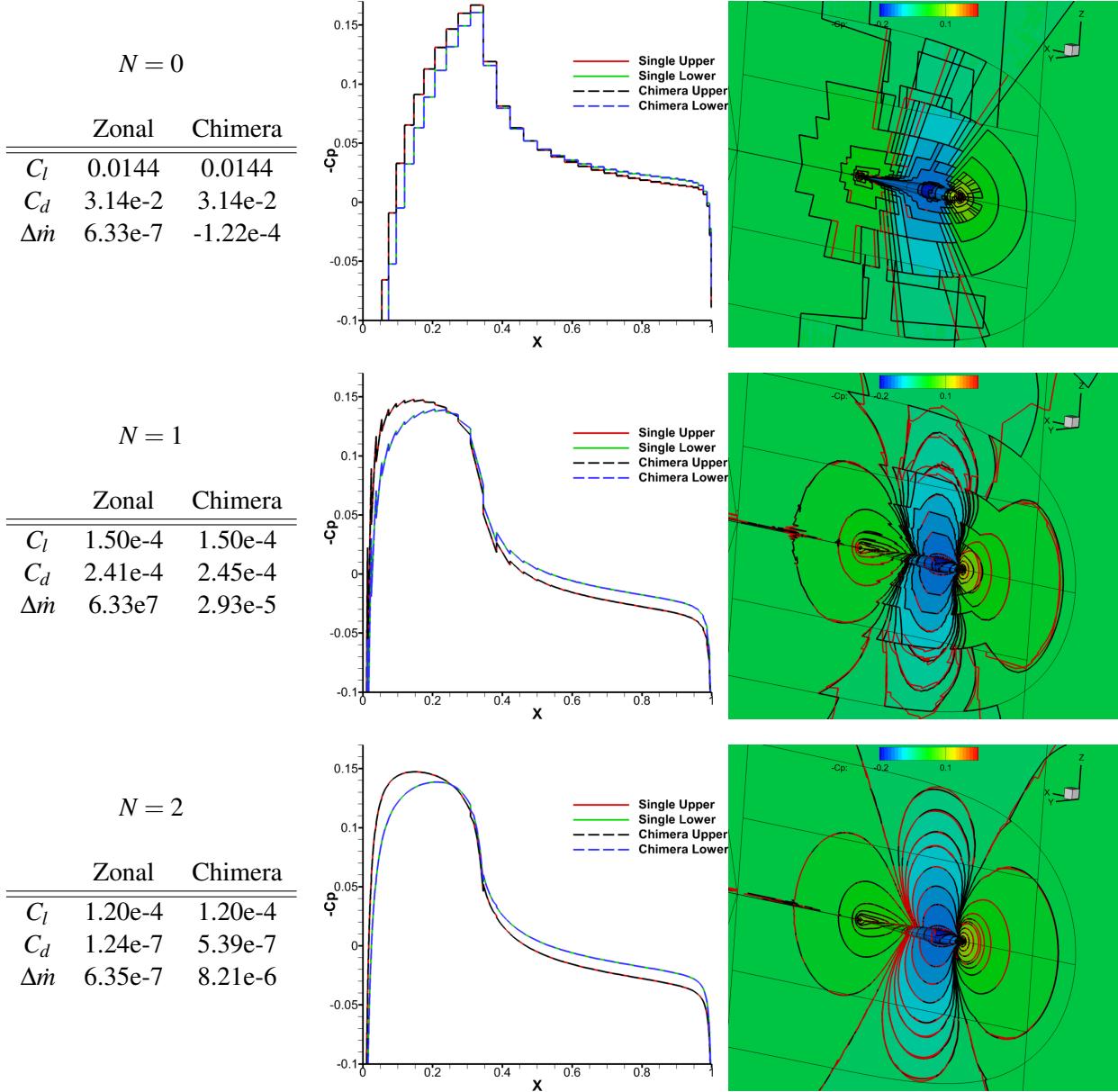


Figure 4.66: Analytical 3-D Body Lift and Drag and Pressure Coefficient

4.3.2.10 Summary

The DG-Chimera scheme has been demonstrated on a set of inviscid subsonic, transonic, and supersonic internal and external flow problems. Inviscid channel flow demonstrated that the numerical mass flux errors associated with the artificial boundaries are consistent for all orders of accuracy and small for $N \geq 1$. The

mass flux error associated with the artificial boundaries is reduced by using a Gauss-Quadrature node count of $N_{GQ} = \lceil 3N/2 \rceil + 1$. Even though the mass flux errors are small, their presences does suggest exploration of methods that can reduce or eliminate these errors and are also extensible to three-dimensions is warranted.

Inviscid internal and external subsonic, transonic, and supersonic flow fields obtained using Chimera overset meshes agree well with flow fields obtained using a single grid with comparable mesh resolution for $N \geq 1$. Notably, the DG-Chimera scheme is able to transfer strong gradients, such as shocks, across artificial boundaries. The scheme was used to compute the inviscid flow about the SKF 1.1. airfoil with a flap; a flow problem that represents traditional use of the Chimera method to represent complex geometry. The convection of an isentropic vortex demonstrates that the 3rd-order and 4th-order DG schemes are able to maintain the pressure deficit associated with the vortex without significant dissipation for a fixed number of degrees of freedom relative to the 1st-order and 2nd-order DG schemes. The artificial boundary did not introduce significant errors in the time accurate calculation. Finally, the solutions computed on a three-dimensional Chimera mesh agree well with solutions computed using a three-dimensional zonal mesh.

4.3.3 Curved Geometry with Viscous Flows

The focus in this section is on viscous Chimera meshes with non-co-located artificial boundaries on curved geometries. In this situation, meshes based on linear cells, which are used for finite volume (FV) and finite difference (FD) schemes, will not produce valid interpolation; unless they are corrected using, for example, the “Projection” features in PEGASUS5[42, 165] and SUGGAR++[110]. Linear cell faces cutting through the curved solid body geometry is the primary cause of the incorrect interpolation stencil. This issue is addressed by using the DG-Chimera scheme along with curved cells capable of matching the CAD interpolation and hence following the boundary of the curved geometry. The use of curved elements for a finite volume discretization was also explored by Noack and Belk in Ref. [166]. An additional advantage of the DG discretization in this regard is that the higher order accuracy permits the use of fewer cells compared to FV and FD schemes to capture a viscous boundary layer.

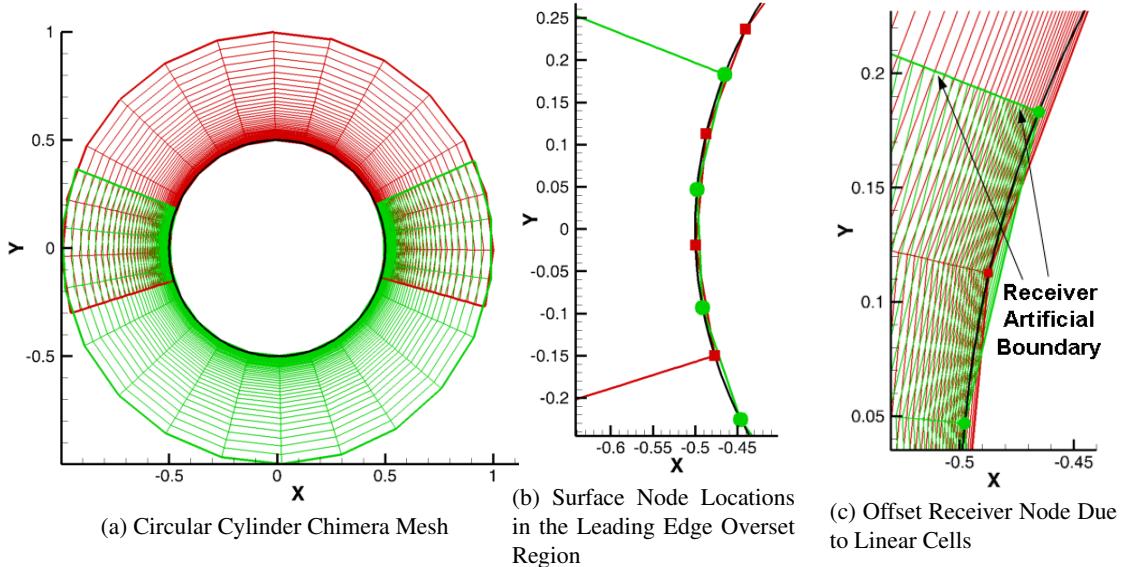


Figure 4.67: Example of Inaccurate Interpolation associated with Overset Meshes on Curved Geometries

The Chimera mesh shown in Fig. 4.67 was contrived to demonstrate issues associated with using Chimera meshes for non-co-located artificial boundaries on curved geometries. The nodes on each mesh used to represent the circle in Fig. 4.67a are placed directly on the surface of the cylinder. Traditional finite volume (FV) and finite difference (FD) schemes rely on linear cells to represent geometric features. On concave surfaces, the face of a linear cell will lie within the actual geometry it is supposed to represent. Since the nodes of the two meshes are not co-located, the surface nodes of the green mesh are offset from the face of the red mesh, and vice versa, as shown in Fig. 4.67b. In addition, as finite volume and finite difference codes typically require a large number of cells normal to the wall to capture a viscous boundary layer, the receiver node on the green mesh, shown in Fig. 4.67c, actually lies within the 4^{th} cell off the surface of the red mesh. This offset not only applies to the surface node, but also to several layers of nodes away from the cylinder in the green mesh. Thus, an interpolation scheme that relies on the red donor cells which contain green receiver nodes will transfer an incorrect boundary layer profile to the green mesh. Much effort has been put into developing “Projection” features in PEGASUS[42, 43] and SUGGAR++[110] to offset the interpolation stencil to account for the offset between the two meshes.

Unlike FV and FD schemes, the DG scheme allows the use of curved cells. Curved cells are capable of better approximating the curvature of a surface without falling within the surface. The mesh representation

of the surface is exact if the order of the cell polynomial is the same as the spline polynomial used to represent the geometry. The DG scheme also requires significantly fewer cells than the FV and FD methods to accurately represent a viscous boundary layer. Thus, the DG scheme is capable of resolving the problems demonstrated in Fig. 4.67 without resorting to corrections.

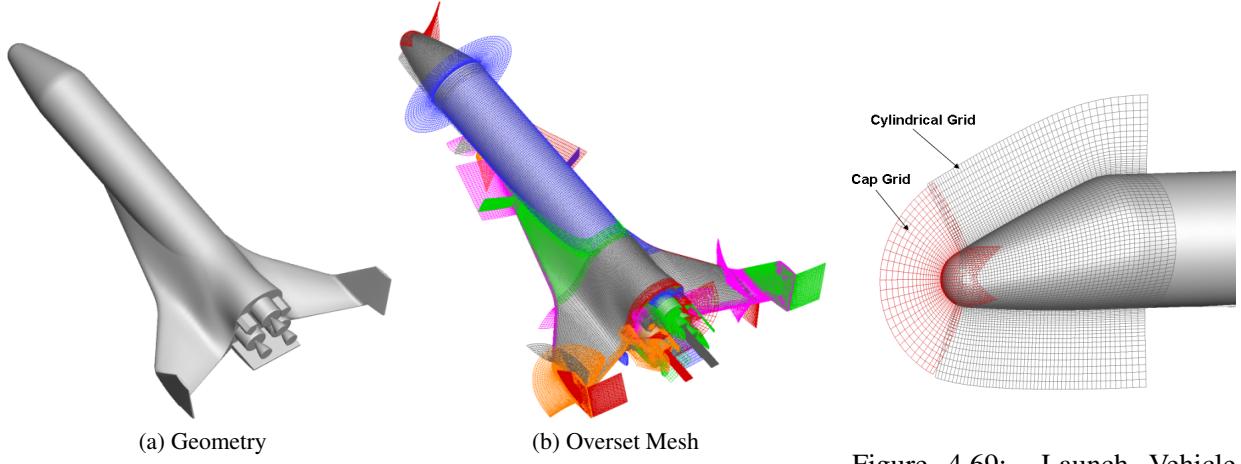


Figure 4.68: Generic First Stage Launch Vehicle

Figure 4.69: Launch Vehicle Nose Section

The use of Chimera meshes based on curved elements is compared here with linear meshes for a traditional finite volume calculation for two two-dimensional geometries. The configurations are inspired by the Chimera mesh of a generic first stage launch vehicle[167] which uses linear overset meshes and is shown in Fig 4.68. The first geometry is representative of grids used to resolve the nose section of the launch vehicle, and the second geometry is a circular cylinder.

4.3.3.1 Nose Section

The first geometry mimics the mesh configuration used to create the overset meshes on the nose section on the launch vehicle, as shown in Fig. 4.69. A cap grid has been placed over the stagnation region of the nose to avoid creating a singularity in the cylindrical grid wrapped around the nose section. A representative two-dimensional geometry defined by the parametric curve

$$x(s) = 1.5s^3, \quad y(s) = \pm 0.5s, \quad s \in [0, 1], \quad (4.19)$$

is used to illustrate in two-dimensions how the curved elements used with the DG solver can resolve issues

associated with the secant mesh representation of the geometry used for a finite volume solver. Solutions are obtained for the flow conditions $M_\infty = 0.5$, $Re = 1,000,000$.

Two meshes generated for the finite volume flow solver OVERFLOW[168] are shown in Fig. 4.70. The single grid in Fig. 4.70a is used to compute a reference solution for the solution computed on the Chimera mesh in Fig. 4.70b. The Chimera mesh was obtained by sub-dividing the single mesh and adding points to generate the overlapping region. By maintaining a similar grid point distribution, differences in the solutions on the two meshes can be attributed to the artificial interfaces. The overlapping region highlighted in Fig. 4.70b was intentionally placed in a high curvature region of the geometry to exacerbate the difficulties associated with finding an acceptable interpolation for the artificial boundary.

The single grid and Chimera mesh for the DG solver are shown in Fig. 4.71. All cells in both the single grid and Chimera mesh are represented using $N_g = 3$ polynomial mappings, which are consistent with the order of the polynomial construction of the geometry. The number of cells in the DG mesh was tailored to yield a comparable number of degrees of freedom (DOF) to the finite volume mesh with a 4th-order DG discretization, i.e. ($N = 3$). Similar to the finite volume meshes, the DG Chimera mesh was generated by sub-dividing the single grid and adding nodes to create the overlapping regions. The overlapping region in the DG Chimera mesh is positioned in the same high curvature region as the finite volume Chimera mesh overlapping region.

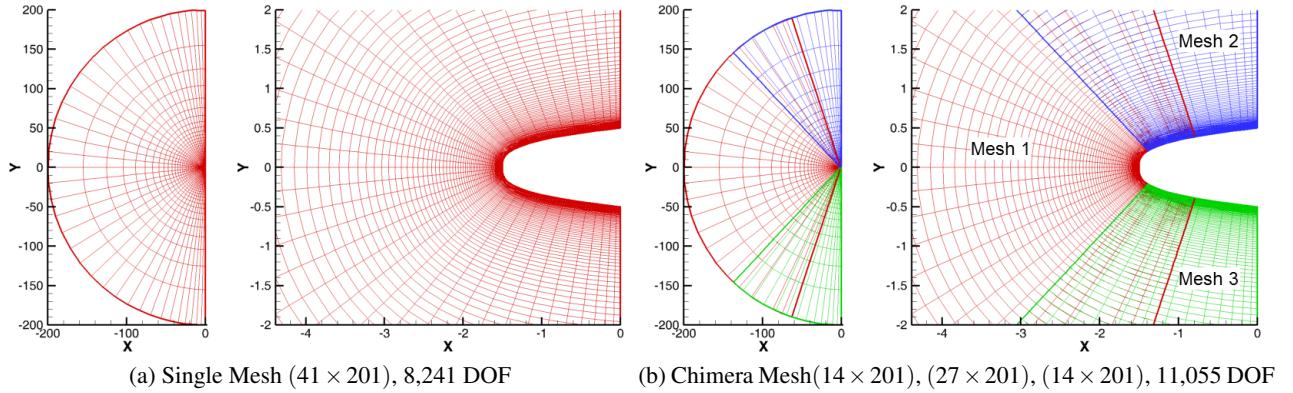


Figure 4.70: Finite Volume Meshes

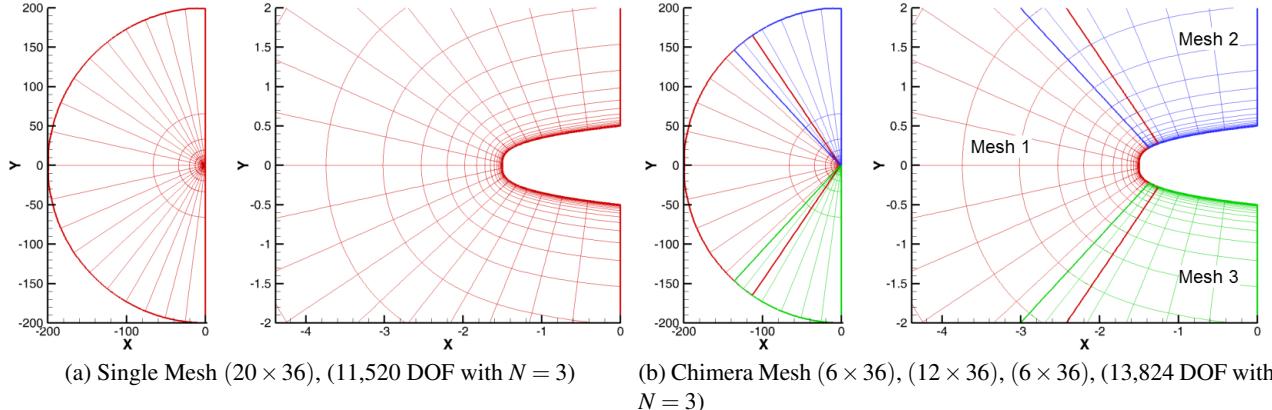


Figure 4.71: Discontinuous Galerkin $N_g = 3$ Polynomial Mapping Meshes

OVERFLOW was used with a 4^{th} -order Roe scheme and a WENO limiter to compute the flow field on the two finite volume meshes. A Riemann invariant boundary condition is imposed on the far-field boundary as well as the outflow boundaries at $x = 0$. The solid surface is assumed to be an adiabatic no-slip wall. Two flow solutions were computed on the Chimera mesh: one which did not use projection, and a second solution which used the projection features of SUGGAR++ to improve the interpolation. Contours of C_p are shown in Fig. 4.72 on both the single grid and Chimera meshes. Figure 4.72d shows the C_p contours computed without projection on the Chimera mesh superimposed on the singe grid contours. Only minor differences can be observed in the two solutions; most notably in the overset region. Similarly, as shown in Fig. 4.72e, only minor differences can be observed between the single grid and Chimera mesh C_p contours when the projection was used. A more notable difference between the single grid solution and Chimera mesh solution without projection is observed in the entropy rise, \hat{s} , as illustrated in Fig. 4.73. The superimposed solutions in Fig. 4.73d exhibit a significant difference in the Chimera solution in the overlapping regions as well as downstream of the overlapping regions compared to the single grid solution. The projection technique is not able to remedy these differences between the single grid and Chimera mesh solutions as shown in Figs. 4.73c and 4.73e.

Solutions on the DG meshes were computed using a 4^{th} -order accurate, $N = 3$, DG formulation. Total pressure and temperature are imposed on the farfield boundary, and a constant pressure is imposed on the $x = 0$ boundaries. The solid surface is assumed to be an adiabatic no-slip wall. C_p contours computed with the DG solver for both the single grid and Chimera grid are shown in Fig. 4.74. The contours from the two

meshes are superimposed in Fig 4.74c. No notable difference is observed between the contours of the two meshes. The entropy contours of the DG solution (See, Fig. 4.75.) are not distinguishable and the entropy rise is only present in the boundary layer.

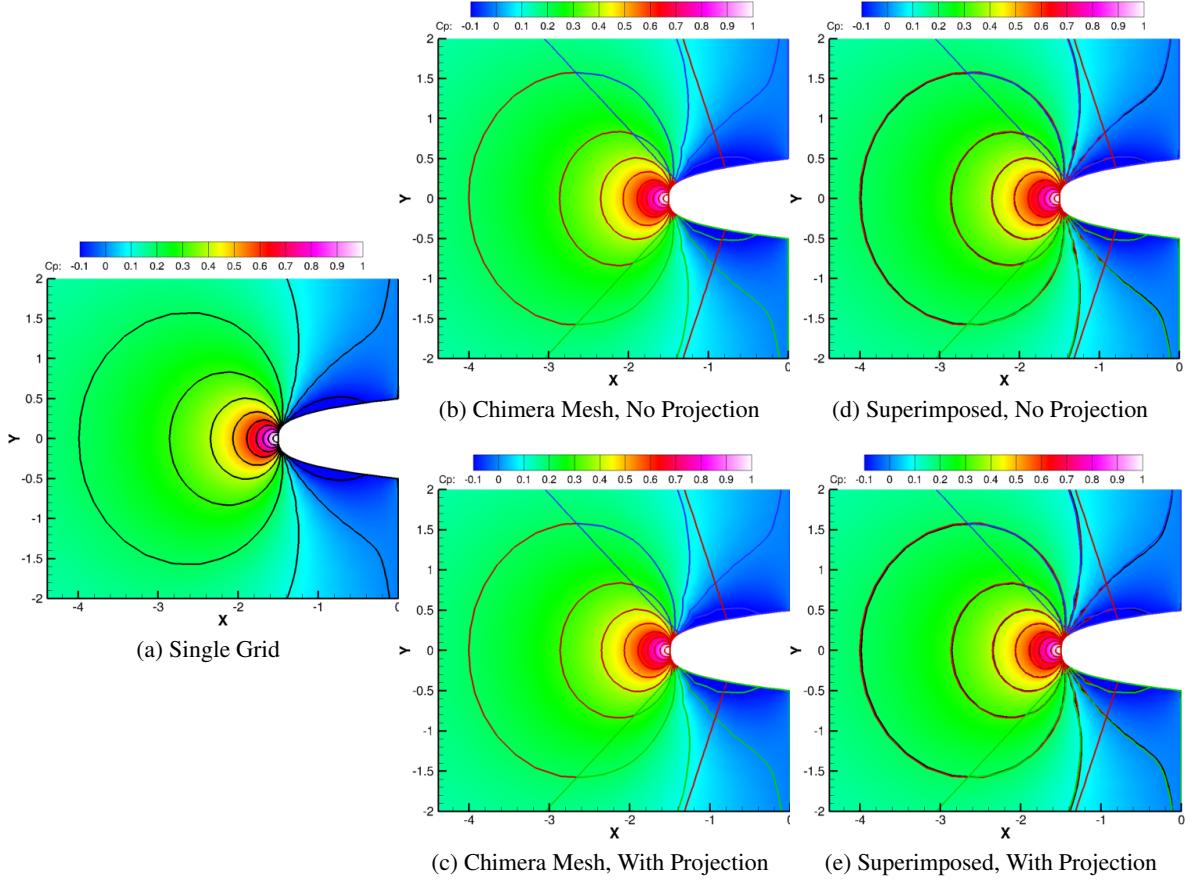


Figure 4.72: Finite Volume C_p Contours on the Generic Nose Section (Single Mesh - Black Lines, Chimera Mesh - Gray Lines)

The overlapping region on the upper surface of the geometry for the finite volume and DG Chimera meshes are shown in Fig. 4.76. As shown in Fig. 4.76a, the interpolated values for the difference stencil on the blue mesh receives values from well outside of the boundary layer due to the secant representation of the geometry. As a result, the interpolated values without projection, i.e., the blue velocity vectors in Fig. 4.77a, do not capture the presence of the boundary layer. However, the projection technique shifts the nodes to the secant surface representation of the mesh and is hence able to capture a boundary layer on the artificial boundary as shown in Fig. 4.77b. The interpolated solution shown in Fig. 4.77c is also able to

capture the boundary layer since the DG Chimera mesh is able to follow the geometry (See, Fig. 4.76.). To further emphasize the interpolation problem, the velocity magnitude at the first point off the wall for the OVERFLOW and DG solvers is shown in Fig. 4.78. The OVERFLOW solution on the Chimera mesh without projection (Fig. 4.78a) exhibits a large rise in the velocity in the overlapping region compared to the solution on the single mesh. The Chimera mesh solution with projection compares favorably with the single grid solution with only small differences between the solutions. The velocity on the DG Chimera mesh one point off the wall agrees well with the single grid DG solution as shown in Fig. 4.78c without any notable differences.

The interpolation problems also lead to differences in C_p and entropy rise in the OVERFLOW solution on the Chimera mesh in comparison with the solution on the single grid, as shown in Figs. 4.79 and 4.80. Notably, when projection is not used, the entropy rise that should be present in the boundary layer on the receiving blue mesh is absent and instead a rise in entropy is observed aft of the artificial boundary in the field away from the wall. While the projection technique is not able to address the differences in C_p , it is able to reintroduce an entropy boundary layer on the artificial boundary as shown in Fig. 4.80b. Though the increased entropy away from the wall on the artificial boundary is still present when projection is used. Again, the DG solution on the Chimera mesh in Figs. 4.79 and 4.80 agrees well with the DG solution on the single grid.

Boundary layer profiles taken a common grid line in both the single and Chimera meshes are shown in Fig. 4.81. Both the boundary layers from the OVERFLOW solution without projection from meshes 1 and 2 of the Chimera mesh have a fuller boundary layer compared to the single mesh solution. However, the single mesh boundary layer profile is recovered when the projection feature is applied, with only a small discrepancy on mesh 2 of the Chimera mesh. The boundary layers from DG solutions on the single and Chimera meshes are indistinguishable.

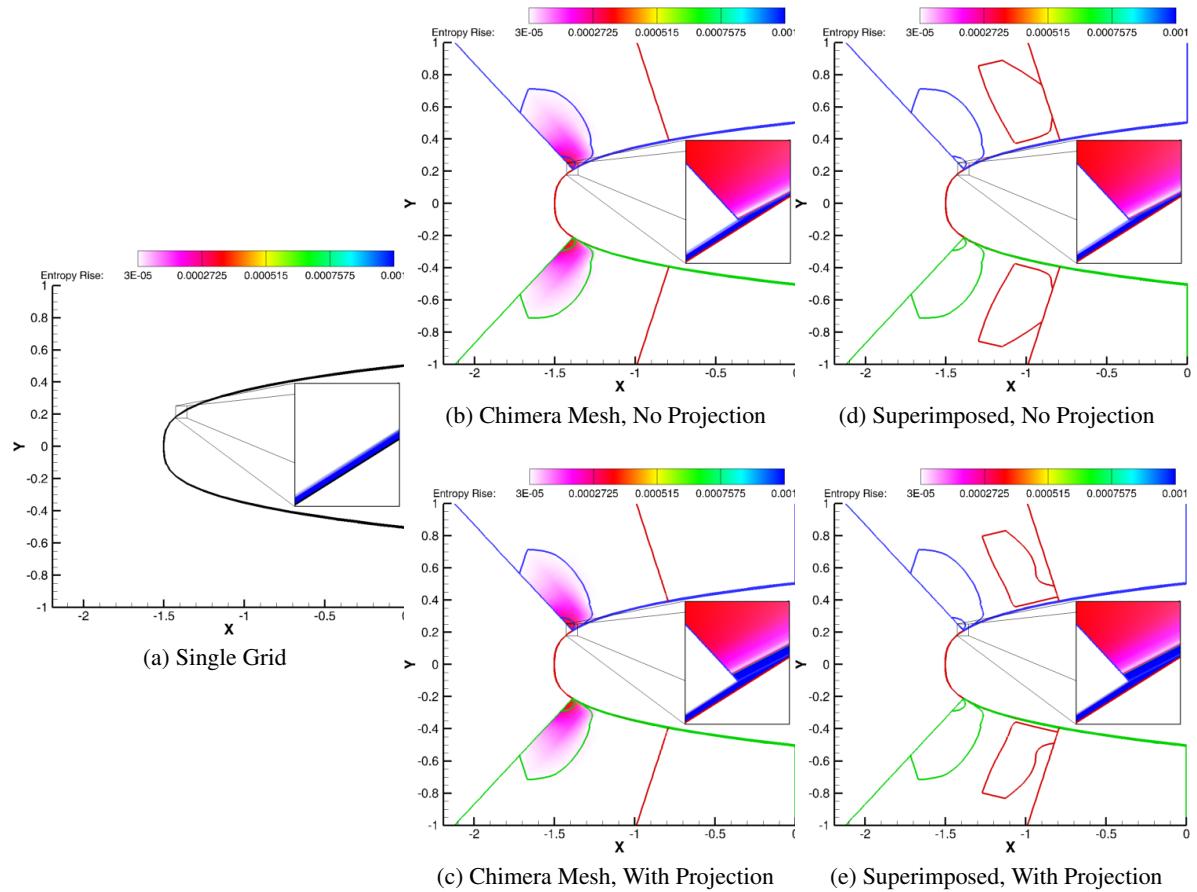


Figure 4.73: Finite Volume Entropy Rise Contours on the Generic Nose Section (Single Mesh - Black Lines, Chimera Mesh - Gray Lines)

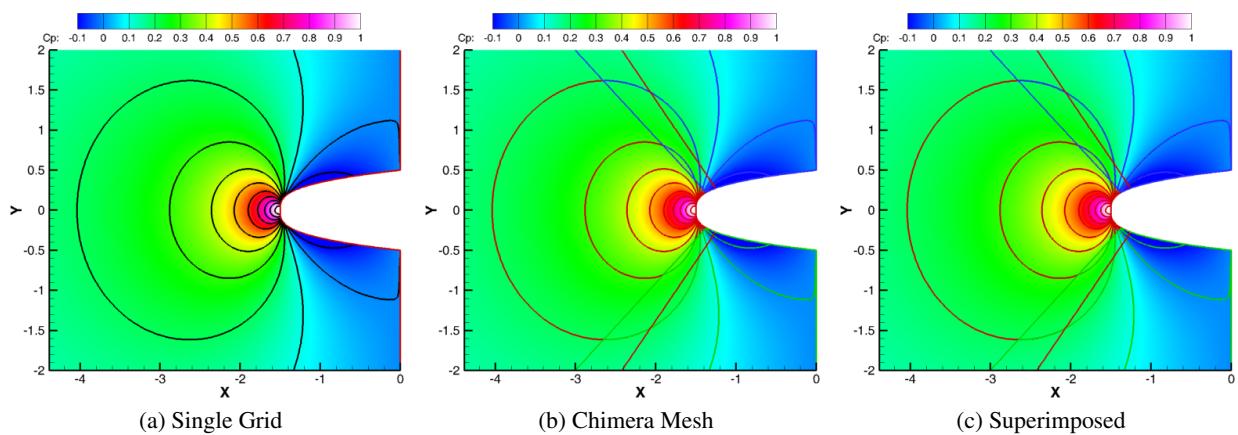


Figure 4.74: Discontinuous Galerkin C_p Contours on the Generic Nose Section

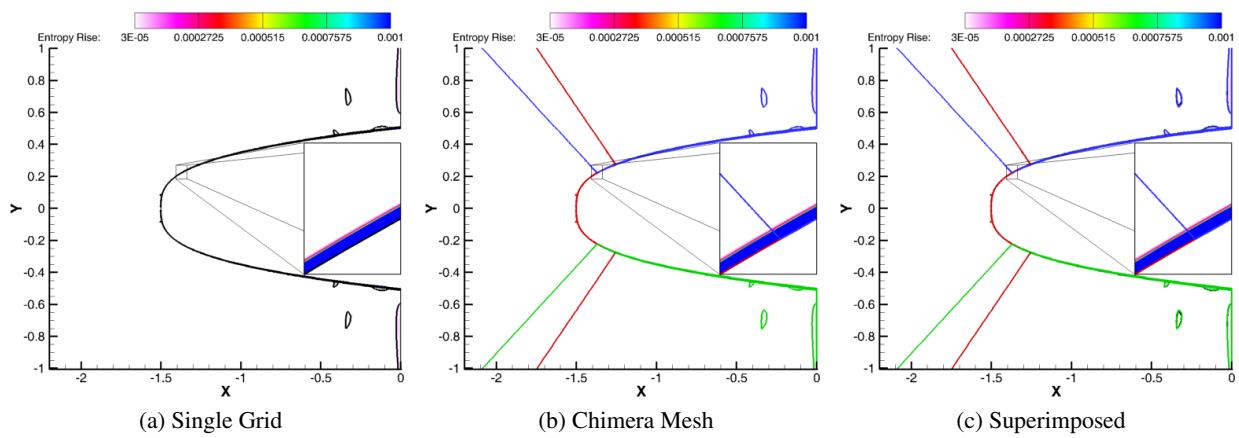


Figure 4.75: Discontinuous Galerkin Entropy Rise Contours on the Generic Nose Section

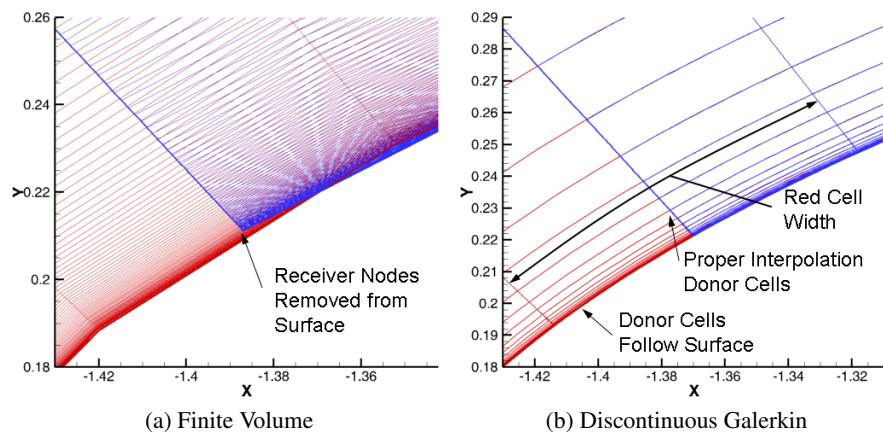


Figure 4.76: Chimera Mesh Overlapping Region

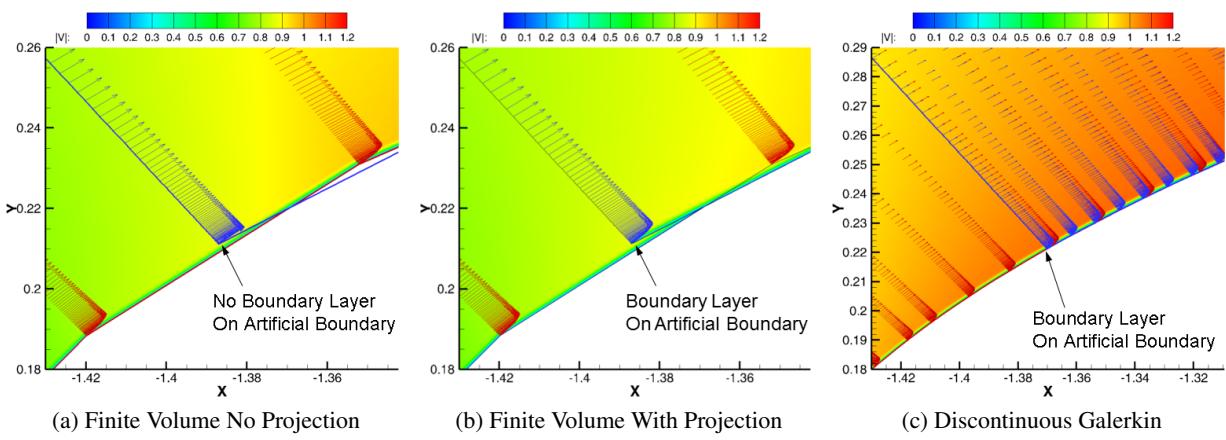


Figure 4.77: $|\vec{V}|$ Contour in the Overlapping Region

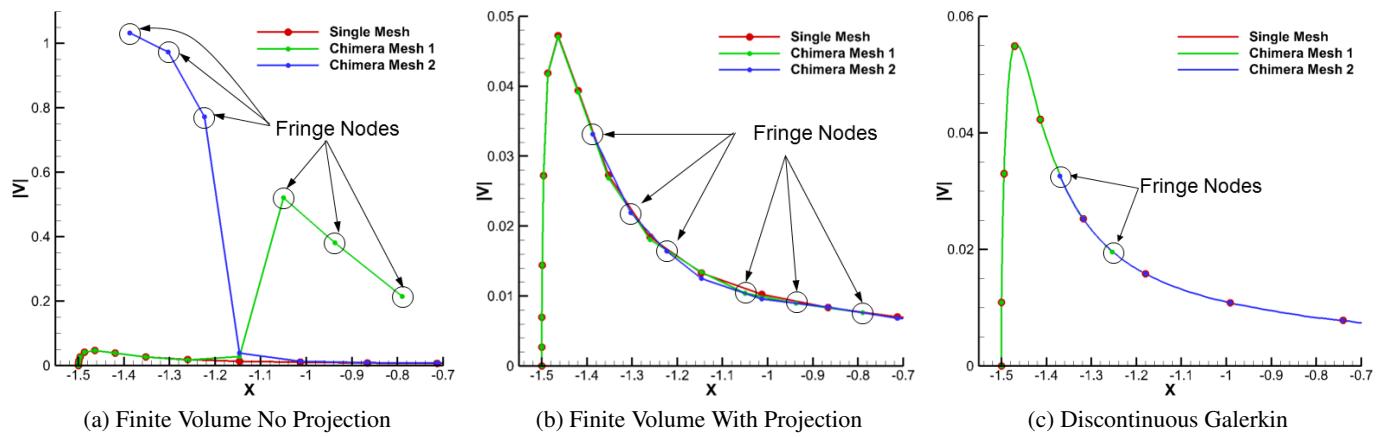


Figure 4.78: Velocity Magnitude One Point off the Wall

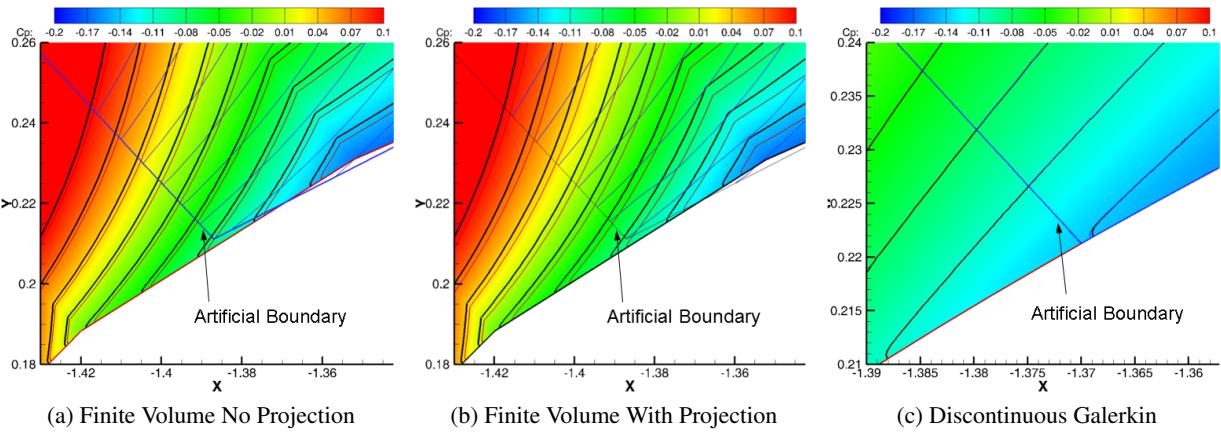


Figure 4.79: C_p Contours in the Overlapping Region (Single Mesh - Black Lines, Chimera Mesh - White/Gray Lines)

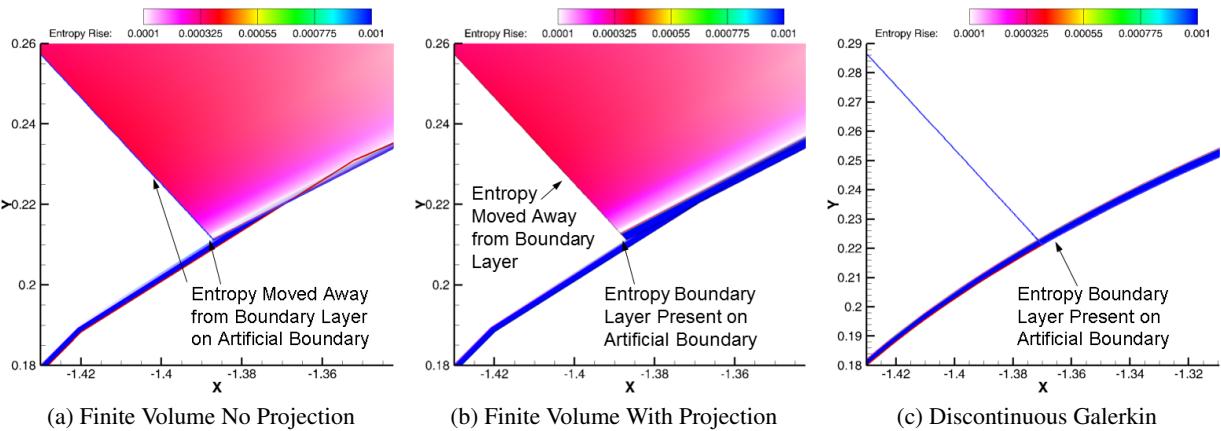


Figure 4.80: Entropy Rise Contours in the Overlapping Region

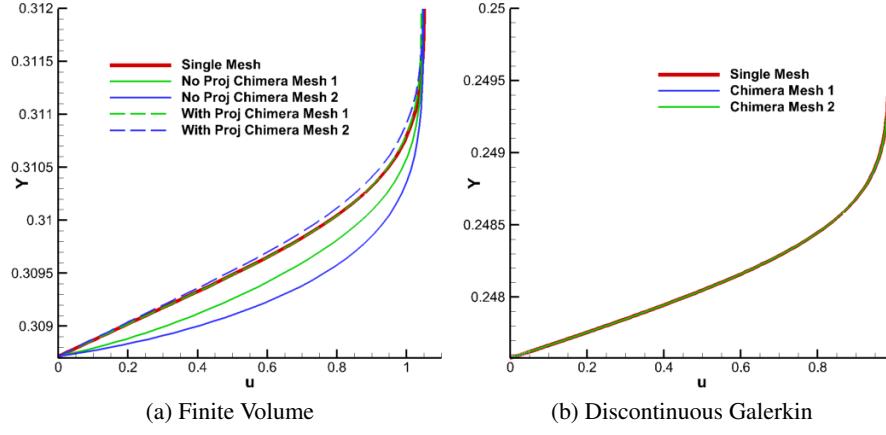


Figure 4.81: Boundary Layer on Common Grid Line

4.3.3.2 Steady Circular Cylinder

The $\text{Re}=40$ flow over a circular cylinder is a steady incompressible flow that has been studied extensively in the literature[169, 170, 171]. A reference Mach number of $M_\infty = 0.1$ was used for all calculations. Experimental and computational work in the literature reports length of the two steady vortices to be between 1.9 and 2.3 cylinder diameters. More recent calculations with compressible and incompressible solvers report a vortex length of approximately 2.2 cylinder diameters[171]. The mesh resolution was adjusted to achieve a wake length of 2.24 cylinder diameters with a minimal number of grid points.

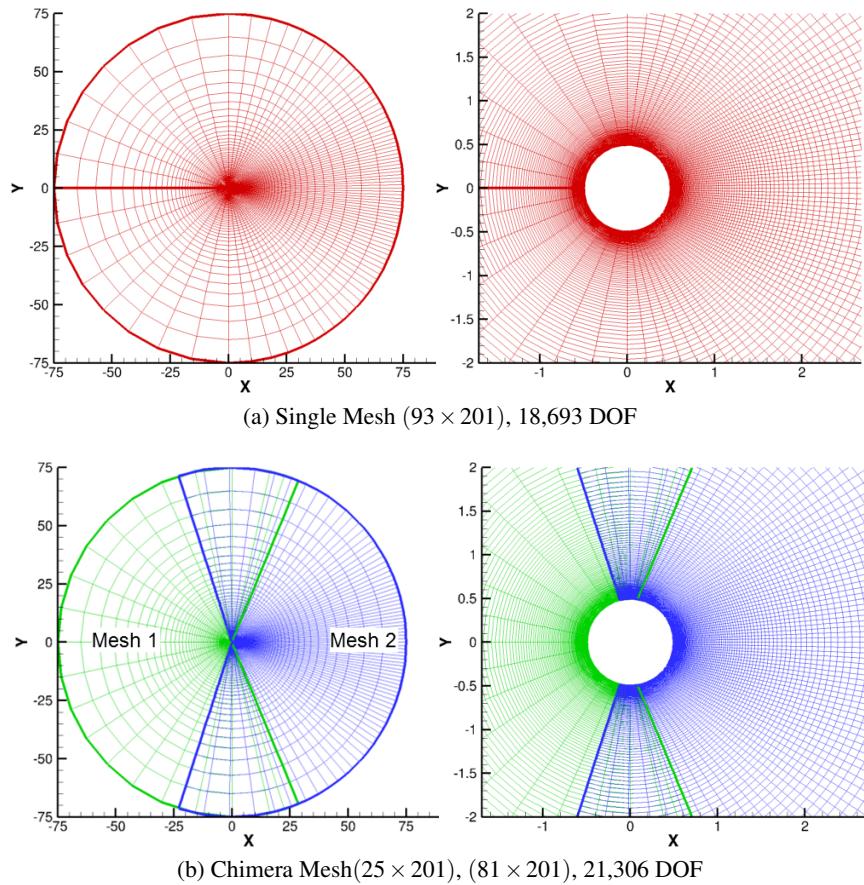


Figure 4.82: Finite Volume Meshes

A finite volume single grid and a Chimera mesh for circular cylinder simulations are shown in Fig. 4.82. The Chimera mesh in Fig. 4.82b was constructed from the single grid in Fig. 4.82a by adding nodes to create the overlapping region. By maintaining the node distribution of the single grid in the Chimera mesh, differences in the solutions on the two meshes can be attributed to the artificial Chimera boundary. The overlapping nodes in the blue mesh were intentionally placed near the center line of the cells in the green mesh in order to produce an inappropriate interpolation.

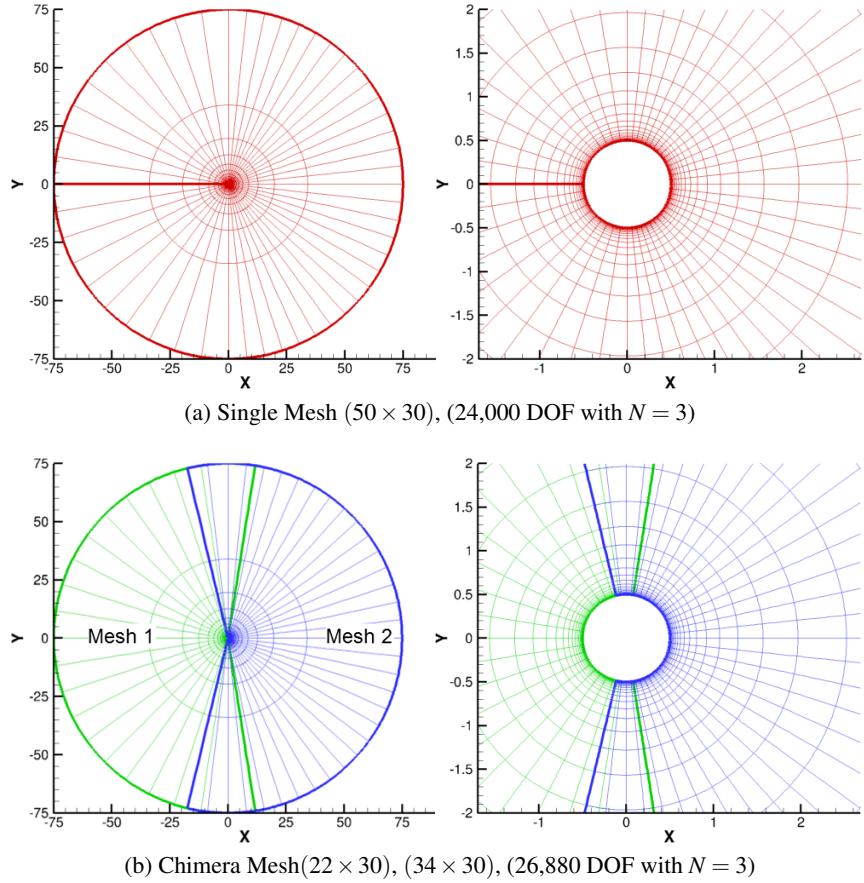


Figure 4.83: Discontinuous Galerkin $N_g = 3$ Polynomial Mapping Meshes

Figure 4.83 shows the discontinuous Galerkin single grid and Chimera mesh with cubic polynomial mappings, $N_g = 3$, for the circular cylinder. Again, the number of cells was tailored to have a comparable number of degrees of freedom to the finite volume grid. In a manner similar to the finite volume meshes, the Chimera mesh in Fig. 4.83b is constructed by splitting the single grid in Fig. 4.83a and adding nodes to create the overlapping regions. The artificial Chimera boundaries are positioned to coincide with the cell centers of the neighboring grid.

Solutions on the two finite volume meshes were computed using OVERFLOW with a 4th-order Roe scheme and a WENO limiter. A Riemann invariant boundary condition is imposed on the far-field boundary. The cylinder surface is assumed to be an adiabatic no-slip wall. Solutions on the Chimera mesh were obtained both with and without using the projection features of SUGGAR++. Contours of C_p from the finite volume solutions on the single grid and Chimera mesh are shown in Fig. 4.84. The Chimera mesh solution

without projection is superimposed on the single grid solution in Fig. 4.84d. There are notable differences in the C_p contour lines of the two solutions, in particular in the wake of the cylinder and in the overlapping region of the Chimera mesh. The Chimera mesh solution obtained with projection exhibits a greater deviation from the single mesh solution, in particular in the overlapping region as shown in Fig. 4.84e. Unlike the contours in C_p , the entropy rise contours from the Chimera mesh solution without projection shown in Fig. 4.85 exhibit a significant difference between the solution obtained on the single grid and Chimera mesh. Again, the Chimera mesh solution with projection further deviates from the single mesh solution as shown in Fig. 4.85e.

The flow fields on the discontinuous Galerkin meshes were obtained using a 4th-order, $N = 3$, formulation. Total temperature and pressure are imposed on the farfield boundary at $x < 0$, and a constant back pressure is imposed on the farfield for $x > 0$. Contours of C_p obtained using the discontinuous Galerkin scheme on the single grid and Chimera mesh are shown in Fig. 4.86. The two solutions are indistinguishable as is illustrated in Fig. 4.86c where the two solutions are superimposed. Similarly, the entropy rise contours shown in Fig. 4.87 do not exhibit a notable difference between solutions obtained with the single grid and the Chimera mesh.

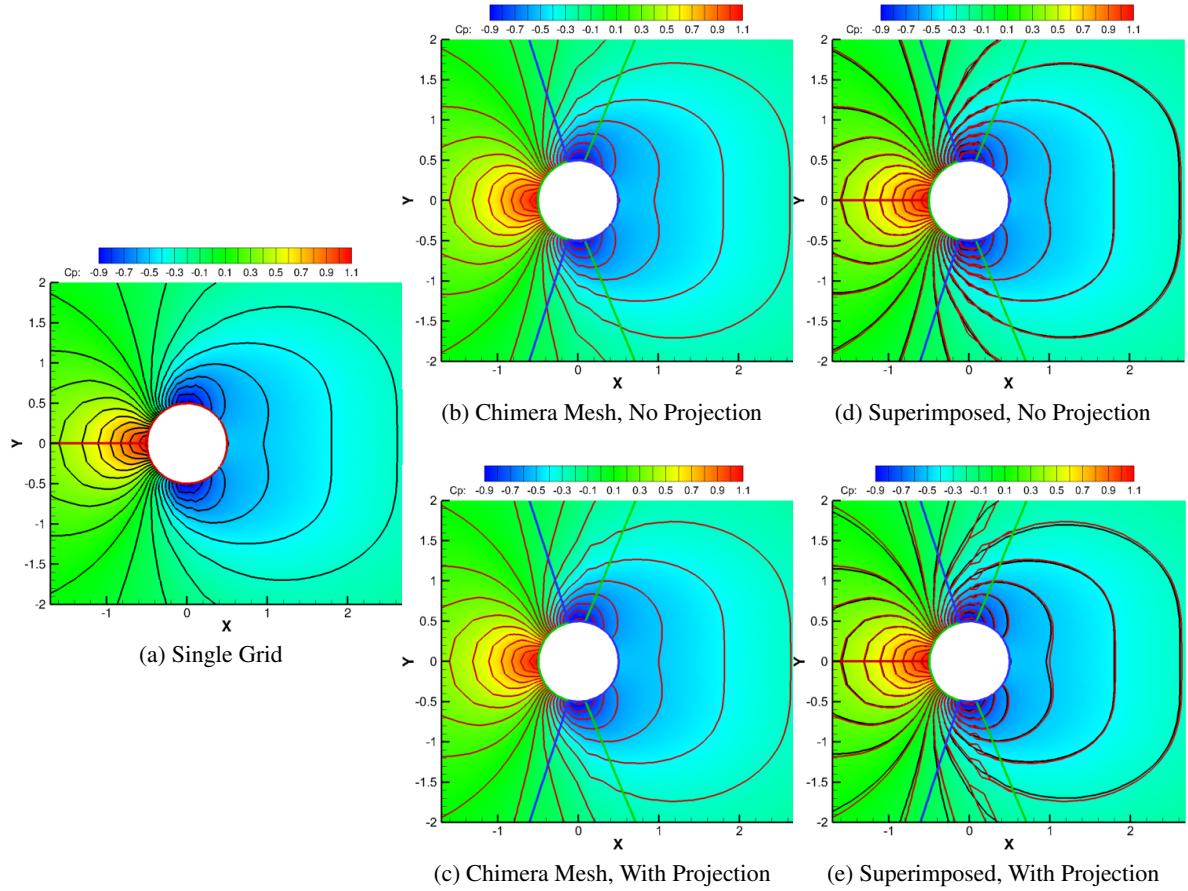


Figure 4.84: Finite Volume C_p Contours on the Circular Cylinder (Single Mesh - Black Lines, Chimera Mesh - White Lines)

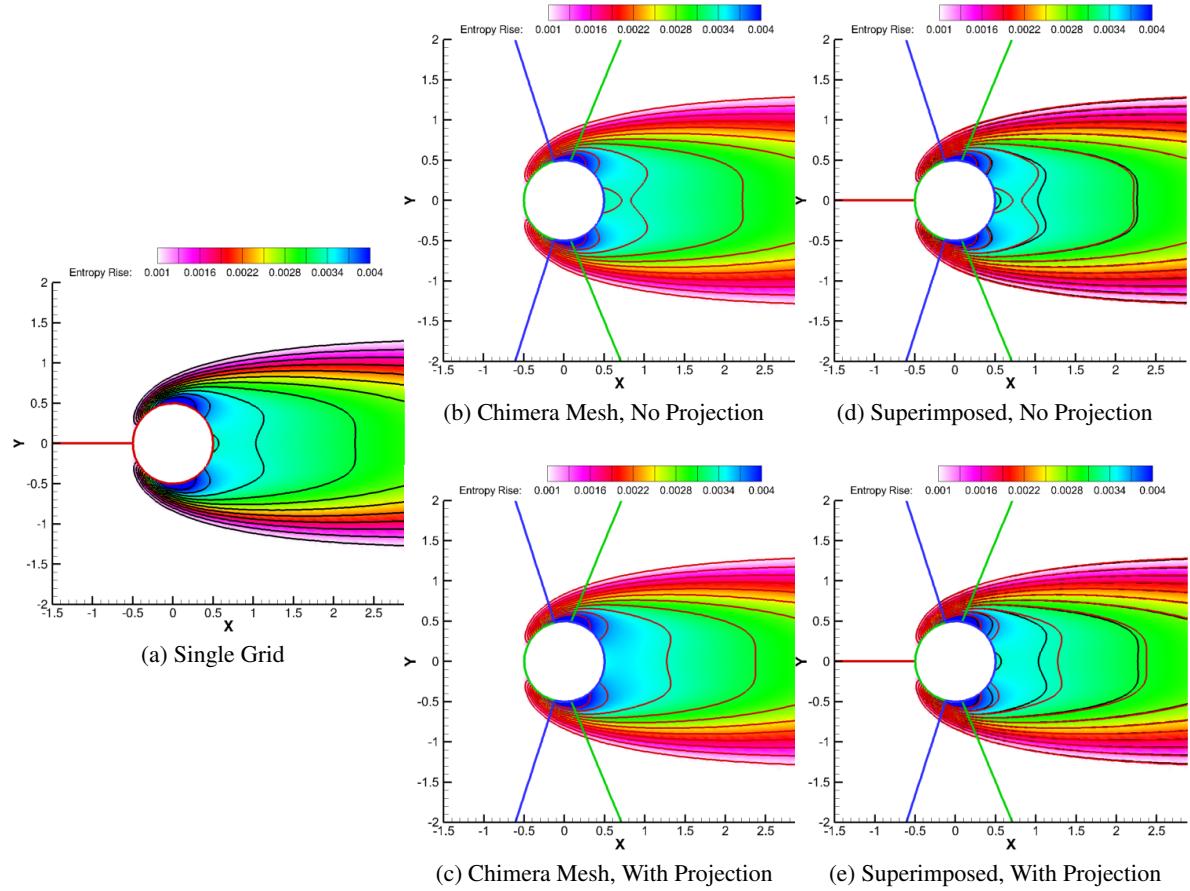


Figure 4.85: Finite Volume Entropy Rise Contours on the Circular Cylinder (Single Mesh - Black Lines, Chimera Mesh - Gray Lines)

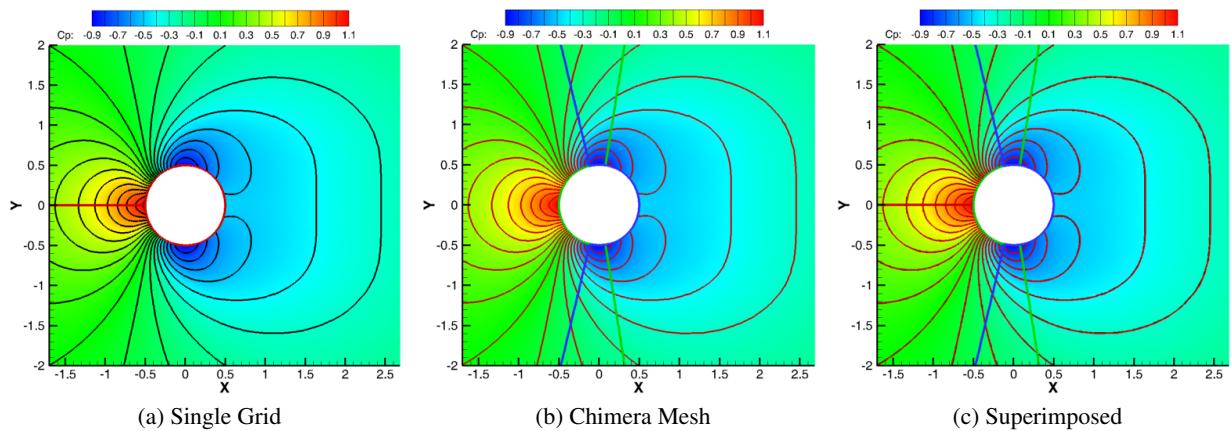


Figure 4.86: Discontinuous Galerkin C_p Contours on the Circular Cylinder ($N = 3$)

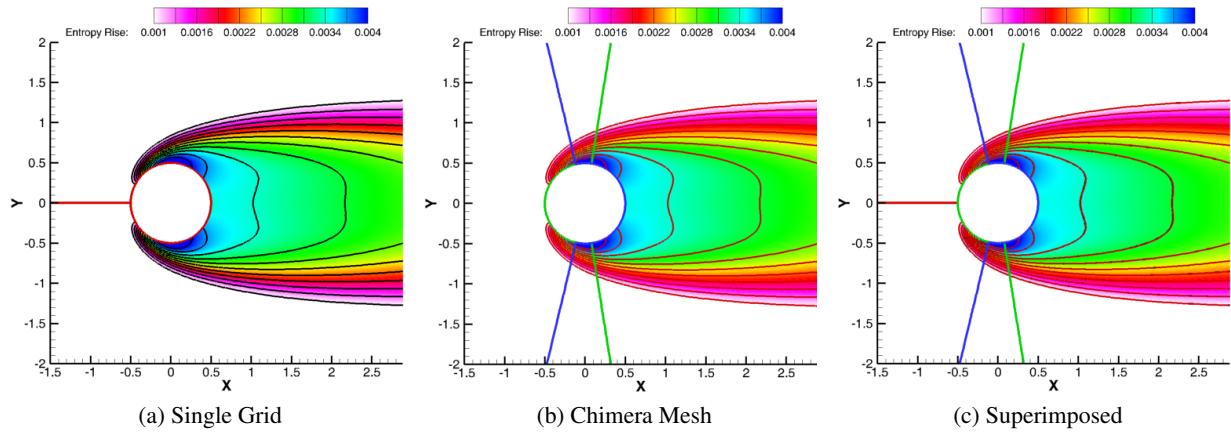


Figure 4.87: Discontinuous Galerkin Entropy Rise Contours on the Circular Cylinder ($N = 3$)

Streamlines are used to visualize the two steady vortices in the wake of the cylinder in Fig. 4.88 for both the OVERFLOW and discontinuous Galerkin calculations, and the length of the wake region is tabulated in Table 4.2. The length of vortices in the solution from finite volume Chimera mesh without projection is slightly longer than that obtained on the single grid. However, the finite volume Chimera mesh solution with projection is significantly shorter than the single mesh solution. The length of the vortices is nearly identical in the discontinuous Galerkin solutions.

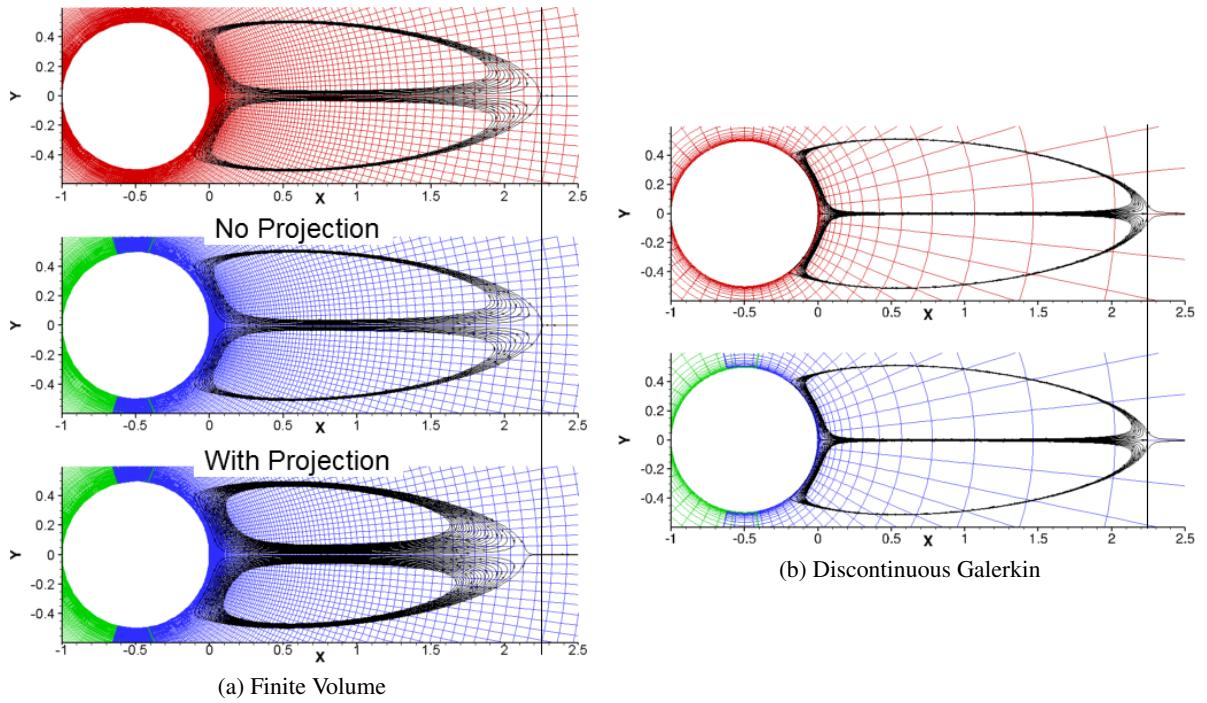


Figure 4.88: Streamlines of Wake Separation Bubble

Table 4.2: Cylinder Wake Vortex Length

	Single Grid	Chimera Mesh
Finite Volume	2.246	2.260 (No Proj.) 2.152 (With Proj.)
Discontinuous Galerkin	2.245	2.246

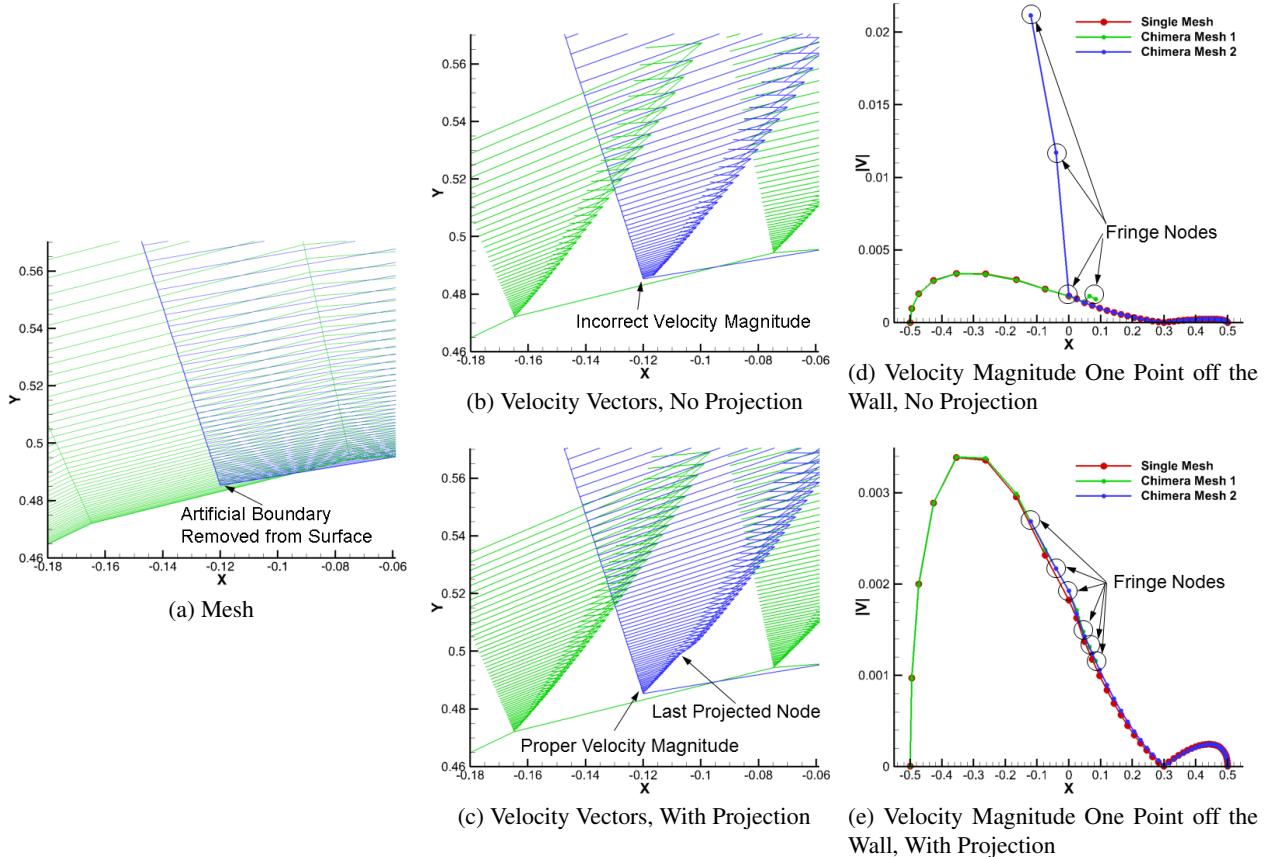


Figure 4.89: Finite Volume Chimera Mesh Overlapping Region

Aside from the overlapping region in the Chimera mesh, the single grid and Chimera mesh share the same node distribution throughout the computational domain. Thus, the discrepancies observed between the finite volume solutions on the two meshes can be attributed to the interpolation. The impact of the interpolation can be visualized by inspecting the flow field in the overlapping region shown in Fig. 4.89a. Without projection, the artificial Chimera boundary on the blue mesh receives information from cells too far removed from the surface in the green mesh due to the secant representation of the geometry in the finite volume Chimera mesh. This improper interpolation produces a boundary layer profile on the artificial Chimera boundary of the blue mesh with higher velocity magnitude than what is found in the boundary layer of the green mesh as shown in Fig. 4.89b. The increased velocity magnitude is particularly evident in Fig. 4.89d where the magnitude of the velocity vector at the first point off the wall in the Chimera mesh is compared to the velocity magnitude of the single grid. The projection feature of SUGGAR++ is able

to remove the high velocity magnitude at the first point off the wall as shown in Figs. 4.89c and 4.89e. However, the projection has introduced a discontinuity in the interpolated boundary layer profile as shown in Fig. 4.89c. This discontinuity coincides with the last point to be shifted towards the wall as part of the projection process. The interpolated velocity magnitude on point off the wall does compare more favorably with the single mesh solution, though differences are still notable.

The solutions on the two discontinuous Galerkin meshes exhibit few differences because the artificial Chimera boundary receives the properly interpolated values without corrections because the scheme utilizes curved elements to represent the geometry (See, Fig. 4.90a.). As a result, the boundary layer profiles on the two grids of the Chimera mesh agree well with each other as shown in Fig. 4.90b, and the velocity magnitude on the first point off the wall on the Chimera mesh also agrees well with that obtained using the single grid as shown in Fig. 4.90c.

Boundary layer profiles extracted from a grid line in the overlap region common to both the single and Chimera mesh are shown in Fig. 4.91. The finite volume solution on mesh 2 of the Chimera mesh agrees well with the single mesh solution. However, the boundary layer profile on mesh 1 of the Chimera mesh differs more significantly from the single mesh solution. The differences in the boundary layer profile between the solutions with and without projection are small. The DG solution exhibits no notable differences between the solutions on the single and Chimera meshes.

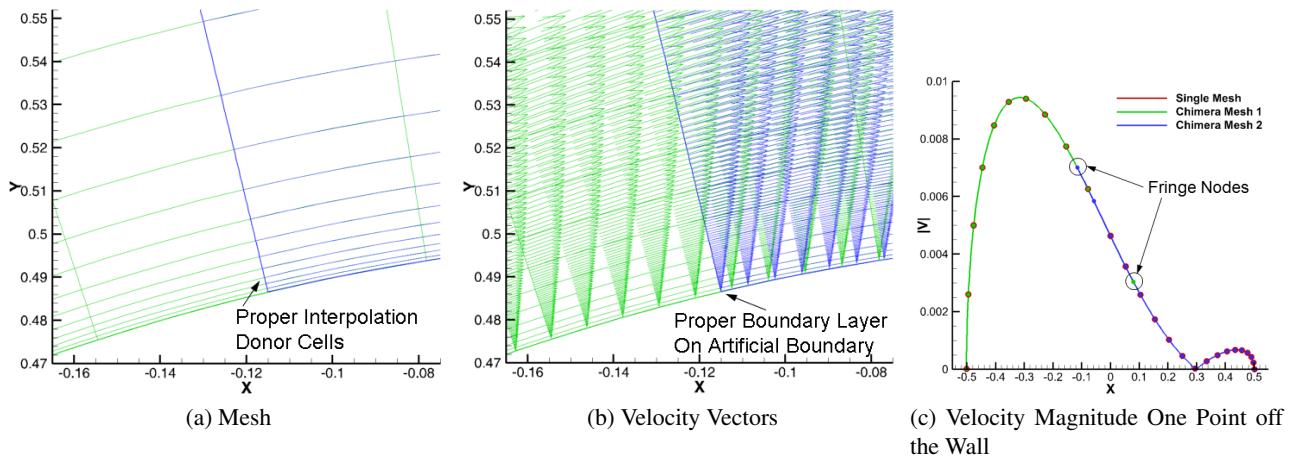


Figure 4.90: Discontinuous Galerkin Chimera Mesh Overlapping Region

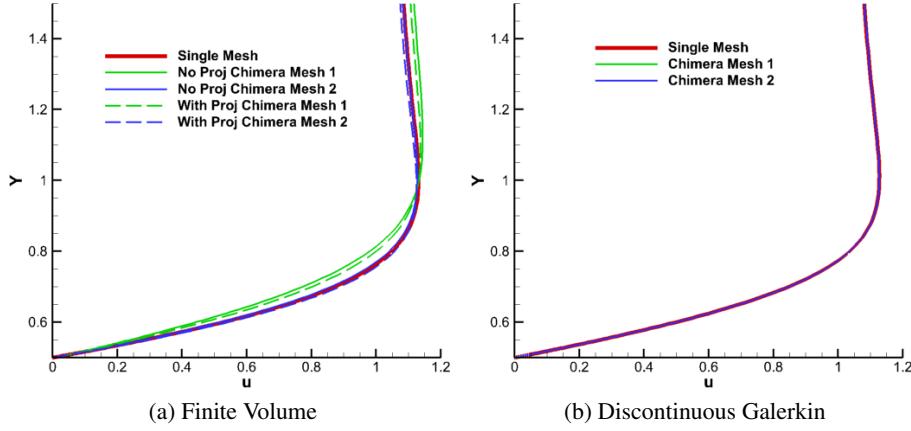


Figure 4.91: Boundary Layer on Common Grid Line

4.3.3.3 Summary

The DG-Chimera scheme has been applied to a set of meshes with curved geometries for viscous flow calculations. The scheme resolves the interpolation issue of overlapping linear meshes on curved geometries, which are used for Finite Volume and Finite Difference Chimera schemes. The DG-Chimera scheme obtains the proper interpolation by accurately representing the geometry using curved cells in lieu of techniques such as the projection method in PEGASUS5 and SUGGAR++.

The Nose Section geometry was used to demonstrate that the DG-Chimera scheme does not introduce an entropy rise of the same magnitude that was observed in the flow computed with a Finite Volume discretization. While the projection method improved the boundary layer profile on the Chimera mesh for the Finite Volume calculations, it did not resolve the issue of the entropy generated on the artificial boundaries. The projection method does not significantly improve the boundary layer profile for the circular cylinder, but it does significantly change the separation length of the steady vortices in the wake of the cylinder. The flow field computed with the DG-Chimera scheme agrees well with the single grid flow field.

4.4 Parallel Performance

This section compares the computational cost of using a traditional explicit formulation of the artificial boundaries with the implicit formulation of the artificial boundaries given in Section 4.1.1. All results

presented here were computed using a cluster of 10 computers, each with a single Intel Core 2 Duo 3.0 GHz processor and 8 GB of RAM. The computers are interconnected using an Ethernet network. This relatively slow Ethernet network favors the traditional explicit artificial boundary formulation that only passes boundary information while computing the right hand side vector of the implicit system relative to the implicit artificial boundary formulation that also communicates boundary information as part of the GMRES iterative matrix solver. Only one core per processor is used in this analysis to maximize network communication and further favor the explicit artificial boundary formulation.

The solver is parallelized for distributed memory calculations using the Message Passing Interface (MPI) standard.[172] A shared memory model using the Boost Thread[173] library is also included in the solver. The fluxes and Jacobians associated with each computational grid are computed concurrently in parallel threads when multiple grids are assigned to a processor. Currently, there is no mechanism in the solver to compute fluxes and Jacobians for multiple grids using a single thread. Hence, two threads are used for the calculation when a Chimera mesh consisting of two grids is assigned to a single processor. Thus, the execution time of the solver in this study with a Chimera mesh that consists of two grids should, ideally, have the same execution time on a single processor as the execution time using two processors with one grid per processor.

High-order solutions are obtained via polynomial sequencing, i.e., solutions with higher-order of approximation are initialized with the converged solution of one order of accuracy lower. The 1st-order accurate calculation, $N = 0$, is initialized with freestream quantities. The parallel performance is assessed using 1, 2, 4, and 8 processors. The number of processors is held constant as the order of the approximation is increased from $N = 0$ to $N = 3$.

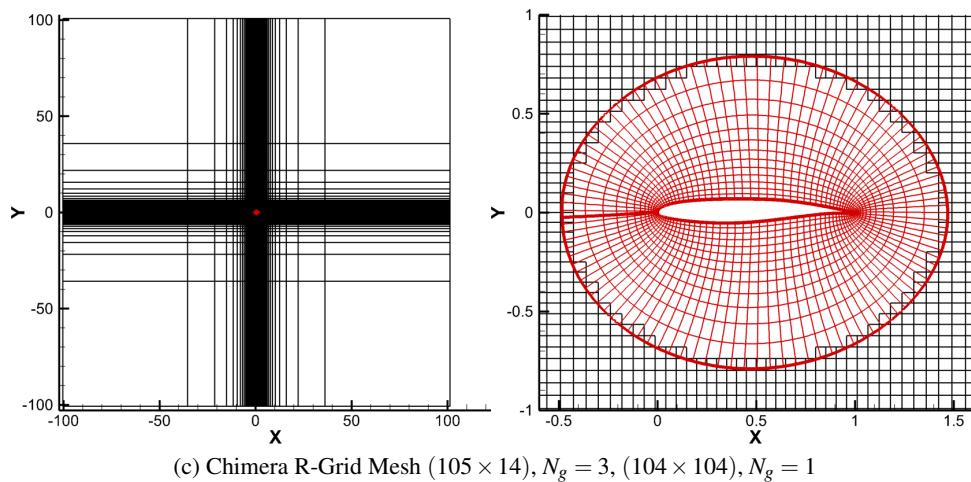
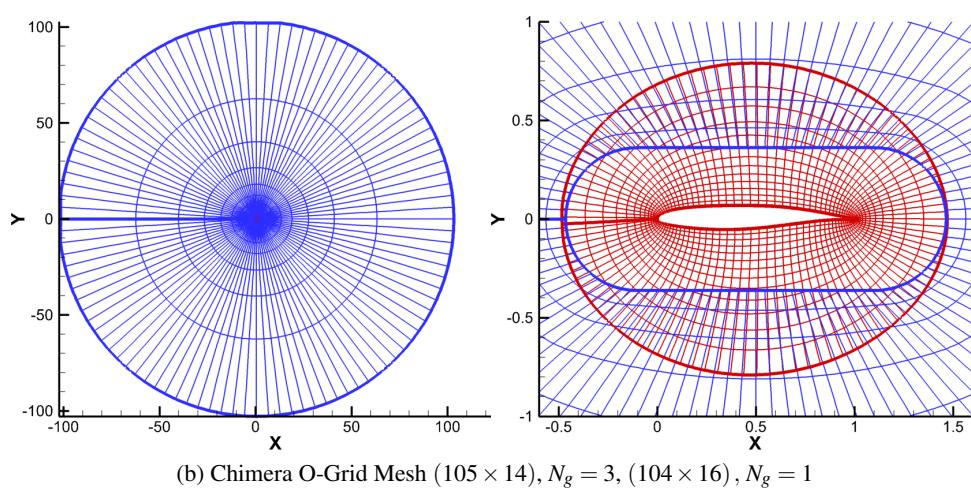
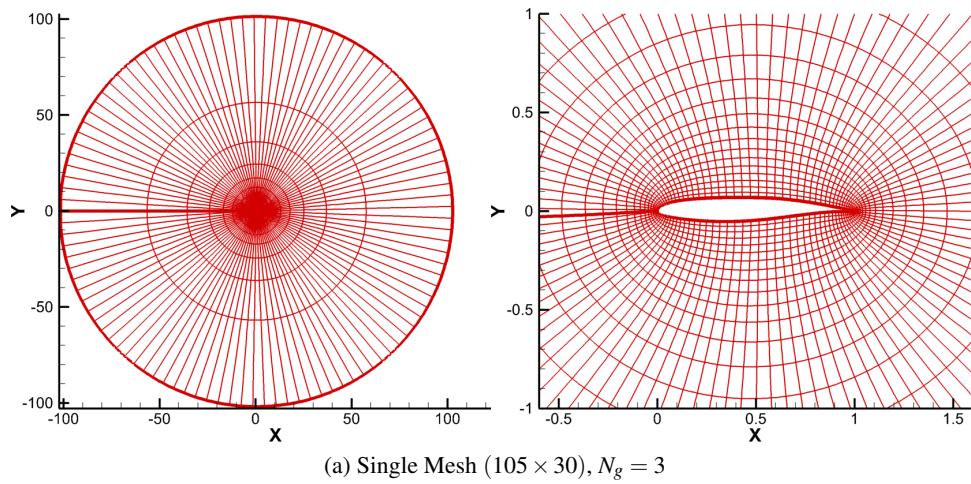


Figure 4.92: SKF 1.1 Airfoil Meshes

Table 4.3: SKF 1.1 Airfoil Degrees of Freedom

	N = 0	N = 1	N = 2	N = 3
Single Grid	3,150	12,600	28,350	50,400
Chimera O-Grid	3,134	12,536	28,206	50,144
Chimera R-Grid	12,286	49,144	110,574	196,576

4.4.1 SKF 1.1 Airfoil

Three different meshes shown in Fig. 4.92 are used to compute the inviscid flow associated with the SKF 1.1 airfoil[146] at $M_\infty = 0.4$ and $\alpha = 2.5^\circ$. The surface of the airfoil is imposed with a slip wall boundary condition and the farfield is imposed with a Riemann invariant[120] boundary condition. The trailing edge of the airfoil is rounded to close it. The center section of the single O-Grid mesh shown in Fig. 4.92a is used in the subsequent Chimera meshes shown in Figs. 4.92b and 4.92c. The Chimera mesh shown in Fig. 4.55b uses another O-Grids as the background grid, while the Chimera grid in Fig. 4.92c uses a rectangular background grid (R-Grid) with a hole cut out of it. The O-Grid mesh with the airfoil geometry consists of cells with cubic, $N_g = 3$, polynomial mappings, while the two background meshes in the Chimera meshes use cell linear, $N_g = 1$, polynomial mappings. The total number of degrees of freedom (DOF) for each order of accuracy are shown in Table 4.3. The single grid and O-Grid Chimera mesh have a comparable number of DOF, and the R-Grid is approximately four times larger. The *CFL* number is increased with each Quasi-Newton iteration using the formula given in Eq. 3.97 with $CFL^0 = 10$, and a solution is considered converged when the L^2 -norm of $\mathcal{R}(Q)$ drops below 5×10^{-10} .

Parallel performance is often assessed by measuring the average execution time of a Quasi-Newton iteration.[174] The average iteration time is typically measured by executing the solver for a fixed number of iterations. Using this metric, the explicit artificial boundaries exhibit a super-linear iteration speedup as shown in Fig. 4.93. The super-linear speedup is a result of GMRES solver converging in fewer iterations when solving the smaller systems of equations on each processor that result from the parallel partitioning. Unfortunately, the iteration speedup does not provide a complete picture of the parallel performance of steady state calculations. This metric is only suitable when the parallel algorithm is identical to the serial algorithm, which is not true for the explicit artificial boundaries. Instead, a more appropriate metric is the solution speedup (See, Fig. 4.94), which is based on the total execution time to reach a converged solution

regardless of the number of Quasi-Newton iterations it takes to reach a converged solution. The solution speedup with explicit artificial boundaries does not exhibit the same super-linear speedup as the iteration speedup. The speedup for the single grid and two processors drops below one, indicating that the parallel execution time is longer than the serial execution time. A similar behavior occurs for the O-Grid and R-Grids calculations using 4 and 8 processors.

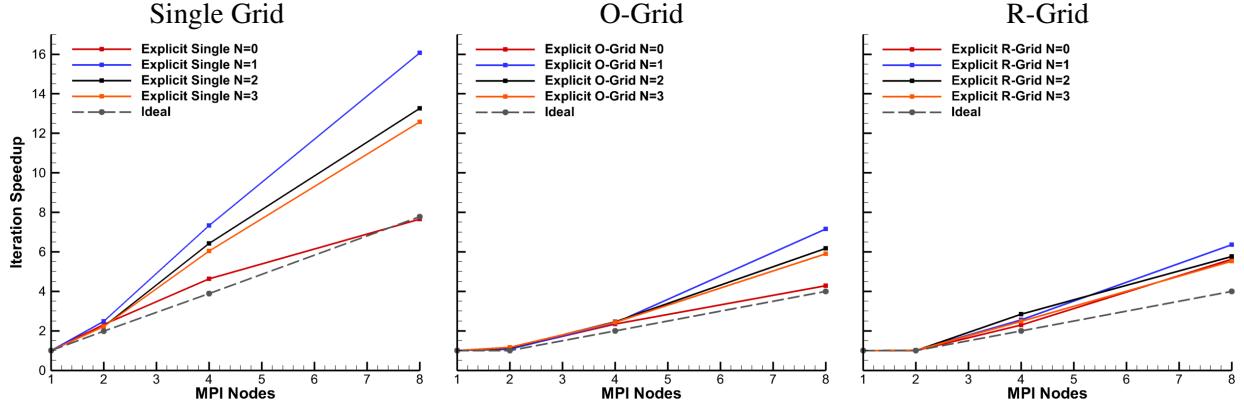


Figure 4.93: SKF 1.1 Airfoil Iteration Speedup with Explicit Artificial Boundaries

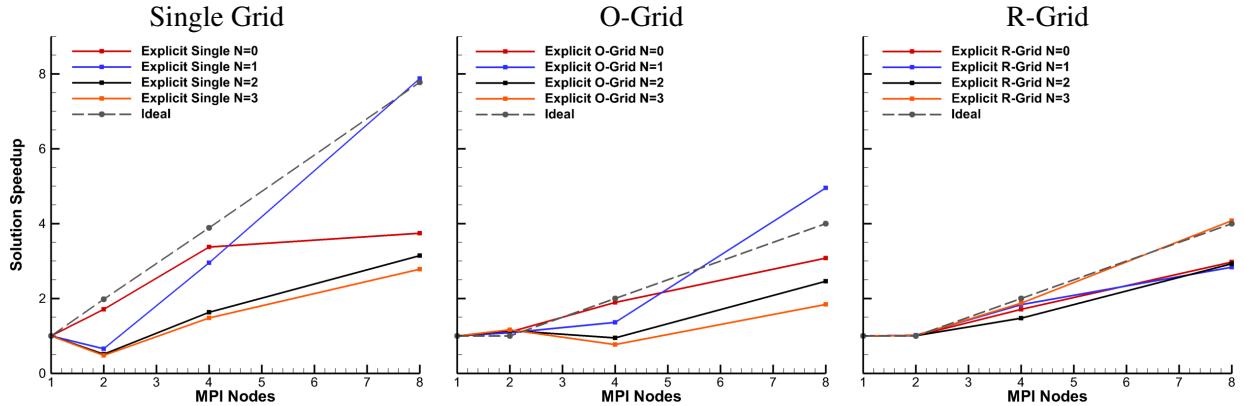


Figure 4.94: SKF 1.1 Airfoil Solution Speedup with Explicit Artificial Boundaries

This is a result of the increased number of Quasi-Newton iterations required to reach the converged solution as shown in Fig. 4.95. The columns of line plots in Fig. 4.95 correspond to the three grids, and the order of the approximation is associated with the rows. The lines in each plot correspond to an increasing number of processors. Note that the single mesh solution using explicit artificial boundaries is equivalent to a solver that does not include the matrix elements associated with the periodic boundary condition. In addition, the convergence history for one processor, $MPI = 1$, and two processors, $MPI = 2$,

are indistinguishable for the O-Grid and R-Grid as these Chimera meshes each consist of two grids. In general, the number of Quasi-Newton iterations tends to increase as the number of processors increases.

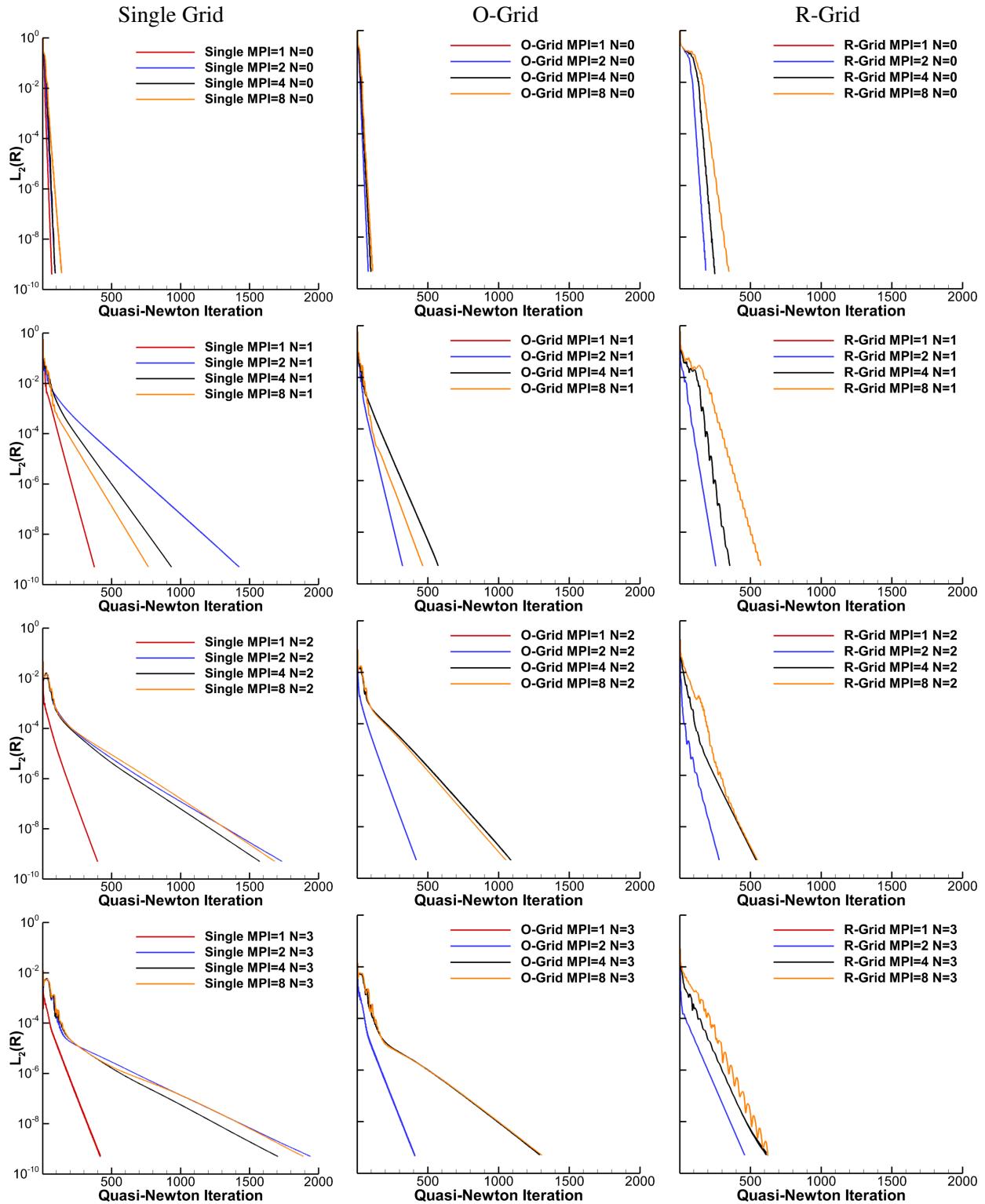


Figure 4.95: SKF 1.1 Airfoil Convergence History with Explicit Artificial Boundaries

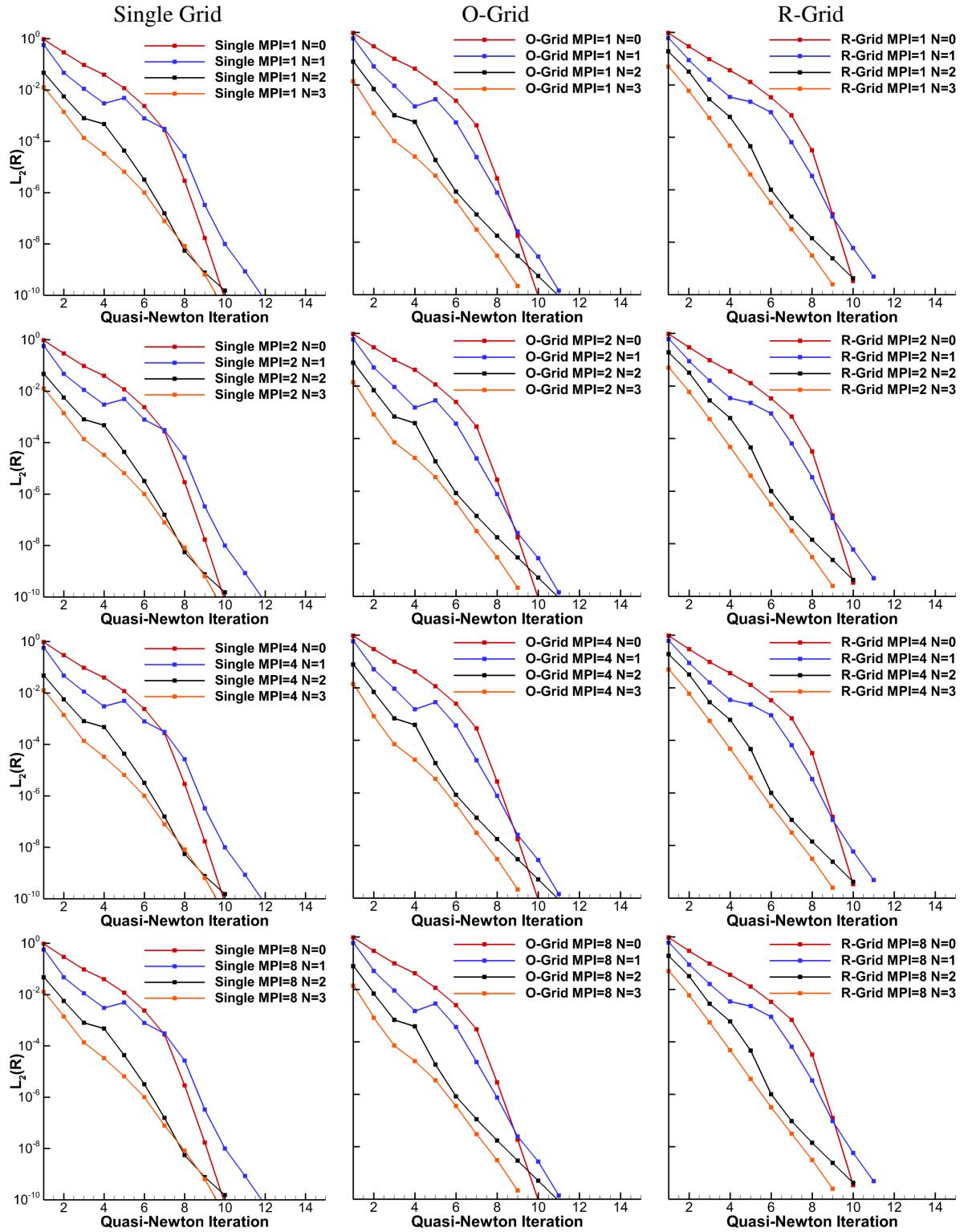


Figure 4.96: SKF 1.1 Airfoil Convergence History with Implicit Artificial Boundaries

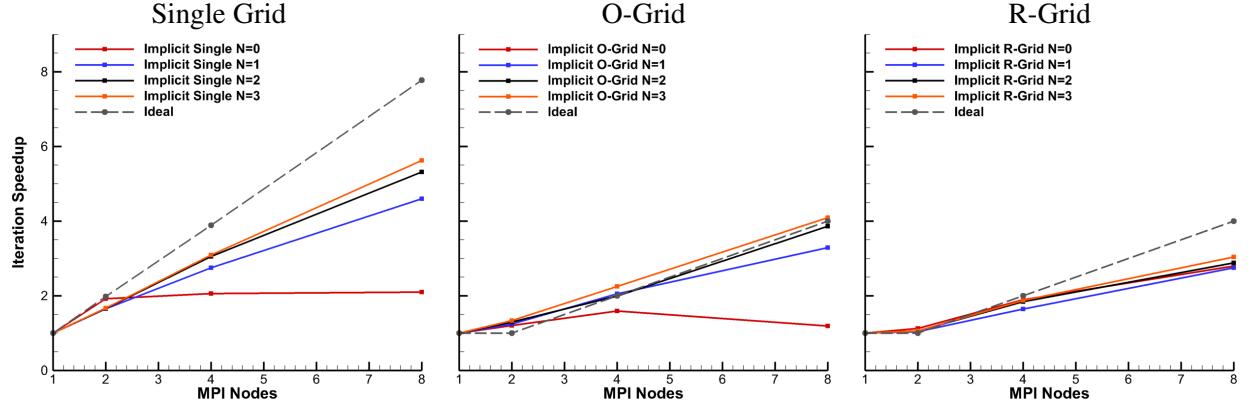


Figure 4.97: SKF 1.1 Airfoil Iteration Speedup with Implicit Artificial Boundaries

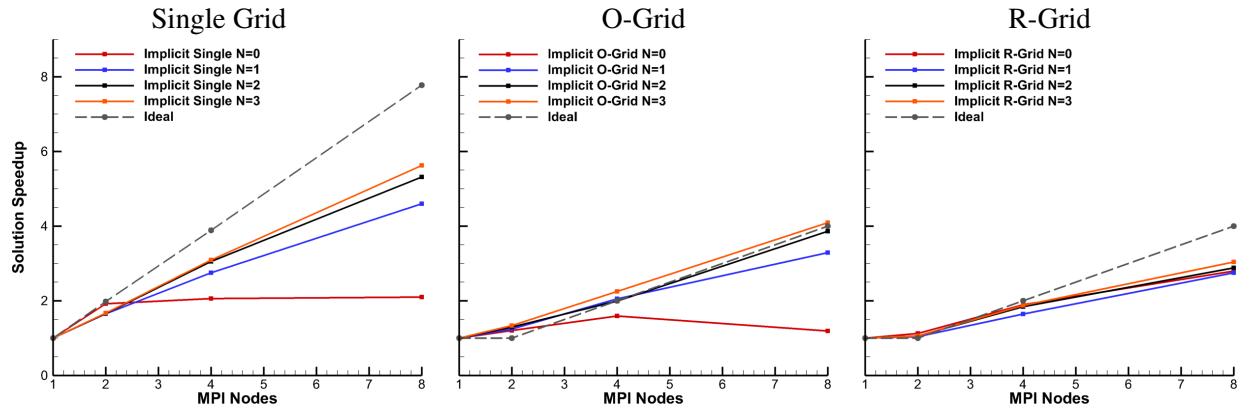


Figure 4.98: SKF 1.1 Airfoil Solution Speedup with Implicit Artificial Boundaries

Unlike the explicit artificial boundaries, the implicit artificial boundary formulation maintains the serial algorithm as demonstrated by the convergence history shown in Fig. 4.96. The rows in Fig. 4.96 correspond to increased number of processors and the lines in each plot correspond to the order of the approximation. The convergence history is nearly identical for each grid as the number of processors is increased. Hence, because the number of Quasi-Newton iterations remains the same, the iteration speedup and solution speedup are also the same as shown in Figs. 4.97 and 4.98.

The total execution time using explicit and implicit artificial boundaries is shown in Fig. 4.99, where the dashed lines are the ideal execution times based on linear speedup and the solid lines is the actual speedup. Here, it is evident that the execution time with implicit artificial boundaries is significantly lower than the execution time with explicit artificial boundaries. In addition, the execution times with implicit artificial boundaries are in general closer to the ideal execution time than the execution times with explicit artificial

boundaries. The ratio of the execution time with explicit and implicit boundaries is shown in Fig. 4.100. In general, the execution time with implicit artificial boundaries is an order of magnitude faster for $N \geq 1$. The speedup with implicit artificial boundaries gets up to nearly 60 times for the Single grid and the O-Grid meshes relative to the execution time with explicit artificial boundaries.

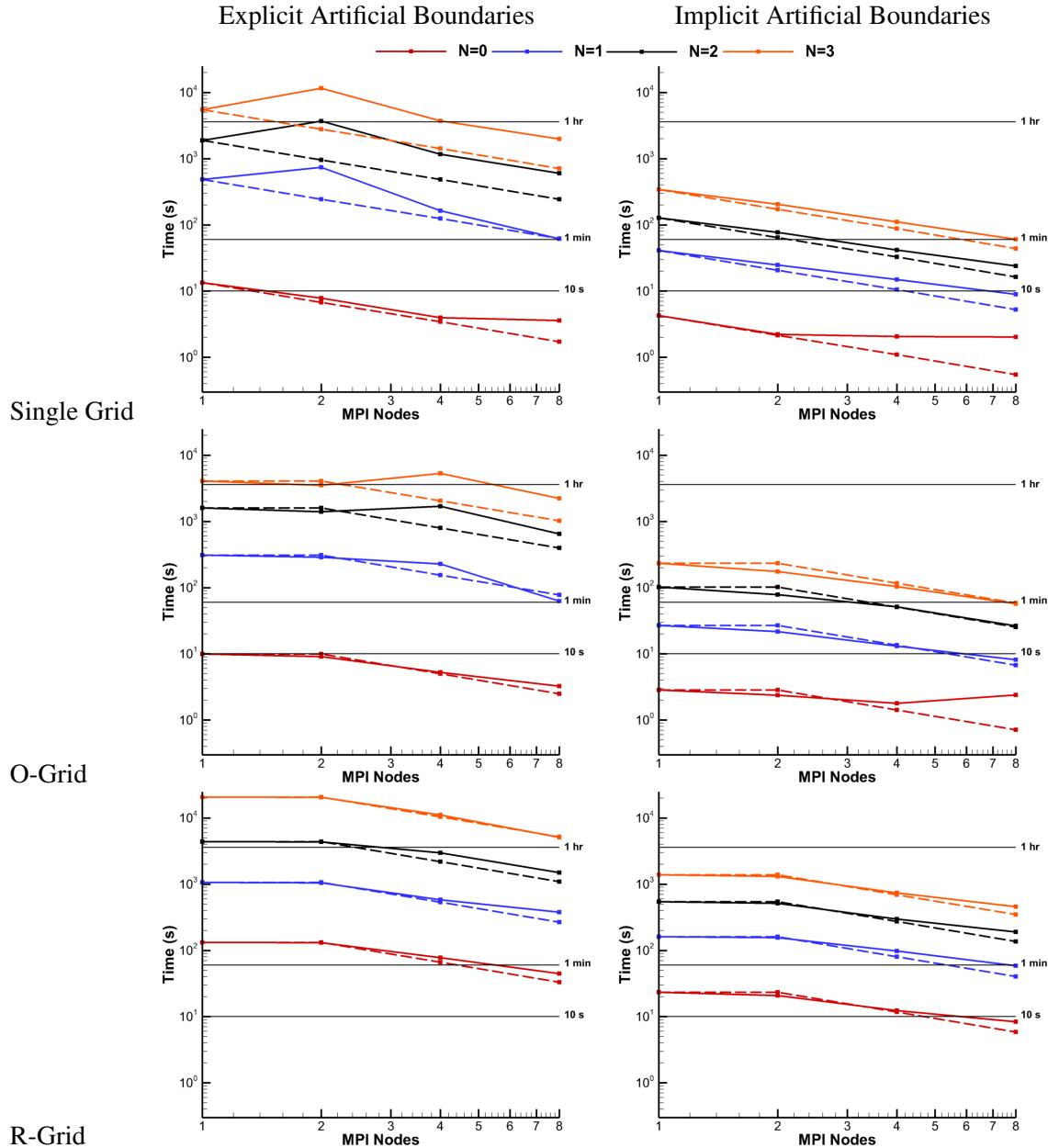


Figure 4.99: SKF 1.1 Airfoil Solution Time (Dashed Lines show Ideal Times)

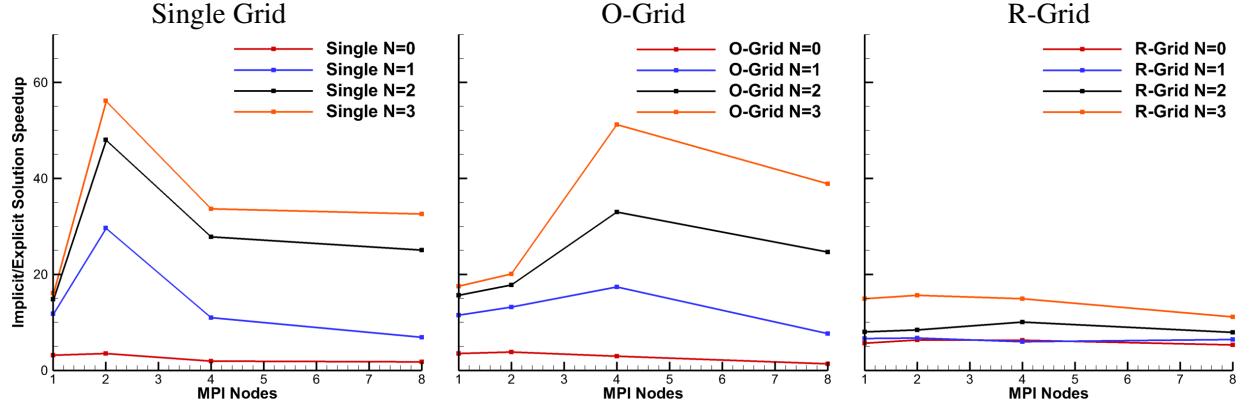


Figure 4.100: SKF 1.1 Airfoil Solution Speedup with Implicit Artificial Boundaries Relative to Explicit Artificial Boundaries

4.4.2 Steady Circular Cylinder at $Re=40$

Three meshes used to compute a viscous flow about a circular cylinder at $M_\infty = 0.2$ and $Re = 40$ are shown in Fig. 4.101. The corresponding degrees of freedom count for each grid and order of approximation is given in table 4.4. The surface of the cylinder is imposed with an adiabatic wall boundary condition and the farfield is imposed with a Riemann invariant[120] boundary condition. The single mesh shown in Fig. 4.101a consists solely of cells with cubic ($N_g = 3$) polynomial mappings. Similar to the SKF 1.1 airfoil, the region defining the cylinder for the single mesh is used to define the cylinder in the two Chimera meshes shown in Figs. 4.101b and 4.101c. Both the background O-Grid and R-Grid in the Chimera meshes are made up of cells defined with linear ($N_g = 1$) polynomial mappings. Minimal overlap is required to implement the artificial boundaries that connect the cylinder mesh with the background meshes. The 2nd-order accurate solutions ($N = 1$) are initialized with freestream quantities, and the subsequent increase in order of accuracy is initialized with the converged solution of one less order of accuracy.

Similar to the SKF airfoil results, the iteration speedup (See, Fig. 4.102) exhibits the ideal linear speedup, while the solution speedup shown in Fig. 4.103 in general exhibits poor parallel performance. The speedup for the R-Grid is less than one when the number of processors is four and above. Hence, the parallel execution time is longer than the serial execution time. The convergence history for the three meshes using explicit artificial boundaries is shown in Fig. 4.104, where the order of approximation is fixed for each row; each line in the plots is associated with the number of processors. Note that the scale for the number

of Quasi-Newton iterations goes up to 2,600 iterations for the Single and O-Grid meshes and 22,000 for the R-Grid. The explicit artificial boundaries required the use of the Quasi-Newton solver with a fixed CFL number. Attempts to allow the CFL number to increase caused the convergence residual to “flat line”. This is the major cause for the large number of Quasi-Newton iterations when using explicit artificial boundaries.

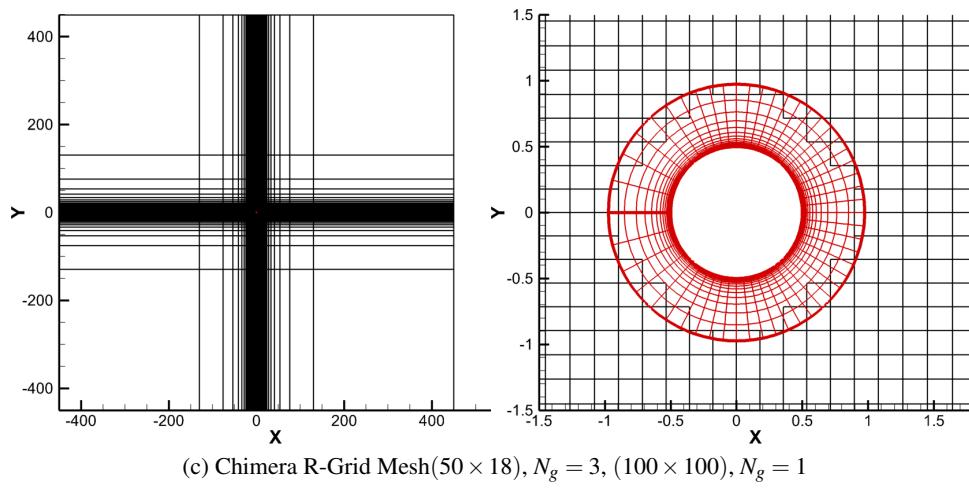
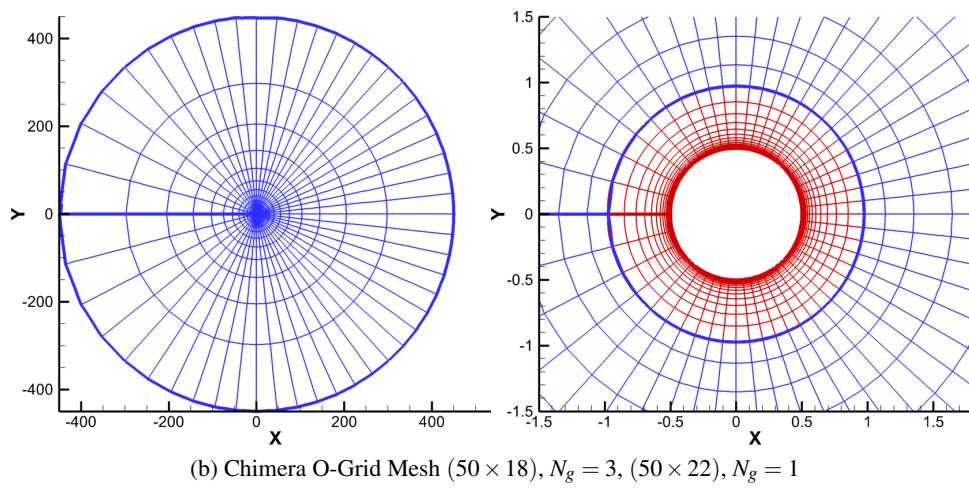
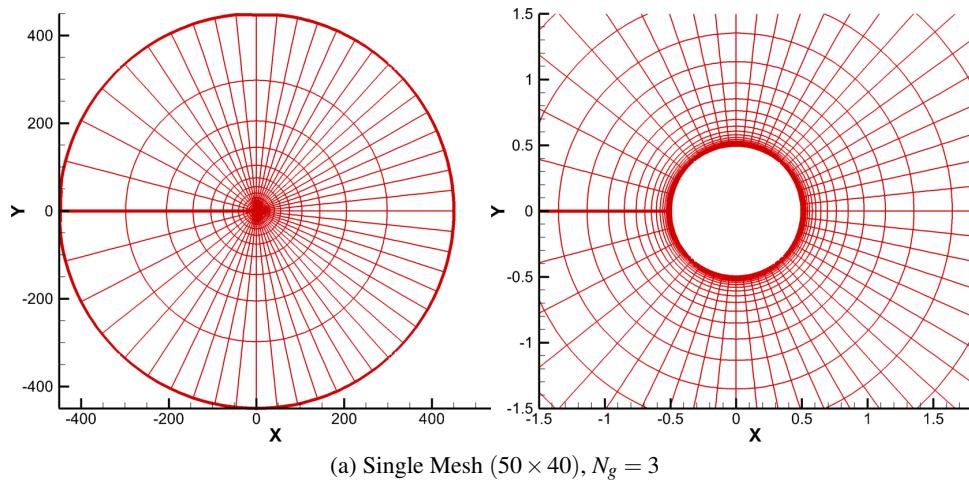


Figure 4.101: Cylinder $Re = 40$ Meshes

Table 4.4: Cylinder $Re = 40$ Degrees of Freedom

	N = 1	N = 2	N = 3
Single Grid	8,000	18,000	32,000
Chimera O-Grid	8,000	18,000	32,000
Chimera R-Grid	43,600	98,100	174,400

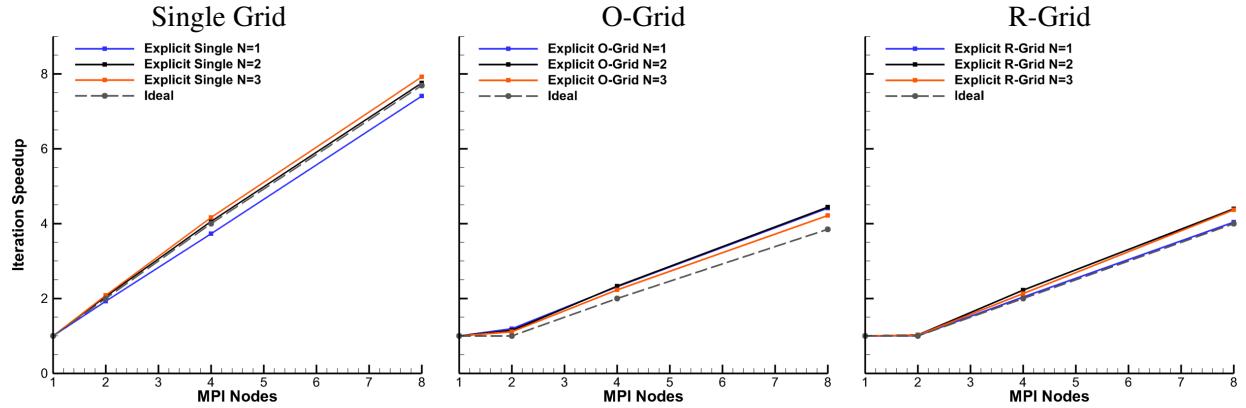


Figure 4.102: Cylinder Iteration Speedup with Explicit Artificial Boundaries

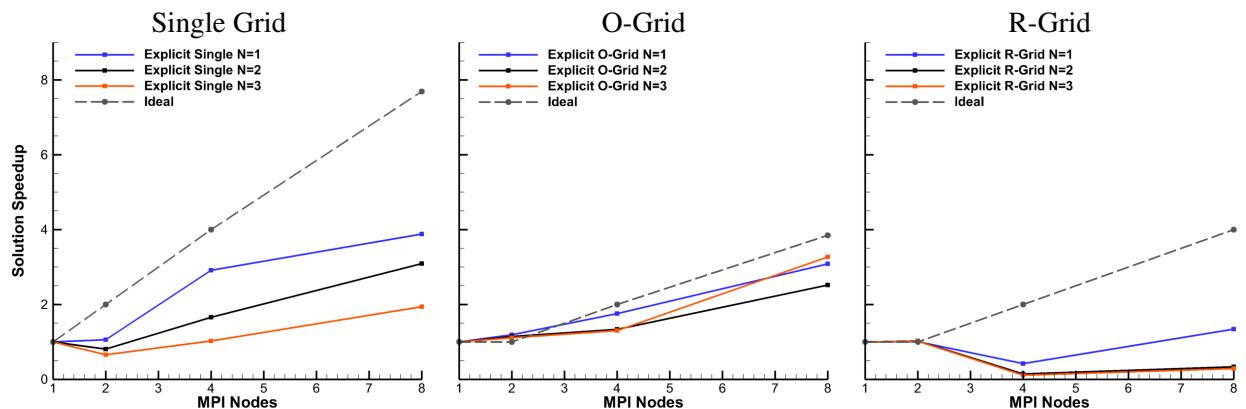


Figure 4.103: Cylinder Solution Speedup with Explicit Artificial Boundaries

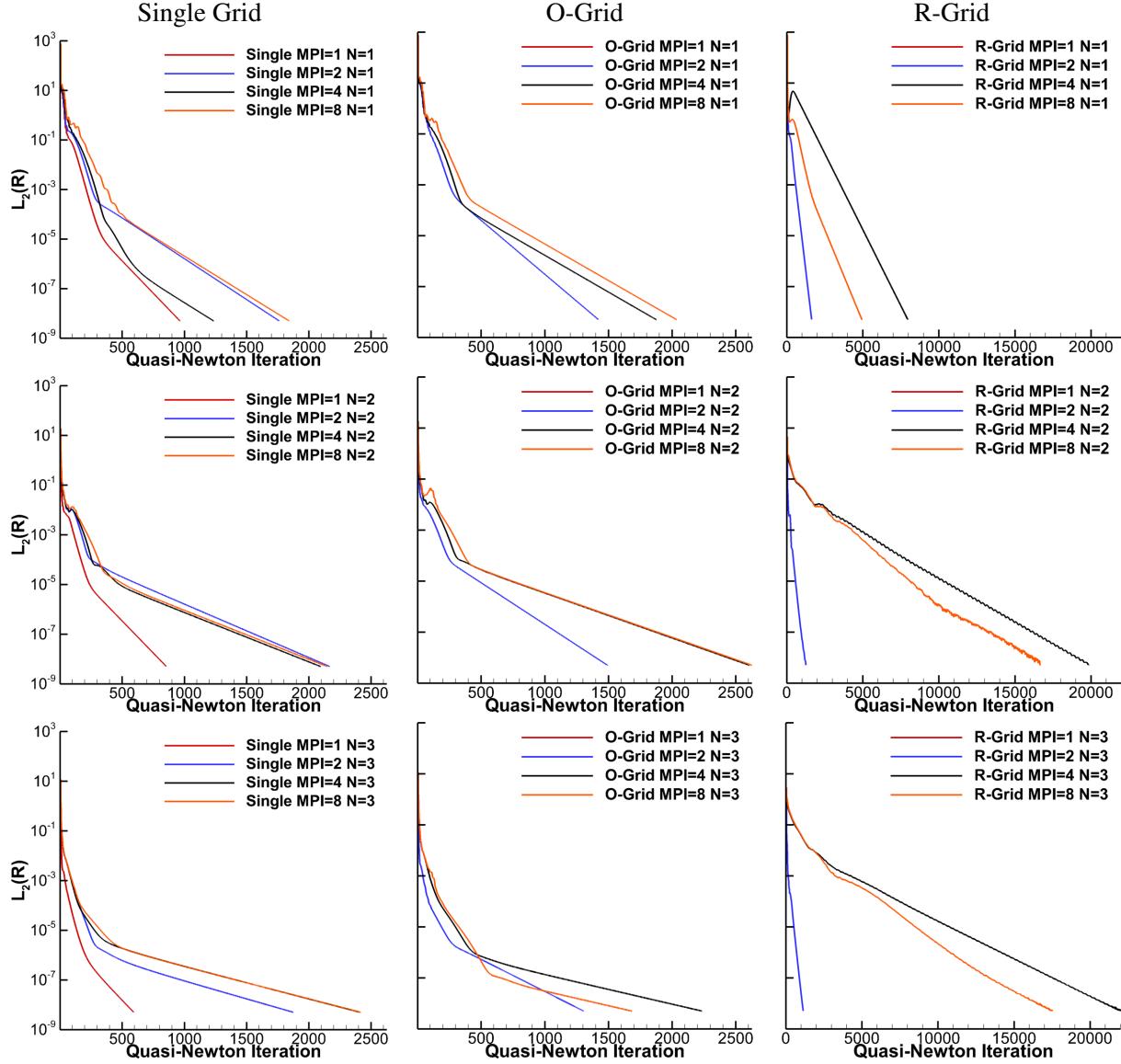


Figure 4.104: Cylinder $Re = 40$ Convergence History with Explicit Artificial Boundaries

The iteration speedup and solution speedup using implicit artificial boundaries are shown in Fig. 4.105 and 4.106. Here, the CFL number is increased using Eq. 3.97 and $CFL^0 = 10$. Similar to the SKF airfoil, the Single Grid and O-Grid iteration and solution speedup are identical. However, the iteration and solution speedup for the R-Grid differ. This disparity is explained by the convergence history shown in Fig. 4.107, where the number of processors is associated with the rows and the lines in each plot represent the order of approximation. The convergence history for the Single Grid and O-Grid requires, in general, less than

10 Quasi-Newton iterations and remain unchanged as the number of processors increases. However, the number of Quasi-Newton iterations required for the R-Grid to converge increases, in general, as the order of approximation or number of processors increase. The number of Quasi-Newton iterations is increasing because the preconditioner is only applied locally and as a result the GMRES iterative solver is not able to converge within the allotted 600 iterations for each Quasi-Newton iteration. Hence, the number of Quasi-Newton iterations increases as a result of only approximately solving the linear system.

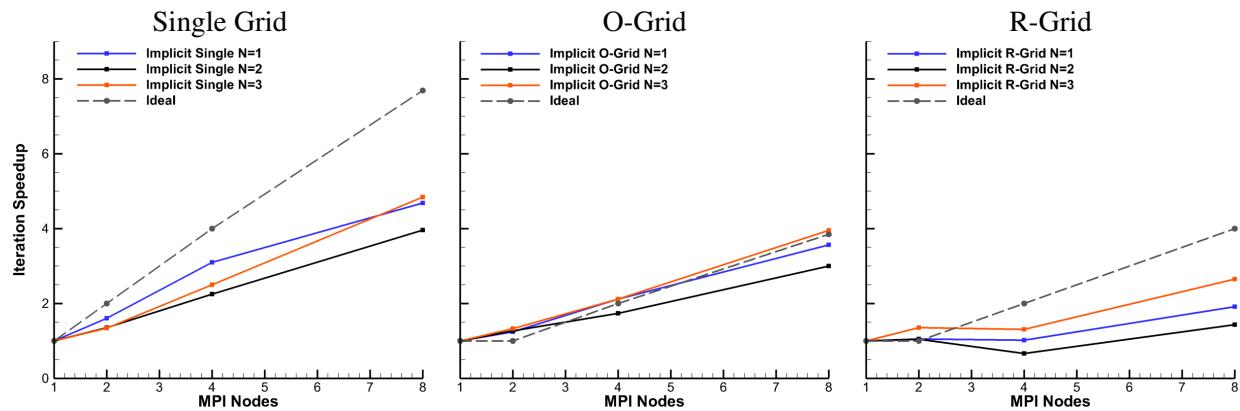


Figure 4.105: Cylinder Iteration Speedup with Implicit Artificial Boundaries

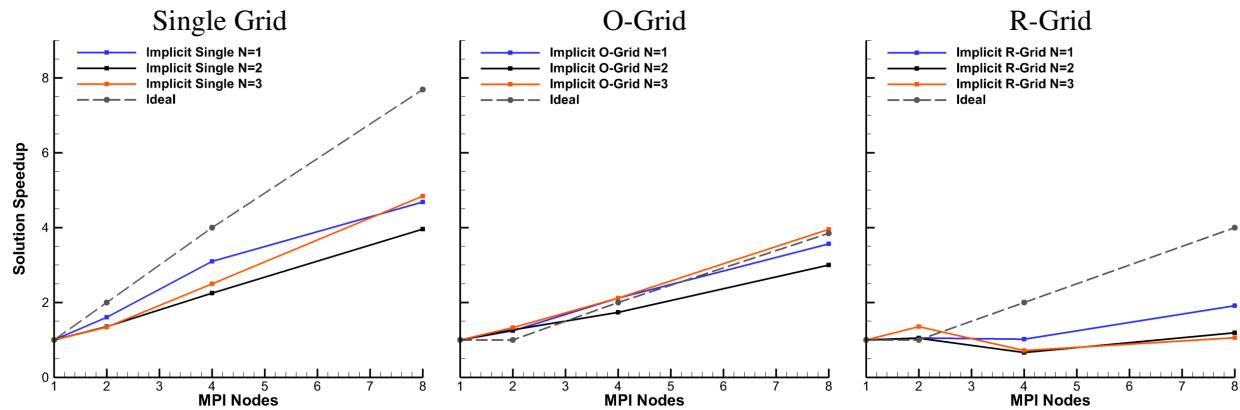


Figure 4.106: Cylinder Solution Speedup with Implicit Artificial Boundaries

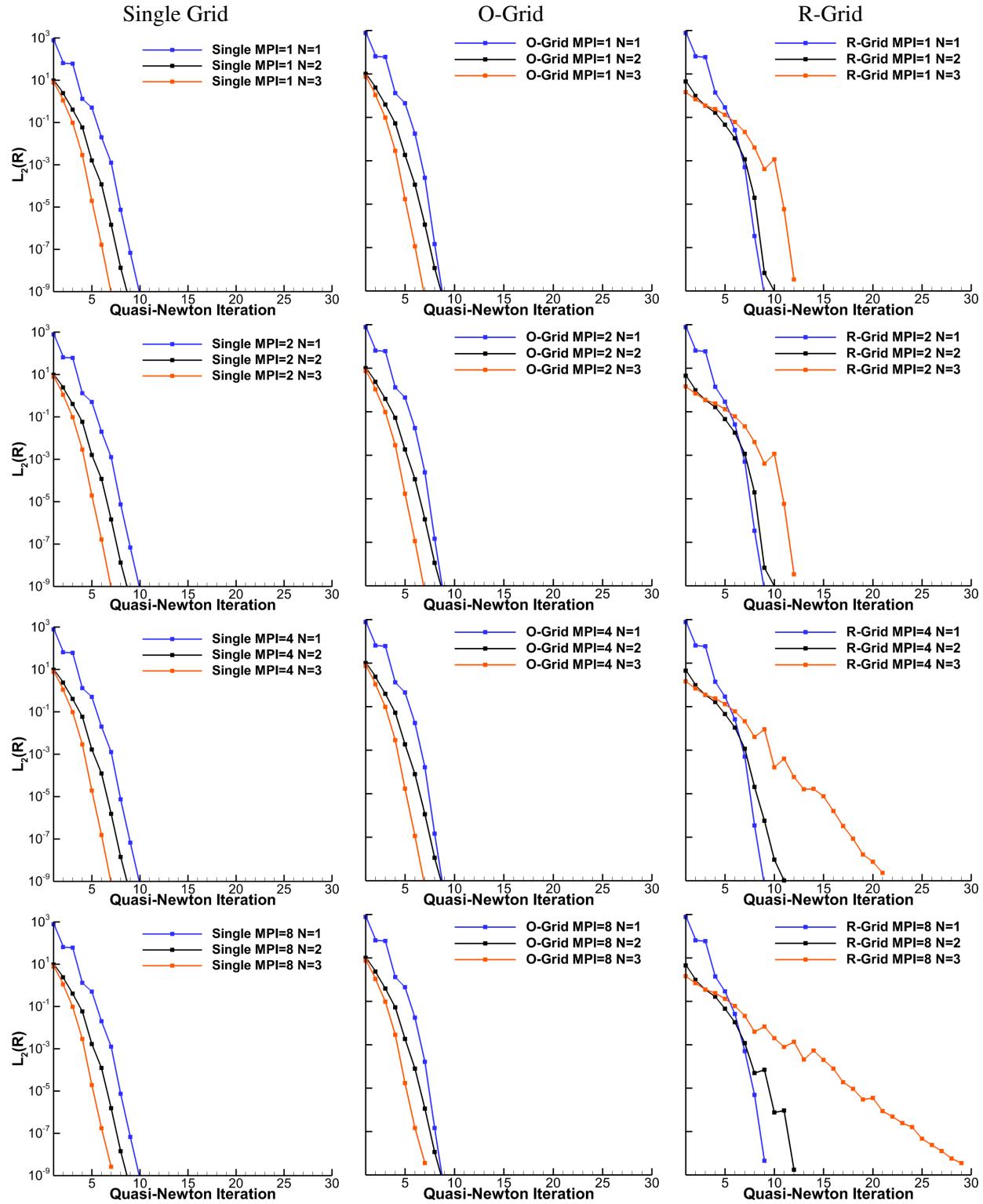


Figure 4.107: Cylinder Convergence History with Implicit Artificial Boundaries

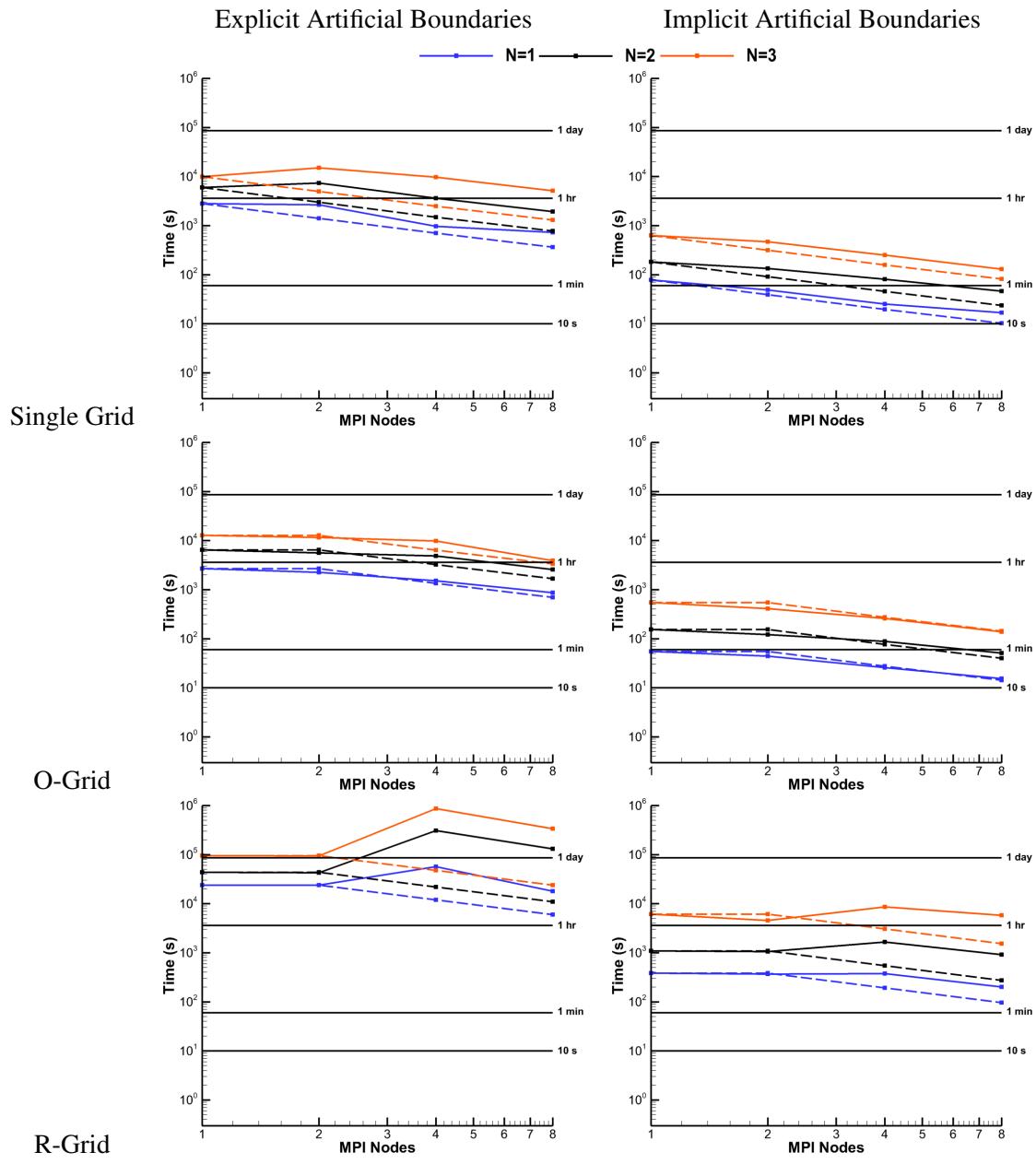


Figure 4.108: Cylinder Solution Time (Dashed Lines show Ideal Times)

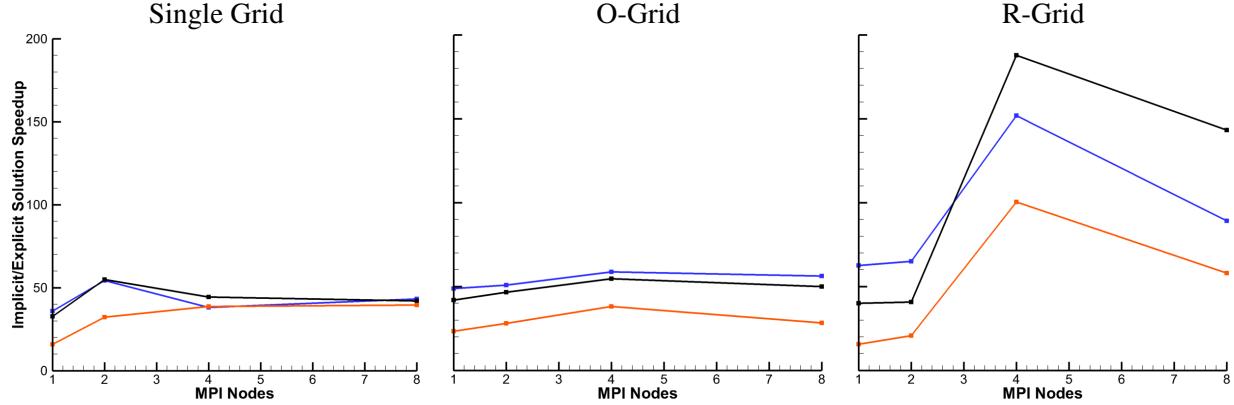


Figure 4.109: SKF 1.1 Airfoil Solution Speedup with Implicit Artificial Boundaries Relative to Explicit Artificial Boundaries

The execution times using explicit and implicit artificial boundaries are shown in Fig. 4.108. The execution time using implicit artificial boundary conditions is consistently significantly faster relative to using explicit artificial boundaries. Notably, the execution time with $N = 3$, four processors, and explicit artificial boundaries is approximately 10 days, whereas the execution time using implicit artificial boundaries is approximately 2 hours. The speedup in execution time using implicit relative to explicit artificial boundaries is shown in Fig. 4.109. The speedup for the Single and O-Grid when using implicit artificial boundaries is between 10 and 60 times faster than the explicit artificial boundaries. The speedup for the R-Grid is approximately two orders of magnitude with four or more processors with a peak speedup that is nearly 200.

4.4.3 Summary

The implicit artificial boundary formulation significantly reduces computational time compared to the standard explicit artificial boundary formulation that only accounts for the artificial boundaries in the residual vector. The implicit artificial boundary formulation exploits the iterative matrix solver to avoid explicitly formulating the Jacobian matrices associated with the artificial boundaries. This significantly reduces the data communicated associated with the implicit artificial boundaries. However, because the artificial boundary Jacobians are not explicitly formulated they are not included in the preconditioner. Thus, the method remains implicit as the grids are partitioned into greater numbers for parallel execution, even though the preconditioner approaches a Jacobi preconditioner. This is a significant improvement over the traditional

explicit Chimera formulation where the time integration scheme becomes more explicit as the grids are partitioned into greater numbers. The execution time of the implicit artificial boundaries is compared with an explicit artificial boundaries by computing a set of steady flow fields using both methods. The implicit method is able to obtain numerical solutions with execution times of up to two orders of magnitude lower than those required by the explicit DG-Chimera method. In addition, the explicit formulation required a fixed CFL number to prevent the convergence rate from stagnating, whereas the *CFL* number was always increased such that $CFL \rightarrow \infty$ with the implicit artificial boundaries. Hence, the implicit method increases robustness relative to the explicit method.

Chapter 5

Conclusion and Future Work

5.1 Summary and Conclusions

This dissertation presents the details of the implementation of a Discontinuous Galerkin Chimera overset solver. The goal was to develop a computational fluid dynamics code that is an improvement in accuracy and efficiency relative to the state of the art. The improved accuracy is achieved through the use of a Discontinuous Galerkin discretization method. Several different strategies are utilized to improve the computational efficiency. For steady state problems, a high convergence rate is achieved by using a Quasi-Newton's method with compete linearization of the discrete partial differential equations. The computational efficiency of the coding strategy is achieved by using Template Expressions, which provide the means for code written in the object oriented programming language C++ to achieve the same computational efficiency of the FORTRAN programming language. Template Expressions are used to develop both dense and block sparse linear algebra libraries that are used throughout the code. A further improvement in the computational efficiency is explored through the use of Quadrature-free integration, where the integrals of the Discontinuous Galerkin discretization are evaluated using symbolic manipulation software. A pre-processing tool, called PyDG, written in the Python programming language is developed. It uses a syntax that mimics the notation for the integrals of the Discontinuous Galerkin method. The PyDG pre-processor evaluates polynomial expressions, such as integration, differentiation, and multiplication, symbolically and produces C++ code that corresponds to that polynomial expression. The PyDG pre-processor was used to develop a C++ polynomial library, DGPoly++, with syntax that allows polynomials to be treated as if they were scalar quantities. For computational efficiency, the polynomial expressions are also implemented using Template Expressions.

The implementation of the Quadrature-free Discontinuous Galerkin discretization in the code is verified by computing solutions to a set of scalar partial differential equations. The implementation is also verified using a set of inviscid and viscous flow problems. Notably, the computational efficiency of the code is demonstrated to be superior relative to commercially available codes for an inviscid flow problem.

The Chimera overset method was implemented artificial boundaries suitable for a Discontinuous Galerkin discretization. The DG-Chimera scheme naturally resolves many of issues associated with Finite Volume and Finite Difference Chimera schemes. Notably, the DG-Chimera scheme does not require fringe points on artificial boundaries, and, hence, cannot have orphan points associated with fringe points. The locality of the cell polynomials of the Discontinuous Galerkin discretization serves as the interpolation scheme for the DG-Chimera method thereby mitigating the need for an additional high-order interpolation scheme. Such interpolation schemes are required in Finite Volume and Finite Difference Chimera schemes and have large stencils. The current DG-Chimera formulation naturally reduces to a zonal method. This means that there is no requirement on the extent of the overlap among grids in the Chimera mesh; proper communication among grids can be established so long as grids in the Chimera mesh overlap or abut. The computational efficiency of the DG-Chimera scheme is significantly enhanced by including the linearization of the artificial boundary equations as part of the Quasi-Newton's method. The DG-Chimera scheme also solves the problem of overlapping grids adjacent to a curved geometry. Traditional Finite Volume and Finite Difference Chimera schemes rely on an ad hoc "projection" method to mitigate the issue of improper interpolation. However, the use of curved cells, which is a necessity for the Discontinuous Galerkin discretization, in the DG-Chimera scheme naturally resolves this issue. The hole cutting process developed here is significantly simpler than hole cutting processes developed for Finite Volume and Finite Difference schemes, because the hole cutting process does not need to impose that any minimal overlap requirements. The hole cutting process is an extension of the Direct Cut method that also accounts for curved cells in the grids. The implementation of the DG-Chimera method is verified by computing solutions to a set of scalar partial differential equations, as well as the Euler and Navier-Stokes equations. Both internal and external subsonic, transonic, and supersonic inviscid flow fields computed using single or zonal grids as well as Chimera meshes demonstrate that the mass flux error associated with the artificial boundaries are small. Furthermore, the mass flux error is demonstrated to be consistent, i.e., it vanishes with grid refinement and/or increase in the order of approximation. Using curved elements to represent curved geometries is demonstrated to produce smaller errors associated with the artificial boundaries relative to a Finite Volume Chimera scheme. Notably, the

Finite Volume Chimera scheme produces a non-physical rise in entropy on the artificial boundaries. This rise is significantly smaller with the DG-Chimera scheme. Finally, the parallel performance of the implicit artificial boundaries for steady inviscid and viscous flow calculations significantly outperforms the explicit artificial boundary scheme with an execution time up to two orders of magnitude faster than the explicit artificial boundaries.

5.2 Future Work

While this dissertation provides promising results for using the high-order Discontinuous Galerkin method, especially combined with the Chimera scheme, some potential future work directions are outlined below:

1. Linear Algebra Algorithms

The execution time for steady state calculations using the Quasi-Newton solver are dominated by the iterative matrix solver. Hence, further improvement in execution time should be pursued through improvements in the iterative solver and pre-conditioner algorithms. In particular, the high convergence rate of the Quasi-Newton solver cannot be achieved if the iterative matrix solver is unable to converge during a Newton sub-iteration. The high computational efficiency of the solver is only achieved by a high convergence rate. Furthermore, storing the global sparse block Jacobian matrix requires significant memory resources, in particular, for calculations in three-dimensions. As such, Jacobian free iterative solvers eliminate the storage requirements by recomputing the matrix each iteration. This comes at a significant increase in computational cost, but eliminates the needs for large amounts of memory. Alternatively, a multi-grid solution method might also eliminate the need for storing the Jacobian matrix.

2. Quadrature-free Integration

The use of quadrature free integration was explored as a possible means of improving the computational efficiency of the Discontinuous Galerkin scheme for non-linear partial differential equations. The Quadrature-free integration has an evident reduction in operation count for linear partial differential equations relative to Gauss-Quadrature integration. However, the operation count is not as evident for non-linear partial differential equations. Unfortunately, a thorough study of the benefits or deficits of the Quadrature-free integration is still lacking. Preliminary results indicate that the execution time

for Quadrature-free integration is comparable to Gauss-Quadrature integration. Possible benefits in robustness that might result from Quadrature-free integration techniques still need to be explored. However, the software development for a Quadrature-free method is significantly more challenging than a Gauss-Quadrature approach. In addition, the file sizes resulting from the PyDG pre-processor currently limits the order of approximation to $N = 3$ in three-dimensions.

3. Discontinuous Galerkin Chimera Scheme

The Discontinuous Galerkin Chimera scheme presented in this dissertation offers solutions to many problems with Finite Volume and Finite Difference Chimera schemes. As such, there are still aspects of the Chimera method that are not explored here. For example, the Chimera overset method is particularly suitable for simulations of bodies in relative motion. Extending the DG-Chimera scheme to simulations with bodies of relative motion is desirable. Again, the lack of fringe points is expected to greatly simplify the dynamic hole cutting algorithm.

The hole cutting processes outlined in this dissertation is nearly complete, aside from issues associated with the possibility of multiple local solutions when determining if a line segment intersects a cell. Multiple global minimization strategies could be considered to resolve this problem.

The issue of multiple overlapping grids is also not explored in this dissertation. The current formulation uses a simple average nodal value from multiple donor cells when multiple grids overlap. While this approach does not appear to introduce significant errors in the few cases where it has been exercised, other possible formulations such as a weighted averaged based on cell size or only using the value from the cell with the best “quality” should be explored.

While the approximation of the BR2 scheme on the artificial boundaries does not appear to introduce significant errors, it is disconcerting that the interior scheme is not recovered on zonal boundaries. Instead, the Symmetric Interior Penalty only relies on the jump and averages of the solution approximation on cell boundaries. As such, this discretization scheme would not require an approximation on artificial boundaries.

The errors in the mass flux are demonstrated to be small for a set of inviscid flow problems. Nonetheless, this is not a complete set of problems, and it is possible that situations arise where the mass flux errors are not negligible. As such, it might be possible to introduce additional equations that explicitly enforce the cancellation of the flux integrals on the artificial boundaries. For example, a Lagrange

multiplier might be suitable to enforce this additional constraint on the discrete partial differential equations.

Finally, the DG-Chimera method is only applied to a limited set of fluid dynamics problems in this dissertation. Further exploration of the applicability of the method to three-dimensional turbulent problems is describable.

Bibliography

- [1] Visbal, M. R. and Rizzetta, D. P., “Large-Eddy Simulation on Curvilinear Grids Using Compact Differencing and Filtering Schemes,” *Journal of Fluids Engineering*, Vol. 124, 2002, pp. 836–847.
- [2] Fidkowski, K., *A High-Order Discontinuous Galerkin Multigrid Solver for Aerodynamic Applications*, Master’s thesis, Massachusetts Institute of Technology, Department of Aeronautics and Astronautics, 2004.
- [3] Oliver, T. A., *Multigrid Solution for High-Order Discontinuous Galerkin Discretizations of the Compressible Navier-Stokes Equations*, Master’s thesis, Massachusetts Institute of Technology, Department of Aeronautics and Astronautics, 2004.
- [4] Visbal, M. R. and Gaitonde, D. V., “High-Order-Accurate Methods for Complex Unsteady Subsonic Flows,” *AIAA Journal*, Vol. 37, No. 10, 1999, pp. 1231–1239.
- [5] Rizzetta, D. P., Visbal, M. R., and Blaisdell, G. A., “A Time-Implicit High Order Compact Differencing and Filtering Scheme for Large-Eddy Simulation,” *International Journal for Numerical Methods in Fluids*, Vol. 42, No. 6, June 2003, pp. 665–693.
- [6] Rizzetta, D. P. and Visbal, M. R., “Numerical Simulation of Separation Control for Transitional Highly Loaded Low-Pressure Turbines,” *AIAA Journal*, Vol. 43, No. 9, 2005, pp. 1958–1967.
- [7] Sherer, S. E., Visbal, M. R., Gordnier, R. E., Yilmaz, T. O., and O.Rockwell, D., “1303 Unmanned Combat Air Vehicle Flowfield Simulations and Comparison with Experimental Data,” *Journal of Aircraft*, Vol. 48, No. 3, 2011, pp. 1005–1019.
- [8] Lesaint, R. and Raviart, P.-A., *Mathematical Aspects of Finite Elements in Partial Differential Equations*, Academic Press, New York, 1974.

- [9] Cockburn, B. and Shu, C.-W., “The Runge-Kutta Local Projection P^1 -Discontinuous Galerkin Finite Element Method for Scalar Conservation Laws,” *IMA Preprint Seris 388*, 1988.
- [10] Cockburn, B. and Shu, C.-W., “TVB Runge-Kutta Local Projection Discontinuous Galerkin Finite Element Method for Conservation Laws III: One-dimensional Systems,” *Journal of Computational Physics*, Vol. 84, No. 186, 1989, pp. 90–113.
- [11] Cockburn, B., Hou, S., and Shu, C.-W., “TVB Runge-Kutta Local Projection Discontinuous Galerkin Finite Element Method for Conservation Laws IV: The Multidimensional Case,” *Mathematisc of Computation*, Vol. 54, No. 186, 1990, pp. 441–435.
- [12] Cockburn, B. and Shu, C.-W., “TVB Runge-Kutta Local Projection Discontinuous Galerkin Finite Element Method for Conservation Laws II: General Framework,” *Mathematics of Computation*, Vol. 52, No. 186, 1989, pp. 411–435.
- [13] Bassi, F., Rebay, S., Mariotti, G., Pedinotti, S., and Savini, M., “A High-Order Accurate Discontinuous Finite Element Method for Inviscid and Viscous Turbomachinery Flows,” 2nd European Conference on Turbomachinery Fluid Dynamics and Thermodynamics, Technologisch Instituut, Antwerpen, Belgium, 1997, pp. 99–108.
- [14] Bassi, F. and Rebay, S., “High-Order Accurate Discontinuous Finite Element Solution of the 2D Euler Equations,” *Journal of Computational Physics*, Vol. 138, 1997, pp. 251–285.
- [15] Cockburn, B. and Shu, C.-W., “The Local Discontinuous Galerkin Method for Time-Dependent Convection-Diffusion Systems,” *SIAM J. Numer. Anal.*, Vol. 35, No. 6, 1998, pp. 2440–2463.
- [16] Arnold, D. N., Brezzi, F., Cockburn, B., and Marini, D., “Discontinuous Galerkin Methods for Elliptic Problems,” *Discontinuous Galerkin Methods*, edited by B. Cockburn, G. E. Karniadakis, and C.-W. Shu, Vol. 11 of *Lecture Notes in Computational Science and Engineering*, Springer, 2000, pp. 89–101.
- [17] Cockburn, B., “Devising Discontinuous Galerkin Methods for Non-linear Hyperbolic Conservation Laws,” *Journal of Computational and Applied Mathematics*, Vol. 128, 2001, pp. 187–204.

- [18] Klaij, C. M., van der Vegt, J. J. W., and van der Ven, H., “Space-time Discontinuous Galerkin Method for the Compressible Navier-Stokes Equations,” *Journal of Computational Physics*, Vol. 217, No. 2, 2006, pp. 589–611.
- [19] Persson, P.-O. and Peraire, J., “Sub-Cell Shock Capturing for Discontinuous Galerkin Methods,” AIAA-Paper 2006-112, 2006.
- [20] Persson, P.-O. and Peraire, J., “An Efficient Low Memory Implicit DG Algorithm for Time Dependent Problems,” AIAA Paper 2006-113, 2006.
- [21] Landmann, B., Kessler, M., Wagner, S., and Kramer, E., “A Parallel Discontinuous Galerkin Code for the Navier-Stokes Equations,” AIAA-Paper 2006-111, 2006.
- [22] Dumbser, M., Käser, M., Titarev, V. A., and Toro, E. F., “Quadrature-free Non-oscillatory Finite Volume Schemes on Unstructured Meshes for Nonlinear Hyperbolic Systems,” *Journal of Computational Physics*, Vol. 226, 2007, pp. 204–243.
- [23] Dumbser, M., Balsara, D. S., Toro, E. F., and Munz, C.-D., “A Unified Framework for the Construction of One-step Finite Volume and Discontinuous Galerkin Schemes on Unstructured Meshes,” *Journal of Computational Physics*, Vol. 227, 2008, pp. 8209–8253.
- [24] Landmann, B., *A Parallel Discontinuous Galerkin Code for the Navier-Stokes and Reynolds-averaged Navier-Stokes Equations*, Ph.D. thesis, Universitat Stuttgart, December 2008.
- [25] Peraire, J. and Persson, P.-O., “The Compact Discontinuous Galerkin (CDG) Method for Elliptic Problems,” *SIAM J. Sci. Comput.*, Vol. 30, No. 4, 2008, pp. 1806–1824.
- [26] Persson, P.-O. and Peraire, J., “Curved Mesh Generation and Mesh Refinement using Lagrangian Solid Mechanics,” AIAA Paper 2009-949, 2009.
- [27] Barter, G. E. and Darmofal, D. L., “Shock Capturing with PDE-Based Artificial Viscosity for DGFEM: Part 1, Formulation,” *Journal of Computational Physics*, Vol. 229, No. 5, 2010, pp. 1810–1827.
- [28] Burgess, N. K., *An Adaptive Discontinuous Galerkin Solver for Aerodynamic Flows*, Ph.D. thesis, University of Wyoming, November 2011.

- [29] Atkins, H. L. and Shu, C.-W., “Quadrature-Free Implementation of Discontinuous Galerkin Method for Hyperbolic Equations,” *AIAA*, Vol. 36, No. 5, May 1998.
- [30] Lockard, D. P. and Atkins, H. L., “Efficient Implementation of the Quadrature-Free Discontinuous Galerkin Method,” *AIAA-Paper 1999-3309*, 1999.
- [31] Galbraith, M. C., Orkwis, P. D., and Benek, J. A., “Automated Quadrature-Free Discontinuous Galerkin Method with a Tailored Recovery Formulation,” *AIAA-Paper 2010-365*, 2010.
- [32] Babuska, I. and Aziz, A. K., “On the Angle Condition in the Finite Element Method,” *SIAM Journal on Numerical Analysis*, Vol. 13, No. 2, 1976, pp. 214–226.
- [33] Barth, T. J., “Numerical Aspects of Computing Viscous High Reynolds Number Flows on Unstructured Meshes,” *AIAA-Paper 1991-0721*, 1991.
- [34] Edelsbrunner, H., Tan, T. S., and Waupotitsch, R., “An $O(n^2 \log n)$ Time Algorithm for the Minmax Angle Triangulation”, *SIAM Journal on Scientific and Statistical Computing*, Vol. 13, No. 4, 1992, pp. 994–1008.
- [35] Kallinderis, Y. and Nakajima, K., “Finite Element Method for Incompressible Viscous Flows on Unstructured Meshes,” *AIAA Journal*, Vol. 32, No. 8, 1994, pp. 1617–1625.
- [36] Lohner, R., “Matching Semi-structured and Unstructured Grids for Navier-Stokes Calculations,” *AIAA-Paper 1993-3348*, 1993.
- [37] van der Vegt, J. J. W. and van der Ven, H., “Discontinuous Galerkin Finite Element Method with Anisotropic Local Grid Refinement for Inviscid Compressible Flow,” *Journal of Computational Physics*, Vol. 141, 1998, pp. 46–77.
- [38] Benek, J. A., Steger, J. L., and Dougherty, F. C., “A Flexible Grid Embedding Technique with Application to the Euler Equations,” *AIAA-Paper 1983-1944*, 1983.
- [39] Chao, J., Haselbacher, A., and Balachandar, S., “A Massively Parallel Multi-Block Hybrid Compact-WENO Scheme for Compressible Flows,” *Journal of Computational Physics*, Vol. 228, 2009, pp. 7473–7491.

- [40] Pascarelli, A. and ad Graham V. Candler, U. P., “Multi-Block Large-Eddy Simulations of Turbulent Boundary Layers,” *Journal of Computational Physics*, Vol. 157, 2000, pp. 256–279.
- [41] Rizzi, A., Eliasson, P., Lindblad, I., Hirsch, C., Lacor, C., and Haeuser, J., “The Engineering of Multiblock/Multigrid Software for Navier-Stokes Flows on Structured Meshes,” *Computers and Fluids*, Vol. 22, 1993, pp. 341–367.
- [42] Rogers, S. E., Suhs, N. E., Dietz, W. E., Nash, S. M., and Onufer, J. T., *PEGASUS User’s Guide Version 5.1k*, NASA Ames Research Center; Micro Craft; MCAT, Inc, Oct 2003, <http://people.nas.nasa.gov/rogers/pegasus/uguide.html>.
- [43] Rogers, S. E., Suhs, N. E., and Dietz, W. E., “PEGASUS 5: An Automated Preprocessor for Overset-Grid Computational Fluid Dynamics,” *AIAA Journal*, Vol. 41, No. 6, 2003, pp. 1037–1045.
- [44] Sherer, E. S., Visbal, M. R., and Galbraith, M. C., “Automated Preprocessing Tools for Use with a Higher-Order Overset-Grid Algorithm,” *AIAA Paper 2006-1147*, Jan. 2006.
- [45] Wang, Z., Fidkowski, K., Abgrall, R., Bassi, F., Caraeni, D., Cary, A., Deconinck, H., Hartmann, R., Hillewaert, K., Huynh, H., Kroll, N., May, G., Persson, P.-O., van Leer, B., and Visbal, M., “High-Order CFD Methods: Current Status and Perspective,” *International Journal for Numerical Methods in Fluids*, Vol. 00, 2012, pp. 1–42.
- [46] Godunov, S. K., “A Finite Difference Method for the Numerical Computation of Discontinuous Solutions of the Equations of Fluid Dynamics,” *Matematicheskii Sbornik*, Vol. 47, 1959, pp. 271–306.
- [47] van Leer, B., “Towards the Ultimate Conservative Difference Scheme. I. The Quest of Monotonicity,” *Lecture Notes in Physics*, Vol. 18, 1973, pp. 163–168.
- [48] van Leer, B., “Towards the Ultimate Conservative Difference Scheme. II. Monotonicity and Conservation Combined in a Second-order Scheme,” *Journal of Computational Physics*, Vol. 14, No. 4, 1974, pp. 361 – 370.
- [49] van Leer, B., “Towards the Ultimate Conservative Difference Scheme III. Upstream-centered Finite-difference Schemes for Ideal Compressible Flow,” *Journal of Computational Physics*, Vol. 23, No. 3, 1977, pp. 263 – 275.

- [50] van Leer, B., “Towards the Ultimate Conservative Difference Scheme. IV. A New Approach to Numerical Convection,” *Journal of Computational Physics*, Vol. 23, No. 3, 1977, pp. 276 – 299.
- [51] van Leer, B., “Towards the Ultimate Conservative Difference Scheme. V. A Second-order Sequel to Godunov’s Method,” *Journal of Computational Physics*, Vol. 32, No. 1, 1979, pp. 101 – 136.
- [52] Kolgan, V. P., “Application of the Principle of Minimizing the Derivative to the Construction of Finite-difference Schemes for Computing Discontinuous Solutions of Gas Dynamics,” *Scientific Notes of TsAGI*, Vol. 3, 1972, pp. 68–77.
- [53] van Leer, B., “A Historical Oversight: Vladimir P. Kolgan and His High-resolution Scheme,” *Journal of Computational Physics*, Vol. 230, No. 7, 2011, pp. 2378 – 2383.
- [54] Harten, A., Engquist, B., Osher, S., and Chakravarthy, S. R., “Uniformly High Order Accurate Essentially Non-oscillatory Schemes, III,” *Journal of Computational Physics*, Vol. 71, No. 2, 1987, pp. 231 – 303.
- [55] Liu, X.-D., Osher, S., and Chan, T., “Weighted Essentially Non-oscillatory Schemes,” *Journal of Computational Physics*, Vol. 115, No. 1, 1994, pp. 200–212.
- [56] Mullenix, N. J. and Gaitonde, D. V., “A Bandwidth and Order Optimized WENO Interpolation Scheme for Compressible Turbulent Flows,” AIAA Paper 2011-366, 2011.
- [57] Lele, S. A., “Compact Finite Difference Schemes with Spectral-Like Resolution,” *Journal of Computational Physics*, Vol. 103, No. 1, 1992, pp. 16–42.
- [58] Gordnier, R. E. and Visbal, M. R., “Numerical Simulation of Delta-Wing Roll,” 1993.
- [59] Rizzetta, D. P. and Visbal, M. R., “Numerical Investigation of Transitional Flow Through a Low-Pressure Turbine Cascade,” AIAA-Paper 2003-3587, 2003.
- [60] Sherer, S. E. and Visbal, M. R., “Implicit Large Eddy Simulations Using a High-Order Overset Grid Solver,” AIAA Paper 2004-2530, 2004.
- [61] Rizzetta, D. P., “Numerical Study of Active Flow Control for a Transitional Highly-Loaded Low-Pressure Turbine,” AIAA Paper 2005-5020, 2005.

- [62] Sherer, S. E., Gordnier, R. E., and Visbal, M. R., “Computational Study of a UCAV Configuration Using a High-Order Overset-Grid Algorithm,” AIAA Paper 2008-626, 2008.
- [63] Galbraith, M. C. and Visbal, M. R., “Implicit Large Eddy Simulation of Low-Reynolds-Number Transitional Flow Past the SD7003 Airfoil,” AIAA-Paper 2010-4737, 2010.
- [64] Fu, H., Lu, P., and Liu, C., “Improvement of Mixing Function for Modified Upwinding Compact Scheme,” AIAA 2011-368, 2011.
- [65] Liu, C. and Oliveira, M., “Modified Upwinding Compact Scheme for Shock and Shock Boundary Layer Interaction,” AIAA Paper 2010-723, 2010.
- [66] Visbal, M. R. and Gaitonde, D. V., “On the Use of Higher-Order Finite-Difference Schemes on Curvilinear and Deforming Meshes,” *Journal of Computational Physics*, Vol. 181, 2002, pp. 155–185.
- [67] Visbal, M. R. and Gaitonde, D. V., “Very High-Order Spatially Implicit Schemes for Computational Acoustics on Curvilinear Meshes,” *Journal of Computational Acoustics*, Vol. 9, No. 4, 2001, pp. 1259–1286.
- [68] Toro, E. F., *Riemann Solvers and Numerical Methods for Fluid Dynamics: A Practical Introduction*, Springer-Verlag, 2nd ed., 1999.
- [69] van Leer, B., “Upwind-difference Methods for Aerodynamic Problems Governed by the Euler Equations,” in *Large-Scale Computations in Fluid Mechanics, Lectures in Applied Mathematics*, 22, 1985.
- [70] McDonald, P. W., “The Computation of Transonic Flow through Two-Dimensional Gas Turbine Cascades,” ASME Paper 71-GT-89, Gas Turbine Conference and Products Show, Houston, Texas, 1971.
- [71] MacCormack, R. W. and Paullay, A. J., “Computational Efficiency Achieved by Time Splitting of Finite Difference Operators,” AIAA Paper 72-154, 1972.
- [72] Delanaye, M. and Liu, Y., “Quadratic Reconstruction Finite Volume Schemes on 3D Arbitrary Unstructured Polyhedral Grids,” AIAA-Paper 1999-3259, 1999.
- [73] Mavriplis, D. J., “An Assessment of Linear Versus Nonlinear Multigrid Methods for Unstructured Mesh Solvers,” *Journal of Computational Physics*, Vol. 175, 2002, pp. 302–325.

- [74] Wang, Z. J., “Spectral (Finite) Volume Method for Conservation Laws on Unstructured Grids,” *Journal of Computational Physics*, Vol. 178, 2002, pp. 210–251.
- [75] Wang, Z. J., “High-Order Methods for the Euler and Navier-Stokes Equations on Unstructured Grids,” *Progress in Aerospace Sciences*, Vol. 43, 2007, pp. 1–41.
- [76] Harris, R., Wang, Z. J., and Liu, Y., “Efficient Implementation of High-Order Spectral Volume Method for Multidimensional Conservation Laws on Unstructured Grids,” AIAA Paper 2007-912, 2007.
- [77] Hesthaven, J. S., “From Electrostatics to Almost Optimal Nodal Sets for Polynomial Interpolation in a Simplex,” *SIAM J. Numer. Anal.*, Vol. 35, No. 2, 1998, pp. 655–676.
- [78] Zhang, M. and Shu, C. W., “An Analysis and a Comparison Between the Discontinuous Galerkin Method and the Spectral Finite Volume Methods,” *Computers and Fluids*, Vol. 34, No. 4-5, 2005, pp. 581–592.
- [79] Sun, Y. and Wang, Z. J., “Evaluation of Discontinuous Galerkin and Spectral Volume Methods for Scalar and System Conservation Laws on Unstructured Grid,” *International Journal for Numerical Methods in Fluids*, Vol. 45, No. 8, 2004, pp. 819–838.
- [80] Reed, W. H. and Hill, T. R., “Triangular Mesh Methods for the Neutron Transport Equation,” Tech. Rep. LA-UR-73-479, Los Alamos Scientific Laboratory, 1973.
- [81] Chavant, G. and Cockburn, B., “The Local Projection Discontinuous Galerkin Finite Element Method for Scalar Conservation Laws,” *IMA Preprint Seris 341*, 1987.
- [82] Bey, K. S. and Oden, J. T., “HP-Version Discontinuous Galerkin Methods for Hyperbolic Conservation Laws,” *Comput. Methods. Appl. Mech. Engrg.*, Vol. 133, 1996, pp. 259–286.
- [83] Bassi, F. and Rebay, S., *GMRES Discontinuous Galerkin solution of the compressible Navier-Stokes equations*, Springer, Berlin, 2000, pp. 197–208.
- [84] Brezzi, F., Marini, L., and Suli, E., “Discontinuous Galerkin Methods for First-Order Hyperbolic Problems,” *Math. Models Methods Appl. Sci.*, Vol. 14, No. 12, 2004, pp. 1893–1903.

- [85] Batten, P., Clarke, N., Lambert, C., and Causon, D. M., “On the Choice of Wavespeeds for the HLLC Riemann Solver,” *SIAM J. Sci. Comput.*, Vol. 18, No. 6, November 1997, pp. 1553–1570.
- [86] Vatsa, V. N. and Wedan, J. L. T. B. W., “Navier-Stokes Computations of Prolate Spheroids at Angle of Attack,” AIAA-Paper 1987-2627, 1987.
- [87] Persson, P.-O. and Peraire, J., “Newton-GMRES Preconditioning for Discontinuous Galerkin Discretizations of the Navier-Stokes Equations,” *SIAM J. Sci. Comput.*, Vol. 30, No. 6, 2008, pp. 2709–2733.
- [88] Hartmann, R. and Houston, P., “Symmetric Interior Penalty DG Method for the Compressible Navier-Stokes Equations I: Method Formulation,” *International Journal of Numerical Analysis and Modeling.*, Vol. 3, No. 1, 2006, pp. 1–20.
- [89] Hartmann, R. and Houston, P., “An Optimal Order Interior Penalty Discontinuous Galerkin Discretization of the Compressible Navier-Stokes Equations,” *Journal of Computational Physics*, Vol. 227, No. 22, 2008, pp. 9670–9685.
- [90] van Leer, B. and Nomura, S., “Discontinuous Galerkin for Diffusion,” AIAA Paper 2005-5108, 2005.
- [91] Burbeau, A., Sagaut, P., and Bruneau, C.-H., “A Propblem-Independent Limiter for High-Order Runge-Kutta Discontinuous Galerkin Methods,” *Journal of Computational Physics*, Vol. 169, 2001, pp. 111–150.
- [92] Qiu, J. and Shu, C.-W., “Hermite WENO Schemes and Their Application as Limiters for Runge-Kutta Discontinuous Galerkin Method: One-dimensional Case,” *Journal of Computational Physics*, Vol. 193, 2003, pp. 115–135.
- [93] Qiu, J. and Shu, C.-W., “Hermite WENO Schemes and Their Application as Limiters for Runge-Kutta Discontinuous Galerkin Method II: Two Dimensional Case,” *Journal of Computational Physics*, Vol. 34, 2005, pp. 642–663.
- [94] Luo, H., Baum, J. D., and Lohner, R., “A Hermite WENO-based Limiter for Discontinuous Galerkin Method on Unstructured Grids,” AIAA Paper 2007-0510, 2007.

- [95] Krivodonova, L., “Limiters for High-Order Discontinuous Galerkin Methods,” *Journal of Computational Physics*, Vol. 226, 2007, pp. 879–896.
- [96] Barter, G. E., *Shock Capturing with PDE-Based Artificial Viscosity for an Adaptive, Higher-Order Discontinuous Galerkin Finite Element Method*, Ph.D. thesis, Massachusetts Institute of Technology, June 2008.
- [97] Nguyen, N. C. and Peraire, J., “An Adaptive Shock-Capturing HDG Method for Compressible Flows,” AIAA Paper 2011-3060, 2011.
- [98] Bassi, F., Crivellini, A., Rebay, S., and Savini, M., “Discontinuous Galerkin solution of the Reynolds-averaged Navier-Stokes and k- ω turbulence model equations,” *Computers and Fluids*, Vol. 34, 2005, pp. 507–540.
- [99] Nguyen, N. C., Persson, P.-O., and Peraire, J., “RANS Solutions Using High Order Discontinuous Galerkin Methods,” AIAA Paper 2007-0914, 2007.
- [100] Hindenlang, F., Neudorfer, J., Gassner, G., and Munz, C.-D., “Unstructured three-dimensional High Order Grids for Discontinuous Galerkin Schemes,” AIAA Paper 2011-3853, 2011.
- [101] Magnus, R. and Yoshihara, H., “Inviscid Transonic Flow Over Airfoils,” AIAA-Paper 1970-47, 1970.
- [102] Benek, J. A., Buning, P. G., and Steger, J. L., “A 3-D Chimera Grid Embedding Technique,” AIAA-Paper 1985-1523, 1985.
- [103] Benek, J. A., Donegan, T. L., and Suhs, N. E., “Extended Chimera Grid Embedding Scheme with Application to Viscous Flows,” AIAA-Paper 1987-1126-CP, 1987.
- [104] Galbraith, M. C., Orkwis, P. D., and Benek, J. A., “Multi-Row Micro-Ramp Actuators for Shock Wave Boundary-Layer Interaction Control,” AIAA-Paper 2009-0321, 2009.
- [105] Dougherty, F. C., Benek, J. A., and Steger, J. L., “On Applications of Chimera Grid Schemes To Store Separation,” Tech. Rep. 88193, NASA Technical Memorandum, October 1985.
- [106] Lijewski, L. E. and Suhs, N. E., “Time-accurate Computational Fluid Dynamics Approach to Transonic Store Separation Trajectory Prediction,” *Journal of Aircraft*, Vol. 31, No. 4, 1994, pp. 886–891.

- [107] Wang, Z. J., “A Conservative Overlapped (Chimera) Grid Algorithm For Multiple Moving Body Flows,” AIAA-Paper 1996-0823, 1996.
- [108] Sherer, S. E., *Investigation of High-Order and Optimized Interpolation Methods with Implementation in a High-Order Overset Grid Fluid Dynamics Solver*, Ph.D. thesis, The Ohio State University, 2002.
- [109] Rai, M. M., “A Conservative Treatment of Zonal Boundaries for Euler Equation Calculations,” AIAA-Paper 1984-0164, 1984.
- [110] Noack, R. W., Boger, D. A., Kunz, R. F., and Carrica, P. M., “Suggar++: An Improved General Overset Grid Assembly Capability,” AIAA-Paper 2009-3992, 2009.
- [111] Hesthaven, J. S. and Warburton, T., *Nodal Discontinuous Galerkin Methods Algorithms, Analysis, and Applications*, Springer, 2008.
- [112] Abramowitz, M. and Stegun, I. A., *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*, Dover, New York, 1965.
- [113] Bassi, F. and Rebay, S., “Numerical Evaluation of Two Discontinuous Galerkin Methods for the Compressible Navier-Stokes Equations,” *International Journal for Numerical Methods in Fluids*, Vol. 40, No. 1-2, 2002, pp. 197–207.
- [114] Orkwis, P. D. and McRae, D. S., “Newton’s Method Solver for High-Speed Viscous Separated Flow-fields,” *AIAA Journal*, Vol. 30, 1992, pp. 78–85.
- [115] Tannehill, J. C., Anderson, D. A., and Pletcher, R. H., *Computational Fluid Mechanics and Heat Transfer*, Taylor & Francis, 2nd ed., 1997.
- [116] Fidkowski, K. J. and Roe, P. L., “An Entropy Adjoint Approach to Mesh Refinement,” *SIAM Journal on Scientific Computing*, Vol. 32, No. 3, 2010, pp. 1261–1287.
- [117] Roe, P. L., “Approximate Riemann Solvers, Parameter Vectors, and Difference Schemes,” *Journal of Computational Physics*, Vol. 43, 1981, pp. 357–372.
- [118] Fidkowski, K. J., Oliver, T. A., Lu, J., and Darmofal, D. L., “p-Multigrid Solutions of High-Order Discontinuous Galerkin Discretizations of the Compressible Navier-Stokes Equations,” *Journal of Computational Physics*, Vol. 207, 2005, pp. 92–113.

- [119] Harten, A. and Hyman, J. H., “Self Adjusting Grid Methods for One-Dimensional Hyperbolic Conservation Laws,” *Journal of Computational Physics*, Vol. 50, 1983, pp. 235–269.
- [120] Jameson, A., “Solution of the Euler Equations for Two Dimensional Transonic Flow by a Multigrid Method,” *Applied Mathematics and Computation*, Vol. 13, 1983, pp. 327–355.
- [121] Blazek, J., *Computational Fluid Dynamics: Principles and Applications*, Elsevier, 2001.
- [122] Veldhuizen, T. L., “Scientific Computing: C++ Versus Fortran: C++ has more than caught up,” *Dr. Dobb’s Journal of Software Tools*, Vol. 22, No. 11, 1997, pp. 34–91.
- [123] Myrnyy, V., “Recursive Function Templates as a Solution of Linear Algebra Expressions in C++,” *CoRR*, Vol. cs.MS/0302026, 2003.
- [124] Abrahams, D., *C++ template metaprogramming : concepts, tools, and techniques from Boost and beyond*, Pearson Education, Inc., 2004.
- [125] Whaley, R. C., Petitet, A., and Dongarra, J. J., “Automated Empirical Optimization of Software and the ATLAS Project,” *Parallel Computing*, Vol. 27, No. 1–2, 2001, pp. 3–35, Also available as University of Tennessee LAPACK Working Note #147, UT-CS-00-448, 2000 (www.netlib.org/lapack/lawns/lawn147.ps).
- [126] Whaley, R. C. and Petitet, A., “Minimizing Development and Maintenance Costs in Supporting Persistently Optimized BLAS,” *Software: Practice and Experience*, Vol. 35, No. 2, February 2005, pp. 101–121, <http://www.cs.utsa.edu/~whaley/papers/spercw04.ps>.
- [127] Tannehill, J. C., Anderson, D. A., and Pletcher, R. H., *Computational Fluid Mechanics and Heat Transfer*, Taylor & Francis, 2nd ed., 1997.
- [128] Saad, Y., *Iterative Methods for Sparse Linear Systems*, SIAM, 2nd ed., 2000.
- [129] SymPy Development Team, *SymPy: Python Library for Symbolic Mathematics*, <http://www.sympy.org>, 2009.
- [130] Car, D. and List, M., *Blockit*, <http://blockit.sourceforge.net>, 2010.
- [131] Butcher, J. B., “Coefficients for the Study of Runge-Kutta Integration Processes,” *J. Austral. Math. Soc*, Vol. 3, 1963, pp. 185–201.

- [132] Alexander, R., “Diagonally Implicit Runge-Kutta Methods for Stiff O.D.E.’s,” *SIAM J. Numer. Anal.*, Vol. 14, No. 6, 1977, pp. 1006–1021.
- [133] Oberkampf, W. L. and Roy, C. J., *Verification and Validation in Scientific Computing*, Cambridge University Press, 2010.
- [134] Polyanin, A. D. and Zaitsev, V. F., *Handbook of Nonlinear Partial Differential Equations*, Chapman & Hall/CRC, 2004.
- [135] Rai, M. M., *A Philosophy for Construction of Solution Adaptive Grids*, Ph.D. thesis, Iowa State University, Ames, 1982.
- [136] Chiocchia, G., “Exact Solutions to Transonic and Supersonic Flows,” Tech. Rep. AR-211, AGARD, 1985.
- [137] Ghia, U., Ghia, K. N., and Shin, C. T., “High-Re Solutions for Incompressible Flow Using Navier-Stokes Equations and a Multigrid Method,” *Journal of Computational Physics*, Vol. 48, 1982, pp. 387–411.
- [138] White, F. M., *Viscous Fluid Flow*, McGraw Hill, 3rd ed., 2006.
- [139] de Berg, M., Cheong, O., and van Kreveld, M., *Computational Geometry: Algorithms and Applications*, Springer, 3rd ed., 2008.
- [140] Rai, M. M., “An Implicit, Conservative, Zonal-Boundary Scheme for Euler Equation Calculations,” *Computers and Fluids*, Vol. 14, 1986, pp. 295–319.
- [141] Quinn, M. J., *Parallel Programming in C with MPI and OpenMP*, McGraw Hill, 2004.
- [142] Belk, D. M. and Maple, R. C., “Automated Assembly of Structured Grids for Moving Body Problems,” AIAA-Paper 95-1680-CP, 1995.
- [143] Petersson, N. A., “Hole-Cutting for Three-Dimensional Overlapping Grids,” *SIAM Journal on Scientific Computing*, Vol. 1999, No. 2, 21, pp. 646–665.
- [144] Noack, R. W., “A Direct Cut Approach for Overset Hole Cutting,” AIAA-Paper 2007-3853, 2007.

- [145] Nelder, J. A. and Mead, R., “A Simplex Method for Function Minimization,” *The Computer Journal*, Vol. 7, No. 4, 1965, pp. 308–313.
- [146] Stanewsky, E. and Thibert, J. J., “Airfoil SKF 1.1 with Maneuver Flap,” Tech. rep., Experimental Data Base for Computer Program Assessment, AGARD-AR-138, 1979.
- [147] Puterbaugh, R., Brown, J., and Battelle, R., “Impact of Heat Exchanger Location on Engine Performance,” SAE Technical Paper 2012-01-2168, 2012.
- [148] Wang, Y. Q., “Laminar Flow Through a Staggered Tube Bank,” *Journal of Thermophysics and Heat Transfer*, Vol. 18, 2004, pp. 557–559.
- [149] Wang, Y. Q., Jackson, P., and Phaneuf, T. J., “Turbulent Flow Through a Staggered Tube Bank,” *Journal of Thermophysics and Heat Transfer*, Vol. 20, No. 4, 2006, pp. 738–747.
- [150] Jiang, Y. and Przekwas, A. J., “Implicit, Pressure-Based Incompressible Navier-Stokes Equations Solver for Unstructured Meshes,” AIAA Paper 1994-0305, 1994.
- [151] Caspers, M. S., Maple, R. C., and Disimile, P. J., “CFD Investigation of Flow Past Idealized Engine Nacelle Clutter,” AIAA Paper 2007-3825, 2007.
- [152] Wang, Z. J., Hariharan, N., and Chen, R., “Recent Development on the Conservation Property of Chimera,” *International Journal of Computational Fluid Dynamics*, Vol. 15, No. 4, 2001, pp. 265–278.
- [153] Wang, Z. J., Buning, P. G., and Benek, J. A., “Critical Evaluation of Conservative and Non-Conservative Interface Treatment for Chimera Grids,” AIAA-Paper 1995-0077, 1995.
- [154] Wang, Z. J. and Yang, H. Q., “A Unified Conservative Zonal Interface Treatment for Arbitrarily Patched and Overlapped Grids,” AIAA-Paper 1994-0320, 1994.
- [155] Moon, Y. J. and Liou, M.-S., “Conservative Treatment of Boundary Interfaces for Overlaid Grids and Multi-level Grid Adaptations,” AIAA-Paper 1989-1980-CP, 1989.
- [156] Cali, P. M. and Couaillier, V., “Conservative Interfacing for Overset Grids,” AIAA-Paper 2000-1008, 2000.

- [157] Meakin, R. L., “On the Spatial and Temporal Accuracy of Overset Grid Methods for Moving Body Problems,” AIAA-Paper 1994-1925, 1994.
- [158] Mattingly, H. D., Heiser, W. H., and Pratt, D. T., *Aircraft Engine Design*, AIAA Education Series, 2nd ed., 2002.
- [159] Dixon, S. L. and Hall, C., *Fluid Mechanics and Thermodynamics of Turbomachinery*, Elsevier, 6th ed., 2010.
- [160] Ni, R., “A Multiple Grid Scheme for Solving the Euler Equations,” In: *Computational Fluid Dynamics Conference, 5th, Palo Alto, Calif., June 22, 23, 1981, Collection of Technical Papers.(A81-37526 16-34) New York, American Institute of Aeronautics and Astronautics, Inc., 1981, p. 257-264.*, Vol. 1, 1981, pp. 257–264.
- [161] Anderson, J. D., *Modern Compressible Flow: With Historical Perspective*, McGraw-Hill, 3rd ed., 2003.
- [162] Erlebacher, G., Hussaini, M. Y., and Shu, C.-W., “Interaction of a Shock with a Longitudinal Vortex,” *Journal of Fluid Mechanics*, Vol. 337, 1997, pp. 129–153.
- [163] Mattsson, K., Svard, M., Carpenter, M., and Nordstrom, J., “High-order Accurate Computations for Unsteady Aerodynamics,” *Computers and Fluids*, Vol. 36, No. 3, 2007, pp. 636–649.
- [164] Persson, P. O., Bonet, J., and Peraire, J., “Discontinuous Galerkin Solution of the Navier-Stokes Equations on Deformable Domains,” *Cmput. Methors Appl. Mech. Engrg.*, Vol. 198, No. 17-20, 2009, pp. 1585–1595.
- [165] Suhs, N. E., Rogers, S. E., and Dietz, W. E., “PEGASUS 5: An Automated Pre-Processor For Overset-Grid CFD,” AIAA-Paper 2002-3186, 2002.
- [166] Noack, R. W. and Belk, D. M., “Improved Interpolation for Viscous Overset Grids,” AIAA-Paper 1997-0199, 1997.
- [167] Hellman, B., Germain, B. S., and Sherer, S., “Critical Flight Conditions of Operational Rocketback Trajectories,” AIAA Paper 2012 (to be published), 2012.

- [168] Nichols, R. H. and Buning, P. G., *User's Manual for OVERFLOW 2.1*, University of Alabama and NASA Langley Research Center, version 2.1t ed., August 2008.
- [169] Coutanceau, M. and Bouard, R., "Experimental Determination of the Main Features of the Viscous Flow in the Wake of a Circular Cylinder in Uniform Translation. Part 1. Steady flow," *Journal of Fluid Mechanics*, Vol. 79, 1977, pp. 231–256.
- [170] Kwak, D. and Chakravarthy, S. R., "A Three-Dimensional Incompressible Navier-Stokes Flow Solver Using Primitive Variables," *AIAA Journal*, Vol. 24, 1984, pp. 390–396.
- [171] Galbraith, M. C. and Abdallah, S., "Implicit Solutions of Incompressible Navier-Stokes Equations Using the Pressure Gradient Method," *AIAA Journal*, Vol. 49, 2011, pp. 2491–2501.
- [172] *Message Passing Interface Forum*, <http://www mpi-forum.org/>, 2012.
- [173] Williams, A. and Escriba, V. J. B., *Boost Thread 3.1.0*, <http://www.boost.org/>, 2012.
- [174] Djomehri, M. J. and Biswas, R., "Performance Enhancement Strategies for Multi-Block Overset Grid CFD Applications," *Parallel Computing*, Vol. 29, 2003, pp. 1791–1810.
- [175] Brenner, S. and Scott, L., *The Mathematical Theory of Finite Element Methods*, Springer-Verlag, 1994.
- [176] Jameson, A. and Turkel, E., "Implicit Scheme and LU-Decompositions," *Mathematics*, Vol. 37, 1981, pp. 385–397.

Appendix A

Non-Dimensionalization of the Total Enthalpy Equation

For an inviscid adiabatic flow, the dimensional total enthalpy, \hat{H}_T , is given by

$$\hat{H}_T = \hat{h} + \frac{|\hat{\vec{V}}|^2}{2}, \quad (\text{A.1})$$

where \hat{h} is the dimensional static enthalpy, and $\hat{\vec{V}}$ is the dimensional velocity vector. For a calorically perfect gas, $\hat{h} = C_p \hat{T}$, where C_p is the specific heat at constant pressure. Hence, Eq. A.1 is written as

$$C_p \hat{H}_T = C_p \hat{T} + \frac{|\hat{\vec{V}}|^2}{2}. \quad (\text{A.2})$$

By introducing the non-dimensional variables T , T_T , and \vec{V} which are related to the dimensional variables \hat{T} , \hat{T}_T , and $\hat{\vec{V}}$ using the dimensional reference quantities \hat{T}_∞ and $\hat{\vec{V}}_\infty$ as

$$\begin{aligned} \hat{T} &= T \hat{T}_\infty, \\ \hat{T}_T &= T_T \hat{T}_\infty, \\ \hat{\vec{V}} &= \vec{V} \hat{\vec{V}}_\infty, \end{aligned} \quad (\text{A.3})$$

Eq. A.2 is written as

$$C_p T_T \hat{T}_\infty = C_p T \hat{T}_\infty + \frac{|\vec{V}|^2 \hat{V}_\infty^2}{2}. \quad (\text{A.4})$$

The reference quantities \hat{T}_∞ and \hat{V}_∞ along with C_p are now gathered

$$T_T = T + \frac{\hat{V}_\infty^2}{C_p \hat{T}_\infty} \frac{|\vec{V}|^2}{2}, \quad (\text{A.5})$$

and using the relationship

$$C_p = \frac{\gamma R}{\gamma - 1}, \quad (\text{A.6})$$

Eq. A.5 is written as

$$T_T = T + \frac{(\gamma - 1) \hat{V}_\infty^2}{\gamma R \hat{T}_\infty} \frac{|\vec{V}|^2}{2}. \quad (\text{A.7})$$

The non-dimensional total temperature equation is obtained by recognizing that the reference speed of sound squared is

$$\hat{c}_\infty^2 = \gamma R \hat{T}_\infty, \quad (\text{A.8})$$

and hence writing A.7 as

$$T_T = T + (\gamma - 1) M_\infty^2 \frac{|\vec{V}|^2}{2}. \quad (\text{A.9})$$

Appendix B

Comparison of Reciprocal Density

Projections via Model and Quadrature Analysis¹

Assume that $\rho = \rho(x)$ is the true partial differential equation density function on a small interval $\tau = [0, h]$ where $0 < h \ll 1$. Further, require that

$$0 < \rho_0 \leq \rho(x) \leq \rho_1 < \infty$$

where ρ_0 and ρ_1 are constants.

We will use the following Sobolev seminorm and norm notation (see [175]):

$$|v|_{m,p,\tau} = \left(\int_{\tau} \left| \frac{\partial^m v}{\partial x^m}(x) \right|^p dx \right)^{1/p} \quad \text{and} \quad \|v\|_{m,p,\tau} = \left(\sum_{j=0}^m |v|_{m,p,\tau}^p \right)^{1/p}.$$

The approximation will be in the space $P_q(\tau)$ which consists of polynomials of degree $\leq q$ on τ . These functions can be expanded in an orthonormal set of Legendre polynomials $\{\phi_0, \phi_1, \dots\}$ defined in the inner product

$$\langle w, z \rangle = \int_{\tau} w(x)z(x) dx.$$

¹This derivation was graciously provided by Dr. Donald French.

Here ϕ_ℓ is exactly a degree ℓ polynomial. (E.g. $\phi_0 = 1/\sqrt{h}$, $\phi_1(x) = 2\sqrt{6} h^{-3/2}(x - h/2)$, ...) We are interested in two different projection approximations of $1/\rho$; a modal approximation and a quadrature projection.

B.1 Modal Approximation

Find $u_M \in P_q(\tau)$ so

$$\int_{\tau} \rho u_M \eta \, dx = \int_{\tau} \eta \, dx \quad \forall \eta \in P_q(\tau).$$

(So, we have $u_M \cong 1/\rho$ since $\rho u_M \cong 1$ in the projection.). This leads to

$$A_M \vec{u}_M = \vec{R}_M$$

where the i th components of \vec{u}_M and \vec{R}_M are labeled $u_{M,i}$ and $R_{M,i}$ with

$$u_M(x) = \sum_{j=0}^q u_{M,j} \phi_j(x), \quad A_{M,ij} = \langle \rho \phi_i, \phi_j \rangle \quad \text{and} \quad R_{M,i} = \langle 1, \phi_i \rangle.$$

Note that A_M is a full matrix but $\vec{R}_M = (\sqrt{h}, 0, \dots, 0)^T$.

We will show that, except for a constant factor ρ_1/ρ_0 , u_M is a *best approximation* of $1/\rho$. That is, for all $\eta \in P_q(\tau)$,

$$\|u_M - 1/\rho\|_{0,2,\tau} \leq \frac{\rho_1}{\rho_0} \|1/\rho - \eta\|_{0,2,\tau}. \quad (\text{B.1})$$

B.2 Quadrature Projection

Find $u_Q \in P_q(\tau)$ so

$$\int_{\tau} u_Q \eta \, dx = (1/\rho, \eta)_h \quad \forall \eta \in P_q(\tau).$$

We define the error in the quadrature;

$$E_Q(f, g) = (f, g)_h - \int_{\tau} f g \, dx.$$

In this case $(f, g)_h$ is a $q+1$ point Gaussian quadrature approximation to $\int_{\tau} f(x)g(x)dx$ which, as is well-known, is exact for integrands in the space $P_{2q+1}(\tau)$. We find that

$$A_Q \vec{u}_Q = \vec{R}_Q \quad \text{with} \quad u_Q(x) = \sum_{j=0}^q u_{Q,j} \phi_j(x) \quad \text{and} \quad R_{Q,i} = (1/\rho, \phi_i)_h.$$

Since the Legendre polynomials form an orthonormal set and $A_{Q,ij} = \int_{\tau} \phi_i \phi_j dx$, we have $A_Q = I$. Thus $\vec{u}_Q = \vec{R}_Q$.

We will use the following numerical integration error estimate which is proved in the Appendix: If $f \in C^{q+1}(\tau)$ and $\sigma \in P_q(\tau)$ then

$$|E_Q(f, \sigma)| \leq Ch^{q+3/2} \|f\|_{q+1, \infty, \tau} \|\sigma\|_{0, 2, \tau} \quad (\text{B.2})$$

We will find that there exists a positive constant C so

$$\|e_Q\|_{0, 2, \tau} \leq C \left(h^{2(q+1)} \|1/\rho\|_{q+1, 2, \tau}^2 + h^{2q+3} \|1/\rho\|_{q+1, \infty, \tau}^2 \right)^{1/2} \quad (\text{B.3})$$

B.3 Preliminary Approximation Theory

For the proofs of (B.1) and (B.3) we well need the following. There are interpolation operators $\pi_h : W^{q+1, p}(\tau) \rightarrow P_q(\tau)$ which, for $\ell \leq r \leq q+1$, satisfy

$$\|(I - \pi_h)f\|_{\ell, p, \tau} \leq Ch^{r-\ell} \|f\|_{r, p, \tau} \quad 1 \leq p \leq \infty \quad (\text{B.4})$$

and $\pi_h \xi = \xi$ for $\xi \in P_q(\tau)$.

We will use the following inverse inequality for $\chi \in P_q(\tau)$; there exists a C which is independent of h so

$$\|\chi\|_{\ell, p, \tau} \leq Ch^{m-\ell+1/p+1/k} \|\chi\|_{m, k, \tau} \quad (\text{B.5})$$

where $0 \leq m \leq \ell$, $1 \leq p \leq \infty$ and $1 \leq k \leq \infty$.

The *arithmetic-geometric mean inequality* states that for scalars a and b , $|ab| \leq \delta a^2 + C_\delta b^2$, where $C_\delta = 1/(4\delta)$ and $\delta > 0$.

B.4 Modal Approximation Error Estimate

We first estimate the error in the modal approximation. Letting $e_M = u_M - 1/\rho$ we note the following orthogonality relation:

$$\int_{\tau} e_M \eta \, dx = 0 \quad \forall \eta \in P_q(\tau).$$

So, using this, we have, for any $\eta \in P_q(\tau)$,

$$\|e_M\|_{0,2,\tau}^2 \leq \frac{1}{\rho_0} \int_{\tau} \rho e_M^2 \, dx = \frac{1}{\rho_0} \int_{\tau} \rho e_M (\eta - 1/\rho) \, dx.$$

We have, after using the Cauchy-Schwarz inequality,

$$\|e_M\|_{0,2,\tau}^2 \leq \frac{\rho_1}{\rho_0} \|e_M\|_{0,2,\tau} \|1/\rho - \eta\|_{0,2,\tau}$$

and this estimate gives (B.1). If we choose $\eta = \pi_h(1/\rho)$ and then use (B.4), we have

$$\|e_M\|_{0,2,\tau} \leq C(\rho_0, \rho_1) h^{q+1} \|1/\rho\|_{q+1,2,\tau}.$$

B.5 Quadrature Approximation Error Estimate

We now turn to the analysis of the quadrature method. The “orthogonality” relation, if we take $e_Q = u_Q - 1/\rho$, is

$$\int_{\tau} e_Q \eta \, dx - E_Q(1/\rho, \eta) = 0 \quad \forall \eta \in P_q(\tau).$$

Using this relation we find that

$$\|e_Q\|_{0,2,\tau}^2 = \int_{\tau} e_Q (\eta - 1/\rho) \, dx + E_Q(1/\rho, u_Q - \eta).$$

The first term on the right side can be handled in a similar way to the modal analysis above. So, to continue, we take $f = 1/\rho$ and $\sigma = u_Q - \eta$ in (B.2) and find, after the triangle inequality on $u_Q - \eta = e_Q + (1/\rho - \eta)$,

$$\|e_Q\|_{0,2,\tau}^2 \leq \left| \int_{\tau} e_Q (\eta - 1/\rho) \, dx \right| + Ch^{q+3/2} \|1/\rho\|_{q+1,\infty,\tau} (\|e_Q\|_{0,2,\tau} + \|1/\rho - \eta\|_{0,2,\tau}).$$

Using the arithmetic-geometric mean we find that there exists a $0 < \delta < 1$ so

$$\|e_Q\|_{0,2,\tau}^2 \leq \delta \|e_Q\|_{0,2,\tau}^2 + C_\delta (\|1/\rho - \eta\|_{0,2,\tau}^2 + h^{2q+3} \|1/\rho\|_{q+1,\infty,\tau}).$$

Choosing $\delta = 1/2$, say, gives

$$\|e_Q\|_{0,2,\tau}^2 \leq C_\delta (\|1/\rho - \eta\|_{0,2,\tau}^2 + h^{2q+3} \|1/\rho\|_{q+1,2,\tau}).$$

Choosing $\eta = \pi_h(1/\rho)$ and using our interpolation estimates, we obtain our estimate (B.3).

B.6 Proof of Quadrature Error Estimate

In this section we provide the proof for our main quadrature error estimate (B.2). We assume that $\sigma \in P_q(\tau)$ and that the rule is exact for integrands in $P_{2q}(\tau)$ (e.g. $E_Q(\chi, 1) = 0$ for all $\chi \in P_{2q}(\tau)$). Our Gauss Quadrature is, in fact, exact on $P_{2q+1}(\tau)$. We also suppose that the weights in the integration rule are positive (This is true for our Gauss Quadrature). Also, we have $|(1, 1)_h| = \text{Meas}(\tau) = h$.

We use the interpolation operator π_h to assist in this analysis; thus, for $\sigma \in P_q(\tau)$,

$$E_Q(f, \sigma) = \int_\tau f \sigma \, dx - ((I - \pi_h)f, \sigma)_h - (\pi_h f, \sigma)_h.$$

Then, since $(\pi_h f, \sigma)_h = \int_\tau (\pi_h f) \sigma \, dx$, due to polynomial exactness, we have

$$\begin{aligned} |E_Q(f, \sigma)| &\leq \left| \int_\tau (I - \pi_h) f \sigma \, dx \right| + |((I - \pi_h) f, \sigma)_h| \\ &\leq \| (I - \pi_h) f \|_{0,\infty,\tau} (\|\sigma\|_{0,1,\tau} + \|\sigma\|_{0,\infty,\tau} |(1, 1)_h|) \\ &\leq Ch^{1/2} \| (I - \pi_h) f \|_{0,\infty,\tau} \|\sigma\|_{0,2,\tau} \end{aligned}$$

We used the inverse inequality and Cauchy-Schwarz on the last step. Using (B.4) we now obtain (B.2).

Appendix C

Iterative Sparse Matrix Solver

Preconditioners

Iterative sparse matrix are suitable for solving linear systems of equations

$$Ax = b \quad (\text{C.1})$$

where A is sparse matrix, i.e. it has few non-zero entries. The performance of an iterative sparse matrix solver can be significantly improved when combined with a preconditioner. The preconditioner is a matrix M with the property that $M^{-1} \approx A^{-1}$. Using a right preconditioning, the system of equations solved with the preconditioner is

$$AM^{-1}u = b, \quad u = Mx. \quad (\text{C.2})$$

The matrix M^{-1} must be relatively inexpensive to compute relative to A^{-1} , but must be a reasonable approximation to A^{-1} in order to accelerate the convergence rate of the iterative sparse matrix solver. Three preconditioners are included in the solver are listed below: LU-SGS, ILU0, and ILU1.

C.1 Lower-Upper Symmetric Gauss Seidel (LU-SGS)

The Lower-Upper Symmetric Gauss-Seidel (LU-SGS) scheme dates back to the work by Jameson and Turkel[176]. A more detailed review of the history of the method is given by Blazek[121]. The sparse matrix A is decomposed into a lower, L , a diagonal, D , and an upper, U , matrix as

$$A = L + D + U. \quad (\text{C.3})$$

The preconditioning matrix M , given by,

$$M = (L + D)D^{-1}(D + U) = (L + D)(I + D^{-1}U) = L + D + U + LD^{-1}U \approx A \quad (\text{C.4})$$

is an approximation to A . The approximation is reasonable so long as the term $LD^{-1}U$ is small. The term $LD^{-1}U$ is small so long as D^{-1} is small, i.e. the matrix A is diagonally dominant. The preconditioner is applied as part of the iterative matrix solver by solving

$$Mw = (L + D)D^{-1}(D + U)w = v, \quad (\text{C.5})$$

where w and v are intermediate vectors of the iterative process. The matrix M is solved in a two step process, first let

$$w' = D^{-1}(D + U)w = (I + D^{-1}U)w. \quad (\text{C.6})$$

Equation C.5 is now

$$(L + D)w' = v, \quad (\text{C.7})$$

which can be solved as

$$w' = D^{-1}(v - Lw'). \quad (\text{C.8})$$

The vector w is now obtained as

$$w = w' - D^{-1}Uw. \quad (\text{C.9})$$

The algorithm can be efficiently implemented by storing D^{-1} in memory for reuse when D^{-1} is a block diagonal matrix.

C.2 Incomplete Lower-Upper (ILU)

The Incomplete Lower-Upper (ILU) preconditioner decomposes the matrix A into a lower matrix, L , and an upper matrix, U , such that

$$LU \approx A. \quad (\text{C.10})$$

Either the upper matrix, U , or the lower matrix, L , has an identity matrix on the diagonal. The upper matrix is set to have an identity on the diagonal here as it reduces the operation count when working with a block sparse matrix. The equations for computing the entries in L and U are obtained by first choosing a non-zero pattern for L and U . Choosing a non-zero pattern such that L has the same non-zero pattern as the lower triangular part of A , and U has the same non-zero pattern as the upper triangular part of A , yields the ILU(0) preconditioner. Including one additional diagonal of non-zero entries in L and \underline{U} results in the ILU(1) preconditioner. The entries in L and U can be computed through a recursive relationship when the matrix A has a regular pattern. For structured meshes, A is a penta- and hepta-diagonal matrix for two- and three-dimensional problems, respectively. The recursive relationships for obtaining L and U are outlined by Saad [128] when L has an identity matrix on the diagonal. The recursive relationships when U has a diagonal identity matrix are provided here.

The ILU preconditioner is applied as part of the iterative matrix solver by solving the system

$$LUw = v \quad (\text{C.11})$$

where w and v are temporary vectors that are part of the iterative matrix solver. The system is solved by defining

$$Uw = w'. \quad (\text{C.12})$$

Equation C.11 is solved in a two step process by first solving

$$Lw' = v, \quad (\text{C.13})$$

and then

$$Uw = w'. \quad (\text{C.14})$$

C.2.1 ILU(0) 2D

The notation for the blocks in the matrix A is given for a 3×3 cell mesh is

$$\begin{bmatrix} D_{0,0} & \xi_{0,0}^{max} & 0 & \eta_{0,0}^{max} & 0 & 0 & 0 & 0 & 0 \\ \xi_{1,0}^{min} & D_{1,0} & \xi_{1,0}^{max} & 0 & \eta_{1,0}^{max} & 0 & 0 & 0 & 0 \\ 0 & \xi_{1,0}^{min} & D_{2,0} & 0 & 0 & \eta_{2,0}^{max} & 0 & 0 & 0 \\ \eta_{0,1}^{min} & 0 & 0 & D_{0,1} & \xi_{0,1}^{max} & 0 & \eta_{0,1}^{max} & 0 & 0 \\ 0 & \eta_{1,1}^{min} & 0 & \xi_{1,1}^{min} & D_{1,1} & \xi_{1,1}^{max} & 0 & \eta_{1,1}^{max} & 0 \\ 0 & 0 & \eta_{2,1}^{min} & 0 & \xi_{2,1}^{min} & D_{2,1} & 0 & 0 & \eta_{2,1}^{max} \\ 0 & 0 & 0 & \eta_{0,2}^{min} & 0 & 0 & D_{0,2} & \xi_{0,2}^{max} & 0 \\ 0 & 0 & 0 & 0 & \eta_{1,2}^{min} & 0 & \xi_{1,2}^{min} & D_{1,2} & \xi_{1,2}^{max} \\ 0 & 0 & 0 & 0 & 0 & \eta_{2,2}^{min} & 0 & \xi_{2,2}^{min} & D_{2,2} \end{bmatrix} \begin{bmatrix} Ax \\ x_0,0 \\ x_1,0 \\ x_2,0 \\ x_0,1 \\ x_1,1 \\ x_2,1 \\ x_0,2 \\ x_1,2 \\ x_2,2 \end{bmatrix} = \begin{bmatrix} b_{0,0} \\ b_{1,0} \\ b_{2,0} \\ b_{0,1} \\ b_{1,1} \\ b_{2,1} \\ b_{0,2} \\ b_{1,2} \\ b_{2,2} \end{bmatrix}, \quad (\text{C.15})$$

where the subscript corresponds to the i and j cell index. For the ILU(0) preconditioner, the L and U matrices are

$$\begin{aligned}
L &= \begin{bmatrix} Ld_{0,0} \\ L\xi_{1,0} \quad Ld_{1,0} \\ L\xi_{2,0} \quad Ld_{2,0} \\ L\eta_{0,1} \quad 0 \quad Ld_{0,1} \\ L\eta_{1,1} \quad L\xi_{1,1} \quad Ld_{1,1} \\ L\eta_{2,1} \quad L\xi_{2,1} \quad Ld_{2,1} \\ L\eta_{0,2} \quad 0 \quad Ld_{0,2} \\ L\eta_{1,2} \quad L\xi_{1,2} \quad Ld_{1,2} \\ L\eta_{2,2} \quad L\xi_{2,2} \quad Ld_{2,2} \end{bmatrix}, \\
U &= \begin{bmatrix} 1 \quad U\xi_{0,0} \quad U\eta_{0,0} \\ 1 \quad U\xi_{1,0} \quad U\eta_{1,0} \\ 1 \quad 0 \quad U\eta_{2,0} \\ 1 \quad U\xi_{0,1} \quad U\eta_{0,1} \\ 1 \quad U\xi_{1,1} \quad U\eta_{1,1} \\ 1 \quad 0 \quad U\eta_{2,1} \\ 1 \quad U\xi_{0,2} \\ 1 \quad U\xi_{1,2} \\ 1 \end{bmatrix}. \tag{C.16}
\end{aligned}$$

The recursive relationships used to compute the entries in L and U are obtained by multiplying L and U and equating the resulting matrix only with non-zero entries in the matrix A , i.e., $LU = A$ only where A is non-zero. This results in the equations

$$\begin{aligned}
L\eta_{i,j} &= \eta_{i,j}^{\min}, \\
L\xi_{i,j} &= \xi_{i,j}^{\min}, \\
Ld_{i,j} + L\xi_{i,j}U\xi_{i-1,j} + L\eta_{i,j}f_{i,j-1} &= D_{i,j}, \\
Ld_{i,j}U\xi_{i,j} &= \xi_{i,j}^{\max}, \\
Ld_{i,j}U\eta_{i,j} &= \eta_{i,j}^{\max},
\end{aligned} \tag{C.17}$$

which are solved to obtain the recursive relationships

$$\begin{aligned}
L\eta_{i,j} &= \eta_{i,j}^{\min}, \\
L\xi_{i,j} &= \xi_{i,j}^{\min}, \\
Ld_{i,j} &= D_{i,j} - L\xi_{i,j}U\xi_{i-1,j} - L\eta_{i,j}U\eta_{i,j-1}, \\
U\xi_{i,j} &= Ld_{i,j}^{-1}\xi_{i,j}^{\max}, \\
U\eta_{i,j} &= Ld_{i,j}^{-1}\eta_{i,j}^{\max},
\end{aligned} \tag{C.18}$$

where i and j are increasing. Block matrices with a negative index when $i = 0$ or $j = 0$ are set to zero. Note that the lower matrix L in Eq. C.18 is identical to the entries in the matrix A , and only the upper matrix U needs to be stored in memory. The forward solve in Eq. C.13 is

$$\begin{aligned}
Lw' &= v \Rightarrow \\
L\xi_{i,j}w'_{i-1,j} + L\eta_{i,j}w'_{i,j-1} + Ld_{i,j}w'_{i,j} &= v_{i,j},
\end{aligned} \tag{C.19}$$

which after solving for $w'_{i,j}$ gives

$$w'_{i,j} = Ld_{i,j}^{-1}(v_{i,j} - (L\xi_{i,j}w_{i-1,j} + L\eta_{i,j}w_{i,j-1})) \tag{C.20}$$

where i and j are increasing. The backward solve in Eq. C.14 is

$$\begin{aligned}
Uw &= w' \Rightarrow \\
w_{i,j} + U\xi_{i,j}w_{i+1,j} + U\eta_{i,j}w_{i,j+1} &= w'_{i,j}
\end{aligned} \tag{C.21}$$

which after solving for $w_{i,j}$ gives

$$w_{i,j} = w'_{i,j} - (U\xi_{i,j}w'_{i+1,j} + U\eta_{i,j}w'_{i,j+1}) \tag{C.22}$$

where i and j are decreasing.

C.2.2 ILU(0) 3D

In three-dimensions, the recursive equations that arise from equating $LU = A$ only where A is non-zero are

$$\begin{aligned}
L\xi_{i,j,k} &= \xi_{i,j,k}^{min}, \\
L\eta_{i,j,k} &= \eta_{i,j,k}^{min}, \\
L\xi_{i,j,k} &= \xi_{i,j,k}^{min}, \\
Ld_{i,j,k} + L\xi_{i,j,k}U\xi_{i-1,j,k} + L\eta_{i,j,k}U\eta_{i,j-1,k} + L\xi_{i,j,k}U\xi_{i,j,k-1} &= D_{i,j,k}, \\
Ld_{i,j,k}U\xi_{i,j,k} &= \xi_{i,j,k}^{max}, \\
Ld_{i,j,k}U\eta_{i,j,k} &= \eta_{i,j,k}^{max}, \\
Ld_{i,j,k}U\xi_{i,j,k} &= \xi_{i,j,k}^{max},
\end{aligned} \tag{C.23}$$

which are solved to obtain the recursive relationships

$$\begin{aligned}
L\zeta_{i,j,k} &= \zeta_{i,j,k}^{\min}, \\
L\eta_{i,j,k} &= \eta_{i,j,k}^{\min}, \\
L\xi_{i,j,k} &= \xi_{i,j,k}^{\min}, \\
Ld_{i,j,k} &= D_{i,j,k} - (L\xi_{i,j,k} U \xi_{i-1,j,k} + L\eta_{i,j,k} U \eta_{i,j-1,k} + L\zeta_{i,j,k} U \zeta_{i,j-1,k}), \\
U\xi_{i,j,k} &= Ld_{i,j,k}^{-1} \xi_{i,j,k}^{\max}, \\
U\eta_{i,j,k} &= Ld_{i,j,k}^{-1} \eta_{i,j,k}^{\max}, \\
U\zeta_{i,j,k} &= Ld_{i,j,k}^{-1} \zeta_{i,j,k}^{\max},
\end{aligned} \tag{C.24}$$

where i , j , and k are increasing. Again, the lower matrix L in Eq. C.18 is identical to the entries in the matrix A , and only the upper matrix U needs to be stored in memory. The forward solve in Eq. C.13 is

$$\begin{aligned}
Lw' &= v \Rightarrow \\
L\xi_{i,j,k} w'_{i-1,j,k} + L\eta_{i,j,k} w'_{i,j-1,k} + L\zeta_{i,j,k} w'_{i,j-1,k} + Ld_{i,j,k} w'_{i,j,k} &= v_{i,j,k},
\end{aligned} \tag{C.25}$$

which after solving for $w'_{i,j,k}$ gives

$$w'_{i,j,k} = Ld_{i,j,k}^{-1} (v_{i,j,k} - (L\xi_{i,j,k} w'_{i-1,j,k} + L\eta_{i,j,k} w'_{i,j-1,k} + L\zeta_{i,j,k} w'_{i,j-1,k})) \tag{C.26}$$

where i , j , and k are increasing. The backward solve in Eq. C.14 is

$$\begin{aligned}
Uw &= w' \Rightarrow \\
w_{i,j,k} + U\xi_{i,j,k} w_{i+1,j,k} + U\eta_{i,j,k} w_{i,j+1,k} + U\zeta_{i,j,k} w_{i,j,k+1} &= w'_{i,j,k},
\end{aligned} \tag{C.27}$$

which after solving for $w_{i,j,k}$ gives

$$w_{i,j,k} = w'_{i,j,k} - (U\xi_{i,j,k} w_{i+1,j,k} + U\eta_{i,j,k} w_{i,j+1,k} + U\zeta_{i,j,k} w_{i,j,k+1}) \tag{C.28}$$

where i , j , and k are decreasing.

C.2.3 ILU(1) 2D

For the ILU(1) preconditioner, the L and U matrices are

$$L = \begin{bmatrix} Ld_{0,0} \\ L\xi_{1,0} & Ld_{1,0} \\ 0 & L\xi_{2,0} & Ld_{2,0} \\ L\eta_{0,1} & Lc_{0,1} & 0 & Ld_{0,1} \\ L\eta_{1,1} & Lc_{1,1} & L\xi_{1,1} & Ld_{1,1} \\ L\eta_{2,1} & 0 & L\xi_{2,1} & Ld_{2,1} \\ L\eta_{0,2} & Lc_{0,2} & 0 & Ld_{0,2} \\ L\eta_{1,2} & Lc_{1,2} & L\xi_{1,2} & Ld_{1,2} \\ L\eta_{2,2} & 0 & L\xi_{2,2} & Ld_{2,2} \end{bmatrix},$$

$$U = \begin{bmatrix} 1 & U\xi_{0,0} & 0 & U\eta_{0,0} \\ 1 & U\xi_{1,0} & Uc_{1,0} & U\eta_{1,0} \\ 1 & 0 & Uc_{2,0} & U\eta_{2,0} \\ 1 & U\xi_{0,1} & 0 & U\eta_{0,1} \\ 1 & U\xi_{1,1} & Uc_{1,1} & U\eta_{1,1} \\ 1 & 0 & Uc_{2,1} & U\eta_{2,1} \\ 1 & U\xi_{0,2} & 0 & \\ 1 & U\xi_{1,2} & & \\ & & 1 \end{bmatrix}. \quad (\text{C.29})$$

where one extra diagonal that was zero in the A matrix is set to be non-zero. Additional equations are required to formulate the recursive relationship. The equations obtained by equating $LU = A$ only where A is non-zero as well as the auxiliary equations for the fill in are

$$\begin{aligned}
L\eta_{i,j} &= \eta_{i,j}^{min}, \\
Lc_{i,j} + L\eta_{i,j}U\xi_{i,j-1} &= 0, \\
L\xi_{i,j} + L\eta_{i,j}Uc_{i,j-1} &= \xi_{i,j}^{min}, \\
Ld_{i,j} + L\xi_{i,j}U\xi_{i-1,j} + L\eta_{i,j}U\eta_{i,j-1} + Lc_{i,j}Uc_{i+1,j-1} &= D_{i,j}, \\
Ld_{i,j}U\xi_{i,j} + Lc_{i,j}U\eta_{i+1,j-1} &= \xi_{i,j}^{max}, \\
L\xi_{i,j}U\eta_{i-1,j} + Ld_{i,j}Uc_{i,j} &= 0, \\
Ld_{i,j}U\eta_{i,j} &= \eta_{i,j}^{max},
\end{aligned} \tag{C.30}$$

which are solved to obtain the recursive relationships

$$\begin{aligned}
L\eta_{i,j} &= \eta_{i,j}^{min}, \\
Lc_{i,j} &= -L\eta_{i,j}U\xi_{i,j-1}, \\
L\xi_{i,j} &= \xi_{i,j}^{min} - L\eta_{i,j}Uc_{i,j-1}, \\
Ld_{i,j} &= D_{i,j} - L\xi_{i,j}U\xi_{i-1,j} - L\eta_{i,j}U\eta_{i,j-1} - Lc_{i,j}Uc_{i+1,j-1}, \\
U\xi_{i,j} &= Ld_{i,j}^{-1}(\xi_{i,j}^{max} - Lc_{i,j}U\eta_{i+1,j-1}), \\
Uc_{i,j} &= Ld_{i,j}^{-1}(-L\xi_{i,j}U\eta_{i-1,j}), \\
U\eta_{i,j} &= Ld_{i,j}^{-1}(\eta_{i,j}^{max}),
\end{aligned} \tag{C.31}$$

where i and j are increasing. Unlike the ILU(0) preconditioner, portions of the L matrix as well as the complete U are stored in memory. The forward solve in Eq. C.13 is

$$\begin{aligned}
Lw' &= v \Rightarrow \\
L\xi_{i,j}w'_{i-1,j} + L\eta_{i,j}w'_{i,j-1} + Lc_{i,j}w'_{i+1,j-1} + Ld_{i,j}w'_{i,j} &= v_{i,j}
\end{aligned} \tag{C.32}$$

which after solving for $w'_{i,j}$ gives

$$w'_{i,j} = Ld_{i,j}^{-1} (v_{i,j} - L\xi_{i,j}w'_{i-1,j} - L\eta_{i,j}w'_{i,j-1} - Lc_{i,j}w'_{i+1,j-1}) \quad (\text{C.33})$$

where i and j are increasing. The backward solve in Eq. C.14 is

$$\begin{aligned} Uw &= w' \Rightarrow \\ w_{i,j} + U\xi_{i,j}w_{i+1,j} + U\eta_{i,j}w_{i,j+1} + Uc_{i,j}w_{i-1,j+1} &= w'_{i,j} \end{aligned} \quad (\text{C.34})$$

which after solving for $w_{i,j}$ gives

$$w_{i,j} = w'_{i,j} - U\xi_{i,j}w_{i+1,j} - U\eta_{i,j}w_{i,j+1} - Uc_{i,j}w_{i-1,j+1} \quad (\text{C.35})$$

where i and j are decreasing.

C.2.4 ILU(1) 3D

In three-dimensions, the recursive equations that arise from equating $LU = A$ only where A is non-zero and including the auxiliary equations are

$$\begin{aligned}
L\xi_{i,j,k} &= \zeta_{i,j,k}^{\min} \\
Le_{i,j,k} + L\xi_{i,j,k}U\xi_{i,j,k-1} &= 0 \\
L\eta_{i,j,k} &= \eta_{i,j,k}^{\min} \\
Lc_{i,j,k} + L\eta_{i,j,k}U\xi_{i,j-1,k} &= 0 \\
L\xi_{i,j,k} + L\eta_{i,j,k}Uc_{i,j-1,k} + L\xi_{i,j,k}Ue_{i,j,k-1} &= \xi_{i,j,k}^{\min} \\
Ld_{i,j,k} + L\xi_{i,j,k}U\xi_{i-1,j,k} + L\eta_{i,j,k}U\eta_{i,j-1,k} + \\
L\zeta_{i,j,k}U\xi_{i,j,k-1} + Lc_{i,j,k}Uc_{i+1,j-1,k} + Le_{i,j,k}Ue_{i+1,j,k-1} &= D_{i,j,k} \quad (\text{C.36}) \\
Ld_{i,j,k}U\xi_{i,j,k} + Lc_{i,j,k}U\eta_{i+1,j-1,k} + Le_{i,j,k}U\zeta_{i+1,j,k-1} &= \xi_{i,j,k}^{\max} \\
L\xi_{i,j,k}U\eta_{i-1,j,k} + Ld_{i,j,k}Uc_{i,j,k} &= 0 \\
Ld_{i,j,k}U\eta_{i,j,k} &= \eta_{i,j,k}^{\max} \\
Ld_{i,j,k}Ue_{i,j,k} + L\xi_{i,j,k}U\xi_{i-1,j,k} &= 0 \\
Ld_{i,j,k}U\xi_{i,j,k} &= \zeta_{i,j,k}^{\max}
\end{aligned}$$

which are solved to obtain the recursive relationships

$$\begin{aligned}
L\zeta_{i,j,k} &= \zeta_{i,j,k}^{\min} \\
Le_{i,j,k} &= -L\zeta_{i,j,k}U\xi_{i,j,k-1} \\
L\eta_{i,j,k} &= \eta_{i,j,k}^{\min} \\
Lc_{i,j,k} &= -L\eta_{i,j,k}U\xi_{i,j-1,k} \\
L\xi_{i,j,k} &= \xi_{i,j,k}^{\min} - (L\eta_{i,j,k}Uc_{i,j-1,k} + L\zeta_{i,j,k}Ue_{i,j,k-1}) \\
Ld_{i,j,k} &= D_{i,j,k} - (L\xi_{i,j,k}U\xi_{i-1,j,k} + L\eta_{i,j,k}U\eta_{i,j-1,k} + L\zeta_{i,j,k}U\zeta_{i,j,k-1} \\
&\quad + Lc_{i,j,k}Uc_{i+1,j-1,k} + Le_{i,j,k}Ue_{i+1,j,k-1}) \\
U\xi_{i,j,k} &= Ld_{i,j,k}^{-1}(\xi_{i,j,k}^{\max} - (Lc_{i,j,k}U\eta_{i+1,j-1,k} + Le_{i,j,k}U\zeta_{i+1,j,k-1})) \\
Uc_{i,j,k} &= Ld_{i,j,k}^{-1}(-L\xi_{i,j,k}U\eta_{i-1,j,k}) \\
U\eta_{i,j,k} &= Ld_{i,j,k}^{-1}\eta_{i,j,k}^{\max} \\
Ue_{i,j,k} &= Ld_{i,j,k}^{-1}(-L\xi_{i,j,k}U\zeta_{i-1,j,k}) \\
U\zeta_{i,j,k} &= Ld_{i,j,k}^{-1}\zeta_{i,j,k}^{\max}
\end{aligned} \tag{C.37}$$

where i , j , and k are increasing. Again, the portions of the lower matrix L and the complete upper matrix U needs to be stored in memory. The forward solve in Eq. C.13 is

$$\begin{aligned}
Lw' &= v \Rightarrow \\
L\xi_{i,j,k}w'_{i-1,j,k} + L\eta_{i,j,k}w'_{i,j-1,k} + L\zeta_{i,j,k}w'_{i,j,k-1} + \\
Lc_{i,j,k}w'_{i+1,j-1,k} + Le_{i,j,k}w'_{i+1,j,k-1} + Ld_{i,j,k}w'_{i,j,k} &= v_{i,j,k}
\end{aligned} \tag{C.38}$$

which after solving for $w'_{i,j,k}$ gives

$$w'_{i,j,k} = Ld_{i,j,k}^{-1}(v_{i,j,k} - (L\xi_{i,j,k}w'_{i-1,j,k} + L\eta_{i,j,k}w'_{i,j-1,k} + L\zeta_{i,j,k}w'_{i,j,k-1} + Lc_{i,j,k}w'_{i+1,j-1,k} + Le_{i,j,k}w'_{i+1,j,k-1})) \tag{C.39}$$

where i , j , and k are increasing. The backward solve in Eq. C.14 is

$$\begin{aligned}
Uw &= w' \Rightarrow \\
w_{i,j,k} + U\xi_{i,j,k}w_{i+1,j,k} + U\eta_{i,j,k}w_{i,j+1,k} + \\
U\zeta_{i,j,k}w_{i,j,k+1} + Uc_{i,j,k}w_{i-1,j+1,k} + Ue_{i,j,k}w_{i-1,j,k+1} &= w'_{i,j,k} \quad (\text{C.40})
\end{aligned}$$

which after solving for $w_{i,j,k}$ gives

$$w_{i,j,k} = w'_{i,j,k} - (U\xi_{i,j,k}w_{i+1,j,k} + U\eta_{i,j,k}w_{i,j+1,k} + U\zeta_{i,j,k}w_{i,j,k+1} + Uc_{i,j,k}w_{i-1,j+1,k} + Ue_{i,j,k}w_{i-1,j,k+1}) \quad (\text{C.41})$$

where i , j , and k are decreasing.

Appendix D

Solution to Multidimensional Burger's Equation¹

Given that the one-dimensional Burger's Equation

$$\frac{1}{2} \frac{\partial}{\partial C} (U^2) - \mu \frac{\partial^2 U}{\partial C^2} = 0 \quad (\text{D.1})$$

has an analytical solution

$$U(C) = -\tanh\left(\frac{C}{2\mu}\right), \quad (\text{D.2})$$

the following formula

$$u(\vec{x}) = U(\vec{c} \cdot \vec{x}) \quad \text{where } \vec{c} = \begin{bmatrix} c_x & c_y & c_z \end{bmatrix} \quad (\text{D.3})$$

can be shown to be a solution to the multidimensional Burger's Equation

$$\frac{1}{2} \nabla \cdot (\vec{c} u^2) - \mu \nabla \cdot \nabla u = 0 \quad \text{where} \quad \nabla = \begin{bmatrix} \partial/\partial x, & \partial/\partial y, & \partial/\partial z \end{bmatrix} \quad (\text{D.4})$$

Thinking of $C = \vec{c} \cdot \vec{x}$ and noting that $\nabla C = \vec{c}$ as well as $\nabla u = \vec{c}(\partial U / \partial C)$, the advective term becomes

$$\frac{1}{2} \nabla \cdot (\vec{c} u^2) = \vec{c} u \nabla u = |\vec{c}|^2 U \frac{\partial U}{\partial C} = \frac{1}{2} |\vec{c}|^2 \frac{\partial}{\partial C} (U^2) \quad (\text{D.5})$$

where $|\vec{c}|^2 = \vec{c} \cdot \vec{c}$ and, of course, $u = U$. Similarly, the diffusion term can be written using the Chain Rule as

¹This derivation was graciously provided by Dr. Donald French.

$$\mu \nabla \cdot \nabla u = \mu \nabla \cdot \left(\vec{c} \frac{\partial U}{\partial C} \right) = \mu |\vec{c}|^2 \frac{\partial^2 U}{\partial C^2}. \quad (\text{D.6})$$

Thus, the multidimensional Burger's equation can be written as a one-dimensional Burger's equation;

$$\frac{1}{2} \nabla \cdot (\vec{c} u^2) - \mu \nabla \cdot \nabla u = |\vec{c}|^2 \left(\frac{1}{2} \frac{\partial}{\partial C} (U^2) + \mu \frac{\partial^2 U}{\partial C^2} \right) = |\vec{c}|^2 \cdot 0 = 0. \quad (\text{D.7})$$

However, the magnitude $|\vec{c}|^2$ should be unity for an unsteady Burger's equation.

Appendix E

Bassi and Rebay Scheme 2 Linearization

The BR2[13, 24] scheme is outlined here for the one-dimensional steady heat equation

$$-U_{xx} = \sigma(x), \quad (\text{E.1})$$

to demonstrate the implicit implementation. Equation E.1 is first put in a weak form by multiplying it with the basis functions ψ and integrating by parts to yield

$$-\int_{\Gamma_e} \psi U_x d\Gamma + \int_{\Omega_e} \psi_x U_x d\Omega = \int_{\Omega_e} \psi \sigma(x) d\Omega. \quad (\text{E.2})$$

The BR2 scheme also relies on volume polynomial lifting operators which are associated with the faces of a cell as shown in Fig. E.1. In higher dimensions, the lifting operators are vectors of polynomials with a vector length equivalent to the number of spatial dimensions of the problem.

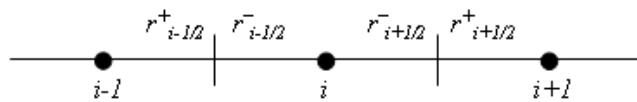


Figure E.1: BR2 lifting operators for cell i

To arrive at the BR2 discretization of Eq. E.2, the analytical solution, U , is approximated with a cell local polynomial expansion of the solution, u , the lifting operators, r , of the i^{th} cell, defined by

$$\begin{aligned}
\int_{\Omega_{i+1}} \psi r_{i+1/2}^+ d\Omega &= \int_{\Gamma_{i+1/2}} \psi \frac{1}{2} [u] \vec{n} d\Gamma, \\
\int_{\Omega_i} \psi r_{i+1/2}^- d\Omega &= \int_{\Gamma_{i+1/2}} \psi \frac{1}{2} [u] \vec{n} d\Gamma, \\
\int_{\Omega_i} \psi r_{i-1/2}^- d\Omega &= \int_{\Gamma_{i-1/2}} \psi \frac{1}{2} [u] \vec{n} d\Gamma, \\
\int_{\Omega_{i-1}} \psi r_{i-1/2}^+ d\Omega &= \int_{\Gamma_{i-1/2}} \psi \frac{1}{2} [u] \vec{n} d\Gamma,
\end{aligned} \tag{E.3}$$

are introduced, and the average operator is used for the boundary integral terms, to yield the final expression for the BR2 scheme

$$-\int_{\Gamma_e} \psi \{u_x + r_i\} d\Gamma + \int_{\Omega_e} \psi_x (u_x + R_i) d\Omega = \int_{\Omega_e} \psi \sigma(x) d\Omega, \tag{E.4}$$

where

$$R_i = \sum r_i^-. \tag{E.5}$$

In order to formulate an implicit scheme, an expansion is expressed as a vector product

$$u(\xi) = \sum_{i=0}^N \psi_i(\xi) u_i = \begin{pmatrix} \psi_0 & \psi_1 & \dots & \psi_N \end{pmatrix} \begin{pmatrix} u_0 \\ u_1 \\ \vdots \\ u_N \end{pmatrix} = \psi \vec{u} \tag{E.6}$$

Thus, ψ is implied to be a row vector of the test functions when multiplying either a matrix or a vector.

By using Eq. E.3, the coefficient vector for the four lifting operators shown in Fig. E.1 are expressed in a matrix form as

$$\begin{aligned}
\vec{r}_{i-1/2}^+ &= V_{i-1}^{-1} (M_{i-1}^{+-} \vec{u}_i - M_{i-1}^{++} \vec{u}_{i-1}) \\
\vec{r}_{i-1/2}^- &= V_i^{-1} (M_i^{--} \vec{u}_i - M_i^{-+} \vec{u}_{i-1}) \\
\vec{r}_{i+1/2}^- &= V_i^{-1} (M_i^{+-} \vec{u}_{i+1} - M_i^{++} \vec{u}_i) \\
\vec{r}_{i+1/2}^+ &= V_{i+1}^{-1} (M_{i+1}^{--} \vec{u}_{i+1} - M_{i+1}^{-+} \vec{u}_i)
\end{aligned} \tag{E.7}$$

where

$$\begin{aligned}
V_i &= \int_{\Omega_e} \psi \psi J_i^{-1} d\Omega_e \\
M_i^{-+} &= \int_{\Gamma_e} \psi(-1) \psi(1) \vec{n}_{i-1/2} d\Gamma \\
M_i^{--} &= \int_{\Gamma_e} \psi(-1) \psi(-1) \vec{n}_{i-1/2} d\Gamma \\
M_i^{++} &= \int_{\Gamma_e} \psi(1) \psi(1) \vec{n}_{i+1/2} d\Gamma \\
M_i^{+-} &= \int_{\Gamma_e} \psi(1) \psi(-1) \vec{n}_{i+1/2} d\Gamma,
\end{aligned} \tag{E.8}$$

which are all matrices. The average operation in the boundary integral of Eq. E.4 for both the left and right face of the cell is written as

$$\begin{aligned}
\{\vec{r}_{i+1/2}\} &= \frac{1}{2} ((\psi(-1) V_{i+1}^{-1} M_{i+1}^{--} + \psi(1) V_i^{-1} M_i^{+-}) \vec{u}_{i+1} - (\psi(-1) V_{i+1}^{-1} M_{i+1}^{-+} + \psi(1) V_i^{-1} M_i^{++}) \vec{u}_i) \\
\{\vec{r}_{i-1/2}\} &= \frac{1}{2} ((\psi(1) V_{i-1}^{-1} M_{i-1}^{+-} + \psi(-1) V_i^{-1} M_i^{--}) \vec{u}_i - (\psi(1) V_{i-1}^{-1} M_{i-1}^{++} - \psi(-1) V_i^{-1} M_i^{-+}) \vec{u}_{i-1}).
\end{aligned} \tag{E.9}$$

The final contributions to the block tri-diagonal system from the boundary integral of the lifting operators from the subtraction $\{\vec{r}_{i+1/2}\} - \{\vec{r}_{i-1/2}\}$.

$$\begin{aligned}
\{\vec{r}_{i+1/2}\} - \{\vec{r}_{i-1/2}\} &= \frac{1}{2} ((\psi(-1) V_{i+1}^{-1} M_{i+1}^{--} + \psi(1) V_i^{-1} M_i^{+-}) \vec{u}_{i+1} \\
&\quad - (\psi(-1) V_{i+1}^{-1} M_{i+1}^{-+} + \psi(1) V_i^{-1} M_i^{++} + \psi(-1) V_i^{-1} M_i^{--} + \psi(1) V_{i-1}^{-1} M_{i-1}^{+-}) \vec{u}_i \\
&\quad + (\psi(1) V_{i-1}^{-1} M_{i-1}^{++} - \psi(-1) V_i^{-1} M_i^{-+}) \vec{u}_{i-1})
\end{aligned} \tag{E.10}$$

The volume integral contribution of the lifting operators to the block tri-diagonal system is

$$\int_{\Omega_e} \psi_x R_i d\Omega \Rightarrow \int_{\Omega_e} \psi_x (\psi V_i^{-1} (-M_i^{-+} \vec{u}_{i-1} + (M_i^{--} - M_i^{++}) \vec{u}_i + M_i^{+-} \vec{u}_{i+1})) d\Omega_e. \tag{E.11}$$