The discontinuous Galerkin (DG) finite element methods have recently become popular for the solution of systems of conservation laws to arbitrary order of accuracy. The DG methods, originally introduced in 1973 for the solution of neutron transport equation, did not come to the attention of computational fluid dynamics (CFD) research scientists and scholars until 1990s, and have enjoyed a prosperous progress for the last few decades. In contrast to the enormous advances in the theoretical and numerical analysis of the DG methods, the development of a viable, attractive, competitive and ultimately superior DG method over the more mature and well-established second-order methods is relatively an untouched area.

The objective of the effort presented in this PhD work is to develop a parallel, implicit reconstructed discontinuous Galerkin (RDG) method using Taylor basis for the solution of the compressible Navier-Stokes equations on 3D hybrid grids. This third-order accurate RDG method is based on a hierarchical weighed essentially non-oscillatory reconstruction scheme, termed as HWENO(P1P2) to indicate that a quadratic polynomial solution is obtained from the underlying linear polynomial DG solution via a hierarchical WENO reconstruction. The HWENO(P1P2) is designed not only to enhance the accuracy of the underlying DG(P1) method but also to ensure non-linear stability of the RDG method. In this reconstruction scheme, a quadratic polynomial (P2) solution is first reconstructed using a least-squares approach from the underlying linear (P1) discontinuous Galerkin solution. The final quadratic solution is then obtained using a Hermite WENO reconstruction, which is necessary to ensure the linear stability of the RDG method on 3D unstructured grids. The first derivatives of the quadratic polynomial solution are then reconstructed using a WENO reconstruction in order to eliminate spurious oscillations in the vicinity of strong discontinuities, thus ensuring the non-linear stability of the RDG method. The parallelization in the RDG method is based on a message passing interface (MPI) programming paradigm, where the METIS library is used for the partitioning of a mesh into subdomain meshes of approximately the same size.

Both multi-stage explicit Runge-Kutta and simple implicit backward Euler methods are implemented for time advancement in the RDG method. In the implicit method, three approaches: analytical differentiation, divided differencing (DD), and automatic differentiation (AD) are developed and implemented to obtain the resulting flux Jacobian matrices. The automatic differentiation is a set of techniques based on the mechanical application of the chain rule to obtain derivatives of a function given as a computer program. By using an AD tool, the manpower can be significantly reduced for deriving the flux Jacobians, which can be quite
complicated, tedious, and error-prone if done by hand or symbolic arithmetic software, depending on the complexity of the numerical flux scheme. In addition, the workload for code maintenance can also be largely reduced in case the underlying flux scheme is updated. The approximate system of linear equations arising from the Newton linearization is solved by the general minimum residual (GMRES) algorithm with lower-upper symmetric gauss-seidel (LU-SGS) preconditioning. This GMRES+LU-SGS linear solver is the most robust and efficient for implicit time integration of the discretized Navier-Stokes equations when the AD-based flux Jacobians are provided other than the other two approaches.

The developed HWENO(P₁P₂) method is used to compute a variety of well-documented compressible inviscid and viscous flow test cases on 3D hybrid grids, including some standard benchmark test cases such as the Sod shock tube, flow past a circular cylinder, and laminar flow past a flat plate. The computed solutions are compared with either analytical solutions or experimental data, if available to assess the accuracy of the HWENO(P₁P₂) method. Numerical results demonstrate that the HWENO(P₁P₂) method is able to not only enhance the accuracy of the underlying HWENO(P₁) method, but also ensure the linear and non-linear stability at the presence of strong discontinuities. An extensive study of grid convergence analysis on various types of elements: tetrahedron, prism, hexahedron, and hybrid prism/hexahedron, for a number of test cases indicates that the developed HWENO(P₁P₂) method is able to achieve the designed third-order accuracy of spatial convergence for smooth inviscid flows: one order higher than the underlying second-order DG(P₁) method without significant increase in computing costs and storage requirements. The performance of the the developed GMRES+LU-SGS implicit method is compared with the multi-stage Runge-Kutta time stepping scheme for a number of test cases in terms of the timestep and CPU time. Numerical results indicate that the overall performance of the implicit method with AD-based Jacobians is order of magnitude better than the its explicit counterpart. Finally, a set of parallel scaling tests for both explicit and implicit methods is conducted on North Carolina State University’s ARC cluster, demonstrating almost an ideal scalability of the RDG method.

Overall, the developed RDG method shows a great potential to become a viable, attractive, competitive and ultimately superior DG method over the current state-of-the-art second-order finite volume methods.
A Parallel Implicit Reconstructed Discontinuous Galerkin Method
for Compressible Flows on Hybrid Grids

by
Yidong Xia

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Aerospace Engineering

Raleigh, North Carolina

2013

APPROVED BY:

Jack R. Edwards Jr.                                   Hassan A. Hassan

Zhilin Li                                           Hong Luo
Chair of Advisory Committee
DEDICATION

To my family, without whose support this work would not have been possible.
BIOGRAPHY

The author was born in Hangzhou, the capital city of Zhejiang Province, China. He received the Bachelor’s degree of science in Aerospace Engineering at Nanjing University of Aeronautics and Astronautics (NUAA), Nanjing, Jiangsu Province, China, in June 2008. He also attended the graduate school at NUAA in August 2008 and studied under the direction of Dr. Yizhao Wu and Dr. Hongqiang Lu. He received the Master’s degree of science in Fluid Mechanics in May 2010. The author was then recruited by the Department of Mechanical and Aerospace Engineering at North Carolina State University (NCSU), Raleigh, North Carolina, USA, and studied in the doctoral program of Aerospace Engineering since August 2010. His academic advisor is Dr. Hong Luo.
I would like to thank my academic advisor Dr. Hong Luo for his great support and help with my research work within the past three years at North Carolina State University.

I would also like to thank my committee members Dr. Jack R. Edwards, Dr. Hassan A. Hassan from the Department of Mechanical and Aerospace Engineering, and Dr. Zhilin Li from the Department of Mathematics for their time of review on my dissertation and valuable suggestions.

I would like to thank my former group member Dr. Behrouz Karami Halashi in the CFD Laboratory for his suggestions in course studies, assistance and encouragement with my research work. I would also like to thank my former group member Dr. Seth Spiegel for his suggestions and help with my research work.

I would also like to thank Dr. Jiufa Cao for creating a lot of illustrations in this work. His time and effort is sincerely appreciated.

Finally, I would like thank my family for their love, support and blessings throughout my doctoral student life.
# TABLE OF CONTENTS

## LIST OF TABLES

- viii

## LIST OF FIGURES

- x

## Chapter 1 Introduction

- 1
  1.1 Background of Discontinuous Galerkin Methods ............................. 1
  1.2 Challenges of Discontinuous Galerkin Methods ................................ 2
    1.2.1 Treatment of Discontinuities ........................................... 2
    1.2.2 Computing Costs ......................................................... 3
  1.3 Answers to the Challenges ..................................................... 4
    1.3.1 Choice of Basis Functions .............................................. 4
    1.3.2 Reconstruction-Based Spatial Formulation ............................ 5
    1.3.3 Convergence Speedup .................................................... 8
  1.4 Motivation and Goals .......................................................... 11

## Chapter 2 Governing Equations of Fluid Dynamics

- 14
  2.1 Navier-Stokes Equations ....................................................... 14
  2.2 Euler Equations ................................................................. 16
  2.3 Nondimensionalization .......................................................... 16

## Chapter 3 Discontinuous Galerkin Spatial Discretization

- 19
  3.1 Weak Formulation ............................................................... 19
  3.2 Basis Functions ................................................................. 21
  3.3 Elemental Calculations ......................................................... 27
    3.3.1 Shape Functions .......................................................... 27
    3.3.2 Numerical Integration .................................................... 30
  3.4 Boundary Conditions ........................................................... 31
    3.4.1 Characteristic Boundary ................................................ 31
    3.4.2 Slip Wall / Symmetry Boundary ....................................... 32
    3.4.3 No-Slip Adiabatic Wall Boundary ................................... 33
    3.4.4 No-Slip Isothermal Wall Boundary .................................. 34

## Chapter 4 Reconstruction Methods

- 35
  4.1 Least-Squares Reconstruction ................................................ 35
  4.2 WENO Reconstruction at P_2: WENO(P_1P_2) .................................. 40
  4.3 WENO Reconstruction at P_1: HWENO(P_1P_2) ................................ 43

## Chapter 5 Temporal Integration Methods

- 46
  5.1 Explicit Time Integration ..................................................... 46
  5.2 Implicit Time Integration ..................................................... 48
  5.3 Solution of the Linear System of Equations ................................ 49
  5.4 Jacobian Matrix ................................................................. 51
    5.4.1 Differentiation by Hand ................................................ 52
A.4.1 8-Node Trilinear Hexahedron .......................... 152
A.4.2 20-Node Curvilinear Hexahedron ....................... 153
A.5 Prismatic Elements ........................................ 155
  A.5.1 6-Node Prism ........................................... 155
  A.5.2 15-node Prism ......................................... 155
A.6 Pyramidal Elements ........................................ 157
  A.6.1 5-Node Pyramid ........................................ 157
  A.6.2 13-Node Pyramid ...................................... 158
Appendix B Tables of Gaussian Quadrature Formulas .............. 160
  B.1 Triangle ................................................. 160
  B.2 Quadrilateral ........................................... 161
  B.3 Tetrahedron ............................................. 162
  B.4 Hexahedron ............................................. 162
  B.5 Prism .................................................. 163
  B.6 Pyramid ................................................ 163
Appendix C Flux Jacobians ..................................... 164
Appendix D Configuration of a Delta Wing ...................... 165
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Reference variables for nondimensionalization of the governing equations</td>
<td>17</td>
</tr>
<tr>
<td>4.1</td>
<td>Cost analysis for different numerical methods on a tetrahedral grid</td>
<td>42</td>
</tr>
<tr>
<td>4.2</td>
<td>Cost analysis for different numerical methods on a hexahedral grid</td>
<td>42</td>
</tr>
<tr>
<td>5.1</td>
<td>Flowchart of the GMRES algorithm</td>
<td>50</td>
</tr>
<tr>
<td>5.2</td>
<td>A simple source program</td>
<td>62</td>
</tr>
<tr>
<td>5.3</td>
<td>A simple differential source program</td>
<td>63</td>
</tr>
<tr>
<td>5.4</td>
<td>Example: implementation of TAPENADE on RDGFLO</td>
<td>64</td>
</tr>
<tr>
<td>5.5</td>
<td>Pseudo code: contribution of face integrals for the block diagonal matrix $D$, lower matrix $L$ and upper matrix $U$</td>
<td>65</td>
</tr>
<tr>
<td>7.1</td>
<td>Discretization errors and convergence rates of the RDG methods obtained on the four successively globally refined hexahedral grids for an inviscid subsonic flow past a circular cylinder at $M_\infty = 0.38$.</td>
<td>76</td>
</tr>
<tr>
<td>7.2</td>
<td>Discretization errors and convergence rates of the RDG methods obtained on the four successively globally refined prismatic grids for an inviscid subsonic flow past a circular cylinder at $M_\infty = 0.38$.</td>
<td>84</td>
</tr>
<tr>
<td>7.3</td>
<td>Discretization errors and convergence rates of the RDG methods obtained on the four successively globally refined prismatic/hexahedral hybrid grids for an inviscid subsonic flow past a circular cylinder at $M_\infty = 0.38$.</td>
<td>84</td>
</tr>
<tr>
<td>7.4</td>
<td>Discretization errors and convergence rates of the RDG methods obtained on the three successively refined tetrahedral grids for an inviscid subsonic flow past a sphere at $M_\infty = 0.5$.</td>
<td>85</td>
</tr>
<tr>
<td>7.5</td>
<td>Discretization errors and convergence rates of the RDG methods obtained on the four successively globally refined prismatic grids for an inviscid subsonic flow through a channel with a bump at $M_\infty = 0.5$.</td>
<td>90</td>
</tr>
<tr>
<td>7.6</td>
<td>Discretization errors and convergence rates of the RDG methods obtained on the three successively refined tetrahedral grids for an inviscid subsonic flow through a channel with a bump at $M_\infty = 0.5$.</td>
<td>95</td>
</tr>
<tr>
<td>7.7</td>
<td>Boundary layer grid resolution for laminar flow past a flat plate at $Re = 100,000$.</td>
<td>108</td>
</tr>
<tr>
<td>7.8</td>
<td>Execution profile of the implicit time integration for RDG($P_1P_2$) on tetrahedral grid for laminar flow past a flat plate at $Re = 100,000$.</td>
<td>117</td>
</tr>
<tr>
<td>A.1</td>
<td>Abbreviation and description for various types of elements</td>
<td>143</td>
</tr>
<tr>
<td>A.2</td>
<td>Shape functions for the 3-node triangle and their derivatives</td>
<td>146</td>
</tr>
<tr>
<td>A.3</td>
<td>Shape functions for the 6-node triangle and their derivatives</td>
<td>146</td>
</tr>
<tr>
<td>A.4</td>
<td>Shape functions for the 4-node quadrilateral and their derivatives</td>
<td>148</td>
</tr>
<tr>
<td>A.5</td>
<td>Shape functions for the 8-node quadrilateral and their derivatives</td>
<td>148</td>
</tr>
<tr>
<td>A.6</td>
<td>Shape functions for the 4-node tetrahedron and their derivatives</td>
<td>150</td>
</tr>
<tr>
<td>A.7</td>
<td>Shape functions for the 10-node tetrahedron and their derivatives</td>
<td>151</td>
</tr>
<tr>
<td>A.8</td>
<td>Shape functions for the 8-node hexahedron and their derivatives</td>
<td>152</td>
</tr>
</tbody>
</table>
Table A.9  Shape functions for the 6-node prism and their derivatives . . . . . . . . 156
Table A.10 Shape functions for the 15-node prism and their derivatives . . . . . . . . 157
Table A.11 Shape functions for the 5-node pyramid and their derivatives . . . . . . . . 157
Table A.12 Shape functions for the 13-node pyramid . . . . . . . . . . . . . . . . . . . . 159

Table B.1 Integration constants of Gaussian quadrature rules for a triangle. . . . . . . 161
Table B.2 Integration constants of Gaussian quadrature rules for a quadrilateral. . . 161
Table B.3 Integration constants of Gaussian quadrature rules for a tetrahedron. . . 162
Table B.4 Integration constants of Gaussian quadrature rules for a hexahedron. . . 162
Table B.5 Integration constants of Gaussian quadrature rules for a prism. . . . . . . 163
Table B.6 Integration constants of Gaussian quadrature rules for a pyramid. . . . . . 163
LIST OF FIGURES

Figure 3.1 Representation of polynomial solutions using finite element shape functions: (a) \( Q_1/P_1 \); (b) \( Q_2/P_2 \). ........................................... 21
Figure 3.2 Representation of polynomial solutions using a Taylor series expansion. ....................... 22
Figure 3.3 Representation of polynomial solutions using a Taylor series expansion for the median dual control volume. ........................................... 25
Figure 3.4 Transformation of an element in \((x, y)\) physical space into a canonical element reference \((\xi, \eta)\) space ........................................... 27
Figure 4.1 Representation of the four face-neighboring cells surrounding a tetrahedron (a) and the six face-neighboring cells surrounding a hexahedron (b). ......................... 36
Figure 6.1 Graph of domain decomposition by METIS for a hybrid prismatic/hexahedral grid for flow past a circular cylinder, 16 partitions ................................. 67
Figure 6.2 Graph of domain decomposition by METIS for a tetrahedral grid (only surface shown) for flow past a wing/pylon/finned-store configuration, 64 partitions ................................. 68
Figure 7.1 The quasi-1D hexahedral grid for the Sod shock tube problem. ................................. 73
Figure 7.2 Comparison of computed density profiles for the Sod shock tube problem obtained by the RDG\((P_1P_1)\), WENO\((P_1P_2)\), and HWENO\((P_1P_2)\) solutions with the analytical solution. ................................. 73
Figure 7.3 Comparison of computed Mach number profiles for the Sod shock tube problem obtained by the RDG\((P_1P_1)\), WENO\((P_1P_2)\), and HWENO\((P_1P_2)\) solutions with the analytical solution. ................................. 74
Figure 7.4 Comparison of computed pressure profiles for the Sod shock tube problem obtained by the RDG\((P_1P_1)\), WENO\((P_1P_2)\), and HWENO\((P_1P_2)\) solutions with the analytical solution. ................................. 74
Figure 7.5 A sequence of four successively globally refined hexahedral grids for inviscid subsonic flow past a circular cylinder. ................................. 77
Figure 7.6 A sequence of four successively globally refined prismatic grids for inviscid subsonic flow past a circular cylinder. ................................. 77
Figure 7.7 A sequence of four successively globally refined prismatic/hexahedral hybrid grids for inviscid subsonic flow past a circular cylinder. ................................. 77
Figure 7.8 Computed density contours in the flow field by the RDG\((P_1P_1)\) (a–d) and WENO\((P_1P_2)\) (e–h) solutions on the four successively refined hexahedral grids for inviscid subsonic flow past a circular cylinder at \(M_\infty = 0.38\). ......................... 78
Figure 7.9 Computed pressure contours in the flow field by the RDG\((P_1P_1)\) (a–d) and WENO\((P_1P_2)\) (e–h) solutions on the four successively refined hexahedral grids for inviscid subsonic flow past a circular cylinder at \(M_\infty = 0.38\). ......................... 78
Figure 7.10 Computed Mach number contours in the flow field by the RDG\((P_1P_1)\) (a–d) and WENO\((P_1P_2)\) (e–h) solutions on the four successively refined hexahedral grids for inviscid subsonic flow past a circular cylinder at \(M_\infty = 0.38\). ......................... 79
Figure 7.11 $L^2$ errors of numerical solutions versus (a) cell size and (b) number of degree of freedom for inviscid subsonic flow past a circular cylinder by RDG($P_1P_1$) and WENO($P_1P_2$) on hexahedral grids.

Figure 7.12 Computed density contours in the flow field by the RDG($P_1P_1$) (a–d) and WENO($P_1P_2$) (e–h) solutions on the four successively refined prismatic grids for inviscid subsonic flow past a circular cylinder at $M_\infty = 0.38$.

Figure 7.13 Computed pressure contours in the flow field by the RDG($P_1P_1$) (a–d) and WENO($P_1P_2$) (e–h) solutions on the four successively refined prismatic grids for inviscid subsonic flow past a circular cylinder at $M_\infty = 0.38$.

Figure 7.14 Computed Mach number contours in the flow field by the RDG($P_1P_1$) (a–d) and WENO($P_1P_2$) (e–h) solutions on the four successively refined prismatic grids for inviscid subsonic flow past a circular cylinder at $M_\infty = 0.38$.

Figure 7.15 $L^2$ errors of numerical solutions versus (a) cell size and (b) number of degree of freedom for inviscid subsonic flow past a circular cylinder by RDG($P_1P_1$) and WENO($P_1P_2$) on prismatic grids.

Figure 7.16 Computed density contours in the flow field by RDG($P_1P_1$) (a–d) and WENO($P_1P_2$) (e–h) solutions on the four successively globally refined prismatic/hexahedral hybrid grids for inviscid subsonic flow past a circular cylinder at $M_\infty = 0.38$.

Figure 7.17 Computed pressure contours in the flow field by RDG($P_1P_1$) (a–d) and WENO($P_1P_2$) (e–h) solutions on the four successively globally refined prismatic/hexahedral hybrid grids for inviscid subsonic flow past a circular cylinder at $M_\infty = 0.38$.

Figure 7.18 Computed Mach number contours in the flow field by RDG($P_1P_1$) (a–d) and WENO($P_1P_2$) (e–h) solutions on the four successively globally refined prismatic/hexahedral hybrid grids for inviscid subsonic flow past a circular cylinder at $M_\infty = 0.38$.

Figure 7.19 $L^2$ errors of numerical solutions versus (a) cell size and (b) number of degree of freedom for inviscid subsonic flow past a circular cylinder by RDG($P_1P_1$) and WENO($P_1P_2$) on prismatic/hexahedral hybrid grids.

Figure 7.20 $L^2$ errors of numerical solutions versus (a) cell size and (b) number of degree of freedom for inviscid subsonic flow past a sphere at $M_\infty = 0.5$, by RDG($P_1P_1$) and WENO($P_1P_2$) on tetrahedral grids.

Figure 7.21 A sequence of four successively globally refined tetrahedral grids for inviscid subsonic flow past a sphere.

Figure 7.22 Computed density contours in the flow field by the RDG($P_1P_1$) (a–c) and WENO($P_1P_2$) (d–f) on the tetrahedral grids for inviscid subsonic flow past a sphere at $M_\infty = 0.5$.

Figure 7.23 Computed pressure contours in the flow field by the RDG($P_1P_1$) (a–c) and WENO($P_1P_2$) (d–f) on the tetrahedral grids for inviscid subsonic flow past a sphere at $M_\infty = 0.5$. 
Figure 7.24 Computed Mach number contours in the flow field by the RDG(P1P1) (a–c) and WENO(P1P2) (d–f) on the tetrahedral grids for inviscid subsonic flow past a sphere at $M_\infty = 0.5$. ................................................................. 88

Figure 7.25 Convergence history for logarithmic density residual with respect to (a) timestep and (b) CPU time respectively for inviscid subsonic flow past a sphere at $M_\infty = 0.5$ on tetrahedral grids (Level 1, 2 and 3), by WENO(P1P2). 88

Figure 7.26 Convergence history for logarithmic density residual with respect to (a) timestep and (b) CPU time respectively for inviscid subsonic flow past a sphere at $M_\infty = 0.5$ on tetrahedral grids (Level 4), by WENO(P1P2). 89

Figure 7.27 Numerical errors versus CPU time for different reconstructed RDG schemes. 89

Figure 7.28 $L^2$ errors of numerical solutions versus cell size (a) and number of degree of freedom (b) for inviscid subsonic flow through a channel with a smooth bump at $M_\infty = 0.5$ by RDG(P1P1) and WENO(P1P2) on prismatic grids. 91

Figure 7.29 A sequence of four successively globally refined prismatic grids for inviscid subsonic flow through a channel with a smooth bump at $M_\infty = 0.5$. .... 91

Figure 7.30 Computed density contours in the flow field by the RDG(P1P1) (a, c, e, g) and WENO(P1P2) (b, d, f, h) solutions on the four successively refined prismatic grids for inviscid subsonic flow through a channel with a smooth bump at $M_\infty = 0.5$. ................................................................. 92

Figure 7.31 Computed pressure contours in the flow field by the RDG(P1P1) (a, c, e, g) and WENO(P1P2) (b, d, f, h) solutions on the four successively refined prismatic grids for inviscid subsonic flow through a channel with a smooth bump at $M_\infty = 0.5$. ................................................................. 93

Figure 7.32 Computed Mach number contours in the flow field by the RDG(P1P1) (a, c, e, g) and WENO(P1P2) (b, d, f, h) solutions on the four successively refined prismatic grids for inviscid subsonic flow through a channel with a smooth bump at $M_\infty = 0.5$. ................................................................. 94

Figure 7.33 $L^2$ errors of numerical solutions versus cell size (a) and number of degree of freedom (b) for inviscid subsonic flow through a channel with a smooth bump at $M_\infty = 0.5$ by RDG(P1P1) and WENO(P1P2) on tetrahedral grids. 96

Figure 7.34 A sequence of four successively globally refined tetrahedral grids for inviscid subsonic flow through a channel with a smooth bump at $M_\infty = 0.5$. .... 96

Figure 7.35 Computed density, pressure and Mach number contours in the flow field by the RDG(P1P1) (a, b, c) and WENO(P1P2) (e, f, g) solutions on the three successively refined tetrahedral grids for inviscid subsonic flow through a channel with a smooth bump at $M_\infty = 0.5$. ................................................................. 97

Figure 7.36 Convergence history for logarithmic density residual with respect to timestep (a) and CPU time (b) respectively for inviscid subsonic flow through a channel with a smooth bump at $M_\infty = 0.5$ on tetrahedral grids (Level 1, 2 and 3), by WENO(P1P2). ................................................................. 97

Figure 7.37 Convergence history for logarithmic density residual with respect to timestep (a) and CPU time (b) respectively for inviscid subsonic flow through a channel with a smooth bump at $M_\infty = 0.5$ on tetrahedral grids (Level 4), by WENO(P1P2). ................................................................. 98
Figure 7.38 Plot of the tetrahedral grid for inviscid transonic flow over the ONERA M6 wing at $M_\infty = 0.699$, $\alpha = 3.06^\circ$: (a) global view; (b) close-up at the leading edge. ................................. 99

Figure 7.39 Plot of the pressure contours on triangular boundary faces of the tetrahedral grid for inviscid transonic flow over the ONERA M6 wing at $M_\infty = 0.699$, $\alpha = 3.06^\circ$. (a) global view; (b) close-up at the leading edge. ................................. 99

Figure 7.40 Plot of pressure coefficient distributions at six span-wise locations for inviscid transonic flow over the ONERA M6 wing at $M_\infty = 0.699$, $\alpha = 3.06^\circ$. ................................. 100

Figure 7.41 Logarithmic density residual versus timestep (a) and CPU time (b) respectively for inviscid transonic flow over the ONERA M6 wing at $M_\infty = 0.699$, $\alpha = 3.06^\circ$, by WENO(P$_1$P$_2$). ................................. 101

Figure 7.42 Plot of the (a) surface unstructured grid and (b) pressure contours for inviscid transonic flow over the ONERA M6 wing at $M_\infty = 0.84$, $\alpha = 3.06^\circ$: ........................................ 101

Figure 7.43 Plot of pressure coefficient distributions at six span-wise locations for inviscid transonic flow over the ONERA M6 wing at $M_\infty = 0.84$, $\alpha = 3.06^\circ$. ................................. 103

Figure 7.44 Computed pressure contours on the unstructured surface mesh of a tetrahedral grid ($Nelem = 319,134$, $Npoin = 147,289$, $Nbfac = 28,738$) obtained by the HWENO(P$_1$P$_2$) solution for a transonic flow over a wing/pylon/finned-store configuration at $M_\infty = 0.95$, $\alpha = 0^\circ$. ................................. 104

Figure 7.45 Comparison of the computed pressure coefficient distributions with experimental data at two span-wise locations for a transonic flow over a wing/pylon/finned-store configuration at $M_\infty = 0.95$, $\alpha = 0^\circ$. ................................. 105

Figure 7.46 Logarithmic density residual versus timestep (a) and CPU time (b) for inviscid subsonic flow over the wing/pylon/finned-store configuration at $M_\infty = 0.95$, $\alpha = 0^\circ$. ................................. 105

Figure 7.47 Computed Mach number contours and unstructured surface mesh of a tetrahedral grid ($Nelem = 253,577$, $Npoin = 48,851$, $Nbfac = 23,616$) for a transonic flow over a complete B747 aircraft at $M_\infty = 0.85$, $\alpha = 2^\circ$. ................................. 106

Figure 7.48 Logarithmic density residual versus timestep (a) and CPU time (b) for a transonic flow over a complete B747 aircraft at $M_\infty = 0.85$, $\alpha = 2^\circ$, on 128 CPUs. ................................. 107

Figure 7.49 Plot of the hexahedral grids for laminar flow past a flat plate at $Re = 100,000$: (a) SR = 1.20 in $y$-direction; (b) SR = 1.30 in $y$-direction. ................................. 112

Figure 7.50 (a) Logarithmic plot of the computed skin friction $c_f$ coefficient distribution obtained on the two hexahedral grids compared with the analytical solution along the flat plate $x \in [0,1]$; (b) close-up comparison of $c_f$ distribution obtained by RDG(P$_1$P$_1$) and RDG(P$_1$P$_2$) for $x \in [0.05,0.1]$. ................................. 112
Figure 7.51 (a) Plot of $x$-velocity $u$ versus similarity variable $\eta$ obtained on the two hexahedral grids and compared with the analytical solutions at downstream coordinates $x = 0.1$, $x = 0.5$ and $x = 0.9$ (in cells adjacent to outflow boundary) in the boundary layer region; (b) close-up comparison of $u$ profiles obtained by RDG(P$_1$P$_1$) and RDG(P$_1$P$_2$) for $\eta \in [4,8]$.  

Figure 7.52 (a) Plot of scaled $y$-velocity $v^+$ versus similarity variable $\eta$ obtained on the two hexahedral grids and compared with the analytical solutions at downstream coordinates $x = 0.1$, $x = 0.5$ and $x = 0.9$ (in cells adjacent to outflow boundary) in the boundary layer region; (b) close-up comparison of $v^+$ profiles obtained by RDG(P$_1$P$_1$) and RDG(P$_1$P$_2$) for $\eta \in [3,6.5]$. 

Figure 7.53 Plot of the prismatic/hexahedral grids for laminar flow past a flat plate at $Re = 100,000$: (a) global view of the domain; (b) local view of boundary layer region. 

Figure 7.54 (a) Logarithmic plot of the computed skin friction $c_f$ coefficient distribution obtained on the prismatic/hexahedral hybrid grid compared with the analytical solution along the flat plate $x \in [0,1]$; (b) close-up comparison of $c_f$ distribution obtained by RDG(P$_1$P$_1$) and RDG(P$_1$P$_2$) for $x \in [0.02,0.1]$. 

Figure 7.55 Plot of the computed velocity profiles versus similarity variable $\eta$ obtained on the prismatic/hexahedral hybrid grid and compared with the analytical solutions at downstream coordinates $x = 0.1$, $x = 0.5$ and $x = 0.9$ (in cells adjacent to outflow boundary) in the boundary layer region: (a) $x$-velocity $u$ versus $\eta$; (b) scaled $y$-velocity $v^+$ versus $\eta$. 

Figure 7.56 Plot of the tetrahedral grids for laminar flow past a flat plate at $Re = 100,000$: (a) global view of the domain; (b) local view of boundary layer region. 

Figure 7.57 (a) Logarithmic plot of the computed skin friction $c_f$ coefficient distribution obtained on the tetrahedral grid compared with the analytical solution along the flat plate $x \in [0,1]$; (b) close-up comparison of $c_f$ distribution obtained by RDG(P$_1$P$_1$) and RDG(P$_1$P$_2$) for $x \in [0.5,0.6]$. 

Figure 7.58 Plot of the computed velocity profiles versus similarity variable $\eta$ obtained on the tetrahedral grid and compared with the analytical solutions at downstream coordinates $x = 0.1$, $x = 0.5$ and $x = 0.9$ (in cells adjacent to outflow boundary) in the boundary layer region: (a) $x$-velocity $u$ versus $\eta$; (b) scaled $y$-velocity $v^+$ versus $\eta$. 

Figure 7.59 Logarithmic density residual versus timestep (a) and CPU time (b) for laminar flow past a flat plate at $Re = 100,000$. 

Figure 7.60 (a) Plot of the surface meshes of the tetrahedral grid for laminar flow past a sphere at $M_\infty = 0.5$ and $Re = 118$; (b) plot of the computed velocity streamtraces on the symmetry plane. 

Figure 7.61 Plot of the computed contours obtained by RDG(P$_1$P$_2$) on the surface meshes in the flow field at $M_\infty = 0.5$ and $Re = 118$: (a) pressure contours; (b) Mach number contours.
Figure 7.62 Logarithmic density residual versus timestep (a) and CPU time (b) for laminar flow past a sphere at \( M_\infty = 0.5 \) and \( Re = 118 \), by RDG(P\(_1\)P\(_1\)).

Figure 7.63 Logarithmic density residual versus timestep (a) and CPU time (b) for laminar flow past a sphere at \( M_\infty = 0.5 \) and \( Re = 118 \), by RDG(P\(_1\)P\(_2\)).

Figure 7.64 Logarithmic density residual versus timestep (a) and CPU time (b) respectively for subsonic flow over a delta wing at \( M_\infty = 0.3 \), \( \alpha = 12.5^\circ \) and \( Re = 4000 \), by RDG(P\(_1\)P\(_2\)).

Figure 7.65 The surface meshes of a tetrahedral grid for subsonic flow over a delta wing at \( M_\infty = 0.3 \), \( \alpha = 12.5^\circ \) and \( Re = 4000 \), by RDG(P\(_1\)P\(_2\)).

Figure 7.66 Computed Mach number contours on the sliced planes of \( x = 0.5 \), \( x = 0.75 \), \( x = 1.0 \) and \( x = 1.2 \) and streamtraces in the flow field for subsonic flow over a delta wing at \( M_\infty = 0.3 \), \( \alpha = 12.5^\circ \) and \( Re = 4000 \), by RDG(P\(_1\)P\(_2\)).

Figure 7.67 A tetrahedral grid for subsonic flow over a sharp-edged slender delta wing at \( M_\infty = 0.3 \), \( \alpha = 20.5^\circ \) and \( Re = 0.95 \times 10^6 \): (a) triangular surface meshes; (b) extracted meshes on sliced plane of \( x/L = 0.9 \).

Figure 7.68 Logarithmic density residual versus (a) timestep and (b) CPU time respectively for subsonic flow over a delta wing at \( M_\infty = 0.3 \), \( \alpha = 20.5^\circ \) and \( Re = 0.95 \times 10^6 \), by RDG(P\(_1\)P\(_2\)).

Figure 7.69 Surface pressure coefficient distributions at four typical downstream intersections for subsonic flow over a delta wing at \( M_\infty = 0.3 \), \( \alpha = 20.5^\circ \) and \( Re = 0.95 \times 10^6 \), by RDG(P\(_1\)P\(_2\)).

Figure 7.70 Parallel speedup and efficiency on a single compute node (up to 16 CPUs): (a) explicit time integration; (b) implicit time integration.

Figure 7.71 Parallel speedup and efficiency on multiple compute nodes (up to 8 nodes with 16 CPUs per node): (a) explicit time integration; (b) implicit time integration.

Figure A.1 Representation of the 3-node triangular element.
Figure A.2 Representation of barycentric coordinates for a triangle in physical space.
Figure A.3 Representation of barycentric coordinates for a reference triangle.
Figure A.4 Representation of the 6-node triangular element.
Figure A.5 Representation of the 4-node quadrilateral element.
Figure A.6 Representation of the 8-node quadrilateral element.
Figure A.7 Representation of the 4-node tetrahedral element.
Figure A.8 Representation of barycentric coordinates for a tetrahedron in physical space.
Figure A.9 Representation of the 10-node tetrahedral element.
Figure A.10 Representation of the 8-node hexahedral element.
Figure A.11 Representation of the 20-node hexahedral element.
Figure A.12 Representation of the 6-node prismatic element.
Figure A.13 Representation of the 15-node prismatic element.
Figure A.14 Representation of the 5-node pyramidal element.
Figure A.15 Representation of the 13-node pyramidal element.
Figure D.1  Configuration of a sharp-edged slender delta wing . . . . . . . . . . . . . . 165
Chapter 1

Introduction

1.1 Background of Discontinuous Galerkin Methods

The discontinuous Galerkin (DG) methods in mathematics form a class of numerical methods for solving differential equations, and have recently become popular for the solution of systems of conservation laws. In particular, they have received considerable interest for problems with a dominant first-order part, e.g. in fluid mechanics, electrodynamics, and plasma physics. The methods were first proposed and analyzed in the early 1970s as a technique to numerically solve partial differential equations. In 1973, Reed and Hill [110] introduced a DG method to solve the hyperbolic neutron transport equation. In 1974, Le Saint and Raviart [74] analyzed the DG methods for linear hyperbolic problems, first derived the priori error estimates and proved rates of convergence. Nowadays, they are widely used in computational fluid dynamics (CFD), computational acoustics, and computational electromagnetics. The discontinuous Galerkin methods combine two advantageous features commonly associated to the finite element (FE) methods and finite volume (FV) methods. As in classical finite element methods, accuracy is obtained by means of high-order polynomial approximation within an element rather than by wide stencils as in the case of finite volume methods. The physics of wave propagation is, however, accounted for by solving the Riemann problems [125] that arise from the discontinuous representation of the solution at element interfaces. In this respect, the methods are therefore similar to finite volume methods. A more comprehensive overview of the discontinuous Galerkin methods is given by Cockburn et. al [32].

The discontinuous Galerkin methods have many features:

- The methods have several useful mathematical properties with respect to conservation, stability and convergence.

- The methods can be easily extended to higher-order (> 2nd) approximation.
• The methods are well suited for complex geometries since they can be applied on unstructured grids. In addition, the methods can also handle non-conforming elements, where the grids are allowed to have hanging nodes.

• The methods are highly parallelizable [88, 72], as they are compact and each element is independent. Since the elements are discontinuous, and the inter-element communications are minimal, domain decomposition can be effectively employed. The compactness also allows for structured and simplified coding for the methods.

• The methods can easily handle adaptive strategies, since refining or coarsening a grid can be achieved without considering the continuity restriction commonly associated with the conforming elements. The methods allow easy implementation of $hp$-refinement [23, 100, 101, 56, 135], for example, the order of accuracy, or shape, can vary from element to element.

• The methods have the ability to compute low Mach number flow problems [81, 13] without recourse to the time-preconditioning techniques normally required for the finite volume methods.

In contrast to the enormous advances in the theoretical and numerical analysis of the DG methods [31, 33, 15, 14, 19, 34, 102, 16, 68, 107, 105, 106, 91, 131, 130, 71, 7, 9, 10, 35, 56, 57, 57, 101, 67, 104], the development of a viable, attractive, competitive, and ultimately superior DG method over the more mature and well-established second order finite volume methods is relatively an untouched area. This is mainly due to the fact that the DG methods have a number of weaknesses that have yet to be addressed, before they can be robustly used for flow problems of practical interest in complex configuration environment. In particular, how to effectively control spurious oscillations in the presence of strong discontinuities, how to reduce the computing costs, and a lack of efficient time integration scheme for both time-accurate and steady-state solutions remain three most challenging and unresolved issues for the DG methods. Indeed, compared with finite volume methods, the DG methods require the solution of systems of equations with more unknowns (number of degree of freedom) for the same grids. Consequently, the DG methods have been recognized as expensive in terms of both computational cost and storage requirement.

1.2 Challenges of Discontinuous Galerkin Methods

1.2.1 Treatment of Discontinuities

Like any other high-order schemes (> 1st order), the discontinuous Galerkin methods will suffer from non-physical oscillations in the vicinity of discontinuities that exist in problems
governed by hyperbolic conservation laws. Two common approaches to addressing this issue are: 1) a discontinuity capturing technique and 2) an appropriate slope limiter. The former adds explicitly consistent artificial viscosity terms to the discontinuous Galerkin discretization. The main disadvantage of this approach is that it usually requires some user-specified parameters, which can be both grid and problem dependent. The classical techniques of slope limiting are not directly applicable for high-order DG methods because of the presence of volume integral terms in the formulation. Therefore, the slope limiter is not integrated in the computation of the residual, but effectively acts as a post-processing filter. Many slope limiters used in the finite volume methods can then be used or modified to meet the needs of the DG methods. Unfortunately, the use of the limiters will reduce the order of accuracy to first order in the presence of discontinuities. Furthermore, the active limiters in the smooth extrema will pollute the solution in the flow field and ultimately destroy the higher-order accuracy of the DG methods. Indeed, it is not an exaggeration to state that the design of efficient, effective, and robust limiters is one of the bottlenecks in the development of the DG methods for solving the conservation laws. Most efforts in the development of the DG methods have primarily been focused on the exploration of their advantages such as higher-order spatial discretization, posteriori error estimation, adaptive algorithms, and parallelization.

1.2.2 Computing Costs

The temporal discretization methods have lagged far behind. Usually, explicit temporal discretizations such as multi-stage TVD (total variation diminishing) Runge-Kutta schemes [31, 15, 14, 34, 33] are used to advance the solution in time. In general, explicit schemes and their boundary conditions are easy to implement, vectorize and parallelize, and require only limited memory storage. However, for large-scale simulations and especially for high-order solutions, the rate of convergence slows down dramatically, resulting in inefficient solution techniques to steady-state solutions. To speed up the convergence, a multigrid strategy [58, 82, 134, 85, 86] or an implicit method is required. In general, implicit methods require the solution of a linear system of equations arising from the linearization of a fully implicit scheme at each timestep or iteration. For the past fifteen years, various authors have contributed remarkable efforts to the development of efficient implicit solution strategies for the DG methods [17, 16, 109, 19, 49, 68, 102, 144, 40]. All these implicit DG methods achieved a significant speedup of convergence in contrast to their explicit counterpart. Unfortunately, the drawback is that they require a considerable amount of memory to store the Jacobian matrix, which may be prohibitive for large-scale problems and high-order solutions. Even in the implementation of the so-called matrix-free implicit methods [79, 93], where only a block diagonal matrix is required to store, the memory requirements can still be extremely demanding. The block diagonal matrix requires a storage of
\((N_{\text{degr}} \times N_{\text{Netot}}) \times (N_{\text{degr}} \times N_{\text{Netot}}) \times N_{\text{elem}}\), where \(N_{\text{degr}}\) is the number of degree of freedom for the polynomial (3 for \(P_1\), 6 for \(P_2\), and 10 for \(P_3\) for triangular element in 2D; 4 for \(P_1\), 10 for \(P_2\), and 20 for \(P_3\) for tetrahedral element in 3D), \(N_{\text{Netot}}\) is the number of components in the solution vector (4 for 2D, and 5 for the three-dimensional Navier-Stokes equations), and \(N_{\text{elem}}\) is the number of elements for the grid. For example, for a 4th-order (cubic polynomial finite element approximation \(P_3\)) DG method in 3D, the storage of this block diagonal matrix alone requires 10,000 words per element. Indeed, it is our belief that a lack of efficient solvers is one of the reasons that the application of the DG methods for engineering-type problems does not exist.

1.3 Answers to the Challenges

1.3.1 Choice of Basis Functions

In the traditional DG methods, either the standard Lagrange or hierarchical node-based finite element basis functions are used to represent numerical polynomial solutions in each element. As a result, the unknowns to be solved are the variables at the nodes and the polynomial solutions are dependent on the shape of elements. For example, for a linear polynomial approximation in 2D, a linear polynomial approximation is used for triangular elements and the unknowns to be solved are the variables at the three vertices and a bi-linear polynomial approximation is used for quadrilateral elements and the unknowns to be solved are the variables at the four vertices. In the present work, the numerical polynomial solutions are represented using a Taylor series expansion at the centroid of the cell, which can be further expressed as a combination of cell-averaged variables and their derivatives at the centroid of the cell. The unknowns to be solved in this formulation are cell-averaged values and their derivatives at the center of the cells, regardless of the element shapes. As a result, this formulation is able to provide a unified framework, where both the cell-centered and vertex-centered finite volume schemes can be viewed as special cases of this discontinuous Galerkin method by choosing reconstruction schemes to compute the derivatives, offer the insight why the DG methods are a better approach than the finite volume methods based on either TVD/MUSCL reconstruction or essentially non-oscillatory (ENO) and weighted ENO (WENO) reconstruction, and possess a number of distinct, desirable, and attractive features and advantages, which can be effectively used to address the shortcomings of the DG methods mentioned above:

- First, the same numerical polynomial solutions are used for any shapes of elements, which can be triangle, quadrilateral, and polygon in 2D, and tetrahedron, pyramid, prism, and hexahedron in 3D. By using this formulation, the DG method can be easily implemented on arbitrary grids. The numerical method based on this formulation has the ability to
compute the 1D, 2D, and 3D problems using the very same code, which greatly alleviates the need and pain for code maintenance and upgrade.

- Secondly, cell-averaged variables and their derivatives are handily available in this formulation. This makes implementation of WENO reconstruction straightforward and efficient, which is required to eliminate the non-physical oscillations in the vicinity of discontinuities.

- Thirdly, the basis functions are hierarchic. This greatly facilitates the implementation of p-multigrid and p-refinement methods.

- Lastly, cell-averaged variable equations are decoupled from their derivative equations in this formulation. This makes the development of a fast, low-storage implicit method possible.

### 1.3.2 Reconstruction-Based Spatial Formulation

In order to reduce the high computing costs associated with the DG methods, Dumbser et al [45, 43, 46] have introduced a new family of reconstructed DG methods, termed $P_nP_m$ schemes and referred to as RDG($P_nP_m$) in this work, where $P_m$ indicates that a piecewise polynomial of degree of $n$ is used to compute the fluxes. The RDG($P_nP_m$) schemes are designed to enhance the accuracy of the discontinuous Galerkin method by increasing the order of the underlying polynomial solution. The beauty of the RDG($P_nP_m$) schemes is that they provide a unified formulation for both the finite volume and DG methods, and contain both the classical finite volume and standard DG methods as two special cases of the RDG($P_nP_m$) schemes, and thus allow for a direct efficiency comparison. When $n = 0$, i.e., a piecewise constant polynomial is used to represent a numerical solution, the RDG($P_0P_m$) scheme is nothing but the classical high-order finite volume scheme, where a polynomial solution of degree $m$ ($m \geq 1$) is reconstructed from a piecewise constant solution. When $m = n$, the reconstruction reduces to the identity operator, and the RDG($P_nP_n$) scheme yields a standard DG method.

Obviously, the construction of an accurate and efficient reconstruction operator is crucial to the success of the RDG($P_nP_m$) schemes. In Dumbser’s work, a higher-order polynomial solution is reconstructed using an $L^2$ projection, requiring it indistinguishable from the underlying DG solutions in the contributing cells in the weak sense. The resulting over-determined system is then solved by using a least-squares method that guarantees exact conservation, not only of the cell averages but also of all higher-order moments in the reconstructed cell itself, such as slopes and curvatures. However, this conservative least-squares reconstruction approach is computationally expensive, as the $L^2$ projection, i.e., the operation of integration, is required to obtain the resulting over-determined system. Furthermore, the reconstruction might be problematic for
a boundary cell, where the number of the adjacent face-neighboring cells might not be enough to provide the necessary information to recover a polynomial solution of a desired order. Fortunately, the projection-based reconstruction is not the only way to obtain a polynomial solution of higher order from the underlying discontinuous Galerkin solutions. In the reconstructed DG method using a Taylor basis [89, 90, 88, 94, 96] developed by Luo et al. for the solution of the compressible Euler/Navier-Stokes equations on arbitrary grids, a higher-order polynomial solution is reconstructed by using a strong interpolation, requiring point values and derivatives to be interpolated on the adjacent face-neighboring cells. The resulting over-determined linear system of equations is then solved in the least-squares sense. This reconstruction scheme only involves the von Neumann neighborhood, and thus is compact, simple, robust, and flexible. Like the projection-based reconstruction, the strong reconstruction scheme guarantees exact conservation, not only of the cell averages but also of their slopes due to a judicious choice of the Taylor basis. More recently, Zhang et al. [145, 146] presented a class of hybrid DG/FV methods for the conservation laws, where the second derivatives in a cell are obtained from the first derivatives in the cell itself and its neighboring cells using a Green-Gauss reconstruction widely used in the finite volume methods. This also provides a fast, simple, and robust way to obtain higher-order polynomial solutions. More recently, Luo et al. [95, 97] have conducted a comparative study for these three reconstructed discontinuous Galerkin methods RDG(P<sub>1</sub>P<sub>2</sub>) for solving the 2D Euler equations on arbitrary grids. It is found that all the three reconstructed discontinuous Galerkin methods can deliver the desired third-order accuracy and significantly improve the accuracy of the underlying second-order DG method, although the least-squares reconstruction method provides the best performance in terms of both accuracy and robustness.

However, the attempt to directly extend the least-squares RDG method to solve the 3D Euler equations on tetrahedral grids is not successful. Like the second-order cell-centered finite volume methods, i.e., RDG(P<sub>0</sub>P<sub>1</sub>), the resultant RDG(P<sub>1</sub>P<sub>2</sub>) method is numerically unstable. Although the RDG(P<sub>0</sub>P<sub>1</sub>) methods are in general stable in 2D and on Cartesian or structured grids in 3D, they suffer from the so-called linear instability on unstructured tetrahedral grids, when the reconstruction stencils only involve von Neumann neighborhood, i.e., adjacent face-neighboring cells [52]. Unfortunately, the least-squares RDG(P<sub>1</sub>P<sub>2</sub>) method exhibits the same linear instability, which can be overcome by using extended reconstruction stencils. However, this is achieved at the expense of sacrificing the compactness of the underlying DG methods. Furthermore, these linear reconstruction-based DG methods will suffer from the non-physical oscillations in the vicinity of strong discontinuities for the compressible Euler equations. Alternatively, the ENO, WENO and Hermite-WENO schemes can be used to reconstruct a higher-order polynomial solution, which can not only enhance the order of accuracy of the underlying DG method but also achieve both the linear and non-linear stability.

The ENO schemes were initially introduced by Harten at el. [55], in which oscillations up
to the order of the truncation error are allowed to overcome the drawbacks and limitations of
limiter-based schemes. ENO schemes avoid interpolation across high-gradient regions through
biasing of the reconstruction. This biasing is achieved by reconstructing the solution on several
stencils at each location and selecting the reconstruction which is in some sense the smoothest.
This allows ENO schemes to retain higher-order accuracy near high-gradient regions. However,
the selection process can lead to convergence problems and loss of accuracy in regions with
smooth solution variations. To counter these problems, the so-called weighted ENO (WENO)
scheme introduced by Liu at al [76] is designed to present better convergence rate for steady-
state problems, better smoothing for the flux vectors, and better accuracy using the same
stencils than the ENO scheme. The WENO scheme uses a suitably weighted combination of all
reconstructions rather than just the one that is judged to be the smoothest. The weighting is
designed to favor the smooth reconstruction in the sense that its weight is small if the oscillation
of a reconstructed polynomial is high, and its weight is order of one if a reconstructed polynomial
has low oscillation. Qiu and Shu initiated the use of the WENO scheme as limiters for the
DG method [107] for solving the 1D and 2D Euler equations on structured grids. Later on,
they constructed a class of WENO schemes based on Hermite polynomials, termed as Hermite
WENO schemes and applied these schemes as limiters for the DG methods [105, 106]. The main
difference between Hermite-WENO and WENO schemes is that the former has a more compact
stencil than the latter for the same order of accuracy.

Unfortunately, implementation of both ENO and WENO schemes is fairly complicated on ar-
bitrary grids, especially in 3D. In fact, there are very few results obtained using ENO/WENO on
unstructured grids in 3D especially for higher-order reconstruction. Harten and Charkravarthy
[54], Abgrall [4], and Sonar [120] presented the first implementation of ENO schemes on un-
structured triangular grids. Implementation of WENO schemes on unstructured triangular grids
was also presented by Friedrich [50] and Hu and Shu [60].

The Hermite-WENO schemes has been developed on 1D and 2D structured grids for the DG
methods by Balsara et al [11], where the Hermite-WENO reconstruction scheme is relatively
simple and straightforward. In the work presented, a Taylor basis [94] reconstruction-based
DG method, RDG(P_1P_2), based on a Hierarchical WENO reconstruction scheme, termed as
HWENO(P_1P_2) [96], is developed for the solution of the compressible Euler and Navier-Stokes
equations on single-type and hybrid unstructured grids in 3D [138]. This reconstructed DG
method is designed not only to reduce the high computing costs of the DG methods, but also to
avoid spurious oscillations in the vicinity of strong discontinuities, thus effectively overcoming
two of the three most severe shortcomings of the DG methods and ensuring the linear and
non-linear stability of the reconstructed DG method. In this RDG(P_1P_2) method, a quadratic
solution is obtained through the HWENO(P_1P_2) reconstruction in three steps:
• Least-squares reconstruction: all the second derivatives ($P^2$) in each cell are first reconstructed using the solution variables ($P^0$) and their first derivatives ($P^1$) from adjacent face-neighboring cells via a strong interpolation.

• WENO reconstruction at $P^2$ – WENO($P^1P^2$): the final second derivatives in each cell are then obtained using a WENO strategy based on the reconstructed second derivatives in the cell itself and its adjacent face-neighboring cells, in order to ensure the linear stability of the RDG method.

• WENO reconstruction at $P^1$ – HWENO($P^1P^2$): the gradients of the quadratic polynomial solutions are then modified using a WENO reconstruction in order to eliminate non-physical oscillations in the vicinity of strong discontinuities, thus ensuring the non-linear stability of the RDG method.

This reconstruction scheme, by taking advantage of handily available and yet valuable information, namely the gradients in the context of the DG methods, only involves von Neumann neighborhood and thus is compact, simple, robust and flexible. As the underlying DG($P^1$) method is second order, and the basis functions are at most linear functions, fewer integration points are then required for both volume and face integrals, and the number of unknowns (the number of degrees of freedom) remains the same as for the DG($P^1$) method. Consequently, this RDG($P^1P^2$) method is more efficient than its third-order DG($P^2$) counterpart.

1.3.3 Convergence Speedup

In general, implicit time integration methods require the solution of a linear system of equations arising from the linearization of a fully implicit scheme at each timestep. Dating back to the 1980s through 90s, significant efforts have been made to develop efficient iterative solution algorithms for the finite volume methods and finite element methods. The most widely used methods to solve a linear system on unstructured grids are iterative solution methods and approximate factorization methods. These methods range from Gauss-Seidel algorithm to Krylov subspace methods that use a variety of preconditioners (see, e.g., [121, 20, 132, 70, 137, 87, 12]) The most successful and effective iterative methods are the Krylov subspace methods [113] such as the GMRES (general minimal residual) method [114] and BiCGSTAB (biconjugate gradient stabilized) method [118] with an ILU (incomplete lower-upper) factorization preconditioner. The drawback is that they require a considerable amount of memory to store the Jacobian matrix, which may be prohibitive for large-scale problems. Later on, the LU-SGS (lower-upper symmetric Gauss-Seidel) method developed first by Jameson and Yoon [65] on structured grids has been successfully generalized and extended to unstructured grids by several authors [119, 98, 117]. The most attractive feature of this approximate factorization method is that the evaluation
and storage of the Jacobian matrix inherent in the original formulation of the LU-SGS method can be completely eliminated by making some approximations to the implicit operator. Luo et al. [79] developed a fast, matrix-free implicit method for the Euler equations on unstructured grids, in which the system of linear equations is solved iteratively by a GMRES algorithm with an LU-SGS preconditioner, namely GMRES+LU-SGS. The advantage of the LU-SGS preconditioner is that it uses the flux Jacobian matrix as a preconditioning matrix, as compared with the ILU preconditioner and consequently, does not require any additional memory storage and computing effort to store and compute the preconditioning matrix. Furthermore, the storage of the Jacobian matrix can be completely eliminated by approximating the Jacobians with numerical fluxes, which leads to a fast, low-storage implicit algorithm. The GMRES+LU-SGS implicit method was later applied to the computation of compressible turbulent flows on unstructured grids [93].

In contrast to the finite volume methods, the development of implicit solution techniques for the high-order discontinuous Galerkin methods lagged much far behind. In 1999, a GMRES discontinuous Galerkin method for the compressible Navier-Stokes equations on 2D unstructured grids was developed by Bassi and Rebay [17]. It was the first implicit DG method that adopted a Krylov subspace algorithm. Also in 1999, an implicit high-order DG method for the compressible Favre-Reynolds averaged Navier-Stokes equations was introduced by Bassi and Rebay [16], which has many impressive features: 1) the method is characterized by a highly compact discretization support even for higher order approximations and this feature can be exploited in the development of implicit integration schemes; 2) turbulence effects are accounted for by means of the low-Reynolds $k$-$\omega$ turbulence model; and 3) a non-standard implementation of the model, whereby the logarithm of $\omega$ rather than $\omega$ itself is used as unknown, was very useful to enhance the stability of the method especially for the higher (third and fourth) order approximations. The content of this work was later extended and presented in a more comprehensive context [19], in which detailed description of the DG discretization of the viscous part of the equations and of several implementation details of the $k$-$\omega$ turbulence model are given. In 2001, an implicit discontinuous spectral Galerkin method for the solution of the compressible Euler equations on 2D structured grids was developed by Rasetarina and Hussaini [109]. In their method, a matrix-free LU-SGS preconditioned Newton–Krylov algorithm is used to solve the implicit system, where the memory requirement for storing the off-diagonal blocks of the Jacobian matrices is largely reduced by the matri-free approach.

The $p$-multigrid techniques, as an alternative strategy of efficient time integration for the discontinuous Galerkin methods, have also been developed by various authors in the past decade. In 2002, a $p$-multigrid discontinuous Galerkin solution strategy for the solution of the Euler equations on 2D unstructured grids was presented by Bassi and Rebay [18], where a series of progressively lower order implicit DG approximations are used as solution smoothers. In 2005,
a $p$-multigrid solution algorithm for a high-order discontinuous Galerkin discretization of the compressible Navier-Stokes equations on 2D unstructured grids was introduced by Fidkowski et al. [49]. The algorithm employs an element line Jacobi smoother in which lines of elements are formed using coupling based on a $p = 0$ discretization of the scalar convection-diffusion equation. In 2006, a $p$-multigrid discontinuous Galerkin method for the compressible Euler equations on both 2D and 3D unstructured grids was presented by Luo et. al [81, 82, 85]. A distinct feature of Luo’s $p$-multigrid method is the application of an explicit smoother on the higher level approximation ($p > 0$) and an implicit smoother on the lowest level approximation ($p = 0$), resulting in a fast as well as low storage method.

More recently, the efforts for the development of the discontinuous Galerkin temporal discretizations gradually shifted to fully implicit schemes, especially in 3D. In 2006, a space–time discontinuous Galerkin method for the compressible Navier-Stokes equations on 3D Cartesian grids is developed by Klaij et al. [68]. A remarkable feature of this method is that it allows local grid adaptation as well as moving and deforming boundaries. In 2008, an implicit discontinuous Galerkin method for RANS simulation utilizing pointwise relaxation algorithm on 3D hybrid grids was introduced by Yasue at al. [144]. In this method, efforts were also made to reduce the computing time by utilizing a $p$-multigrid scheme along with the full-order implicit method, and also by solving a simplified matrix instead of a fully loaded dense matrix in the implicit matrix inversion. In 2011, a Newton–Krylov matrix-free discontinuous Galerkin solver for the NS/RANS equations on 3D structured grids is presented by Crivellini and Bassi [40], and compared with its matrix based counterpart. However, a common drawback associated with these implicit high-order discontinuous Galerkin methods is that they all require a considerable amount of static memory to store the Jacobian matrix, which may be prohibitive for large-scale problems (millions of elements or even more). Even with the so-called “matrix-free” implicit methods, where only a block diagonal matrix is required to store, the memory requirements can still be far from an affordable range for simulations of practical importance.

In order to take up only a minimal static memory while retaining the superiority of accuracy to the more established second-order implicit finite volume methods, an efficient implicit method for the RDG($P_1P_2$) scheme was initially developed for the compressible Euler and Navier-Stokes equations on unstructured tetrahedral grids in 3D [139, 142, 140], and then extended to hybrid grids [141]. A notable feature of this implicit RDG($P_1P_2$) method is that the Jacobian matrix $J = \partial \mathbf{R}/\partial \mathbf{U}$, i.e., linearization of the right-hand-side (RHS) vector $\mathbf{R}$ in terms of the conserved state vector $\mathbf{U}$, is evaluated based on the underlying second-order DG($P_1$) operator instead of the reconstructed third-order RDG($P_1P_2$) operator. In fact, it is not hard to see that the full linearization of the RDG($P_1P_2$) method is practically not accessible in a closed form, especially when the highly non-linear HWENO($P_1P_2$) reconstruction is included. Furthermore, the memory requirement and time of computing the exact Jacobian matrix would be highly
demanding. Nevertheless, a well-approximated Jacobian matrix is preferred in the implicit DG methods because the efficiency of a preconditioned linear solver depends closely on the quality of the Jacobian matrix, which will become more sensitive as the order of the methods increase.

In the early stage of study, a fast matrix-free approach to formulating the Jacobian matrix by spectral approximation was successfully extended from an implicit finite volume method for the Euler and Navier-Stokes equations on unstructured grids [79, 93], to an implicit DG method for the Euler equations on 2D structured grids [109]. Unfortunately, the extension of such approach to the Euler and Navier-Stokes equations on unstructured tetrahedral grids was not successful for higher-order DG methods. Numerical tests on inviscid flow problems indicated that a severe CFL constraint was observed and the convergence rate slowed down significantly when only small CFLs can be used [139]. Indeed, convergence would be even hard to achieve with this approach applied to the viscous flow problems due to the very ill condition number of the roughly approximated Jacobian matrix, indicating the necessity of using a more accurately approximated Jacobian matrix for the implicit RDG(P1P2) method. Several approaches are proposed and implemented in this work including 1) analytical differentiation, 2) divided differencing (DD), and 3) automatic differentiation (AD). Automatic differentiation is a set of techniques based on the mechanical application of the chain rule to obtain derivatives of a function given as a computer program. By using an AD tool [2, 26, 25], the labor of a programmer can be significantly reduced for assembly of the Jacobian matrix, which can be quite complicated, tedious and error-prone if done by hand or symbolic arithmetic software, depending on the complexity of the numerical flux scheme discretized the discontinuous Galerkin context. In addition, the workload for code maintenance can also be largely reduced in case the underlying flux schemes are updated. In this work, the approximate system of linear equations arising from the Newton’s linearization is solved iteratively by a GMRES+LU-SGS algorithm. The developed GMRES+LU-SGS linear solver is most robust and efficient in terms of convergence rate when the AD-based Jacobians are provided other than the other two approaches.

1.4 Motivation and Goals

The discontinuous Galerkin (DG) methods have recently become a popular and promising class of numerical discretization methods for the solution of systems of conservation laws to arbitrary order of accuracy. However, in contrast to the enormous advances in the theoretical and numerical analysis of the DG methods, the development of a viable, attractive, competitive and ultimately superior DG method over the more mature and well-established second-order methods is relatively an untouched area.

The objective of the effort presented in this PhD work is to develop a parallel, implicit reconstructed discontinuous Galerkin (RDG) method using Taylor basis for the solution of
the compressible Navier-Stokes equations on 3D hybrid grids. This third-order accurate RDG method is based on a hierarchical weighed essentially non-oscillatory reconstruction scheme, termed as HWENO(P₁P₂) to indicate that a quadratic polynomial solution is obtained from the underlying linear polynomial DG solution via a hierarchical WENO reconstruction. The HWENO(P₁P₂) is designed not only to enhance the accuracy of the underlying DG(P₁) method but also to ensure non-linear stability of the RDG method. In this reconstruction scheme, a quadratic polynomial (P₂) solution is first reconstructed using a least-squares approach from the underlying linear (P₁) discontinuous Galerkin solution. The final quadratic solution is then obtained using a Hermite WENO reconstruction, which is necessary to ensure the linear stability of the RDG method on 3D unstructured grids. The first derivatives of the quadratic polynomial solution are then reconstructed using a WENO reconstruction in order to eliminate spurious oscillations in the vicinity of strong discontinuities, thus ensuring the non-linear stability of the RDG method. The parallelization in the RDG method is based on a message passing interface (MPI) programming paradigm, where the METIS library is used for the partitioning of a mesh into subdomain meshes of approximately the same size.

Both multi-stage explicit Runge-Kutta and simple implicit backward Euler methods are implemented for time advancement in the RDG method. In the implicit method, three approaches: analytical differentiation, divided differencing (DD), and automatic differentiation (AD) are developed and implemented to obtain the resulting flux Jacobian matrices. The automatic differentiation is a set of techniques based on the mechanical application of the chain rule to obtain derivatives of a function given as a computer program. By using an AD tool, the manpower can be significantly reduced for deriving the flux Jacobians, which can be quite complicated, tedious, and error-prone if done by hand or symbolic arithmetic software, depending on the complexity of the numerical flux scheme. In addition, the workload for code maintenance can also be largely reduced in case the underlying flux scheme is updated. The approximate system of linear equations arising from the Newton linearization is solved by the general minimum residual (GMRES) algorithm with lower-upper symmetric gauss-seidel (LU-SGS) preconditioning. This GMRES+LU-SGS linear solver is the most robust and efficient for implicit time integration of the discretized Navier-Stokes equations when the AD-based flux Jacobians are provided other than the other two approaches.

The question whether the discontinuous Galerkin methods will replace the finite volume methods in future CFD applications cannot be answered by now, but this PhD work is dedicated to contributing some facts to assess the practicality of the DG methods for large-scale problems. This dissertation serves as only a first step toward the completion of the RDGFLO framework for simulating flow problems in reality-like complex environment, i.e., multi-phase/multi-species flows through a reactor cooling system of grid bundles.

The outline of the dissertation is organized as follows. In Chapter 2, the governing equations
and nondimensionalization are briefly described. In Chapter 3, the spatial discretization of these equations by the DG method is discussed in more detail. The reconstruction methods based on the second-order DG($P_1$) method are presented in Chapter 4. The temporal integration methods (both explicit and implicit) are discussed in Chapter 5. Chapter 6 delas with the parallel implementation strategy. Numerical results for a variety of compressible inviscid/viscous flow test cases are presented in Chapter 7. Finally the conclusion and future work is discussed in Chapter 8.
Chapter 2

Governing Equations of Fluid Dynamics

In this chapter, the governing equations of the physical flow models used in this work are briefly described: the Navier-Stokes equations (§2.1) and the Euler equations (§2.2). The nondimensionalization of the governing equations are described in the last section.

2.1 Navier-Stokes Equations

The Navier-Stokes equations governing unsteady compressible viscous flows can be expressed as

$$\frac{\partial U}{\partial t} + \frac{\partial F_k(U)}{\partial x^k} = \frac{\partial G_k(U)}{\partial x^k}$$  \hspace{1cm} (2.1)

where the summation convention ($k = 1, 2, 3$) has been used. The conservative variable vector $U$ is defined by

$$U = \begin{pmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \\ \rho e \end{pmatrix}$$  \hspace{1cm} (2.2)

where $\rho$, $p$, and $e$ denote the density, pressure, and specific total energy of the fluid, respectively, and $u$, $v$, and $w$ are the velocity components of the flow in the coordinate direction $x$, $y$ and $z$. The pressure can be computed from the equation of state

$$p = (\gamma - 1)\rho \left( e - \frac{1}{2}(u^2 + v^2 + w^2) \right)$$  \hspace{1cm} (2.3)
which is valid for perfect gas, and the ratio of the specific heats $\gamma$ is assumed to be constant and equal to 1.4. Furthermore, the specific total enthalpy $h$ is defined as

$$h = e + \frac{p}{\rho}$$  \hspace{1cm} (2.4)

In Eq. 2.1, the advective (inviscid) flux tensor $\mathbf{F} = (\mathbf{F}_x, \mathbf{F}_y, \mathbf{F}_z)$ is defined by

$$\mathbf{F}_x = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ \rho w u \\ u(\rho e + p) \end{pmatrix} \quad \mathbf{F}_y = \begin{pmatrix} \rho v \\ \rho vu \\ \rho v^2 + p \\ \rho vw \\ v(\rho e + p) \end{pmatrix} \quad \mathbf{F}_z = \begin{pmatrix} \rho w \\ \rho wu \\ \rho w^2 + p \\ \rho vw \\ w(\rho e + p) \end{pmatrix}$$  \hspace{1cm} (2.5)

and the viscous flux tensor $\mathbf{G}$ is defined by

$$\mathbf{G}_x = \begin{pmatrix} 0 \\ \tau_{xx} \\ \tau_{xy} \\ \tau_{xz} \\ \tau_{xx} \rho u + \tau_{xy} u + \tau_{xz} v + q_x \end{pmatrix}$$

$$\mathbf{G}_y = \begin{pmatrix} 0 \\ \tau_{yx} \\ \tau_{yy} \\ \tau_{yz} \\ \tau_{yx} \rho v + \tau_{yy} u + \tau_{yz} v + q_y \end{pmatrix}$$  \hspace{1cm} (2.6)

$$\mathbf{G}_z = \begin{pmatrix} 0 \\ \tau_{zx} \\ \tau_{zy} \\ \tau_{zz} \\ \tau_{zx} \rho w + \tau_{zy} u + \tau_{zz} v + q_z \end{pmatrix}$$

where the viscous stress tensor $\tau$ is expressed as

$$\begin{pmatrix} \tau_{xx} & \tau_{xy} & \tau_{xz} \\ \tau_{yx} & \tau_{yy} & \tau_{yz} \\ \tau_{zx} & \tau_{zy} & \tau_{zz} \end{pmatrix}$$

The Newtonian fluid with the Stokes hypothesis is valid under the current framework, since
only air is considered. Thus $\tau$ is symmetric and the tensor is a linear function of the velocity gradients

$$\tau_{ij} = \mu \left( \frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) - \frac{2}{3} \mu \frac{\partial u_k}{\partial x_k} \delta_{ij}$$

(2.7)

where $\delta_{ij}$ is the Kronecker delta function, and $\mu$ represents the molecular viscosity coefficient (often referred to as dynamic viscosity coefficient as well), which can be determined through Sutherland’s law

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

(2.8)

where $\mu_0$ denotes the viscosity coefficient at the reference temperature $T_0$, and $S$ is a constant which is assumed the value $S = 110K$. The temperature of the fluid $T$ is determined by

$$T = \frac{P}{\rho R}$$

(2.9)

where $R$ denotes the universal gas constant for perfect gas.

The heat flux vector $q_j$, which is formulated according to Fourier’s law, is given by

$$q_j = -\lambda \frac{\partial T}{\partial x_j}$$

(2.10)

where $\lambda$ is the thermal conductivity coefficient and expressed as

$$\lambda = \frac{\mu c_p}{Pr}$$

(2.11)

where $c_p$ is the specific heat capacity at constant pressure and $Pr$ is the nondimensional laminar Prandtl number, which is taken as 0.7 for air.

2.2 Euler Equations

If the effect of viscosity and thermal conduction are neglected in Eq. 2.1, we arrived at the Euler equations expressed as below, which govern unsteady compressible inviscid flows

$$\frac{\partial U}{\partial t} + \frac{\partial F_k(U)}{\partial x_k} = 0$$

(2.12)

2.3 Nondimensionalization

The governing equations are often put into the nondimensional form. The advantage in doing so is that the characteristic parameters such as Mach number, Reynolds number, and Prandtl number can be varied independently. Also, by nondimensionalizing the governing equations, the
flow variables are “normalized”, so that their values fall between certain prescribed limits such as 0 and 1. Many different nondimensionalizing procedures are possible. In this work, we use the following four reference variables: length, density, velocity and temperature. The choice of each reference variable is summarized in Table 2.1.

Table 2.1: Reference variables for nondimensionalization of the governing equations

<table>
<thead>
<tr>
<th>Variable</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length $L_{ref}$</td>
<td>Problem dependent (cylinder diameter, plate length, etc) $d, l$</td>
</tr>
<tr>
<td>Density $\rho_{ref}$</td>
<td>Freestream density $\rho_\infty$</td>
</tr>
<tr>
<td>Velocity $V_{ref}$</td>
<td>Freestream speed of sound $a_\infty$</td>
</tr>
<tr>
<td>Temperature $T_{ref}$</td>
<td>Freestream temperature $T_\infty$</td>
</tr>
</tbody>
</table>

The nondimensional variables are denoted by an overbar

$$\bar{L} = \frac{L}{L_{ref}}, \quad \bar{\rho} = \frac{\rho}{\rho_\infty}, \quad \bar{u} = \frac{u}{a_\infty}, \quad \bar{v} = \frac{v}{a_\infty}, \quad \bar{w} = \frac{w}{a_\infty}, \quad \bar{T} = \frac{T}{T_\infty},$$

and accordingly, the derived normalized variables are expressed in the following manner

$$\bar{p} = \frac{p}{\rho_\infty a_\infty^2}, \quad \bar{h} = \frac{h}{a_\infty^2}, \quad \bar{c}_p = \frac{c_p}{a_\infty^2/T_\infty}, \quad \bar{\mu} = \frac{\mu}{\rho_\infty a_\infty}$$

It is also trivial to derive the nondimensional equation of state as

$$\bar{p} = \frac{1}{\gamma} \bar{\rho} \bar{T}$$

The freestream Mach number $M_\infty$ is defined as

$$M_\infty = \frac{V_\infty}{a_\infty}$$

The freestream Reynolds number $Re_\infty$ is determined as

$$Re_\infty = \frac{\rho_\infty a_\infty L_{ref}}{\mu_\infty}$$

The Prandtl number is written as

$$Pr = \frac{\mu_\infty c_p}{\lambda}$$
In the normalized governing equations, the nondimensional viscous stress tensor is

\[ \tilde{\tau}_{ij} = \tilde{\mu} \left( \frac{\partial \tilde{u}_i}{\partial x_j} + \frac{\partial \tilde{u}_j}{\partial x_i} - \frac{2}{3} \frac{\partial \tilde{u}_k}{\partial x_k} \delta_{ij} \right) \]

and the nondimensional heat flux \( \tilde{q}_j \) vector is

\[ \tilde{q}_j = -\tilde{\mu} \tilde{c}_p \frac{1}{\Pr} \frac{\partial \tilde{T}}{\partial x_j} \]

The nondimensional molecular viscosity coefficient \( \tilde{\mu} \) is computed with the dimensionless Sutherland’s law

\[ \tilde{\mu} = \frac{M_\infty}{Re_\infty} T_\infty^\frac{3}{2} \left( \frac{1}{T} + \frac{S}{T_\infty} \right) \]

For the installation of a specific flow problem, the nondimensional input parameters include two fixed-value quantities \( \tilde{\rho}_\infty = 1.0 \) and \( a_\infty = 1.0 \), and five user-adjustable quantities: \( M_\infty \), angle of attack \( \alpha \), yaw angle \( \beta \), \( Re_\infty \) and \( Pr \). With these inputs, a uniform flow field is prescribed for a steady-state problem at the initialization stage and the corresponding conservative variables are

\[ \tilde{\rho}_\infty = 1.0 \]
\[ \tilde{\rho}_u_\infty = M_\infty \cos \alpha \cos \beta \]
\[ \tilde{\rho}_v_\infty = M_\infty \cos \alpha \sin \beta \]
\[ \tilde{\rho}_w_\infty = M_\infty \sin \alpha \]
\[ \tilde{\rho}_e_\infty = \frac{1}{\gamma(\gamma - 1)} + \frac{1}{2} \frac{M_\infty^2}{\gamma} \]

The other derived dimensionless variables are

\[ \tilde{p}_\infty = \frac{1}{\gamma} \]
\[ \tilde{\mu}_\infty = \frac{M_\infty}{Re_\infty} \]
\[ \tilde{c}_p = \frac{1}{\gamma - 1} \]
\[ \tilde{\lambda} = \tilde{\mu} \frac{1}{Pr} \frac{1}{\gamma - 1} \]

From the next chapter, all variables and equations that appear in the text are assumed to be in the nondimensional system and therefore the overbar sign will be dropped for simplicity.
Chapter 3

Discontinuous Galerkin Spatial Discretization

In the present work, the governing equations are discretized in space by using the discontinuous Galerkin method. The outline of this chapter is organized in the following. In section 3.1, the discontinuous Galerkin discretization for the Navier-Stokes equations is described in detail. The basis functions preferably chosen for the convenient implementation of arbitrary grids and reconstruction are introduced in section 3.2. The boundary conditions are described in section 3.3. Finally the finite element integration and the quadrature rules for several grid topologies are stated in section 3.4.

3.1 Weak Formulation

The Navier-Stokes equations Eq. 2.1 are discretized using a discontinuous Galerkin finite element formulation. To formulate the discontinuous Galerkin method, we first introduce the following weak formulation, which is obtained by multiplying the above conservation law by a test function $W$, integrating over the domain $\Omega$, and then performing an integration by parts,

$$
\int_{\Omega} \frac{\partial U}{\partial t} W \, d\Omega + \int_{\Gamma} F_k n_k \, d\Gamma - \int_{\Omega} F_k \frac{\partial W}{\partial x_k} \, d\Omega = \int_{\Gamma} G_k n_k \, d\Gamma - \int_{\Omega} G_k \frac{\partial W}{\partial x_k} \, d\Omega \quad (3.1)
$$

where $\Gamma(= \partial \Omega)$ denotes the boundary of $\Omega$, and $n_j$ the unit outward normal vector to the boundary. We assume that the domain $\Omega$ is subdivided into a collection of non-overlapping arbitrary elements $\Omega_e$. Then we introduce the following broken Sobolev space $V^p_h$,

$$
V^p_h = \{ v_h \in [L^2(\Omega)]^m : v_h|_{\Omega_e} \in [V^m_p] \ \forall \Omega_e \in \Omega \} \quad (3.2)
$$
which consists of discontinuous vector-values polynomial functions of degree $p$, and where $m$ is the dimension of the unknown vector and

$$V_p^m = \text{span}\left\{ \prod_{i} x_i^{\alpha_i} : 0 \leq \alpha_i \leq p, 0 \leq i \leq d \right\}$$  \hspace{1cm} (3.3)$$

where $\alpha$ denotes a multi-index and $d$ is the dimension of space. Then, we can obtain the following semi-discrete form by applying weak formulation on each element $\Omega_e$, find $U \in V_h^p$ such as

$$\frac{d}{dt} \int_{\Omega_e} U_h W_h \, d\Omega + \int_{\Gamma_e} F_k(U_h) n_k W_h \, d\Gamma - \int_{\Omega_e} \frac{\partial W_h}{\partial x_k} \partial B_i \, d\Omega = \int_{\Gamma_e} G_k(U_h) n_k W_h \, d\Gamma - \int_{\Omega_e} \frac{\partial W_h}{\partial x_k} \partial B_i \, d\Omega \quad \forall W_h \in V_h^p$$  \hspace{1cm} (3.4)$$

where $U_h$ and $W_h$ represent the finite element approximations to the analytical solution $U$ and the test function $W$ respectively, and they are approximated by a piecewise polynomial function of degrees $p$, which are discontinuous between the cell interfaces. Assume that $B$ is the basis of polynomial function of degrees $p$, this is then equivalent to the following system of $N$ equations,

$$\frac{d}{dt} \int_{\Omega_e} U_h B_i \, d\Omega + \int_{\Gamma_e} F_k(U_h) n_k B_i \, d\Gamma - \int_{\Omega_e} \frac{\partial B_i}{\partial x_k} \partial B_i \, d\Omega = \int_{\Gamma_e} G_k(U_h) n_k B_i \, d\Gamma - \int_{\Omega_e} \frac{\partial B_i}{\partial x_k} \partial B_i \, d\Omega \quad 1 \leq i \leq N$$  \hspace{1cm} (3.5)$$

where $N$ is the dimension of the polynomial space. Since the numerical solution $U_h$ is discontinuous between element interfaces, the interface fluxes are not uniquely defined. The flux function $F_k(U_h)n_k$ appearing in the second terms of Eq. 3.5 is replaced by a numerical Riemann flux function $H_k(U_{Lh}^R, U_{Rh}^L, n_k)$ where $U_{Lh}^R$ and $U_{Rh}^L$ are the conservative state vectors at the left and right side of the element boundary. This scheme is called discontinuous Galerkin method of degree $p$, or in short notation DG(P) method. By simply increasing the degree $p$ of the polynomials, the DG methods of corresponding higher order are obtained.

In our work, the inviscid flux function is approximated using the HLLC approximate Riemann solver [126], which has been successfully used to compute compressible viscous and turbulent flows on both structured grids [22] and unstructured grids [80]. This HLLC scheme is found to have the following properties: (1) exact preservation of isolated contact and shear waves, (2) positivity-preserving of scalar quantity, (3) enforcement of entropy condition. In addition, the implementation of HLLC Riemann solver is easier and the computational cost is lower compared with some other available Riemann solvers.

Among the many possible schemes [14, 19, 34, 23, 102, 51, 91, 131, 130, 108] that are developed for the discretization of the viscous fluxes for discontinuous Galerkin methods, we have
chosen to implement the famous second Bassi-Rebay scheme (BR2) [19] for the discretization of the viscous fluxes, as the BR2 scheme is the only one proposed in the literature to achieve optimal order of accuracy of compactness.

3.2 Basis Functions

In the traditional DG method, numerical polynomial solutions $U_h$ in each element are represented using either standard Lagrange finite element or hierarchical node-based basis as follows:

$$U_h = \sum_{i=1}^{N} U_i B_i(x)$$  \hspace{1cm} (3.6)

As a result, the unknowns to be solved are the variables at the nodes $U_i$, as illustrated in Figure 3.1 for linear and quadratic polynomial approximations, where polynomial solutions are dependent on the shape of elements. For example, for a linear polynomial approximation in 2D as shown in Figure 3.1, a linear polynomial is used for triangular elements and the unknowns to be solved are the variables at the three vertices and a bi-linear polynomial is used for quadrilateral elements and the unknowns to be solved are the variables at the four vertices. However, the numerical polynomial solutions $U$ can be expressed in other forms as well. In our work, the numerical polynomial solutions are represented using a Taylor series expansion at the centroid of the cell. In order to present the main ideas, let us consider $P_2$ approximation in 3D, where numerical solutions on each cell are approximated using a quadratic polynomial. If we do a Taylor series expansion at the centroid of the cell, the quadratic polynomial solutions can...
be expressed as follows:

\[ \begin{align*}
U_h &= U_c + \frac{\partial U}{\partial x} \bigg|_c (x - x_c) + \frac{\partial U}{\partial y} \bigg|_c (y - y_c) + \frac{\partial U}{\partial z} \bigg|_c (z - z_c) \\
&+ \frac{\partial^2 U}{\partial x^2} \bigg|_c \frac{(x - x_c)^2}{2} + \frac{\partial^2 U}{\partial y^2} \bigg|_c \frac{(y - y_c)^2}{2} + \frac{\partial^2 U}{\partial z^2} \bigg|_c \frac{(z - z_c)^2}{2} \\
&+ \frac{\partial^2 U}{\partial x \partial y} \bigg|_c (x - x_c)(y - y_c) + \frac{\partial^2 U}{\partial x \partial z} \bigg|_c (x - x_c)(z - z_c) + \frac{\partial^2 U}{\partial y \partial z} \bigg|_c (y - y_c)(z - z_c)
\end{align*} \tag{3.7}\]

which can be further expressed as cell-averaged values and their derivatives at the centroid of the cell:

\[ \begin{align*}
U_h &= \tilde{U} + \frac{\partial U}{\partial x} \bigg|_c (x - x_c) + \frac{\partial U}{\partial y} \bigg|_c (y - y_c) + \frac{\partial U}{\partial z} \bigg|_c (z - z_c) \\
&+ \frac{\partial^2 U}{\partial x^2} \bigg|_c \left( \frac{(x - x_c)^2}{2} - \frac{1}{\Omega_e} \int_{\Omega_e} \frac{(x - x_c)^2}{2} \, d\Omega \right) \\
&+ \frac{\partial^2 U}{\partial y^2} \bigg|_c \left( \frac{(y - y_c)^2}{2} - \frac{1}{\Omega_e} \int_{\Omega_e} \frac{(y - y_c)^2}{2} \, d\Omega \right) \\
&+ \frac{\partial^2 U}{\partial z^2} \bigg|_c \left( \frac{(z - z_c)^2}{2} - \frac{1}{\Omega_e} \int_{\Omega_e} \frac{(z - z_c)^2}{2} \, d\Omega \right) \\
&+ \frac{\partial^2 U}{\partial x \partial y} \bigg|_c (x - x_c)(y - y_c) - \frac{1}{\Omega_e} \int_{\Omega_e} (x - x_c)(y - y_c) \, d\Omega \\
&+ \frac{\partial^2 U}{\partial x \partial z} \bigg|_c (x - x_c)(z - z_c) - \frac{1}{\Omega_e} \int_{\Omega_e} (x - x_c)(z - z_c) \, d\Omega \\
&+ \frac{\partial^2 U}{\partial y \partial z} \bigg|_c (y - y_c)(z - z_c) - \frac{1}{\Omega_e} \int_{\Omega_e} (y - y_c)(z - z_c) \, d\Omega
\end{align*} \tag{3.8}\]

where \( \tilde{U} \) is the mean value of \( U \) in this cell. The unknowns to be solved in this formulation are the cell-averaged variables and their derivatives at the center of the cells, regardless of element shapes, as shown in Figure 3.2. In this case, the dimension of the polynomial space is ten and
the ten basis functions are

\[
B_1 = 1 \\
B_2 = x - x_c \\
B_3 = y - y_c \\
B_4 = z - z_c \\
B_5 = \frac{(x - x_c)^2}{2} - \frac{1}{\Omega_e} \int_{\Omega_e} \frac{(x - x_c)^2}{2} d\Omega \\
B_6 = \frac{(y - y_c)^2}{2} - \frac{1}{\Omega_e} \int_{\Omega_e} \frac{(y - y_c)^2}{2} d\Omega \\
B_7 = \frac{(z - z_c)^2}{2} - \frac{1}{\Omega_e} \int_{\Omega_e} \frac{(z - z_c)^2}{2} d\Omega \\
B_8 = (x - x_c)(y - y_c) - \frac{1}{\Omega_e} \int_{\Omega_e} (x - x_c)(y - y_c) d\Omega \\
B_9 = (x - x_c)(z - z_c) - \frac{1}{\Omega_e} \int_{\Omega_e} (x - x_c)(z - z_c) d\Omega \\
B_{10} = (y - y_c)(z - z_c) - \frac{1}{\Omega_e} \int_{\Omega_e} (y - y_c)(z - z_c) d\Omega
\]  

and the discontinuous Galerkin formulation Eq. 3.5 then leads to the following ten equations

\[
\frac{d}{dt} \int_{\Omega_e} \ddot{U} d\Omega + \int_{\Gamma_e} F_k(U_h) n_k d\Gamma = \int_{\Gamma_e} G_k(U_h) n_k d\Gamma \\
\sum_{j=2}^{10} \int_{\Omega_e} B_i B_j \frac{d}{dt} \begin{pmatrix}
\frac{\partial U}{\partial x} c \\
\frac{\partial U}{\partial y} c \\
\frac{\partial U}{\partial z} c \\
\frac{\partial^2 U}{\partial x^2} c \\
\frac{\partial^2 U}{\partial y^2} c \\
\frac{\partial^2 U}{\partial z^2} c \\
\frac{\partial^2 U}{\partial x \partial y} c \\
\frac{\partial^2 U}{\partial x \partial z} c \\
\frac{\partial^2 U}{\partial y \partial z} c
dataOmega
dataOmega
dataOmega
dataOmega
dataOmega
dataOmega
dataOmega
dataOmega
dataOmega
dataOmega
\end{pmatrix} d\Omega + \int_{\Gamma_e} F_k(U_h) n_k B_i d\Gamma - \int_{\Omega_e} F_k(U_h) \frac{\partial B_i}{\partial x_k} d\Omega \\
= \int_{\Gamma_e} G_k(U_h) n_k B_i d\Gamma - \int_{\Omega_e} G_k(U_h) \frac{\partial B_i}{\partial x_k} d\Omega \quad 2 \leq i \leq 10
\]  

Note that in this formulation, the equations for the cell-averaged variables are decoupled from the equations for their derivatives due to the judicious choice of the basis functions in our formulation and the fact

\[
\int_{\Omega_e} B_i B_j d\Omega = 0 \quad 2 \leq i \leq 10
\]  

Using this formulation, the similarity and difference between DG and FV methods become clear, and the advantage of the DG methods is especially evident in comparison with the FV
methods. In fact, the discretized governing equations for cell-averaged variables Eq. 3.10 and the assumption of a polynomial solution on each cell Eq. 3.7 are exactly the same for both methods. In other words, this DG method provides a unified formulation, where the existing finite volume methods can be recovered virtually. For example, the application of this DG method to the median dual control volume of a given mesh will lead to the classic vertex-centered finite volume schemes as shown in Figure 3.3, while the application of this DG method to the cell itself of any given mesh will lead to the classic cell-centered finite volume scheme as shown in Figure 3.2. The only difference between them is the way how to obtain the polynomial solutions, i.e., how to compute the derivatives of high-order polynomial solutions ($\geq 2$). In the finite volume methods, the derivatives of the polynomial solutions of degree $p$ are reconstructed using cell-averaged values of the flow variables in the neighboring cells, which can be obtained using either TVD/MUSCL [128, 129] or ENO/WENO [5, 50, 53, 60, 76, 120] reconstruction schemes. Unfortunately, the multi-dimensional TVD/MUSCL reconstruction schemes of arbitrary order based on the extension of one-dimensional MUSCL approach, which are praised to achieve high-order accuracy for multi-dimensional problems, suffer from two serious flaws in the context of unstructured grids: (1) uncertainty and arbitrariness in choosing the stencils and methods to compute the derivatives. This explains why a nominally second order finite volume scheme is hardly able to deliver a formal solution of second order accuracy in practice for unstructured grids. (2) Extended stencils required for the reconstruction of higher-order ($\geq 2$nd) polynomial solutions. This is exactly the reason why the current finite volume methods using the TVD/MUSCL reconstruction are not practical at higher-order and have remained second order on unstructured grids. When the ENO/WENO reconstruction schemes are used for the construction of a polynomial of degree $p$ on unstructured grids, the dimension of the polynomial space, $N=N(p,d)$ depends on the degree of polynomials of the expansion $p$, and the number of spatial dimensions $d$. One must have three, six and ten stencils in 2D and four, ten and twenty cells in 3D for the construction of a linear, quadratic and cubic Lagrange polynomial, respectively. Undoubtedly, it is an overwhelming challenge, if not practically impossible, as to judiciously choose a set of admissible and proper stencils that have such a large number of cells on unstructured grids especially for higher order polynomials and higher dimensions. This explains why the application of higher-order ENO/WENO methods hardly exist on unstructured grids, in spite of their tremendous success on structured grids and their superior performance over the TVD/MUSCL methods. Unlike the FV methods, where the derivatives are reconstructed using the mean values of the neighboring cells, the DG methods solve the equations for the derivatives in a manner similar to the mean variables. This is natural, unique, compact, rigorous and element mathematically in contrast with arbitrariness characterizing the reconstruction schemes used in the FV methods with respect to how to compute the derivatives and how to choose the stencils. It is our belief that this is one of the main reasons why the
Figure 3.3: Representation of polynomial solutions using a Taylor series expansion for for the median dual control volume.

second order DG methods are more accurate than the FV methods using either TVD/MUSCL or ENO/WENO reconstruction schemes, which has been numerically demonstrated [84]. In addition, the perception that the DG methods are more expensive than the FV methods in terms of both computing costs and storage requirements, is actually baseless, as the storage of derivatives is also required for the FV methods as well in the context of unstructured grids, and solving the discretized equations of the derivatives are relatively inexpensive due to the fact that the numerical Riemann fluxes, representing the most dominant CPU consuming operation, are already computed for the cell-averaged equations, and the fact that a quadrature-free formulation can be used to significantly reduce the number of flux evaluations, and thus the computational costs associated with numerical quadrature. Furthermore, the higher-order DG methods can be easily constructed by simply increasing the degree $p$ of the polynomials locally, in contrast to the finite volume methods which use the extended stencils to achieve higher-order of accuracy. Some other methods such as ADER scheme by Titarev and Toro [44, 124], compact finite differencing scheme by Lele [123], and Visbal and Gaitonde [133], and CE/SE scheme by Chang [30] also solve the governing equations for the derivatives instead of using the reconstruction schemes. Note that in the case of $P_1$ approximation (piecewise linear), this formulation results in the so-called moment approach to the approximation of the weak solution of the Euler equations, that was first introduced by van Leer [129], and was also used by Allmaras et al. [8], Agarwal et al. [6], and Huynh [63].

This formulation has a number of distinct, desirable, and attractive features and advantages in the context of DG methods. First, the same numerical polynomial solutions are used for any shape of elements, which can be triangle, quadrilateral and polygon in 2D, and tetrahedron, pyramid, prism and hexahedron in 3D. Using this formulation, the DG method can be easily implemented on arbitrary meshes. The numerical method based on this formulation has the
ability to compute 1D, 2D and 3D problems using the very same code, which greatly alleviates the need and pain for code maintenance and upgrade. Secondly, the cell-averaged variables and their derivatives are handily available in this formulation. This makes implementation of WENO limiter straightforward and efficient [76, 83, 107, 105, 106], that is required to eliminate non-physical oscillations in the vicinity of discontinuities. Thirdly, the basis functions are hierarchic. This greatly facilitates implementation of p-multigrid methods [82, 85] and p-refinement. Lastly, the cell-averaged variable equations are decoupled from their derivative equations in this formulation, which makes development of fast, low-storage implicit methods possible.

In the implementation of this DG method, the basis functions are actually normalized in order to improve the conditioning of the system matrix Eq. 3.11 as follows:

\[
\begin{align*}
\tilde{B}_1 &= 1, & \tilde{B}_2 &= \frac{x - x_c}{\Delta x}, & \tilde{B}_3 &= \frac{y - y_c}{\Delta y}, & \tilde{B}_4 &= \frac{z - z_c}{\Delta z} \\
\tilde{B}_5 &= \frac{(x - x_c)^2}{2\Delta x^2} - \frac{1}{\Omega_e} \int_{\Omega_e} \left( \frac{(x - x_c)^2}{2\Delta x^2} \right) d\Omega \\
\tilde{B}_6 &= \frac{(y - y_c)^2}{2\Delta y^2} - \frac{1}{\Omega_e} \int_{\Omega_e} \left( \frac{(y - y_c)^2}{2\Delta y^2} \right) d\Omega \\
\tilde{B}_7 &= \frac{(z - z_c)^2}{2\Delta z^2} - \frac{1}{\Omega_e} \int_{\Omega_e} \left( \frac{(z - z_c)^2}{2\Delta z^2} \right) d\Omega \\
\tilde{B}_8 &= \frac{(x - x_c)(y - y_c)}{\Delta x\Delta y} - \frac{1}{\Omega_e} \int_{\Omega_e} \left( \frac{(x - x_c)(y - y_c)}{\Delta x\Delta y} \right) d\Omega \\
\tilde{B}_9 &= \frac{(x - x_c)(z - z_c)}{\Delta x\Delta z} - \frac{1}{\Omega_e} \int_{\Omega_e} \left( \frac{(x - x_c)(z - z_c)}{\Delta x\Delta z} \right) d\Omega \\
\tilde{B}_{10} &= \frac{(y - y_c)(z - z_c)}{\Delta y\Delta z} - \frac{1}{\Omega_e} \int_{\Omega_e} \left( \frac{(y - y_c)(z - z_c)}{\Delta y\Delta z} \right) d\Omega
\end{align*}
\]

where \(\Delta x = 0.5(x_{\text{max}} - x_{\text{min}})\), \(\Delta y = 0.5(y_{\text{max}} - y_{\text{min}})\), \(\Delta z = 0.5(z_{\text{max}} - z_{\text{min}})\) and \(x_{\text{max}}, y_{\text{max}}, z_{\text{max}}\) and \(x_{\text{min}}, y_{\text{min}}, z_{\text{min}}\) are the maximum and minimum coordinates in the cell \(\Omega_e\) in \(x\), \(y\) and \(z\) directions, respectively. Accordingly, the scaled first and second derivatives of the unknown variables are denoted as

\[
\begin{align*}
U_{x,c} &= \frac{\partial U}{\partial x} \Big|_c \Delta x \\
U_{y,c} &= \frac{\partial U}{\partial y} \Big|_c \Delta y \\
U_{z,c} &= \frac{\partial U}{\partial z} \Big|_c \Delta z \\
U_{xx,c} &= \frac{\partial^2 U}{\partial x^2} \Big|_c \Delta x^2 \\
U_{yy,c} &= \frac{\partial^2 U}{\partial y^2} \Big|_c \Delta y^2 \\
U_{zz,c} &= \frac{\partial^2 U}{\partial z^2} \Big|_c \Delta z^2 \\
U_{xy,c} &= \frac{\partial^2 U}{\partial x \partial y} \Big|_c \Delta x \Delta y \\
U_{xz,c} &= \frac{\partial^2 U}{\partial x \partial z} \Big|_c \Delta x \Delta z \\
U_{yz,c} &= \frac{\partial^2 U}{\partial y \partial z} \Big|_c \Delta y \Delta z
\end{align*}
\]
Finally, the quadratic polynomial solutions can be rewritten as

\[
U_h = \tilde{U} + U_{x,c}\tilde{B}_2 + U_{y,c}\tilde{B}_3 + U_{z,c}\tilde{B}_4 + U_{xx,c}\tilde{B}_5 + U_{yy,c}\tilde{B}_6 + U_{zz,c}\tilde{B}_7 + U_{xy,c}\tilde{B}_8 + U_{xz,c}\tilde{B}_9 + U_{yz,c}\tilde{B}_{10}
\]  

(3.15)

This is especially helpful and important to remove the stiffness of the system matrix for higher-order DG approximations.

### 3.3 Elemental Calculations

In this section, the use of isoparametric elements for 3D domain calculations and boundary calculations for discontinuous Galerkin methods is introduced in brief, since the content of this part can also be referred in some textbooks [59, 75]. The numerical integration for the discretized weak formulation of the governing equations is also described concisely.

#### 3.3.1 Shape Functions

First, our focus is on boundary calculations. For an isoparametric formulation, the nodes for numerical integration in the faces are mapped to a reference (or canonical) face, and all ensuing calculations are conducted in the mapped face. An example is shown in Figure 3.4, where a curvilinear quadrilateral face is mapped to its corresponding 2D reference face in the normalized coordinate system. The transformation between the face in physical space and the reference face is invertible, or in other words, the Jacobian of the transformation matrix is positive. The quadrilateral reference face is chosen to be a square, where the local coordinates \( \xi \) and \( \eta \) are normalized, \( \{(\xi, \eta) \mid \xi, \eta \in [-1, +1]\} \). If we treat the coordinate variables \( x, y \) and \( z \) themselves
as functions on $\Gamma$, then they can be approximated using the standard finite element shape functions in the form

$$
x(\xi, \eta) = \sum_{i} x_i \phi_i(\xi, \eta), \quad y(\xi, \eta) = \sum_{i} y_i \phi_i(\xi, \eta), \quad z(\xi, \eta) = \sum_{i} z_i \phi_i(\xi, \eta)
$$

(3.16)

where $(x_i, y_i, z_i)$ are the $(x, y, z)$ coordinates of local nodal point $i$ of face $\Gamma_e$. $N_f$ is equal to the number of the shape functions for the face. Note that by this transformation, the mapping can be constructed from every face in the physical space to the reference face using the shape functions. The 4-node and shape functions and derivatives for linear quadrilateral element and 8-node shape functions for curvilinear quadrilateral element are expressed in §A.2.2 and §A.2.3, respectively.

Note that the functions $x, y$ and $z$ are differentiable with respect to the local coordinates $\xi$ and $\eta$ in reference space:

$$
dx(\xi, \eta) = \frac{\partial x}{\partial \xi} d\xi + \frac{\partial x}{\partial \eta} d\eta = \sum_{i} x_i \frac{\partial \phi_i(\xi, \eta)}{\partial \xi} d\xi + \sum_{i} x_i \frac{\partial \phi_i(\xi, \eta)}{\partial \eta} d\eta
$$

$$
dy(\xi, \eta) = \frac{\partial y}{\partial \xi} d\xi + \frac{\partial y}{\partial \eta} d\eta = \sum_{i} y_i \frac{\partial \phi_i(\xi, \eta)}{\partial \xi} d\xi + \sum_{i} y_i \frac{\partial \phi_i(\xi, \eta)}{\partial \eta} d\eta
$$

(3.17)

$$
dz(\xi, \eta) = \frac{\partial z}{\partial \xi} d\xi + \frac{\partial z}{\partial \eta} d\eta = \sum_{i} z_i \frac{\partial \phi_i(\xi, \eta)}{\partial \xi} d\xi + \sum_{i} z_i \frac{\partial \phi_i(\xi, \eta)}{\partial \eta} d\eta
$$

Thus a differential face area $dA$ in the $(x, y, z)$ space can be written in terms of the reference coordinates $(\xi, \eta)$,

$$
dA(\xi, \eta) = |J_{\Gamma}(\xi, \eta)| d\xi d\eta
$$

(3.18)

with the Jacobian of the transformation $|J_{\Gamma}(\xi, \eta)|$ calculated by

$$
|J_{\Gamma}(\xi, \eta)| = \left[ \left( \frac{\partial x}{\partial \xi} \frac{\partial x}{\partial \eta} \right)^2 + \left( \frac{\partial y}{\partial \xi} \frac{\partial y}{\partial \eta} \right)^2 + \left( \frac{\partial z}{\partial \xi} \frac{\partial z}{\partial \eta} \right)^2 \right]^{1/2}
$$

(3.19)

in which all the partial derivative entries of the matrices can be calculated according to Eq. 3.17. If it is a linear face, i.e., 3-node triangle or a bilinear face, i.e., 4-node quadrilateral, $|J_{\Gamma}(\xi, \eta)|$ is nothing but the ratio of the area of face in physical space to the area of face in reference space.
Moreover, the unit vector normal to the differential face area \( dA \) can be calculated by

\[
\mathbf{n}_r = \frac{1}{|J_T(\xi, \eta)|} \begin{vmatrix}
\frac{\partial y}{\partial \xi} & \frac{\partial y}{\partial \eta} & \frac{\partial z}{\partial \xi} \\
\frac{\partial z}{\partial \xi} & \frac{\partial z}{\partial \eta} & \frac{\partial x}{\partial \xi} \\
\frac{\partial x}{\partial \xi} & \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \xi}
\end{vmatrix}
\]

(3.20)

Similarly, \( \mathbf{n}_r \) is a constant vector normal to a linear/bilinear face. Note that the grid topology should guarantee that the direction of \( \mathbf{n}_r \) points outward of its host cell.

Shape functions for triangular faces can also be expressed in terms of barycentric coordinates \((L_1, L_2, L_3)\) (or area coordinates):

\[
\phi_i(\xi, \eta) = \phi_i(L_1, L_2, L_3) , \text{ with } L_1 + L_2 + L_3 = 1
\]

which are discussed in detail in §A.1.1. Accordingly, the 3-node shape functions and derivatives for linear triangle and 6-node shape functions for curvilinear triangle are expressed in §A.1.2 and §A.1.3, respectively.

If extended to domain calculations, the coordinate variables \(x, y\) and \(z\) are treated as functions of local reference coordinates \(\{\xi, \eta, \zeta\} \mid \xi, \eta, \zeta \in [-1, +1]\) on \(\Omega\), then approximated using the standard finite element shape functions in the form

\[
\begin{align*}
x(\xi, \eta, \zeta) &= \sum_{i} x_i \phi_i(\xi, \eta, \zeta), \\
y(\xi, \eta, \zeta) &= \sum_{i} y_i \phi_i(\xi, \eta, \zeta), \\
z(\xi, \eta, \zeta) &= \sum_{i} z_i \phi_i(\xi, \eta, \zeta)
\end{align*}
\]

(3.21)

where \(N_e\) is equal to the number of the shape functions for the element. The 8-node shape functions and derivatives for linear hexahedron and 20-node shape functions and derivatives for trilinear hexahedron are expressed in §A.4.1 and §A.4.2, respectively. The functions \(x, y\) and \(z\) are differentiable with respect to the coordinates \(\xi, \eta\) and \(\zeta\) in reference space:

\[
\begin{align*}
dx(\xi, \eta, \zeta) &= \frac{\partial x}{\partial \xi} d\xi + \frac{\partial x}{\partial \eta} d\eta + \frac{\partial x}{\partial \zeta} d\zeta = \sum_{i} x_i \frac{\partial \phi_i}{\partial \xi} d\xi + \sum_{i} x_i \frac{\partial \phi_i}{\partial \eta} d\eta + \sum_{i} x_i \frac{\partial \phi_i}{\partial \zeta} d\zeta \\
dy(\xi, \eta, \zeta) &= \frac{\partial y}{\partial \xi} d\xi + \frac{\partial y}{\partial \eta} d\eta + \frac{\partial y}{\partial \zeta} d\zeta = \sum_{i} y_i \frac{\partial \phi_i}{\partial \xi} d\xi + \sum_{i} y_i \frac{\partial \phi_i}{\partial \eta} d\eta + \sum_{i} y_i \frac{\partial \phi_i}{\partial \zeta} d\zeta \\
dz(\xi, \eta, \zeta) &= \frac{\partial z}{\partial \xi} d\xi + \frac{\partial z}{\partial \eta} d\eta + \frac{\partial z}{\partial \zeta} d\zeta = \sum_{i} z_i \frac{\partial \phi_i}{\partial \xi} d\xi + \sum_{i} z_i \frac{\partial \phi_i}{\partial \eta} d\eta + \sum_{i} z_i \frac{\partial \phi_i}{\partial \zeta} d\zeta
\end{align*}
\]

(3.22)

Then a differential domain volume \(dV\) in the \((x, y, z)\) space can be written in terms of the reference coordinates \((\xi, \eta, \zeta)\),

\[
dV(\xi, \eta, \zeta) = |J_\Omega(\xi, \eta)| d\xi d\eta d\zeta
\]

(3.23)
with the Jacobian of the transformation \(|J_\Omega(\xi, \eta)|\) calculated by

\[
|J_\Omega(\xi, \eta, \zeta)| = \left| \begin{array}{ccc}
\frac{\partial x}{\partial \xi} & \frac{\partial x}{\partial \eta} & \frac{\partial x}{\partial \zeta} \\
\frac{\partial y}{\partial \xi} & \frac{\partial y}{\partial \eta} & \frac{\partial y}{\partial \zeta} \\
\frac{\partial z}{\partial \xi} & \frac{\partial z}{\partial \eta} & \frac{\partial z}{\partial \zeta}
\end{array} \right|
\] (3.24)

in which all the partial derivative entries of the matrices can be calculated according to Eq. 3.22.

Similar to the linear triangular elements, shape functions for tetrahedral elements can also be expressed in terms of barycentric coordinates \((L_1, L_2, L_3, L_4)\):

\[
\phi_i(\xi, \eta, \zeta) = \phi_i(L_1, L_2, L_3, L_4), \text{ with } L_1 + L_2 + L_3 + L_4 = 1
\]

which are discussed in detail in §A.3.1. Accordingly, the 4-node shape functions and derivatives for linear tetrahedron and 10-node shape functions and derivatives for bilinear tetrahedron are expressed in §A.3.2 and §A.3.3, respectively. In addition, the shape functions for pyramid and prism (wedge) are also listed in Appendix A. Some other nonstandard types of element like wrick (prism + pyramid) are not included in our study, since they are rarely used for hybrid grids in CFD.

### 3.3.2 Numerical Integration

Numerical integration for the discontinuous Galerkin formulation is often carried out using the numerical quadrature formulae. In our study, we adopt the Gaussian integration quadrature formulae in discontinuous finite element calculations since they present the best accuracy for a given number of points. The objective of the effort discussed in this section is only limited to apply the numerical integration to the semi-discretized weak formulation of the governing equations as shown in Eq. 3.5, but not trying to provide the detailed procedures of numerical integration, as the principles and complete concepts can be referred to some well-accepted textbooks [59, 75].

Numerical integration over a control face, i.e., inviscid flux integration over a triangle, can be performed with respect to the reference coordinates \((\xi, \eta)\):

\[
\int_{\Gamma_e} \mathbf{F}_k(U_h) \mathbf{n}_k B_i \, d\Gamma = \int_{\Gamma_e} \mathbf{H}_k \left( \sum_{j=1}^{N} B^L_j \mathbf{u}^L_j, \sum_{j=1}^{N} B^R_j \mathbf{u}^R_j, \mathbf{n}_k \right) B^L_i \, d\Gamma
\]

\[
= \sum_{l=1}^{M} \mathbf{H}_k \left( \sum_{j=1}^{N} B^L_j \mathbf{u}^L_j, \sum_{j=1}^{N} B^R_j \mathbf{u}^R_j, \mathbf{n}_k \right) B^L_i w_l |J_{\Gamma_e}(\xi_l, \eta_l)| \quad 1 \leq i \leq N
\] (3.25)
where the basis functions $B_L$ and $B_R$ are the functions of the coordinates $(x, y, z)$, and thus also the functions of the integration points $(\xi, \eta)$ in reference space. In Eq. 3.25, $w_I$ are the associated weighting factors, and $M$ is the total number of integration points, which indicates the integration order as listed in Table B.1. For example, if $M = 1$, an exact integration can be obtained for a polynomial of degree $p = 1$; for $M = 3$, integration is exact for $p \leq 2$; and for $M = 4$, $p \leq 3$. The weights also exist for quadrilateral faces, which are listed in Table B.2.

Similarly, numerical integration over a control volume, i.e., inviscid flux integration over a tetrahedron can be conducted using the reference coordinates $(\xi, \eta, \zeta)$:

$$\int_{\Omega_e} F_k(U_h) \frac{\partial B_i}{\partial x_k} \, d\Omega = \sum_{i=1}^{M} F_k \left( \sum_{j=1}^{N} B_j u_j \frac{\partial B_i}{\partial x_k} w_I |J_{\Omega_e}(\xi_l, \eta_l, \zeta_l)| \right) \quad 1 \leq i \leq N \quad (3.26)$$

The procedures for volume integration are basically the same for tetrahedron, pyramid, prism and hexahedron, and the weighting coefficients for these types of elements can be found in Appendix B.

### 3.4 Boundary Conditions

All the boundary conditions are weakly imposed in this work. Different from using the so-called ghost boundary cell in the finite volume methods, the DG methods usually employ the so-called boundary state vector $U^b_h(U_h, U_\infty)$ constructed at the boundary integration point, which is a function of the state vector $U_h$ of interior cell and the known free-stream vector $U_\infty$.

#### 3.4.1 Characteristic Boundary

If a boundary face is tagged as “inflow” or “outflow” or “farfield” in a grid, it will be uniformly treated as a characteristic boundary in the solver. The actual flow condition at the boundary face is determined by the Mach number $M^b$ derived from the mean state vector ($P_0$) in the interior cell. Four conditions are categorized based on the value of $M^b$, and then the $U^b_h$ vector and the approach of flux evaluation are specified accordingly.

**supersonic inflow ($M^b \leq -1$)**

$$H^b = F(U_\infty)$$

where the first-order free-stream values $U_\infty$ are prescribed at the integration point of the boundary face for flux evaluation.
**subsonic inflow** (−1 < \( M^b < 0 \))

\[
U^b_h = \begin{pmatrix}
\rho_{\infty} \\
\rho u_{\infty} \\
\rho v_{\infty} \\
\rho w_{\infty} \\
\rho c_{\infty}
\end{pmatrix}
\]

\( H^b = H(U^-, U^b_h, n^b) \)

where \( U_{\infty} \) are prescribed to the boundary state vector at the integration point, and the boundary flux is evaluated using a numerical flux scheme.

**subsonic outflow** (0 ≤ \( M^b < 1 \))

\[
U^b_h = \begin{pmatrix}
\rho^- \\
\rho u^- \\
\rho v^- \\
\rho w^- \\
\gamma (p_{\infty} - \frac{1}{2} \rho^{|V^-|^2})
\end{pmatrix}
\]

\( H^b = F(U^b_h) \)

where \( V^- = (u^-, v^-, w^-) \), and the first-order free-stream pressure \( p_{\infty} \) is imposed for the boundary state vector at the integration point, and the rest of required variables are computed from the interior cell.

**supersonic outflow** (\( M^b ≥ 1 \))

\( H^b = F(U^-) \)

where the interior state vector is extrapolated at the integration point of the boundary face and used for evaluation of the boundary flux.

### 3.4.2 Slip Wall / Symmetry Boundary

At the slip wall / symmetry boundary, the flow tangency requirement \( \mathbf{V} \cdot \mathbf{n} = 0 \) needs to be satisfied at the integration point. Instead of using the face unit normal vector \( \mathbf{n}^b \), the approximate physical unit normal vector \( \mathbf{n}^g \), which is a more accurate representation of normal direction on the physical entity, is required for the solver to obtain steady convergence and formal order of accuracy [71]. The vector \( \mathbf{n}^g \) is obtained through a quadratic representation of the solid wall boundary, i.e., 6-node curvilinear triangle and 8-node curvilinear quadrilateral as shown in Figure A.4 and Figure A.6, which is adopted in accordance with the quadratic spatial discretization. A comprehensive description of various types of higher-order elements can be found...
in Appendix A. Unlike those approaches that remodel and approximate the curved geometries with either additional geometric information or extended stencils on the linear elements [71, 72], i.e., 3-node triangle and 4-node bilinear quadrilateral, in this work the curved triangular or quadrilateral surface meshes are directly generated provided with the CAD geometric configurations during pre-processing, indicating the robustness of the code to deal with arbitrary complex geometries without recourse to any special treatment. Thus \( \mathbf{n}^g \) is approximated by Eq. 3.20 at each integration point.

The velocity at the integration point is then computed as

\[
\begin{align*}
u^b &= u^- - 2 \left( \mathbf{V}^- \cdot \mathbf{n}^g \right) n^g_x \\
v^b &= v^- - 2 \left( \mathbf{V}^- \cdot \mathbf{n}^g \right) n^g_y \\
w^b &= w^- - 2 \left( \mathbf{V}^- \cdot \mathbf{n}^g \right) n^g_z
\end{align*}
\]

The density and total energy at the integration point are computed from the interior cell. The boundary state vector and the numerical flux is then computed as

\[
\begin{align*}
\mathbf{U}_b^h &= \begin{pmatrix}
\rho^- \\
\rho^- u^b \\
\rho^- v^b \\
\rho^- w^b \\
\rho e^-
\end{pmatrix} \\
\mathbf{H}_b^h &= \mathbf{H}(\mathbf{U}_h^-, \mathbf{U}_h^b, \mathbf{n}^b)
\end{align*}
\]

where one can see that \( \mathbf{n}^g \) is only required to compute \( \mathbf{V}^b \), and \( \mathbf{n}^b \) is still used in flux evaluation.

3.4.3 No-Slip Adiabatic Wall Boundary

At the no-slip adiabatic wall boundary, the zero velocity condition and the zero normal temperature gradient condition at the wall face are prescribed at the integration point

\[
\begin{align*}
u^b &= v^b = w^b = 0 \\
\left( \frac{\partial T}{\partial n^b} \right) &= \left( \nabla T \right)^b \cdot \mathbf{n}^b = 0
\end{align*}
\]

where the rest of the required state variables are computed from the interior cell

\[
\mathbf{U}_h^b = \begin{pmatrix}
\rho^- \\
0 \\
0 \\
0 \\
\rho e^-
\end{pmatrix}
\]
To take the possible jump at the wall boundary into consideration, the corrected gradient $(\nabla U_h + r_h)^b$ is adopted as the auxiliary gradient, where $r_h$ is the so-called local lifting operator computed at the wall boundary [19]. The numerical flux at the no-slip adiabatic wall boundary is computed as

$$H^b = F \left( U_h^b, (\nabla U_h + r_h)^b \right)$$

### 3.4.4 No-Slip Isothermal Wall Boundary

At the no-slip isothermal wall boundary, the zero velocity condition and the fixed temperature $T_{wall}$ condition at the wall face are prescribed at the integration point

$$u^b = v^b = w^b = 0 \quad T^b = 2T_{wall} - T^- \quad \rho e^b = \frac{\rho^- R T^b}{\gamma - 1}$$

where $\bar{R}$ is the non-dimensional universal gas constant for perfect gas, and the rest of the required state variables are computed from the interior cell

$$U_h^b = \begin{pmatrix} \rho^- \\ 0 \\ 0 \\ 0 \\ \rho e^b \end{pmatrix}$$

Similar to the adiabatic wall treatment, the corrected gradient $(\nabla U_h + r_h)^b$ is used to account for the possible jump at the wall boundary. The numerical flux at the no-slip isothermal wall boundary is computed as

$$H^b = F \left( U_h^b, (\nabla U_h + r_h)^b \right)$$
Chapter 4

Reconstruction Methods

In this chapter, a three-step reconstruction scheme which consists of the least-squares reconstruction and the hierarchical WENO reconstruction is presented. The outline of this chapter is organized in the following. In §4.1, a general procedure of the least-squares reconstruction is described and four major methods for solving an over-determined system of linear equations are introduced. The hierarchical WENO reconstruction is then described in §4.2 for WENO reconstruction at $P_2$, and §4.3 for WENO reconstruction at $P_1$, respectively.

4.1 Least-Squares Reconstruction

As a continuation of the discontinuous Galerkin discretization described earlier in §3.2 where a number of advantages of the DG methods are stated, it is also noted that people need to be able to properly address several major drawbacks of the DG methods before they can be implemented robustly and efficiently in engineering applications.

In comparison to the reconstructed FV methods, the DG methods have a significant drawback in that they require more degrees of freedom, an additional domain integration, and more Gauss quadrature points for the boundary integration, and therefore more computational costs and storage requirements. On the one hand, the reconstruction methods that the FV methods use to achieve higher-order accuracy are relatively inexpensive but less accurate and robust. One the other hand, the DG methods that can be viewed as a different way to extend a FV method to higher orders are accurate and robust but costly. It is only natural and tempting to combine the efficiency of the reconstruction methods and the accuracy of the DG methods. This idea was originally introduced by Dumbser et al. in the frame of $P_nP_m$ scheme [44, 45, 43], termed RDG($P_nP_m$) in this work, where $P_n$ indicates that a piecewise polynomial of degree of $n$ is used to represent a DG solution, and $P_m$ represents a reconstructed polynomial solution of degree of $m$ ($n \leq m$) that is used to compute the fluxes and source terms. The beauty of
RDG(P^nP^m) schemes is that they provide a unified formulation for both the FV and DG methods, and contain both the classical finite volume and standard discontinuous Galerkin methods as two special cases of RDG(P^nP^m) schemes, and thus allow for a direct efficiency comparison. When \( n = 0 \), i.e. a piecewise constant polynomial is used to represent a numerical solution, RDG(P^0P^m) is nothing but the classical high order finite volume schemes, where a polynomial solution of degree \( m \) \((m \leq 1)\) is reconstructed from a piecewise constant solution. When \( m = n \), the reconstruction reduces to the identity operator, and RDG(P^nP^m) scheme yields a standard DG method. Clearly, an accurate and efficient reconstruction is the key ingredient in extending the underlying DG method to higher order accuracy. Although the discussion in this work is mainly focused on the linear DG method RDG(P_1P_2), its extension to higher order DG methods is straightforward. In the case of DG(P_1) method, a linear polynomial solution \( U_i \) in any cell \( i \) is

\[
U_i = \tilde{U}_i + U_{x,i}\tilde{B}_2 + U_{y,i}\tilde{B}_3 + U_{z,i}\tilde{B}_4
\]

Using this underlying linear polynomial DG solution in the neighboring cells, one can reconstruct a quadratic polynomial solution \( U_i^R \) as follows:

\[
U_i = \tilde{U}_i^R + U_{x,i}^R\tilde{B}_2 + U_{y,i}^R\tilde{B}_3 + U_{z,i}^R\tilde{B}_4 + U_{xx,i}^R\tilde{B}_5 + U_{yy,i}^R\tilde{B}_6 + U_{zz,i}^R\tilde{B}_7 + U_{xy,i}^R\tilde{B}_8 + U_{xz,i}^R\tilde{B}_9 + U_{yz,i}^R\tilde{B}_{10}
\]

In order to maintain the compactness of the DG methods, the reconstruction is required to involve only von Neumann neighborhood, i.e., the adjacent cells that share a face with the cell \( i \).
under consideration, as shown in Figure 4.1. There are ten degrees of freedom, and therefore ten unknowns must be determined. The first four unknowns can be trivially obtained, by requiring the consistency of the RDG method with the underlying DG method: (1) The reconstruction scheme must be conservative, and (2) The values of the reconstructed first derivatives are equal to the ones of the first derivatives of the underlying DG solution at the centroid $i$. Due to the judicious choice of Taylor basis in our DG formulation, these four degrees of freedom simply coincide with the ones from the underlying DG solution, i.e.,

$$\tilde{U}_i^R = \tilde{U}_i \quad U_{x,i}^R = U_{x,i} \quad U_{y,i}^R = U_{y,i} \quad U_{z,i}^R = U_{z,i}$$ (4.3)

As a result, only six second derivatives need to be determined. This can be accomplished by requiring that the point-wise values and first derivatives of the reconstructed solution and of the underlying DG solution are equal at the cell centers for all the adjacent face neighboring cells. Consider a neighboring cell $j$, one requires

$$U_{j} = \tilde{U}_i^R + U_{x,i}^R \tilde{B}_2^j + U_{y,i}^R \tilde{B}_3^j + U_{z,i}^R \tilde{B}_4^j$$

$$+ U_{xx,i}^R \tilde{B}_5^j + U_{yy,i}^R \tilde{B}_6^j + U_{zz,i}^R \tilde{B}_7^j + U_{xy,i}^R \tilde{B}_8^j + U_{xz,i}^R \tilde{B}_9^j + U_{yz,i}^R \tilde{B}_{10}^j$$

$$\frac{\partial U}{\partial x} \bigg|_j = U_{x,i}^R \frac{1}{\Delta x_i} + U_{xx,i}^R \frac{\tilde{B}_2^j}{\Delta x_i} + U_{xy,i}^R \frac{\tilde{B}_3^j}{\Delta x_i} + U_{xz,i}^R \frac{\tilde{B}_4^j}{\Delta x_i}$$ (4.4)

$$\frac{\partial U}{\partial y} \bigg|_j = U_{y,i}^R \frac{1}{\Delta y_i} + U_{yy,i}^R \frac{\tilde{B}_3^j}{\Delta y_i} + U_{xy,i}^R \frac{\tilde{B}_4^j}{\Delta y_i}$$

$$\frac{\partial U}{\partial z} \bigg|_j = U_{z,i}^R \frac{1}{\Delta z_i} + U_{zz,i}^R \frac{\tilde{B}_4^j}{\Delta z_i} + U_{yx,i}^R \frac{\tilde{B}_3^j}{\Delta z_i} + U_{yz,i}^R \frac{\tilde{B}_2^j}{\Delta z_i}$$

where the basis functions $B$ are evaluated at the center of cell $j$, i.e., $B = B(x_j, y_j, z_j)$. The
the size of the resulting non-square matrix is 16

a pyramid, a prism and a hexahedron are four, five, five and six, respectively. Consequently,

which leads to a non-square matrix. The numbers of the face-neighboring cells for a tetrahedron,

Similar equations can be written for all the cells connected to the cell

Finally, this group of equations can be written in a matrix form as follows:

\[
\begin{pmatrix}
\tilde{B}_5^j & \tilde{B}_6^j & \tilde{B}_7^j & \tilde{B}_8^j & \tilde{B}_9^j & \tilde{B}_10^j \\
\tilde{B}_5^j & 0 & 0 & \tilde{B}_4^j & 0 \\
0 & \tilde{B}_3^j & 0 & \tilde{B}_4^j & 0 \\
0 & 0 & \tilde{B}_4^j & \tilde{B}_2^j & \tilde{B}_3^j
\end{pmatrix}
\begin{pmatrix}
U_{xx,i}^R \\
U_{yy,i}^R \\
U_{zz,i}^R \\
U_{xy,i}^R \\
U_{xz,i}^R \\
U_{yz,i}^R
\end{pmatrix}
= \begin{pmatrix}
\Delta x_i \ddot{U}_{x,j} - \ddot{U}_{x,i} \\
\frac{\Delta y_i}{\Delta x_j} \ddot{U}_{y,j} - \ddot{U}_{y,i} \\
\frac{\Delta z_i}{\Delta y_j} \ddot{U}_{z,j} - \ddot{U}_{z,i}
\end{pmatrix}
\]

(4.6)

Similar equations can be written for all the cells connected to the cell \(i\) with a common face,

which leads to a non-square matrix. The numbers of the face-neighboring cells for a tetrahedron,
a pyramid, a prism and a hexahedron are four, five, five and six, respectively. Consequently,
the size of the resulting non-square matrix is 16 \(\times\) 6, 20 \(\times\) 6, 20 \(\times\) 6 and 24 \(\times\) 6, respectively. In the present work, this over-determined linear system of 16, or 20, or 24 equations for 6
unknowns is solved in the least-squares sense using both the normal equation approach and the QR decomposition to obtain the second derivatives of the reconstructed quadratic polynomial solution.

With the normal equation approach, a symmetric linear system of equations is yielded by pre-multiplying through by matrix transpose as follows

\[
\begin{pmatrix}
  a_{11} & a_{12} & a_{13} & a_{14} & a_{15} & a_{16} \\
  a_{12} & a_{22} & a_{23} & a_{24} & a_{25} & a_{26} \\
  a_{13} & a_{23} & a_{33} & a_{34} & a_{35} & a_{36} \\
  a_{14} & a_{24} & a_{34} & a_{44} & a_{45} & a_{46} \\
  a_{15} & a_{25} & a_{35} & a_{45} & a_{55} & a_{56} \\
  a_{16} & a_{26} & a_{36} & a_{46} & a_{56} & a_{66}
\end{pmatrix}
\begin{pmatrix}
  U_{x,x,i}^R \\
  U_{y,y,i}^R \\
  U_{z,z,i}^R \\
  U_{x,y,i}^R \\
  U_{x,z,i}^R \\
  U_{y,z,i}^R
\end{pmatrix}
= \begin{pmatrix}
  b_1 \\
  b_2 \\
  b_3 \\
  b_4 \\
  b_5 \\
  b_6
\end{pmatrix}
\] (4.7)

where

\[
  a_{11} = \sum_j (\tilde{B}_j^i \tilde{B}_j^i + \tilde{B}_j^i \tilde{B}_j^i) \quad a_{12} = \sum_j (\tilde{B}_j^i \tilde{B}_j^i) \quad a_{13} = \sum_j (\tilde{B}_j^i \tilde{B}_j^i) \\
  a_{14} = \sum_j (\tilde{B}_j^i \tilde{B}_j^i + \tilde{B}_j^i \tilde{B}_j^i) \quad a_{15} = \sum_j (\tilde{B}_j^i \tilde{B}_j^i + \tilde{B}_j^i \tilde{B}_j^i) \quad a_{16} = \sum_j (\tilde{B}_j^i \tilde{B}_j^i) \\
  a_{22} = \sum_j (\tilde{B}_j^i \tilde{B}_j^i + \tilde{B}_j^i \tilde{B}_j^i) \quad a_{23} = \sum_j (\tilde{B}_j^i \tilde{B}_j^i) \quad a_{24} = \sum_j (\tilde{B}_j^i \tilde{B}_j^i + \tilde{B}_j^i \tilde{B}_j^i) \\
  a_{25} = \sum_j (\tilde{B}_j^i \tilde{B}_j^i) \quad a_{26} = \sum_j (\tilde{B}_j^i \tilde{B}_j^i + \tilde{B}_j^i \tilde{B}_j^i) \quad a_{33} = \sum_j (\tilde{B}_j^i \tilde{B}_j^i + \tilde{B}_j^i \tilde{B}_j^i) \\
  a_{34} = \sum_j (\tilde{B}_j^i \tilde{B}_j^i) \quad a_{35} = \sum_j (\tilde{B}_j^i \tilde{B}_j^i + \tilde{B}_j^i \tilde{B}_j^i) \quad a_{36} = \sum_j (\tilde{B}_j^i \tilde{B}_j^i + \tilde{B}_j^i \tilde{B}_j^i) \\
  a_{44} = \sum_j (\tilde{B}_j^i \tilde{B}_j^i + \tilde{B}_j^i \tilde{B}_j^i) \quad a_{45} = \sum_j (\tilde{B}_j^i \tilde{B}_j^i + \tilde{B}_j^i \tilde{B}_j^i) \quad a_{46} = \sum_j (\tilde{B}_j^i \tilde{B}_j^i + \tilde{B}_j^i \tilde{B}_j^i) \\
  a_{55} = \sum_j (\tilde{B}_j^i \tilde{B}_j^i + \tilde{B}_j^i \tilde{B}_j^i) \quad a_{56} = \sum_j (\tilde{B}_j^i \tilde{B}_j^i + \tilde{B}_j^i \tilde{B}_j^i) \quad a_{66} = \sum_j (\tilde{B}_j^i \tilde{B}_j^i + \tilde{B}_j^i \tilde{B}_j^i)
\]

and

\[
  b_1 = \mathbf{R}_1 \tilde{B}_1^i + \mathbf{R}_2 \tilde{B}_2^i \quad b_2 = \mathbf{R}_1 \tilde{B}_1^i + \mathbf{R}_3 \tilde{B}_3^i \quad b_3 = \mathbf{R}_1 \tilde{B}_1^i + \mathbf{R}_4 \tilde{B}_4^i \quad b_4 = \mathbf{R}_1 \tilde{B}_1^i + \mathbf{R}_2 \tilde{B}_2^i + \mathbf{R}_3 \tilde{B}_3^i + \mathbf{R}_4 \tilde{B}_4^i \quad b_5 = \mathbf{R}_1 \tilde{B}_1^i + \mathbf{R}_3 \tilde{B}_3^i + \mathbf{R}_4 \tilde{B}_4^i \quad b_6 = \mathbf{R}_1 \tilde{B}_1^i + \mathbf{R}_3 \tilde{B}_3^i + \mathbf{R}_4 \tilde{B}_4^i
\]

This linear system of \(6 \times 6\) can be then trivially solved to obtain the second derivatives of the reconstructed quadratic polynomial solution. Furthermore, in implementation, the left-hand-side matrix in Eq. 4.7 can either be pre-computed for only once and stored at an extra expense.
of static memory requirement equal to \(36 \times Nelem\) words, or computed during the loop in each
time step with a dynamic memory allocation.

The QR decomposition (also called QR factorization) of a matrix is a decomposition of a
matrix \(A\) into a product \(A = QR\) of an orthogonal matrix \(Q\) and an upper triangular matrix
\(R\). QR decomposition is often used to solve the linear least squares system of equations, and
is the basis for a particular eigenvalue algorithm, the QR algorithm. In our framework, the
QR decomposition is implemented by calling the routines from the LAPACK (Linear Algebra
PACKage) library. Compared with the normal equation approach, QR decomposition is a little
more expensive in computing time, but is also observed to yield a slightly higher order of
accuracy in a very few numerical experiments in chapter 7.

4.2 WENO Reconstruction at \(P_2\): WENO\((P_1P_2)\)

This linear reconstruction-based RDG\((P_1P_2)\) method is able to achieve the designed third order
of accuracy and significantly improve the accuracy of the underlying second-order DG method
for solving the 2D compressible Euler equations on arbitrary grids [90, 88, 146, 95]. However,
when used to solve the 3D compressible Euler equations on tetrahedral grids or prismatic
grids, this RDG method suffers from the so-called linear instability, that is also observed in
the second-order cell-centered finite volume methods, i.e., RDG\((P_0P_1)\) [52]. This linear insta-
lability is attributed to the fact that the reconstruction stencils only involve the von Neumann
neighborhood, i.e., adjacent face-neighboring cells [52]. The linear stability can be achieved
using extended stencils, which will unfortunately sacrifice the compactness of the underlying
DG methods. Furthermore, such a linear reconstruction-based DG method will suffer from
non-linear instability, leading to non-physical oscillations in the vicinity of strong discontinu-
ities for the compressible Euler equations. Alternatively, the ENO (Essentially Non-Oscillatory)
scheme, WENO (Weighted Essentially Non-Oscillatory) scheme, and Hermite WENO scheme
can be used to reconstruct a higher-order polynomial solution, which can not only enhance
the order of accuracy of the underlying DG method but also achieve both linear and non-linear
stability. In the present work, the reconstructed quadratic polynomial based on the Hermite
WENO on cell \(i\) are a convex combination of the least-squares reconstructed second derivatives
at the cell itself and its face-neighboring cells,

\[
\frac{\partial^2 U}{\partial x_m \partial x_n} |_{i}^{\text{WENO}} = \sum_{k=1}^{1+N_{es}} w_k \frac{\partial^2 U}{\partial x_m \partial x_n} |_{k}
\]

where \(N_{es}\) denotes the number of its face-neighboring cells (\(N_{es}\) is 4 for tetrahedron, 5 for pyra-
mid and prism, and 6 for hexahedron), and the normalized nonlinear weights \(w_k\) are computed
\[ w_k = \frac{\tilde{w}_k}{1 + N_{es}} \sum_{i=1}^{\tilde{w}_i} \] (4.9)

The non-normalized nonlinear weights \( \tilde{w}_i \) are functions of the linear weights \( \lambda_i \) and the so-called oscillation indicator \( o_i \)

\[ \tilde{w}_k = \frac{\lambda_i}{(\varepsilon + o_i)^\gamma} \] (4.10)

where \( \varepsilon \) is a small positive number used to avoid division by zero, and \( \gamma \) an integer parameter to control how fast the non-linear weights decay for non-smooth stencils. The oscillation indicator \( o_k \) for the reconstructed second order polynomials is simply defined as

\[ o_k = \left[ \frac{\partial^2 U}{\partial x_m \partial x_n} \right]_{k}^{2} \] (4.11)

where the convention of summation over repeated indices \( m \) and \( n \) \((m, n = 1, 2, 3)\) are used. Note that the least-squares reconstructed polynomial at the cell itself serves as the central stencil and the least-squares reconstructed polynomials on its face-neighboring cells act as biased stencils in this Hermite WENO reconstruction. This reconstructed quadratic polynomial solution is then used to compute the domain and boundary integrals of the underlying DG(P\(_1\)) method in Eq. 3.5. The resulting DG method, termed a “reconstructed DG” method (RDG(P\(_1\)P\(_2\)) in short notation), is expected to have third order of accuracy at a moderate increase of computing costs in comparison to the underlying DG(P\(_1\)) method. The extra costs are mainly due to the least-squares reconstruction, which is relatively cheap in comparison to the evaluation of fluxes, and an extra Gauss quadrature point, which is required to calculate both domain and boundary integrals on tetrahedral cells. In comparison to DG(P\(_2\)), this represents a significant saving in terms of flux evaluations. Furthermore, the number of degrees of freedom is considerably reduced, which leads to a significant reduction in memory requirements, and from which implicit methods will benefit tremendously. The cost analysis for the RDG(P\(_0\)P\(_1\)) (FV(P\(_1\))), RDG(P\(_1\)P\(_1\)) (DG(P\(_1\))), RDG(P\(_1\)P\(_2\)) and RDG(P\(_2\)P\(_2\)) (DG(P\(_2\))) on a tetrahedral cell is summarized in Table 4.1, where the memory requirement for storing only the implicit diagonal matrix is given as well, and which grows quadratically with the order of the DG methods. We would like to emphasize that the storage requirements for the implicit DG methods are extremely demanding, especially for higher-order DG methods. A similar summary of cost analysis on a hexahedral grid is presented in Table 4.2, where the numbers of Gauss quadrature points for both the domain and boundary integrals required by the RDG(P\(_1\)P\(_2\)) method are the same with the RDG(P\(_1\)P\(_1\)) method, and are a huge reduction in comparison with the RDG(P\(_2\)P\(_2\)) method.
Note that this RDG method is not compact anymore, as the neighbors neighbors are used in updating the solution. However, the stencils used in the reconstruction are compact, involving only von Neumann neighbors. Consequently, the resultant RDG method can be implemented in a compact manner. If not specially stated, “RDG(P_1P_2) method” refers to WENO(P_1P_2), and “RDG(P_1P_2) solution” refers to the WENO(P_1P_2) reconstructed quadratic polynomial solution in chapter 7. As demonstrated in chapter 7, this RDG(P_1P_2) method is able to achieve the designed third order of accuracy for inviscid smooth flows and maintain the desired linear stability.

### Table 4.1: Cost analysis for different numerical methods on a tetrahedral grid

<table>
<thead>
<tr>
<th></th>
<th>RDG(P_0P_1)</th>
<th>RDG(P_1P_1)</th>
<th>RDG(P_1P_2)</th>
<th>RDG(P_2P_2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of quadrature points</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>for boundary integrals</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of quadrature points</td>
<td>0</td>
<td>4</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>for domain integrals</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reconstruction</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Order of accuracy</td>
<td>$O(h^2)$</td>
<td>$O(h^2)$</td>
<td>$O(h^3)$</td>
<td>$O(h^3)$</td>
</tr>
<tr>
<td>Storage for implicit diagonal</td>
<td>25 words per</td>
<td>400</td>
<td>400</td>
<td>2500</td>
</tr>
<tr>
<td>matrix</td>
<td>element</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Table 4.2: Cost analysis for different numerical methods on a hexahedral grid

<table>
<thead>
<tr>
<th></th>
<th>RDG(P_0P_1)</th>
<th>RDG(P_1P_1)</th>
<th>RDG(P_1P_2)</th>
<th>RDG(P_2P_2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of quadrature points</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>9</td>
</tr>
<tr>
<td>for boundary integrals</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of quadrature points</td>
<td>0</td>
<td>8</td>
<td>8</td>
<td>27</td>
</tr>
<tr>
<td>for domain integrals</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reconstruction</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Order of accuracy</td>
<td>$O(h^2)$</td>
<td>$O(h^2)$</td>
<td>$O(h^3)$</td>
<td>$O(h^4)$</td>
</tr>
<tr>
<td>Storage for implicit diagonal</td>
<td>25 words per</td>
<td>400</td>
<td>400</td>
<td>2500</td>
</tr>
<tr>
<td>matrix</td>
<td>element</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
4.3 WENO Reconstruction at P1: HWENO(P1P2)

Although the WENO(P1P2) method does not introduce any new oscillatory behavior for the reconstructed curvature terms (second derivatives) due to the WENO reconstruction, it cannot remove inherent oscillations in the underlying DG(P1) solutions. Consequently, the WENO(P1P2) method still suffers from the non-linear instability for flows with strong discontinuities. In order to eliminate non-physical oscillations in the vicinity of strong discontinuities and thus maintain the non-linear instability, the first derivatives need to be reconstructed using a WENO reconstruction. The resulting reconstructed discontinuous Galerkin method based on this Hierarchical WENO reconstruction is termed as HWENO(P1P2) [96] in this work, where a hierarchical reconstruction (successively from high order to low order) strategy [143] is adopted.

The WENO reconstruction for the first derivatives is based on the reconstructed quadratic polynomial solutions of the flow variables for each cell in the grid. The stencils are only chosen in the von Neumann neighborhood. More precisely, for a tetrahedral cell \(i\), the following four stencils \((i,j_1, j_2, j_3)\), \((i,j_1, j_2, j_4)\), \((i,j_1, j_3, j_4)\) and \((i,j_2, j_3, j_4)\), where \(j_1, j_2, j_3\) and \(j_4\) designate the four adjacent face-neighboring cells of the cell \(i\) are chosen to construct a Lagrange polynomial such that

\[
U_j = \tilde{U}_i^R + U_{x,i}^{R} \tilde{B}_2^j + U_{y,i}^{R} \tilde{B}_3^j + U_{z,i}^{R} \tilde{B}_4^j + U_{xx,i}^{R} \tilde{B}_5^j + U_{xy,i}^{R} \tilde{B}_6^j + U_{xz,i}^{R} \tilde{B}_7^j + U_{yy,i}^{R} \tilde{B}_8^j + U_{yz,i}^{R} \tilde{B}_9^j + U_{zz,i}^{R} \tilde{B}_{10}^j
\] (4.12)

where \(U_j\) refers to the point-wise value of the reconstructed polynomial solution at centroid of cell \(j\) and the normalized basis functions \(B\) are evaluated at the center of cell \(j\), i.e., \(\tilde{B} = \tilde{B}(x_j, y_j, z_j)\). In addition, the following four stencils \((i,j_1)\), \((i,j_2)\), \((i,j_3)\) and \((i,j_4)\) are chosen to construct a Hermite polynomial such that

\[
\frac{\partial U}{\partial x} \bigg|_j = U_{x,i}^{R} \frac{1}{\Delta x_i} + U_{xx,i}^{R} \frac{\tilde{B}_2^j}{\Delta x_i} + U_{xy,i}^{R} \frac{\tilde{B}_3^j}{\Delta y_i} + U_{xz,i}^{R} \frac{\tilde{B}_4^j}{\Delta z_i}
\]

\[
\frac{\partial U}{\partial y} \bigg|_j = U_{y,i}^{R} \frac{1}{\Delta y_i} + U_{yy,i}^{R} \frac{\tilde{B}_2^j}{\Delta y_i} + U_{xy,i}^{R} \frac{\tilde{B}_3^j}{\Delta x_i} + U_{yz,i}^{R} \frac{\tilde{B}_4^j}{\Delta z_i}
\] (4.13)

\[
\frac{\partial U}{\partial z} \bigg|_j = U_{z,i}^{R} \frac{1}{\Delta z_i} + U_{zz,i}^{R} \frac{\tilde{B}_2^j}{\Delta z_i} + U_{xy,i}^{R} \frac{\tilde{B}_3^j}{\Delta y_i} + U_{yz,i}^{R} \frac{\tilde{B}_4^j}{\Delta x_i}
\]

These eight reconstructed gradients \((U_{x,i}^{R}, U_{y,i}^{R}, \text{ and } U_{z,i}^{R})\) serving as the biased stencils and the gradient from the DG solution itself at cell \(i\) \((U_{x,i}, U_{y,i}, \text{ and } U_{z,i})\) acting as the central stencil are used to modify the first derivatives based on the WENO reconstruction as a convex
combination of these nine derivatives

$$\left. \frac{\partial U}{\partial x_m} \right|_{\text{WENO}} = \sum_{k=1}^{N_{\text{sten}}} w_k \left. \frac{\partial U}{\partial x_m} \right|_k$$

(4.14)

where, $N_{\text{sten}}$ denotes the number of stencils. In general, $N_{\text{sten}}$ is equal to \(C_{N_{\text{es}}}^3 + N_{\text{es}} + 1\) for an element (9 for a tetrahedron, 16 for a pyramid and prism, and 27 for a hexahedron). The normalized nonlinear weights $w_k$ are computed as

$$w_k = \frac{\tilde{w}_k}{\sum_{i=1}^{N_{\text{sten}}} \tilde{w}_i}$$

(4.15)

The non-normalized nonlinear weights $\tilde{w}_i$ are functions of the linear weights $\lambda_i$ and the so-called oscillation indicator $o_i$

$$\tilde{w}_k = \frac{\lambda_i}{(\varepsilon + o_i)^\gamma}$$

(4.16)

where $\varepsilon$ is a small positive number used to avoid division by zero, and $\gamma$ an integer parameter to control how fast the non-linear weights decay for non-smooth stencils. The oscillation indicator $o_k$ for the reconstructed first order polynomials is simply defined as

$$o_k = \left[ \left( \left. \frac{\partial U}{\partial x_m} \right|_k \right)^2 \right]^{\frac{1}{2}}$$

(4.17)

where the convention of summation over repeated indices $m$ ($m = 1, 2, 3$) is used. The present choice of stencils is symmetric, and compact, as the van Neumann neighbors are only involved in the reconstruction. This means that no additional data structure is required for our HWENO ($P_1P_2$) method. Note that this WENO reconstruction at $P_1$ is the extension of a HWENO limiter developed for the DG ($P_1$) by Luo et.al [83]. From the perspective of both computational cost and solution accuracy, the above WENO reconstruction at $P_1$ should only be used in the regions where strong discontinuities exist. This can be accomplished using the so-called discontinuity detectors, which are helpful to distinguish regions where solutions are smooth and discontinuous. The beauty of this WENO reconstruction is that in case that the reconstruction is mistakenly applied in the smooth cells, the uniform high-order accuracy can still be maintained, unlike the slope limiters, which, when applied near smooth extrema, will have a profoundly adverse impact on solution in the smooth region, leading to the loss of the original high-order accuracy. This remarkable feature of the WENO reconstruction in turn alleviates the burden on the discontinuity detectors, as no discontinuity detectors can really either in theory or in practice make a distinction between a stagnation point and a shock wave, as flow gradients
near the stagnation point are even larger than the ones near the shock wave in some cases. The developed HWENO(P₁P₂) scheme is assessed and validated to achieve the designed third order of accuracy for both internal and external smooth inviscid flow problems: one order of accuracy higher than the underlying DG(P₁) method, as presented in Reference [96]. In chapter 7, the WENO reconstruction at P₁ is only activated for flow problems with discontinuities by definition, e.g., Sod shock tube problem, and transonic flow over an M6 wing.
Chapter 5

Temporal Integration Methods

In this chapter, both the explicit and implicit time integration schemes for driving the flow field to steady state are introduced, while the emphasis is put for the latter. The outline of this chapter is organized in the following. The explicit multi-stage Runge-Kutta scheme is briefly introduced in section 5.1. A general procedure of the implicit time integration scheme is described in section 5.2. The specific methods employed in this work for solving a linear system of equations are presented in detail for the rest of this chapter.

5.1 Explicit Time Integration

The spatial discretization of the governing equations with the reconstruction-based discontinuous Galerkin method leads to a system of ordinary differential equations (ODEs) in time and Eq. 3.5 can be written in an elemental semi-discrete form as

$$M\frac{dU}{dt} = R(U)$$  \hspace{1cm} (5.1)

where $U = (U_1, U_2, \ldots, U_k, \ldots, U_{Nelem})^T$ is the global solution vector of $Ndegr \times Netot \times Nelem$ degrees of freedom to be evolved in time. Each $U_k$ itself is a vector, which represents solution of $Ndegr \times Netot$ degrees of freedom for the $k$-th element, for which the normalized
linear \((P_1)\) solution vector can be written as follows:

\[
\mathbf{U}_k = \begin{pmatrix}
\rho, & \frac{\partial \rho}{\partial x} \Delta x, & \frac{\partial \rho}{\partial y} \Delta y, & \frac{\partial \rho}{\partial z} \Delta z, \\
\rho u, & \frac{\partial \rho u}{\partial x} \Delta x, & \frac{\partial \rho u}{\partial y} \Delta y, & \frac{\partial \rho u}{\partial z} \Delta z, \\
\rho v, & \frac{\partial \rho v}{\partial x} \Delta x, & \frac{\partial \rho v}{\partial y} \Delta y, & \frac{\partial \rho v}{\partial z} \Delta z, \\
\rho w, & \frac{\partial \rho w}{\partial x} \Delta x, & \frac{\partial \rho w}{\partial y} \Delta y, & \frac{\partial \rho w}{\partial z} \Delta z,
\end{pmatrix}
\tag{5.2}
\]

Likewise, \(\mathbf{R} = (\mathbf{R}_1, \mathbf{R}_2, \ldots, \mathbf{R}_k, \ldots, \mathbf{R}_{N_{elem}})^T\) represents the global residual vector, where \(\mathbf{R}_k\) is the elemental residual vector for the \(k\)-th element, and approaches zero for a steady-state solution. Since the shape functions \(\mathbf{B}_i\) are nonzero within the \(k\)-th element only, the global mass matrix \(\mathbf{M}\) has a block-diagonal structure that couples the \(N_{degr} \times N_{tot}\) degrees of freedom of each component of the solution vector only within \(\Omega_k\). The \(k\)-th elemental mass matrix for the linear DG\((P_1)\) method can be written as follows

\[
\mathbf{M}_k = \begin{pmatrix}
\int_{\Omega_k} \tilde{B}_1 \tilde{B}_1 \, d\Omega & \int_{\Omega_k} \tilde{B}_1 \tilde{B}_2 \, d\Omega & \int_{\Omega_k} \tilde{B}_1 \tilde{B}_3 \, d\Omega & \int_{\Omega_k} \tilde{B}_1 \tilde{B}_4 \, d\Omega \\
\int_{\Omega_k} \tilde{B}_2 \tilde{B}_1 \, d\Omega & \int_{\Omega_k} \tilde{B}_2 \tilde{B}_2 \, d\Omega & \int_{\Omega_k} \tilde{B}_2 \tilde{B}_3 \, d\Omega & \int_{\Omega_k} \tilde{B}_2 \tilde{B}_4 \, d\Omega \\
\int_{\Omega_k} \tilde{B}_3 \tilde{B}_1 \, d\Omega & \int_{\Omega_k} \tilde{B}_3 \tilde{B}_2 \, d\Omega & \int_{\Omega_k} \tilde{B}_3 \tilde{B}_3 \, d\Omega & \int_{\Omega_k} \tilde{B}_3 \tilde{B}_4 \, d\Omega \\
\int_{\Omega_k} \tilde{B}_4 \tilde{B}_1 \, d\Omega & \int_{\Omega_k} \tilde{B}_4 \tilde{B}_2 \, d\Omega & \int_{\Omega_k} \tilde{B}_4 \tilde{B}_3 \, d\Omega & \int_{\Omega_k} \tilde{B}_4 \tilde{B}_4 \, d\Omega
\end{pmatrix}
\tag{5.3}
\]

Due to its symmetry and Eq. 3.12, only the volume of the element and other six components \(\int_{\Omega_k} \tilde{B}_i \tilde{B}_j \, d\Omega \) \((j \geq i)\) need to be actually pre-computed and stored, as shown in the following equation

\[
\mathbf{M}_k = \begin{pmatrix}
V_k & 0 & 0 & 0 \\
0 & \int_{\Omega_k} \tilde{B}_2 \tilde{B}_2 \, d\Omega & \int_{\Omega_k} \tilde{B}_2 \tilde{B}_3 \, d\Omega & \int_{\Omega_k} \tilde{B}_2 \tilde{B}_4 \, d\Omega \\
0 & \int_{\Omega_k} \tilde{B}_3 \tilde{B}_2 \, d\Omega & \int_{\Omega_k} \tilde{B}_3 \tilde{B}_3 \, d\Omega & \int_{\Omega_k} \tilde{B}_3 \tilde{B}_4 \, d\Omega \\
0 & \int_{\Omega_k} \tilde{B}_4 \tilde{B}_2 \, d\Omega & \int_{\Omega_k} \tilde{B}_4 \tilde{B}_3 \, d\Omega & \int_{\Omega_k} \tilde{B}_4 \tilde{B}_4 \, d\Omega
\end{pmatrix}
\tag{5.4}
\]

The following explicit three-stage third-order TVD Runge-Kutta scheme [32, 33]

\[
\mathbf{U}^{(1)} = \mathbf{U}^n + \Delta t \mathbf{M}^{-1} \mathbf{R} \left( \mathbf{U}^n \right)
\]

\[
\mathbf{U}^{(2)} = \frac{3}{4} \mathbf{U}^n + \frac{3}{4} \left[ \mathbf{U}^{(1)} + \Delta t \mathbf{M}^{-1} \mathbf{R} \left( \mathbf{U}^{(1)} \right) \right]
\tag{5.5}
\]

\[
\mathbf{U}^{(n+1)} = \frac{1}{3} \mathbf{U}^n + \frac{2}{3} \left[ \mathbf{U}^{(2)} + \Delta t \mathbf{M}^{-1} \mathbf{R} \left( \mathbf{U}^{(2)} \right) \right]
\]
is widely used to advance the solution in time. This method linearly stable for a CFL number less than or equal to $1/(2p + 1)$.

### 5.2 Implicit Time Integration

If we choose to integrate the spatially discretized governing equations implicitly in time, we can apply the backward Euler scheme to Eq. 5.1

$$M\frac{(U^{n+1} - U^n)}{\Delta t} = R(U^{n+1})$$  \hspace{1cm} (5.6)

Since the governing equations are nonlinear, Eq. 5.6 is a system of nonlinear equations for the global solution vector $U^{n+1}$. In order to solve this type of equations, we can linearize the right-hand-side vector $R$ with respect to the global solution vector $U$ at the current timestep

$$R(U^{n+1}) \approx R(U^n) + \left(\frac{\partial R}{\partial U}\right)^n (U^{n+1} - U^n)$$  \hspace{1cm} (5.7)

where $\left(\frac{\partial R}{\partial U}\right)^n$ is the so-called Jacobian matrix of the system evaluated at timestep $n$, and denoted symbolically as $J(U^n)$, which involves the linearization of both the inviscid and viscous flux functions.

If we plug Eq. 5.7 into Eq. 5.6 and move $\left(\frac{\partial R}{\partial U}\right)^n (U^{n+1} - U^n)$ to the left side, it then leads to a delta form of the equations

$$A \Delta U^n = \left(M \frac{\Delta t}{M} - \left(\frac{\partial R}{\partial U}\right)^n\right) \Delta U^n = R(U^n)$$  \hspace{1cm} (5.8)

where $\Delta t$ is the time increment and $\Delta U^n = U^{n+1} - U^n$ is the difference of global solution vector between time level $n$ and $n + 1$. Note that if $\Delta t$ tends to infinity, the scheme reduces to the standard Newtons method with a property of quadratic convergence for solving a system of nonlinear equations.

In our framework, a linear system of equations can be solved iteratively by using either the Symmetric Gauss-Seidel (SGS) method, or the Generalized Minimal Residual (GMRES) method developed by Saad and Schulz [114] with LU-SGS preconditioning [79, 93], namely GMRES+LU-SGS. Since GMRES is among the most popular and efficient implicit algorithms in CFD [32, 17, 109, 58, 49, 84, 86, 82, 85, 19, 79, 93, 139, 16, 40], we stick to using the GMRES+LU-SGS scheme throughout the numerical test cases if implicit time advancement is required.
5.3 Solution of the Linear System of Equations

The linear system of equations $A \Delta U^n = R$ is at first left-multiplied by an inverse of the preconditioning matrix $P$

$$ P^{-1} A \Delta U^n = P^{-1} R (U^n) $$

(5.9)

where the matrix $P$ consists of the strict upper $U$, lower $L$ and diagonal $D$ matrices

$$ P = (D + L) D^{-1} (D + U) $$

(5.10)

where $L$ and $U$ are stored via an face-based data structure. In order to construct an elemental Jacobian matrix for cell $i$, the contributions from $U$, $L$ and $D$ in face integrals are computed as below

$$ U = \int_{\Gamma_{ij}} \frac{\partial H_{\text{inv}} (U_i, U_j, n_{ij})}{\partial U_j} B_{d,ij} d\Gamma - \int_{\Gamma_{ij}} \frac{\partial H_{\text{vis}} (U_i, U_j, n_{ij})}{\partial U_j} B_{d,ij} d\Gamma $$

(5.11)

$$ L = \int_{\Gamma_{ij}} - \frac{\partial H_{\text{inv}} (U_i, U_j, n_{ij})}{\partial U_i} B_{d,ij} d\Gamma + \int_{\Gamma_{ij}} \frac{\partial H_{\text{vis}} (U_i, U_j, n_{ij})}{\partial U_i} B_{d,ij} d\Gamma $$

(5.12)

$$ D_{\Gamma} = \sum_{i<j}^{i<j} \left( \int_{\Gamma_{ij}} \frac{\partial H_{\text{inv}} (U_i, U_j, n_{ij})}{\partial U_i} B_{d,ij} d\Gamma - \int_{\Gamma_{ij}} \frac{\partial H_{\text{vis}} (U_i, U_j, n_{ij})}{\partial U_i} B_{d,ij} d\Gamma \right) $$

(5.13)

The contribution to $D$ from domain integrals is

$$ D_\Omega = - \int_{\Omega_i} \frac{\partial F_k (U_i)}{\partial U_i} \frac{\partial B_{d,i}}{\partial x_k} d\Omega + \int_{\Omega_i} \frac{\partial G_k (U_i)}{\partial U_i} \frac{\partial B_{d,i}}{\partial x_k} d\Omega $$

(5.14)

In Eq. 5.11, Eq. 5.12, Eq. 5.13 and Eq. 5.14, the subscript $d$ denotes the index of DOFs, $1 \leq d \leq N\text{degr}$; $i$ and $j$ denote cell $i$ and its adjacent face-neighboring cell $j$, respectively. Finally, the time derivative term $\frac{M}{\Delta t}$ is added to the $D$, and the elemental block diagonal matrix is as below

$$ D_i = \frac{M_i}{\Delta t} + D_{\Gamma_i} + D_{\Omega_i} = \frac{M_i}{\Delta t} - J_i $$

(5.15)

in which the crucial part is the assembly of the Jacobian matrix $J_i$, and we will devote an effort to it in §5.4. The global block diagonal matrix requires a storage of $Nelem \times (N\text{degr} \times Netot)^2$ units. Both the upper and lower matrices require a storage of $Nafac \times (N\text{degr} \times Netot)^2$ units, where $Nafac$ is the number of faces.

GMRES is one of the most robust Krylov methods, but it is also highly expensive in terms of storage, because all the computed vectors in the orthogonal sequence need to be retained. In order to alleviate the memory requirements, a restarted sub-version denoted as GMRES($m$)
is used, where after a user specified amount of \( m \) iterations, the stored data is cleared and the intermediate data is used as the initial solution for the next \( m \) iterations. There is no definite rules for the choice of \( m \). Usually, we choose \( m = 10 \) as a rule of thumb. The structure of the GMRES(\( m \)) algorithm is presented in Table 5.1 to solve the linear system of equations within each timestep. The GMRES algorithm can be partly matrix-free, for a finite difference approach

Table 5.1: Flowchart of the GMRES algorithm

| DO \( l = 1, m \) | \( m \) restarted iterations |
| \( v_0 = R - AU_0 \) | initial residual |
| \( r_0 := P^{-1}v_0 \) | preconditioning step |
| \( \beta := ||r_0||_2 \) | initial residual norm |
| \( v_1 := r_0/\beta \) | define initial Krylov |
| DO \( j = 1, k \) | inner iteration |
| \( y_j := \partial R(U)/\partial U \Delta U \) | matrix-vector product |
| \( w_j = P^{-1}y_j \) | preconditioning step |
| DO \( i = 1, j \) | Gram-Schmidt step |
| \( h_{i,j} = (w_j, v_i) \) | ... |
| \( w_j = w_j - h_{i,j}v_i \) | ... |
| END DO | |
| \( h_{j+1,j} := ||w_j||_2 \) | define Krylov vector |
| \( v_{j+1} := w_j/h_{j+1,j} \) | |
| END DO | |
| \( z := \min \beta e - H\hat{z} ||_2 \) | solve least squares |
| \( \Delta U := \Delta U_0 + \sum_{i=1}^m v_i z_i \) | approximate solution |
| IF \( ||\beta e - H\hat{z}||_2 < \epsilon \) EXIT | convergence check |
| \( \Delta U_0 := \Delta U \) | restart |
| END DO | |

can be used to compute the matrix-vector products instead of calculating and storing the full matrix as shown below

\[
\frac{\partial R(U)}{\partial U} \Delta U \approx \frac{R(U + \epsilon \cdot \Delta U) - R(U)}{\epsilon}
\] (5.16)

where the residual vector \( R(U + \epsilon \cdot \Delta U) \) of the governing equations is computed with the state variable vector \( U \) added by a set of perturbed state quantities. The scalar parameter \( \epsilon \) can be computed according to the formula proposed by Pernice and Walker [103]

\[
\epsilon = \epsilon \sqrt{1 + ||U||_2^2} / ||\Delta U||_2^2
\] (5.17)
where the parameter $\epsilon$ is a user-defined input and the choice of this parameter is between the round-off and truncation error. In our applications, $\epsilon$ is used for all the computations without problem. The GMRES algorithm requires one flux evaluation per time step and one flux evaluation per inner GMRES iteration. The primary storage is dictated by the LU-SGS preconditioning, which requires the upper, lower and diagonal matrices to be stored for every non-zero element in the approximate left-hand-side matrix in Eq. 5.8. The additional storage associated to the GMRES algorithm is an array of size $(k + 2) \times Nelem \times (Ndegr \times Netot)$, where $k$ is the number of search directions. Since the GMRES algorithm is completely separated from the RHS computation procedure, the memory which is used to compute the fluxes can also be shared by the procedure of solving the linear system.

### 5.4 Jacobian Matrix

To formulate the Jacobian matrix, the exact linearization of flux functions can be considered. But the process can be quite complicated due to the fact that it involves the linearization of the Hermite-WENO reconstruction, which is highly nonlinear in nature and practically not accessible in a closed form, and the computing time and storage requirement would be prohibitively demanding. Thus, approximate Jacobians are usually formulated instead of the exact ones for the implicit finite volume / discontinuous Galerkin methods.

In order to make a balance between the speedup and extra computational cost by implementing an implicit algorithm, the Jacobian matrix is evaluated based on the linear ($P_1$) linearization $J_{P_1}$ of the second-order RHS operator $R_{P_1}$, instead of the reconstructed third-order RHS operator $R_{P_1P_2}$. Eq. 5.8 can be rewritten in the form of approximate linear system as follows

$$\left( \frac{M}{\Delta t} - J_{P_1} \right) \Delta U^n = R_{P_1P_2}$$

Eq. 5.18

Due to the inexact representation of the LHS matrix used in Eq. 5.18, the quadratic convergence of the Newtons method can no longer be achieved. On the other side, the advantage of doing so lies in the fact that the implicit RDG($P_1P_2$) method only requires a $P_1$-like linear system with significantly reduced memory requirement and computing time, while the scheme still remains to be third-order accurate in space.

How to obtain the linearization (or differentiation) of a set of complicated functions, e.g., upwind numerical flux functions, is not trivial work. In general, the following four approaches are widely adopted in a variety of applications:

- **differentiation by hand**
  Manual implementation of analytic derivative formulae typically results in very efficient derivative code. However, the implementation is tedious and error-prone.
• **symbolic differentiation**
  Computer algebra packages manipulate expressions by repeatedly applying the chain rule so that there is no truncation error. However, the resulting expression for the derivative involves the parameters with respect to which one is differentiating. This can lead to an excessive growth of the length of the expression.

• **numerical differentiation**
  Divided differencing (DD) is based on some truncation of the Taylor series. It is easy to implement by evaluating the underlying function using perturbations of the input parameters. However, a suitable perturbation is often hard to find because a small perturbation decreasing the truncation error will increase the cancellation error.

• **automatic differentiation**
  Automatic differentiation (AD), also called algorithmic differentiation, is a technology for automatically augmenting computer programs, including arbitrarily complex simulations, with statements for the computation of derivatives, also known as sensitivities.

### 5.4.1 Differentiation by Hand

**Inviscid Flux**

For the linearization of inviscid flux functions, we use an implicit HLLC flux scheme with the frozen acoustic wave-speed originally introduced by Batten et al. [22] for the finite volume methods, which demonstrated an uncompromised speed of convergence and robustness for smooth flows. In this work, we have extended this scheme into the discontinuous Galerkin space.

First, the explicit form of the HLLC flux evaluated at the integration point of face $ij$ is defined by

$$H_{\text{HLLC}}(U_l, U_r, n_{ij}) = \begin{cases} 
H_l(U_l) & \text{if } S_L > 0 \\
H(U_l^*) & \text{if } S_L \leq 0 < S_M \\
H(U_r^*) & \text{if } S_M \leq 0 \leq S_R \\
H_r(U_r) & \text{if } S_R < 0 
\end{cases} \quad (5.19)$$

where the subscript $l$ and $r$ denote the elemental state vectors $U_i$ from cell $i$ and $U_j$ from cell $j$ evaluated at the face integration point respectively as follows

$$U_l = \begin{pmatrix} 
\rho_l \\
(\rho u)_l \\
(\rho v)_l \\
(\rho w)_l \\
(\rho e)_l 
\end{pmatrix} = \sum_{id} M_{\text{deg}} \begin{pmatrix} 
\tilde{B}_{id,l,i} U_{id,1,i} \\
\tilde{B}_{id,l,i} U_{id,2,i} \\
\tilde{B}_{id,l,i} U_{id,3,i} \\
\tilde{B}_{id,l,i} U_{id,4,i} \\
\tilde{B}_{id,l,i} U_{id,5,i} 
\end{pmatrix}, \quad U_r = \begin{pmatrix} 
\rho_r \\
(\rho u)_r \\
(\rho v)_r \\
(\rho w)_r \\
(\rho e)_r 
\end{pmatrix} = \sum_{id} M_{\text{deg}} \begin{pmatrix} 
\tilde{B}_{id,r,j} U_{id,1,j} \\
\tilde{B}_{id,r,j} U_{id,2,j} \\
\tilde{B}_{id,r,j} U_{id,3,j} \\
\tilde{B}_{id,r,j} U_{id,4,j} \\
\tilde{B}_{id,r,j} U_{id,5,j} 
\end{pmatrix} \quad (5.20)$$
where $M_{degr}$ is the number of DOFs for the polynomials ($M_{degr} = 1$ for RDG(P0P0), $M_{degr} = 4$ for RDG(P0P1) and RDG(P1P1), $M_{degr} = 10$ for RDG(P1P2) and RDG(P2P2)).

In Eq. 5.19, the symbols with superscript * are defined by

$$ U_i^* = \begin{pmatrix} \rho_i^* \\ (\rho u)_i^* \\ (\rho v)_i^* \\ (\rho w)_i^* \\ (\rho e)_i^* \end{pmatrix} = \Omega_i \begin{pmatrix} \rho_l (S_L - q_l) \\ (S_L - q_l)(\rho u)_l + (p^* - p_l)n_x \\ (S_L - q_l)(\rho v)_l + (p^* - p_l)n_y \\ (S_L - q_l)(\rho w)_l + (p^* - p_l)n_z \\ (S_L - q_l)(\rho e)_l - p_lq_l + p^* S_M \end{pmatrix} \tag{5.21} $$

$$ U_r^* = \begin{pmatrix} \rho_r^* \\ (\rho u)_r^* \\ (\rho v)_r^* \\ (\rho w)_r^* \\ (\rho e)_r^* \end{pmatrix} = \Omega_r \begin{pmatrix} \rho_r (S_R - q_r) \\ (S_R - q_r)(\rho u)_r + (p^* - p_r)n_x \\ (S_R - q_r)(\rho v)_r + (p^* - p_r)n_y \\ (S_R - q_r)(\rho w)_r + (p^* - p_r)n_z \\ (S_R - q_r)(\rho e)_r - p_rq_r + p^* S_M \end{pmatrix} \tag{5.22} $$

$$ H_i^* \equiv H(U_i^*) = \begin{pmatrix} \rho_l^* S_M \\ (\rho u)_l^* S_M + p^* n_x \\ (\rho v)_l^* S_M + p^* n_y \\ (\rho w)_l^* S_M + p^* n_z \\ ((\rho e)_l^* + p^*) S_M \end{pmatrix}, \quad H_r^* \equiv H(U_r^*) = \begin{pmatrix} \rho_r^* S_M \\ (\rho u)_r^* S_M + p^* n_x \\ (\rho v)_r^* S_M + p^* n_y \\ (\rho w)_r^* S_M + p^* n_z \\ ((\rho e)_r^* + p^*) S_M \end{pmatrix} \tag{5.23} $$

$$ \Omega_l \equiv (S_L - S_M)^{-1}, \quad \Omega_r \equiv (S_R - S_M)^{-1}, \tag{5.24} $$

$$ p^* = \rho_l (q_l - S_L)(q_l - S_M) + p_l = \rho_r (q_r - S_R)(q_r - S_M) + p_r \tag{5.25} $$

$$ q_l \equiv u_l n_x + v_l n_y + w_l n_z, \quad q_r \equiv u_r n_x + v_r n_y + w_r n_z \tag{5.26} $$

with $(n_x, n_y, n_z)^T$ being the unit vector normal to face $ij$. $S_M$ is taken from Batten et al. [21]:

$$ S_M = \frac{\rho_r q_r (S_R - q_r) - \rho_l q_l (S_L - q_l) + p_l - p_r}{\rho_r (S_R - q_r) - \rho_l (S_L - q_l)} \tag{5.27} $$

and $S_L, S_R$ are taken from Einfeldt et al. [47]:

$$ S_L = \min \left[ \lambda_1(U_l), \lambda_1(U^{\text{Roe}}) \right], \quad S_R = \max \left[ \lambda_m(U^{\text{Roe}}), \lambda_m(U_r) \right] \tag{5.28} $$

where $\lambda_1(U^{\text{Roe}})$ and $\lambda_m(U^{\text{Roe}})$ are the smallest and largest eigenvalues of the Roe matrix [112].

53
The implicit form (differentiation) of the HLLC flux functions in Eq. 5.19 are given by

\[
\mathbf{H}_{\text{HLLC}}^{n+1}(\mathbf{U}_l, \mathbf{U}_r, \mathbf{n}_{ij}) = \begin{cases} 
\mathbf{H}_l^n + \frac{\partial \mathbf{H}_l}{\partial \mathbf{U}_l} \Delta \mathbf{U}_l & \text{if } S_L > 0 \\
(\mathbf{H}_l^n)^n + \frac{\partial \mathbf{H}_l}{\partial \mathbf{U}_l} \Delta \mathbf{U}_l + \frac{\partial \mathbf{H}_l^*}{\partial \mathbf{U}_r} \Delta \mathbf{U}_r & \text{if } S_L \leq 0 < S_M \\
(\mathbf{H}_r^n)^n + \frac{\partial \mathbf{H}_r}{\partial \mathbf{U}_l} \Delta \mathbf{U}_l + \frac{\partial \mathbf{H}_r^*}{\partial \mathbf{U}_r} \Delta \mathbf{U}_r & \text{if } S_M \leq 0 \leq S_R \\
\mathbf{H}_r^n + \frac{\partial \mathbf{H}_r}{\partial \mathbf{U}_r} \Delta \mathbf{U}_r & \text{if } S_R < 0 
\end{cases} 
\tag{5.29}
\]

where for the supersonic case, \( \frac{\partial \mathbf{H}_l}{\partial \mathbf{U}_l} \) and \( \frac{\partial \mathbf{H}_r}{\partial \mathbf{U}_r} \) are linearization of the original flux functions, i.e., \( \frac{\partial \mathbf{H}}{\partial \mathbf{U}} = \frac{\partial}{\partial \mathbf{U}} \left( \mathbf{F}_x n_x + \mathbf{F}_y n_y + \mathbf{F}_z n_z \right) \) and can be computed according to Appendix C. For the subsonic case, the HLLC Jacobian matrices are given by

\[
\frac{\partial \mathbf{H}_l^*}{\partial \mathbf{U}_l} = \begin{pmatrix} 
\left( \frac{\partial \rho}{\partial \mathbf{U}_l} \right)^T S_M + \left( \frac{\partial \rho}{\partial \mathbf{U}_l} \right)^T \rho_l^* \\
\left( \frac{\partial \rho}{\partial \mathbf{U}_l} \right)^T S_M + \left( \frac{\partial \rho}{\partial \mathbf{U}_l} \right)^T \rho_l \\
\left( \frac{\partial \rho}{\partial \mathbf{U}_l} \right)^T S_M + \left( \frac{\partial \rho}{\partial \mathbf{U}_l} \right)^T \rho_l \\
\left( \frac{\partial \rho}{\partial \mathbf{U}_l} \right)^T S_M + \left( \frac{\partial \rho}{\partial \mathbf{U}_l} \right)^T \rho_l 
\end{pmatrix}
\tag{5.30}
\]

and

\[
\frac{\partial \mathbf{H}_r^*}{\partial \mathbf{U}_r} = \begin{pmatrix} 
\left( \frac{\partial \rho}{\partial \mathbf{U}_r} \right)^T S_M + \left( \frac{\partial \rho}{\partial \mathbf{U}_r} \right)^T \rho_l \\
\left( \frac{\partial \rho}{\partial \mathbf{U}_r} \right)^T S_M + \left( \frac{\partial \rho}{\partial \mathbf{U}_r} \right)^T \rho_l \\
\left( \frac{\partial \rho}{\partial \mathbf{U}_r} \right)^T S_M + \left( \frac{\partial \rho}{\partial \mathbf{U}_r} \right)^T \rho_l \\
\left( \frac{\partial \rho}{\partial \mathbf{U}_r} \right)^T S_M + \left( \frac{\partial \rho}{\partial \mathbf{U}_r} \right)^T \rho_l 
\end{pmatrix}
\tag{5.31}
\]

In Eq. 5.30 and Eq. 5.31, the vectors \( \frac{\partial S_M}{\partial \mathbf{U}_l}, \frac{\partial S_M}{\partial \mathbf{U}_r}, \frac{\partial p^*}{\partial \mathbf{U}_l}, \) and \( \frac{\partial p^*}{\partial \mathbf{U}_r} \) can be
computed approximately. The $S_M$ derivatives are computed as follows

\[
\frac{\partial S_M}{\partial U_l} = \tilde{\rho}^{-1} \begin{pmatrix}
-q_l^2 + \psi_l(\gamma - 1)/2 + S_MS_L \\
n_x(2q_l - S_L - S_M) - (\gamma - 1)u_l \\
n_y(2q_l - S_L - S_M) - (\gamma - 1)v_l \\
n_z(2q_l - S_L - S_M) - (\gamma - 1)w_l \\
\gamma - 1
\end{pmatrix}
\tag{5.32}
\]

\[
\frac{\partial S_M}{\partial U_r} = \tilde{\rho}^{-1} \begin{pmatrix}
-q_r^2 - \psi_r(\gamma - 1)/2 - S_RS_R \\
n_x(-2q_r + S_R + S_M) + (\gamma - 1)u_r \\
n_y(-2q_r + S_R + S_M) + (\gamma - 1)v_r \\
n_z(-2q_r + S_R + S_M) + (\gamma - 1)w_r \\
-(\gamma - 1)
\end{pmatrix}
\tag{5.33}
\]

where $\psi \equiv u^2 + v^2 + w^2$ and $\tilde{\rho} \equiv \rho_r(S_R - q_r) - \rho_l(S_L - q_l)$. Taking differentiation of Eq. 5.25 with respect to $U_l$ and $U_r$ respectively gives

\[
\frac{\partial p^*}{\partial U_l} = \rho_r(S_R - q_r) \frac{\partial S_M}{\partial U_l}, \quad \frac{\partial p^*}{\partial U_r} = \rho_l(S_L - q_l) \frac{\partial S_M}{\partial U_r}
\tag{5.34}
\]

The remaining terms in Eq. 5.30 and Eq. 5.31 are given as follows

\[
\frac{\partial \rho^*_l}{\partial U_l} = \Omega_l \begin{pmatrix}
S_L \\
-n_x \\
n_x \\
n_z
\end{pmatrix} + \Omega_l \rho_l \frac{\partial S_M}{\partial U_l}
\tag{5.35}
\]

\[
\frac{\partial \rho^*_r}{\partial U_r} = \Omega_l \rho_l \frac{\partial S_M}{\partial U_r}
\tag{5.36}
\]

\[
\frac{\partial (\rho u)_l^*}{\partial U_l} = \Omega_l \begin{pmatrix}
q_u l - n_x \psi_l (\gamma - 1)/2 \\
S_L - q_l + n_x (\gamma - 2)u_l \\
u_l n_y + n_x (\gamma - 1)v_l \\
u_l n_z + n_x (\gamma - 1)w_l \\
-(\gamma - 1)n_x
\end{pmatrix} + \Omega_l \left( n_x \frac{\partial p^*_l}{\partial U_l} + (\rho u)_l^* \frac{\partial S_M}{\partial U_l} \right)
\tag{5.37}
\]

\[
\frac{\partial (\rho u)_r^*}{\partial U_r} = \Omega_l \left( n_x \frac{\partial p^*_l}{\partial U_r} + (\rho u)_l^* \frac{\partial S_M}{\partial U_r} \right)
\tag{5.38}
\]
\[
\frac{\partial (\rho v)^*_l}{\partial U_l} = \Omega_l \left( q_l v_l - n_y \psi_l (\gamma - 1)/2 \right) + \Omega_l \left( n_y \frac{\partial p^*}{\partial U_l} + (\rho v)^*_l \frac{\partial S_M}{\partial U_l} \right) \tag{5.39}
\]

\[
\frac{\partial (\rho w)^*_l}{\partial U_l} = \Omega_l \left( n_y \frac{\partial p^*}{\partial U_l} + (\rho w)^*_l \frac{\partial S_M}{\partial U_l} \right) \tag{5.40}
\]

\[
\frac{\partial (\rho w)^*_r}{\partial U_r} = \Omega_l \left( n_z \frac{\partial p^*}{\partial U_l} + (\rho w)^*_l \frac{\partial S_M}{\partial U_l} \right) \tag{5.41}
\]

\[
\frac{\partial (\rho e)^*_l}{\partial U_l} = \Omega_l \left( ((\rho e)_l + p_l) q_l / \rho_l - q_l \psi_l (\gamma - 1)/2 \right) + \Omega_l \left( \frac{\partial p^*}{\partial U_l} S_M + (p^* + (\rho e)^*_l) \frac{\partial S_M}{\partial U_l} \right) \tag{5.42}
\]

\[
\frac{\partial (\rho e)^*_r}{\partial U_r} = \Omega_l \left( \frac{\partial p^*}{\partial U_r} S_M + (p^* + (\rho e)^*_l) \frac{\partial S_M}{\partial U_l} \right) \tag{5.43}
\]

\[
\frac{\partial (\rho e)_l}{\partial U_l} = \Omega_l \left( \frac{\partial p^*}{\partial U_l} S_M + (p^* + (\rho e)_l) \frac{\partial S_M}{\partial U_l} \right) \tag{5.44}
\]

In the case where \( S_M < 0 \), the relevant HLLC Jacobian matrices are obtained by simply interchanging subscript \( l \leftrightarrow r \) and \( L \leftrightarrow R \) in Eq. 5.35 through Eq. 5.44. This completes the definition of the frozen acoustic wavespeed version of the implicit HLLC flux. In addition, it was also found by Batten et al. [22] that very little speedup could be further achieved by computing the fully linearized implicit HLLC flux, due to the fact that the extra work required to compute the Jacobians including the differentiations of the acoustic wavespeed does not significantly favor this version over the approximated form.

Instead of evaluating the Jacobians based on the flux scheme in the RHS operator which is usually quite complicated, a much more simplified approach is also available for differentiation by hand. A fast matrix-free implicit finite volume method for compressible flows on unstructured grids was introduced by Luo et al. [79, 93], in which the Jacobian matrix is computed based on a simple spectral-type flux function, i.e., the Lax-Friedrichs scheme instead of the one used in the RHS operator. It was proved to be highly cost-effective in the finite volume context by carrying out numerical test cases on complex geometric configurations [79, 93]. This approach is named
as the unmatching Jacobians (UJ) in this work, which is to distinguish from the RHS flux-based matched Jacobians (MJ), i.e., HLLC Jacobians. We have also extended this approach into the discontinuous Galerkin space to construct the elemental Jacobian matrix. The contributions from upper matrix $U$, lower matrix $L$ and diagonal matrix $D$ in face integrals are computed as below:

$$U = \int_{\Gamma_{ij}} \frac{1}{2} \left( \frac{\partial F_k (U_{ij}) n_k B_{d,ij}}{\partial U_j} - |\lambda_{lr}| I_{Netot \times Netot} B_{d,ij} \right) d\Gamma \quad (5.45)$$

$$L = \int_{\Gamma_{ij}} -\frac{1}{2} \left( \frac{\partial F_k (U_{ij}) n_k B_{d,ij}}{\partial U_i} + |\lambda_{lr}| I_{Netot \times Netot} B_{d,ij} \right) d\Gamma \quad (5.46)$$

$$D_{\Gamma} = \sum_{i<j} \left\{ \int_{\Gamma_{ij}} \frac{1}{2} \left( \frac{\partial F_k (U_{ij}) n_k B_{d,ij}}{\partial U_i} + |\lambda_{lr}| I_{Netot \times Netot} B_{d,ij} \right) d\Gamma \right\} \quad (5.47)$$

where the spectral radii $|\lambda_{lr}|$ on the face integration point is given by

$$|\lambda_{lr}| = \frac{1}{2} \left( |(V_l + V_r) \cdot n_{ij}| + a_t + a_r \right) \quad (5.48)$$

where $a$ denotes the local speed of sound.

A matrix-free approach [79, 93] is used along with the unmatching Jacobians (UJ) for the lower (forward) and upper (backward) sweep of preconditioning, where the product of Jacobian matrix and incremental state variable vector is approximated by the increment of flux vector. The major advantage of doing so is that no static memory is required for storing the lower and upper matrices at all, which results in a significant reduction in memory requirement. Rasetinarera and Hussaini [109] extended this matrix-free implicit scheme into the discontinuous Galerkin space for compressible inviscid flows on 2D structured grids. Nevertheless, it still remains to be explored whether the unmatching Jacobians are able to achieve equivalent or better performance than the matched Jacobians for compressible flows in 3D. Comparison of these two types of Jacobians evaluation is carried out in §7.

**Viscous Flux**

In contrast to the various successful schemes to treat the inviscid flux, an optimal scheme to discretize the viscous flux is still under quest for the discontinuous Galerkin methods. Among the most remarkable are the first Bassi-Rebay (BR1) scheme that was introduced by Bassi and Rebay [14] in 1997 and the famous second Bassi-Rebay (BR2) scheme [16] that was introduced in 1999. The BR2 scheme was designed to overcome several deficiencies of the BR1 scheme, and received a wide acceptance in the DG community. In Bassi and Rebay’s 2005 publication [19], the Reynolds-Averaged Navier-Stokes (RANS) equations and $k$-$\varepsilon$ turbulence model were first discretized in the discontinuous Galerkin space and integrated implicitly in time with the
analytically derived approximate Jacobians.

To apply the implicit BR2 scheme, the implicit form of the viscous flux evaluated at the integration point of face $ij$ is defined by

$$
H_{BR2}^{n+1}(U_l, \nabla U_l + r_l, U_r, \nabla U_r + r_r)
$$

$$
= H_{BR2}^n(U_l, \nabla U_l + r_l, U_r, \nabla U_r + r_r)
$$

$$
+ \frac{\partial H_{BR2}}{\partial U_l} \Delta U_l + \frac{\partial H_{BR2}}{\partial U_l} \Delta U_r
$$

$$
+ \frac{\partial H_{BR2}}{\partial (\nabla U_l + r_l)} \frac{\partial \nabla U_l}{\partial U_l} \Delta U_l + \frac{\partial H_{BR2}}{\partial (\nabla U_r + r_r)} \frac{\partial \nabla U_r}{\partial U_r} \Delta U_r
$$

$$
+ \frac{\partial H_{BR2}}{\partial (\nabla U_l + r_l)} \frac{\partial \nabla r_l}{\partial U_l} \Delta U_l + \frac{\partial H_{BR2}}{\partial (\nabla U_r + r_r)} \frac{\partial \nabla r_r}{\partial U_r} \Delta U_r
$$

(5.49)

where $r$ is the so-called local lifting operator for interior faces, and defined by

$$
\int_{\Omega_i} r_l B_i \, d\Omega = \int_{\Gamma_{ij}} \frac{1}{2} (U_r - U_l) n_k B_i \, d\Gamma
$$

(5.50)

On the boundary faces the local lifting operator is defined by

$$
\int_{\Omega_i} r_l B_i \, d\Omega = \int_{\Gamma_{ij}} (U_b - U_l) n_k B_i \, d\Gamma
$$

(5.51)

The locality means that the integration is evaluated on face $ij$ only.

The domain integral evaluated at the integration point of cell $i$ is defined by

$$
G_{BR2}^{n+1}(U_l, \nabla U_l + R_l) = G_{BR2}^n(U_l, \nabla U_l + R_l)
$$

$$
+ \frac{\partial G_{BR2}}{\partial U_l} \Delta U_l + \frac{\partial G_{BR2}}{\partial (\nabla U_l + R_l)} \frac{\partial \nabla U_l}{\partial U_l} \Delta U_l + \frac{\partial G_{BR2}}{\partial (\nabla U_l + R_l)} \frac{\partial \nabla R_l}{\partial U_l} \Delta U_l
$$

(5.52)

where $R$ is the so-called global lifting operator for elements, and defined by

$$
\int_{\Omega_i} R_l B_i \, d\Omega = \int_{\partial \Omega_i} \frac{1}{2} (U_r - U_l) n_k B_i \, d\Gamma
$$

(5.53)

The relation between the local and global lifting operators is given by

$$
R = \sum_{\Gamma_{ij}} r
$$

(5.54)

The BR2 scheme was successfully implemented in various studies in the DG community.
and many remarkable Navier-Stokes solvers based on have been developed recently. In 2006, Klaij et.al [68] introduced a space-time discontinuous Galerkin method for the compressible Navier-Stokes equations, in which the procedures to assemble the approximate Jacobians of the BR2 viscous flux function were described in detail. In 2008, Landmann [72] developed a parallel discontinuous Galerkin code for the Navier-Stokes and RANS equations on 2D structured/unstructured grids, in which differentiation of the BR2 viscous flux function were carried out with the aid of the algebraic package Maple. In 2010, Yasue et.al [144] developed an implicit discontinuous Galerkin method for RANS simulation, in which the inviscid flux was linearized using a spectral method while the linearization of BR2 viscous flux was accurately approximated. More recently, Crivellini et.al [40] introduced an implicit matrix-free discontinuous Galerkin solver, in which a matrix-free GMRES algorithm was extended into the discontinuous Galerkin space for viscous and turbulent aerodynamic simulations. All these works discussed above indicate that the key to a robust and efficient implicit DG Navier-Stokes solver is the quality of the linearized viscous terms. On the other side, a common disadvantage of these solvers is that they are required to devote large amount of CPU time to computing the viscous Jacobian matrix for each or every several Newton iterations, as the complexity of computation increases cubically with the order of the considered DG methods.

5.4.2 Divided Differencing

A well-accepted and widely-used approach to evaluating the elemental Jacobian matrix for implicit finite volume methods are the so-called divided differencing (DD) method, for example, the one-sided differencing of this type

\[
\frac{\partial R_I(U_i)}{\partial U_{iJ}} = \frac{R_I(U_i + \varepsilon \cdot e_J) - R_I(U_i)}{\varepsilon}, \quad \varepsilon \in \mathbb{R}
\]  

(5.55)

where \(R_I\) is the \(I\)-th component of the elemental residual vector \(R(U_i)\) for cell \(i\), \(e_J\) is the \(J\)-th unit vector associated with the variable component \(U_{iJ}\) of the elemental state vector \(U_i\) \((1 \leq I, J \leq Netot\) for finite volume discretization, \(1 \leq I, J \leq (Ndegr \times Netot)\) for discontinuous Galerkin discretization), and \(\varepsilon\) is the perturbation parameter. In Eq. 5.55, the Jacobian matrix is computed in a column-wise manner: with a perturbation \(\varepsilon \cdot e_J\), a column of the Jacobian matrix components \(J_{I,J}(U_i)\) \((1 \leq I \leq (Ndegr \times Netot))\) are obtained via \(R_I(U_i + \varepsilon \cdot e_J)\) as shown below

\[
J(U_i) = \begin{pmatrix}
J_{1,1} & \cdots & J_{1,J} & \cdots & J_{1,Ndegr \times Netot} \\
J_{2,1} & \cdots & J_{2,J} & \cdots & J_{2,Ndegr \times Netot} \\
\vdots & \vdots & \vdots & \vdots & \vdots \\
J_{Ndegr \times Netot,1} & \cdots & J_{Ndegr \times Netot,J} & \cdots & J_{Ndegr \times Netot,Ndegr \times Netot}
\end{pmatrix}
\]
Thus for a standard implicit DG($P_n$) method, it requires $N_{degr} \times N_{tot}$ times of $R_I(U_i + \varepsilon \cdot e_J)$ evaluation ($N_{degr}$ is 1 for $P_0$, 4 for $P_1$ and 10 for $P_2$) to assemble an elemental Jacobian matrix.

So here the question is how many calls of RHS evaluation routine is required to obtain a global Jacobian matrix? Obviously the way is not to compute the elemental Jacobian matrix one by another, which requires $N_{elem} \times N_{degr} \times N_{tot}$ calls of the RHS routine in total, which is too expensive. Alternatively, a grouping algorithm is employed to split the elements into a certain number of groups, $N_{group}$, where no two element shares a common interface in each group. The range of effect of an elemental perturbation $(U_i + \varepsilon \cdot e_J)$ is only within cell $i$ itself, but not propagating to any other cells when computing $\partial R_I(U_i)/\partial U_{i,j}$ within each group. Therefore only $N_{group} \times N_{degr} \times N_{tot}$ calls of the RHS routine are required to assemble the global Jacobian matrix. As one can see, the smaller $N_{group}$ is, the less computing time the divided differencing approach takes. In general, the value of $N_{group}$ depends on how the elements are numbered in a grid. A smallest number of $N_{group}$ is usually the result of an element-renumbering algorithm, which is reported in Reference [77].

As discussed above, the divided differencing approach can be naturally extended to the implicit RDG methods, whose advantage lies in the fact that an arbitrary complicated numerical flux function can be differentiated in a manner of black box, by multiple times of evaluation of the residual vector. Furthermore, this approach is easy and quick to implement in an CFD code, and does not need additional maintenance if the numerical flux function is updated or changed. These are probably the two major reasons for the wide acceptance of the divided difference approach in formulating the Jacobian matrix in implicit finite volume methods [28, 27].

However, one should also realize that the divided differencing approach does have two primary disadvantages. For the first, the quality of approximation depends on the choice of $\varepsilon$, which is always unknown a priori. Theoretically, the value of $\varepsilon$ should be small enough to minimize the truncation error that results from truncating the polynomial solution, while it should also be large enough to avoid the cancellation errors using finite precision arithmetic when evaluating the numerator in Eq. 5.55. The second disadvantage is the high computational cost, especially for the higher-order DG methods. For example, the DG($P_1$) method has 4 times of the number of DOFs of the finite volume methods ($4 \times N_{tot}$ versus $N_{tot}$ for an element), so it will cost the DG($P_1$) method at least 16 times of the computing time for assembling an elemental Jacobian matrix than the finite volume methods, not to mention the extra time spent on loops over the integration points. Thus, the computational costs of the Jacobian evaluation would grow quadratically for higher-order DG methods if we stick to the divided differencing approach. Even though, we have been using this approach in the RDG methods only as a benchmark to assess and validate the other approaches of Jacobian formulation. For the one-sided approximation, a simple rule of thumb is to choose $\varepsilon$ about the square root of machine precision. Other approaches for the choice of $\varepsilon$ take into account of the dependence of the state vector $U$ [103].
In our test cases, the value of $\varepsilon = 1.0 \times 10^{-7}$ is assumed for all the computations without any problems. However, the choice of perturbation parameter $\varepsilon$ for the divided differencing approach is case-dependent and a particular source of uncertainty during computation, if the flow field in solving exhibits physical phenomena of multiple scales. A proper perturbation parameter may also depend on the Reynolds number. Above all, for the high-order DG methods, the limitations of huge computational costs for the divided differencing approach make it acceptable only within the interest of academic research. In order to facilitate high performance computation, more effective approaches of Jacobian evaluation are desired.

5.4.3 Automatic Differentiation

Automatic Differentiation (AD) is a set of techniques based on the mechanical application of the chain rule to obtain derivatives of a function given as a computer program. AD exploits the fact that every computer program, no matter how complicated, executes a sequence of elementary arithmetic operations such as additions or elementary functions such as $\exp()$. By applying the chain rule of derivative calculus repeatedly to these operations, derivatives of arbitrary order can be computed automatically, and accurate to working precision. Conceptually, AD is different from symbolic differentiation and approximations by divided differencing.

Automatic differentiation is used in the following areas:

- Numerical Methods
- Sensitivity Analysis
- Design Optimization
- Data Assimilation & Inverse Problems

Just like divided differencing, AD requires only the source program $C$. But instead of executing $C$ on different sets of inputs, it builds a new, augmented code $C$, that computes the analytical derivatives along with the source program. This new program is called the differentiated program. Precisely speaking, each time the source program holds some value $v$, the differentiated program holds an additional value $dv$, which is the differential of $v$. In principle, arbitrarily complex functions can be differentiated. The largest application to date is a 1.6 million line FEM code written in Fortran 77 [26]. An incomplete list of AD tools can be found at Community Portal for Automatic Differentiation [2].

There are usually two modes in an AD tool: forward mode and reverse mode. The derivatives of dependent with respect to independent variables computed by these modes are mathematically equivalent, but the time and memory requirements of computing them may differ. The
forward mode computes derivatives of intermediate variables with respect to independent vari-
ables and propagates from one statement to the next according to the chain rule. The reverse
mode computes derivatives of dependent variables with respect to intermediate variables and
propagates from one statement to the previous statement according to the chain
rule. Thus, the reverse mode requires a reversal of the program execution. The forward mode is
sometimes referred to as direct or tangent linear mode whereas the reverse mode is also called
backward, adjoint or cotangent linear mode.

Implementations of AD can be broadly classified into two categories. AD tools based on
operator overloading exploit the fact that modern programming languages offer the possibility
to redefine the semantics of elementary operators. AD tools based on source-to-source transfor-
mation change the semantics by explicitly rewriting the code. Each of these approaches has its
advantages and disadvantages [25].

In this work, we have applied automatic differentiation to evaluating the flux Jacobian
matrix along with differentiation by hand and divided differencing. TAPENADE is a source-
to-source AD tool [3]. Given a FORTRAN77, FORTRAN95, or C source program, it generates
its derivative in forward (tangent) or reverse (adjoint) mode. TAPENADE is directly accessible
through a web server, or can be downloaded locally, between which it is preferred in our work
to obtaining the source code returned by the web server. Although there is no algorithmic
difference between the source codes from the web server and a locally installed package, there
are some advantages in sticking to the web server. First, one does not need to install, and
second, one doesn't need to re-install to get updates of the tool.

A simple hand-made example is presented to demonstrate how TAPENADE web server gen-
erates the differential of a function. First, a FORTRAN77 file test.f which contains the routine
func as shown in Table 5.2 is uploaded as a source file. Multiple source files or header files can
also be provided. Then, the name of the top routine func, the dependent output variable array
p_y, and the independent input variable array p_x are specified, and the differentiation mode
Tangent Multidirectional Mode is chosen.

Table 5.2: A simple source program

<table>
<thead>
<tr>
<th>!... This routine demonstrates a simple algebraic function</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUBROUTINE FUNC(p_x, p_y)</td>
</tr>
<tr>
<td>IMPLICIT NONE</td>
</tr>
<tr>
<td>REAL*8 p_x(2), p_y(2)</td>
</tr>
<tr>
<td>p_y(1) = 2.0<em>p_x(1) + 3.0</em>p_x(2)</td>
</tr>
<tr>
<td>p_y(2) = 4.0*p_x(1)*p_x(2)</td>
</tr>
<tr>
<td>RETURN</td>
</tr>
<tr>
<td>END</td>
</tr>
</tbody>
</table>
TAPENADE will generate a file `test_dv.f` which contains the differentiation routine `func_dv` as shown in Table 5.3. There are two more inputs `p_xd(nbdirsmax, 2)` and `nbdirs`, and one more output `p_yd(nbdirsmax, 2)` for the new routine. The `nbdirsmax` is a user-defined integer in `DIFFSIZES.inc` as a header file. Like all the other AD tools, TAPENADE does not directly provide the routine for Jacobian matrix evaluation. Instead, it can provide the Jacobian-vector products based on user’s inputs. In order to obtain the Jacobian matrix, `Tangent Multidirectional Mode` serves the purpose of obtaining the whole Jacobian matrix, or arbitrary columns in the matrix. The variable `nbdirs` is the number of directions that the user want to do loops over the Jacobian-vector products process. It is set equal to the subscript size of the vector `p_x`. If `p_xd` is set equal to the identity matrix, then the computed matrix `p_yd` is just the transpose of the Jacobian matrix. The user can transpose `p_yd` back to obtain the Jacobian matrix.

Table 5.3: A simple differential source program

```fortran
!... Differentiation of func in forward (tangent) mode: (multi-directional mode)
!... variations of useful results: p_y
!... with respect to varying inputs: p_x
!... RW status of diff variables: p_x:in p_y:out
SUBROUTINE FUNC_DV(p_x, p_xd, p_y, p_yd, nbdirs)
IMPLICIT NONE
INCLUDE 'DIFFSIZES.inc'
!... Hint: nbdirsmax should be the maximum number of differentiation directions
REAL*8 p_x(2), p_y(2)
REAL*8 p_xd(nbdirsmax, 2), p_yd(nbdirsmax, 2)
INTEGER nd
INTEGER nbdirs
INTEGER ii1
DO nd = 1, nbdirs
  DO ii1 = 1, 2
    p_yd(nd, ii1) = 0.0
  ENDDO
  p_yd(nd, 1) = 2.0*p_xd(nd, 1) + 3.0*p_xd(nd, 2)
  p_yd(nd, 2) = 4.0*(p_xd(nd, 1)*p_x(2)+p_x(1)*p_xd(nd, 2))
ENDDO
p_y(1) = 2.0*p_x(1) + 3.0*p_x(2)
p_y(2) = 4.0*p_x(1)*p_x(2)
RETURN
END
```

Recall from §5.3, a face-based data structure is used to store the lower matrix `L` and the upper matrix `U`, and we still need to preserve this structure when the flux Jacobian matrix is evaluated by the AD tool. In order to make the least code modification, only the manually derived routines of Jacobian matrix evaluation are replaced with the corresponding AD generated source routines. Table 5.4 presents a series of examplary procedures to implement TAPENADE automatic differentiation on the source routine `rhsbp1`, whose functionality is to compute the
contribution of integrals to the left and right elemental RHS vectors across an interior face. The AD generated source code $rhsbp1_dv$ is then integrated in RDGFLO to compute the elemental Jacobian matrices for the left and right cells, and also the corresponding lower and upper matrices, as outlined in Table 5.5.

The Jacobian matrix is computed once per time step and used for either preconditioning in the restarted GMRES solver or SGS iterations. In this work, we normally use the manually derived Jacobians for the Euler equations, and use the AD generated Jacobians for the Navier-Stokes equations, as they turned out to be most efficient for the inviscid and viscous flow problems respectively. A comparison among the manually derived Jacobians, divided differencing (DD) numerical Jacobians and automatic differentiation (AD) generated Jacobians will be given in the results chapter.

Table 5.4: Example: implementation of TAPENADE on RDGFLO

<table>
<thead>
<tr>
<th>Procedure</th>
<th>User’s inputs or choice</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Select the input language</td>
<td>Fortran 95</td>
<td>RDGFLO is written in Fortran95</td>
</tr>
<tr>
<td>Upload source code</td>
<td>top.F</td>
<td>It contains the top routine to be differentiated</td>
</tr>
<tr>
<td>Name of the top routine (given)</td>
<td>rhsbp1_</td>
<td>Integrals of RHS over a face</td>
</tr>
<tr>
<td>Dependent output variables for the given routine</td>
<td>p_rzl, p_rzr</td>
<td>Local elemental RHS vectors for left and right cells across a face</td>
</tr>
<tr>
<td>Independent input variables for the given routine</td>
<td>p_unl, p_unr</td>
<td>Local elemental state vectors for left and right cells across a face</td>
</tr>
<tr>
<td>Differentiate in Tangent Multidirectional Mode</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Generated source code (returned)</td>
<td>rhsbp1_dv_</td>
<td>Differentials of RHS over a face</td>
</tr>
<tr>
<td>Dependent output variables for the returned routine</td>
<td>p_rzl, p_rzr, p_rzld, p_rzrd</td>
<td></td>
</tr>
<tr>
<td>Independent input variables for the returned routine</td>
<td>p_unl, p_unr, p_unld, p_unrd, nbdirs</td>
<td></td>
</tr>
</tbody>
</table>
Table 5.5: Pseudo code: contribution of face integrals for the block diagonal matrix $D$, lower matrix $L$ and upper matrix $U$

```
subroutine adlhsbounp1(... , unkno, diago, lower, upper)
  implicit none
  ...
  real*8, dimension(Mdegr, Netot, Nsize) : unkno
  real*8, dimension(Ndofe, Ndofe, Nelem) : diago
  real*8, dimension(Ndofe, Ndofe, Nafac) : lower, upper
  ! local input arrays
  real*8, dimension(Ndofe) : p_unl, p_unr, p_rzl, p_rzr
  real*8, dimension(Ndofe,Ndofe) : p_unld, p_unrd, p_rzld, p_rzrd
  !
  ! loop over the interior faces
  do ifa = Nbfac+1, Nafac
    !
    ! conservative variables at the left & right cells
    p_unl(1 : Ndofe) := unkno(1 : Ndegr, 1 : Netot, iel)
    p_unr(1 : Ndofe) := unkno(1 : Ndegr, 1 : Netot, ier)
    ! initialize the local RHS vectors
    p_rzl(1 : Ndofe) := 0.0 ; p_rzr(1 : Ndofe) := 0.0
    !
    ! initialize the identity matrix & number of directions for diago at left cell and lower matrix
    p_unld = 0.0 ; p_unrd = 0.0
    do ii = 1, Ndofe ; p_unld(ii, ii) = 1.0 ; enddo
    nbdirs = Ndofe
    ! differentiation of the local face integral
    call rhsbp1_dv(... , p_unl, p_unld, p_unr, p_unrd, p_rzl, p_rzld, p_rzr, p_rzrd, nbdirs)
    ! transpose and get the Jacobian matrix at left cell, scatter to diago & lower
    if(iel .le. nelem) diago(:, :, iel) = diago(:, :, iel) + transpose(p_rzld(:, :))
    lower(:, :, ifa) = lower(:, :, ifa) - transpose(p_rzrd(:, :))
    !
    ! initialize the identity matrix & number of directions for diago at right cell and upper matrix
    p_unld = 0.0
    do ii = 1, Ndofe ; p_unrd(ii, ii) = 1.0 ; enddo
    ! differentiation of the local face integral
    call rhsbp1_dv(... , p_unl, p_unld, p_unr, p_unrd, p_rzl, p_rzld, p_rzr, p_rzrd, nbdirs)
    ! transpose and get the Jacobian matrix at right cell, scatter to diago & upper
    if(ier .le. nelem) diago(:, :, ier) = diago(:, :, ier) - transpose(p_rzld(:, :))
    upper(:, :, ifa) = upper(:, :, ifa) + transpose(p_rzrd(:, :))
    !
    ! end of loop over the interior faces
    enddo
  ! return
end
```
Chapter 6

Parallelization

As stated in chapter 1, the compactness of the DG and also RDG methods makes them ideally suited for the implementation of parallel computing on regular clusters and supercomputers. In order to guarantee the maximal portability of the code and flexibility for parallel programming. The RDGFLO framework utilizes the Message Passing Interface (MPI) library for parallel communication between user threads. The parallel strategy is designed for architectures with distributed memory. The outline of this chapter is organized in the following. The grid partitioning is briefly introduced in section 6.1. The data structures are then described in section 6.2. Finally, the parallel communication is discussed in section 6.3.

6.1 Grid Partitioning

Grid partitioning is usually the first task we should carry out before the parallel code can read in a series of partitioned grids. The original grid generated by some grid generation software is called the global domain in our context. The global domain is then decomposed into local domains (subdomains) and each processor is allocated the workload for at least one local domains. Domain decomposition into a user-specified number of local domains of high quality is a very complex task. Thus the RDGFLO code employs the open-source software package METIS [66] for domain decomposition in the pre-processing stage. METIS is a set of serial programs for partitioning graphs, partitioning finite element meshes, and producing fill reducing orderings for sparse matrices. The algorithms implemented in METIS are based on the multilevel recursive-bisection, multilevel k-way, and multi-constraint partitioning schemes developed in the Kyrypis Lab at the Department of Computer Science and Engineering at the University of Minnesota in the Twin Cities of Minneapolis. Two examples are shown in Figure 6.1 and Figure 6.2. The first is a hybrid prismatic/hexahedral grid with 1,536 elements for flow around a circular cylinder is split into 16 local domains. The second is a tetrahedral grid with 328,370...
elements for flow over a wing/ pyl on/finned-store configuration.

6.2 Data Structures

The data structures for parallel communication between different threads are organized in three categories: (1) elemental geometric information, (2) elemental solution vector and (3) statistical indicators. At the beginning of a parallel computation, each local domain ASCII file is read in by one thread and thanks to METIS, the total element number per thread is as equal as possible. This distribution is maintained throughout the whole computation with a static load balancing. In order to implement the first two categories, we need to create the so-called partition boundary ghost elements for the boundary faces of the local domains, which physically connect to other local domains if mapped into the graph of the global domain. The partition boundary ghost elements for a particular local domain resemble the ones from its partitioning adjacent local domains as they should have the same geometric information like the elemental mass matrix coefficients, coordinates of the cell centers, and normalization factors($\Delta x, \Delta y, \Delta z$), and the same solution vectors during the advancement in time. The geometric information for the partition boundary ghost elements is exchanged at initialization of the computation and
Figure 6.2: Graph of domain decomposition by METIS for a tetrahedral grid (only surface shown) for flow past a wing/pylon/finned-store configuration, 64 partitions

does not need synchronization since the grid is kept static under the current development. The statistical indicators mean the global residual vectors, global $L^2$ norm and some other suggestive indices dumped every user-specified number of time steps. These indicative figures are computed locally and finally collected and reported by one thread (usually Rank 0). An exchange object is built which possesses a send and receive buffer that contain the data to be exchanged between threads. The buffers are the consecutive partition boundary face-based degrees of freedom which are projected from the elemental degrees of freedom. Also keep in mind that the relative orientation of the corresponding partition boundary faces should be correct.

6.3 MPI Communication

The communication in RDGFLO in parallel mode is managed by the described exchange object above, which utilizes nothing but the necessary standard MPI commands like nonblocking send, nonblocking receive and wait commands. Parallelization is implemented for both the explicit and implicit RDG methods.

First we introduce the communication for explicit methods. During the reconstruction stage,
the ghost solution vectors need to be first updated by the exchange object after the second
derivatives are obtained via least-squares reconstruction. While all the threads are at work,
a synchronization checkpoint is put in place to ensure that each single thread proceeds to a
next section that requires the updated ghost solution vectors at the partition ghost elements. A
second update of ghost solution vectors is performed when the second derivatives are modified
after the HWENO reconstruction. Then the boundary and domain integrals are computed. Note
that the boundary integrals are evaluated twice across a partition face on both threads that
share this face. Alternatively, we can also choose to exchange fluxes in order get them evaluated
only on one thread. But this would rather increase the latency per time step. In practice, we
adopt to evaluate the partition boundary fluxes twice. At the end of a Runge-Kutta stage, the
ghost solution vectors are also updated.

For the implicit RDG methods, the communication in the reconstruction stage is untouched.
Two options are available for solving the linear system of equations. First, if the SGS solver
is employed, the ghost solution vectors need only to be updated at the end of each time step.
Second, if the GMRES+LU-SGS solver is used, the ghost vectors with respect to the temporary
vectors which store the solution of the linear system of equations are required to be updated
at the end of each Newton iteration. Note that, either the SGS iteration or the LU-SGS pre-
conditioning is carried out locally, so the latency of converging speed in terms of time step is
proportional to the number of threads. Again, the ghost solution vectors are updated at the
end of every time step.
Chapter 7

Numerical Results

The content of this chapter is split into numerical experiments of inviscid flow (§7.1) and viscous flow (§7.2) problems, and scaling tests of parallel performance (§7.3). For all the test cases, computation is carried out on a Dell Precision T7400 personal workstation computer (2.98 GHz Xeon CPU with 18 GBytes memory) using openSUSE 12.2 Linux operating system with the Intel FORTRAN + MPICH2 compiler. For the test of parallel performance, computation is conducted on a Root Cluster for Research into Scalable Computer Systems (ARC) running the Red Hat 4.1.2-51 Linux operating system at the Department of Computer Science, North Carolina State University, with the code compiled using the PGI FORTRAN + OpenMPI package. More details of the cluster is described in §7.3.

The following $L^2$ norm of the entropy production is used as the error measurement for the steady-state inviscid flow problems

$$\|\varepsilon\|_{L^2(\Omega)} = \sqrt{\int_{\Omega} \varepsilon^2 \, d\Omega} = \sqrt{\sum_{i=1}^{N_{\text{elem}}} \int_{\Omega_i} \varepsilon^2 \, d\Omega}$$

where the entropy production $\varepsilon$ is defined as

$$\varepsilon = \frac{S - S_\infty}{S_\infty} = \frac{p}{p_\infty} \left( \frac{\rho_\infty}{\rho} \right)^\gamma - 1$$

Note that the entropy production, where the entropy is defined as $S = (p/\rho)^\gamma$, is a very good criterion to measure accuracy of the numerical solutions, since the flow under consideration is isentropic.

The order of accuracy for spatial discretization of a scheme can be assessed in a piecewise manner between two successively refined grids with a characteristic grid size $h$ and $h/2$ respectively, and computed as such a slope: the logarithmic difference of two $L^2$ norms over the
difference of their corresponding logarithmic cell sizes expressed in the following

\[ m = \left| \frac{\log_{10} \varepsilon_h - \log_{10} \varepsilon_{h/2}}{\log_{10} h - \log_{10} h/2} \right| = \left| \frac{\log_{10} \varepsilon_h/\varepsilon_{h/2}}{\log_{10} 2} \right| \]

where \( m \) is taken from the denotation of the RDG(P\(_n\)P\(_m\)) scheme with a designed \( O(h^{m+1}) \) order of accuracy. The actual order of accuracy for a specific problem can be obtained by computing on a series of globally successively refined grids and taking an arithmetic averaged value of these piecewise slopes. The RDG(P\(_1\)P\(_1\)) and RDG(P\(_1\)P\(_2\)) methods are therefore expected to deliver the \( O(h^2) \) and \( O(h^3) \) order of accuracy, respectively.

Furthermore, a definition of the characteristic grid size \( h \) is required in order to plot a figure of piecewise slopes with respect to \( h \) for the purpose of demonstration, although it is not required to compute the slope itself, as one can see from the above equation. \( h \) can be defined as the actual cell sizes of a series of globally successively grids. Alternatively, \( h \) can be computed in terms of total number of degree of freedom for a grid as below

\[ h = \frac{1}{N_{\text{dimn}} \sqrt{N_{\text{degr}} \times N_{\text{elem}}}} \]

where the index of root \( N_{\text{dimn}} \) is the dimension of a problem type: \( N_{\text{dimn}} = 1 \) for 1D-type problems where grids are refined in one direction only i.e., \( x \)-direction and \( N_{\text{elem}} \) grows linearly; \( N_{\text{dimn}} = 2 \) for 2D-type problems where grids are refined in a plane i.e., \( x-y \) plane and \( N_{\text{elem}} \) grows quadratically; \( N_{\text{dimn}} = 3 \) for 3D-type problems where grids are refined in 3D space and \( N_{\text{elem}} \) grows cubically.

The converging speed (called convergence rate as well) for the implicit GMRES+LU-SGS scheme is assessed for both inviscid and viscous steady-state problems, where the baseline results for comparison are obtained by the explicit three-stage TVD Runge-Kutta time stepping scheme. This explicit scheme is also used for computing unsteady flows, i.e., Sod shock tube problem. To plot a flow variable on the surface of the solid body in 2D demonstration, its value at two end points of a face on the solid body are drawn using a line. This is the most accurate way to represent the P\(_1\) solution for profile plotting, as the solution is linear on each face and multiple values exist for a vertex due to the discontinuous representation of DG solution.

### 7.1 Inviscid Cases

In this section, a variety of numerical experiments of compressible inviscid flow problems computed by the RDG methods are presented. Firstly, the implementation of the Riemann solver is validated with the test case of Sod shock tube problem. Secondly, a series of grid convergence
analysis on both external and internal flow problems are performed in order to verify the order of accuracy for the inviscid RDG code. Thirdly, the stability and robustness of the RDG methods is demonstrated at the presence of weak and strong discontinuities. The fourth test case is to assess the performance of the RDG methods for a transonic flow over a complex geometrical configuration with strong shock waves. Finally, a transonic flow over a complete Boeing 747 aircraft is presented to assess and verify our RDG methods for computing complicated flows of practical importance.

### 7.1.1 Sod Shock Tube

The shock tube problem constitutes a particularly interesting and difficult test case, since it presents an exact solution to the full system of one-dimensional Euler equations containing simultaneously a shock wave, a contact discontinuity, and expansion fan. This test case is chosen to validate the implementation of the HLLC Riemann solver and demonstrate the robustness of the RDG methods. The initial conditions in the present computation are the following:

\[
\begin{align*}
\rho_L &= 1.000, & u_L &= 0, & p_L &= 1.0 & \text{for} & 0.0 \leq x \leq 0.5, \\
\rho_R &= 0.125, & u_R &= 0, & p_R &= 0.1 & \text{for} & 0.5 \leq x \leq 1.0.
\end{align*}
\]

This is a 3D simulation of the 1D problem. Figure 7.1 shows the hexahedral grid used in computation. The grid consists of 50 cells in the \(x\)-direction, 1 cell in \(y\)-direction and 1 cell in \(z\)-direction. Figure 7.2, Figure 7.3 and Figure 7.4 show a comparison of the density, Mach number, and pressure profiles obtained by the unlimited RDG(\(P_1P_1\)), WENO(\(P_1P_2\)), HWENO(\(P_1P_2\)), and the exact solutions, respectively. As expected, the unlimited RDG(\(P_1P_1\)) and WENO(\(P_1P_2\)) solutions exhibit small oscillations in the vicinity of discontinuities and yield a sharp resolution for both contact and discontinuity and shock wave. In comparison, one can observe that the use of HWENO(\(P_1P_2\)) reconstruction is able to eliminate the spurious oscillations in the shock wave region. In the rest region of domain, the HWENO(\(P_1P_2\)) solutions are not as good as those obtained by the underlying RDG(\(P_1P_1\)) solutions, indicating that the HWENO reconstruction does increase slightly the absolute error, although it can keep the designed order of accuracy of the original DG method as shown in the next section. Most importantly, the reconstruction scheme used in the DG method does not create negative density, and thus does not lead to the breakdown of the solution process. This can be attributed to the robustness of the underlying RDG(\(P_1P_1\)) method upon which a reconstruction is based.
Figure 7.1: The quasi-1D hexahedral grid for the Sod shock tube problem.

Figure 7.2: Comparison of computed density profiles for the Sod shock tube problem obtained by the RDG(P₁P₁), WENO(P₁P₂), and HWENO(P₁P₂) solutions with the analytical solution.
Figure 7.3: Comparison of computed Mach number profiles for the Sod shock tube problem obtained by the RDG(P₁P₁), WENO(P₁P₂), HWENO(P₁P₂) solutions with the analytical solution.

Figure 7.4: Comparison of computed pressure profiles for the Sod shock tube problem obtained by the RDG(P₁P₁), WENO(P₁P₂), and HWENO(P₁P₂) solutions with the analytical solution.
7.1.2 Subsonic Flow past a Circular Cylinder

This is a well-known test case: inviscid subsonic flow past a circular cylinder at a Mach number of $M_{\infty} = 0.38$. This test case is chosen to verify if a formal order of the convergence rate of the RDG methods can be achieved for the compressible Euler equations on arbitrary grids. This is a 3D simulation of the 2D problem. Three types of grids: hexahedral, prismatic and prismatic/hexahedral hybrid grids are tested respectively.

Figure 7.5 shows the four successively globally refined O-type hexahedral grids having $16 \times 4$, $32 \times 8$, $64 \times 16$, $128 \times 32$ cells in $x-y$ plane and 1 cell in $z-$direction. The radius of the cylinder is $r_1 = 0.5$, and the domain is bounded by $r_b = 20$. Numerical solutions to this problem are computed using the RDG(P₁P₁) and WENO(P₁P₂) methods on these four grids to obtain quantitative measurement of the discretization errors and order of accuracy. The detailed results for this test case are presented in Table 7.1. They show the grid size, the number of degrees of freedom, the $L^2$ error of the solutions, and the order of convergence. Figure 7.8, Figure 7.9 and Figure 7.10 show the computed density, pressure and Mach number contours in the flow field obtained by the RDG(P₁P₁) and WENO(P₁P₂) solutions on the four hexahedral grids, respectively. One can see that the results obtained by RDG(P₁P₂) are more accurate than the ones obtained by RDG(P₁P₁). Both the RDG(P₁P₁) and WENO(P₁P₂) solutions are virtually identical on the finest grid for this case. However, the WENO(P₁P₂) does yield a slightly more accurate solution than the RDG(P₁P₁) at the same grid resolution. This can be seen in Figure 7.11(a), providing the details of the spatial convergence of each method for this numerical experiment. As expected, the RDG(P₁P₁) method exhibits a full $O(h^2)$ order of convergence. The WENO(P₁P₂) method does offer a full $O(h^3)$ order of convergence, adding one order of accuracy to the underlying RDG(P₁P₁) method. Figure 7.11(b) illustrates that the higher order DG methods require significantly fewer degrees of freedom than the lower one for the same accuracy.

Next, computation is performed on the four successively globally refined O-type prismatic grids having $16 \times 8$, $32 \times 16$, $64 \times 32$, $128 \times 64$ cells in $x-y$ plane and 1 cell in $z-$direction, as shown in Figure 7.6. The radius of the cylinder and the outer bound of the domain are remained the same as the hexahedral grids. Numerical solutions are computed again using the RDG methods on these four grids to obtain quantitative measurement of the discretization errors and order of accuracy. The detailed results for this test case are presented in Table 7.2. Figure 7.12, Figure 7.13 and Figure 7.14 show the computed density, pressure and Mach number contours in the flow field obtained by the RDG(P₁P₁) and WENO(P₁P₂) solutions on the four prismatic grids, respectively. Once again, it can be observed that the results obtained by the WENO(P₁P₂) method are more accurate than the ones obtained by the RDG(P₁P₁) method. Both the RDG(P₁P₁) and WENO(P₁P₂) solutions are virtually identical on the finest grid.
whereas WENO(P1P2) yields a more accurate solution than RDG(P1P1) at the same grid resolution, which can be seen in Figure 7.15, providing the details of the spatial convergence of each method for this test case. The RDG(P1P1) method delivers a full $O(h^2)$ order of convergence and the WENO(P1P2) method does achieve a full $O(h^3)$ order of convergence which adds one order of accuracy to the underlying RDG(P1P1) method.

The last test in this numerical experiment is on the four successively globally refined prismatic/hexahedral hybrid grids, which have $(16 + 8) \times 4$, $(32 + 16) \times 8$, $(64 + 32) \times 16$ and $(128 + 64) \times 32$ cells in $x - y$ plane and 1 cell in $z$-direction, as shown in Figure 7.7. Numerical solutions are computed using the RDG methods on these four grids so as to obtain the order of accuracy. The detailed results are presented in Table 7.3. Figure 7.16, Figure 7.17 and Figure 7.18 demonstrate the computed density, pressure and Mach number contours in the flow field obtained by the RDG(P1P1) and WENO(P1P2) solutions on the four hybrid grids, respectively. It is evident to see that the results obtained by the WENO(P1P2) method exhibit more accurate contour distributions than the ones obtained by the RDG(P1P1) method on the grids of lower resolution. Although the RDG(P1P1) and WENO(P1P2) solutions are virtually identical on the finest grid, the WENO(P1P2) method does yield a more accurate solution than the RDG(P1P1) method if one compares the levels of discretization errors as illustrated in Figure 7.19. In conclusion of this test case, the WENO(P1P2) method has been verified being able to deliver the designed $O(h^3)$ order of accuracy of convergence on hybrid grids, which largely improves the accuracy of the underlying RDG(P1P1) method without significant extra computational cost and storage requirement.

Table 7.1: Discretization errors and convergence rates of the RDG methods obtained on the four successively globally refined hexahedral grids for an inviscid subsonic flow past a circular cylinder at $M_\infty = 0.38$.

<table>
<thead>
<tr>
<th>Grid</th>
<th>No. DOFs</th>
<th>$L^2$-error-P1P1</th>
<th>Order-P1P1</th>
<th>$L^2$-error-P1P2</th>
<th>Order-P1P2</th>
</tr>
</thead>
<tbody>
<tr>
<td>16 × 4</td>
<td>256</td>
<td>-0.86941E+00</td>
<td>-</td>
<td>-0.80491E+00</td>
<td>-</td>
</tr>
<tr>
<td>32 × 8</td>
<td>1,024</td>
<td>-0.13597E+01</td>
<td>1.629</td>
<td>-0.15768E+01</td>
<td>2.193</td>
</tr>
<tr>
<td>64 × 16</td>
<td>4,096</td>
<td>-0.20821E+01</td>
<td>2.400</td>
<td>-0.27377E+01</td>
<td>3.746</td>
</tr>
<tr>
<td>128 × 32</td>
<td>16,384</td>
<td>-0.28501E+01</td>
<td>2.551</td>
<td>-0.39997E+01</td>
<td>3.548</td>
</tr>
</tbody>
</table>
Figure 7.5: A sequence of four successively globally refined hexahedral grids for inviscid subsonic flow past a circular cylinder.

Figure 7.6: A sequence of four successively globally refined prismatic grids for inviscid subsonic flow past a circular cylinder.

Figure 7.7: A sequence of four successively globally refined prismatic/hexahedral hybrid grids for inviscid subsonic flow past a circular cylinder.
Figure 7.8: Computed density contours in the flow field by the RDG(P₁P₁) (a–d) and WENO(P₁P₂) (e–h) solutions on the four successively refined hexahedral grids for inviscid subsonic flow past a circular cylinder at $M_\infty = 0.38$.

Figure 7.9: Computed pressure contours in the flow field by the RDG(P₁P₁) (a–d) and WENO(P₁P₂) (e–h) solutions on the four successively refined hexahedral grids for inviscid subsonic flow past a circular cylinder at $M_\infty = 0.38$. 

78
Figure 7.10: Computed Mach number contours in the flow field by the RDG(P₁P₁) (a–d) and WENO(P₁P₂) (e–h) solutions on the four successively refined hexahedral grids for inviscid subsonic flow past a circular cylinder at $M_\infty = 0.38$.

Figure 7.11: $L^2$ errors of numerical solutions versus (a) cell size and (b) number of degree of freedom for inviscid subsonic flow past a circular cylinder by RDG(P₁P₁) and WENO(P₁P₂) on hexahedral grids.
Figure 7.12: Computed density contours in the flow field by the RDG($P_1P_1$) (a–d) and WENO($P_1P_2$) (e–h) solutions on the four successively refined prismatic grids for inviscid subsonic flow past a circular cylinder at $M_{\infty} = 0.38$.

Figure 7.13: Computed pressure contours in the flow field by the RDG($P_1P_1$) (a–d) and WENO($P_1P_2$) (e–h) solutions on the four successively refined prismatic grids for inviscid subsonic flow past a circular cylinder at $M_{\infty} = 0.38$. 

80
Figure 7.14: Computed Mach number contours in the flow field by the RDG(P₁P₁) (a–d) and WENO(P₁P₂) (e–h) solutions on the four successively refined prismatic grids for inviscid subsonic flow past a circular cylinder at $M_\infty = 0.38$.

Figure 7.15: $L^2$ errors of numerical solutions versus (a) cell size and (b) number of degree of freedom for inviscid subsonic flow past a circular cylinder by RDG(P₁P₁) and WENO(P₁P₂) on prismatic grids.
Figure 7.16: Computed density contours in the flow field by RDG(P1P1) (a–d) and WENO(P1P2) (e–h) solutions on the four successively globally refined prismatic/hexahedral hybrid grids for inviscid subsonic flow past a circular cylinder at $M_\infty = 0.38$.

Figure 7.17: Computed pressure contours in the flow field by RDG(P1P1) (a–d) and WENO(P1P2) (e–h) solutions on the four successively globally refined prismatic/hexahedral hybrid grids for inviscid subsonic flow past a circular cylinder at $M_\infty = 0.38$.  

82
Figure 7.18: Computed Mach number contours in the flow field by RDG(P1P1) (a–d) and WENO(P1P2) (e–h) solutions on the four successively globally refined prismatic/hexahedral hybrid grids for inviscid subsonic flow past a circular cylinder at $M_\infty = 0.38$.

Figure 7.19: $L^2$ errors of numerical solutions versus (a) cell size and (b) number of degree of freedom for inviscid subsonic flow past a circular cylinder by RDG(P1P1) and WENO(P1P2) on prismatic/hexahedral hybrid grids.
Table 7.2: Discretization errors and convergence rates of the RDG methods obtained on the four successively globally refined prismatic grids for an inviscid subsonic flow past a circular cylinder at $M_\infty = 0.38$.

<table>
<thead>
<tr>
<th>Grid</th>
<th>No. DOFs</th>
<th>$L^2$-error-P$_1$P$_1$</th>
<th>Order-P$_1$P$_1$</th>
<th>$L^2$-error-P$_1$P$_2$</th>
<th>Order-P$_1$P$_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>16 $\times$ 8</td>
<td>512</td>
<td>-0.12456E+01</td>
<td>-</td>
<td>-0.14009E+01</td>
<td>-</td>
</tr>
<tr>
<td>32 $\times$ 16</td>
<td>2,048</td>
<td>-0.19753E+01</td>
<td>2.424</td>
<td>-0.25497E+01</td>
<td>3.816</td>
</tr>
<tr>
<td>64 $\times$ 32</td>
<td>8,192</td>
<td>-0.28013E+01</td>
<td>2.744</td>
<td>-0.38685E+01</td>
<td>4.381</td>
</tr>
<tr>
<td>128 $\times$ 64</td>
<td>32,768</td>
<td>-0.36030E+01</td>
<td>2.663</td>
<td>-0.50007E+01</td>
<td>3.761</td>
</tr>
</tbody>
</table>

Table 7.3: Discretization errors and convergence rates of the RDG methods obtained on the four successively globally refined prismatic/hexahedral hybrid grids for an inviscid subsonic flow past a circular cylinder at $M_\infty = 0.38$.

<table>
<thead>
<tr>
<th>Grid</th>
<th>No. DOFs</th>
<th>$L^2$-error-P$_1$P$_1$</th>
<th>Order-P$_1$P$_1$</th>
<th>$L^2$-error-P$_1$P$_2$</th>
<th>Order-P$_1$P$_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>(16 + 8) $\times$ 4</td>
<td>384</td>
<td>-0.12907E+01</td>
<td>-</td>
<td>-0.12000E+01</td>
<td>-</td>
</tr>
<tr>
<td>(32 + 16) $\times$ 8</td>
<td>1,536</td>
<td>-0.19912E+01</td>
<td>2.326</td>
<td>-0.23306E+01</td>
<td>3.762</td>
</tr>
<tr>
<td>(64 + 32) $\times$ 16</td>
<td>6,144</td>
<td>-0.27710E+01</td>
<td>2.593</td>
<td>-0.36842E+01</td>
<td>4.500</td>
</tr>
<tr>
<td>(128 + 64) $\times$ 32</td>
<td>24,576</td>
<td>-0.35798E+01</td>
<td>2.697</td>
<td>-0.46498E+01</td>
<td>3.219</td>
</tr>
</tbody>
</table>

7.1.3 Subsonic Flow past a Sphere

In this numerical example, an inviscid subsonic flow past a sphere at a freestream Mach number of $M_\infty = 0.5$ is chosen to assess the performance of the RDG(P$_1$P$_1$) and WENO(P$_1$P$_2$) methods for external flows. The surface triangular meshes of a sequence of the four successively refined tetrahedral grids for computation are shown in Figure 7.21 respectively: the coarse grid (Level 1) shown in Figure 7.21a consists of 535 elements, 167 nodes and 244 boundary faces; the medium grid (Level 2) shown in Figure 7.21b consists of 2,426 elements, 589 nodes and 640 boundary faces; the fine grid (Level 3) shown in Figure 7.21c consists of 16,467 elements, 3,425 nodes and 2,372 boundary faces, and the finest grid (Level 4) shown in Figure 7.21d consists of 124,706 elements, 23,462 nodes and 9,072 boundary faces. The cell size is halved between consecutive grids. Note that only a quarter of the configuration is modeled due to symmetry of the problem. Computations are conducted on the first three grids (Level 1, 2 and 3) by using the RDG(P$_1$P$_1$) and WENO(P$_1$P$_2$) methods to obtain a quantitative measurement of the dis-
cretization errors and order of convergence, as shown in Table 7.4. In average, both RDG(P₁P₁) and WENO(P₁P₂) have achieved almost a formal order of accuracy of convergence, being 2.00 and 3.01, respectively, as Figure 7.20 shows the plot of the $L^2$ errors of numerical solutions versus cell size and DOFs respectively for the RDG methods, convincingly demonstrating the benefits of using the reconstructed DG method. Figure 7.22, Figure 7.23 and Figure 7.24 show the computed density, pressure and Mach number contours in the flow field obtained by RDG(P₁P₁) and WENO(P₁P₂) on the three grids, respectively. One can see that the results obtained by WENO(P₁P₂) are more accurate than the ones obtained by RDG(P₁P₁).

Table 7.4: Discretization errors and convergence rates of the RDG methods obtained on the three successively refined tetrahedral grids for an inviscid subsonic flow past a sphere at $M_\infty = 0.5$.

<table>
<thead>
<tr>
<th>Grid</th>
<th>No. DOFs</th>
<th>$L^2$-error-P₁P₁</th>
<th>Order-P₁P₁</th>
<th>$L^2$-error-P₁P₂</th>
<th>Order-P₁P₂</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level 1</td>
<td>2,140</td>
<td>-0.1732E+01</td>
<td>-</td>
<td>-0.196E+01</td>
<td>-</td>
</tr>
<tr>
<td>Level 2</td>
<td>9,704</td>
<td>-0.2302E+01</td>
<td>1.895</td>
<td>-0.284E+01</td>
<td>2.924</td>
</tr>
<tr>
<td>Level 3</td>
<td>65,868</td>
<td>-0.2933E+01</td>
<td>2.094</td>
<td>-0.377E+01</td>
<td>3.094</td>
</tr>
</tbody>
</table>

Figure 7.25a and Figure 7.25b show the WENO(P₁P₂) convergence history of logarithmic density residual with respect to timestep and CPU time on the grids Level 1, 2 and 3. In comparison, the implicit method is about four orders of magnitude faster than its explicit counterpart, indicating a significant saving in terms of computing time for external flow problems. A decreased converging speed with respect to timestep is observed on grids of higher level for the implicit WENO(P₁P₂) methods, due to the use of approximated Jacobian matrix. Computations are also conducted on the grid Level 4, but only for the implicit WENO(P₁P₂) methods. The speed of convergence for UJ-GMRES is over three times slower than MJ-GMRES as shown in Figure 7.26, indicating the limitation of the simplified, unmatching Jacobians used as the preconditioning matrix in implicit higher-order DG methods. Figure 7.27 shows the plot of the numerical error against the CPU time required for the WENO(P₁P₂) and RDG(P₁P₁) methods on the fine grid (Level 3) and the RDG(P₀P₁) method on the finest grid (Level 4). As can be clearly seen from Figure 7.27, the WENO(P₁P₂) method provides a significant saving (probably orders of magnitude when a high accuracy is requested) in CPU time in comparison with the second-order finite volume method (RDG(P₀P₁)), convincingly demonstrating the superior performance of the WENO(P₁P₂) method. Indeed, as indicated in Figure 7.27, the RDG(P₀P₁) method needs one order of magnitude more degrees of freedom, and consequently CPU time.
than the WENO(P\textsubscript{1}P\textsubscript{2}) method in order to achieve an error level below -3.

Figure 7.20: \(L^2\) errors of numerical solutions versus (a) cell size and (b) number of degree of freedom for inviscid subsonic flow past a sphere at \(M_\infty = 0.5\), by RDG(P\textsubscript{1}P\textsubscript{1}) and WENO(P\textsubscript{1}P\textsubscript{2}) on tetrahedral grids.

Figure 7.21: A sequence of four successively globally refined tetrahedral grids for inviscid subsonic flow past a sphere.
Figure 7.22: Computed density contours in the flow field by the RDG($P_1P_1$) (a–c) and WENO($P_1P_2$) (d–f) on the tetrahedral grids for inviscid subsonic flow past a sphere at $M_\infty = 0.5$.

Figure 7.23: Computed pressure contours in the flow field by the RDG($P_1P_1$) (a–c) and WENO($P_1P_2$) (d–f) on the tetrahedral grids for inviscid subsonic flow past a sphere at $M_\infty = 0.5$. 

87
Figure 7.24: Computed Mach number contours in the flow field by the RDG($P_1P_1$) (a–c) and WENO($P_1P_2$) (d–f) on the tetrahedral grids for inviscid subsonic flow past a sphere at $M_\infty = 0.5$.

Figure 7.25: Convergence history for logarithmic density residual with respect to (a) timestep and (b) CPU time respectively for inviscid subsonic flow past a sphere at $M_\infty = 0.5$ on tetrahedral grids (Level 1, 2 and 3), by WENO($P_1P_2$).
Figure 7.26: Convergence history for logarithmic density residual with respect to (a) timestep and (b) CPU time respectively for inviscid subsonic flow past a sphere at $M_\infty = 0.5$ on tetrahedral grids (Level 4), by WENO(P$_1$P$_2$).

Figure 7.27: Numerical errors versus CPU time for different reconstructed RDG schemes.
### 7.1.4 Subsonic Flow through a Channel with a Smooth Bump

In this numerical example, an inviscid subsonic flow through a channel with a smooth bump on the lower surface at a freestream Mach number of $M_\infty = 0.5$ is chosen to verify if a formal order of accuracy can be achieved for the RDG($P_1P_1$) and WENO($P_1P_2$) methods to compute internal flows. This is a 3D simulation of the 2D problem. Two types of grids: prismatic and tetrahedral grids are assessed respectively.

Figure 7.29a through Figure 7.29d display the four successively globally refined prismatic grids having 128, 512, 2048 and 8192 cells as shown in $x$-$y$ plane and 1 cell in $z$-direction. Figure 7.30, Figure 7.31 and Figure 7.32 show the computed density, pressure and Mach number contours in the flow field obtained by the RDG($P_1P_1$) and WENO($P_1P_2$) solutions on the four prismatic grids, respectively. Also, numerical solutions to this problem are computed on these four grids to obtain a quantitative measurement of discretization errors and order of accuracy as presented in Table 7.5. As expected, the RDG($P_1P_1$) method exhibits a full $O(h^2)$ order of convergence, being 2.469 in average. The WENO($P_1P_2$) method does offer a full $O(h^3)$ order of convergence, being 3.223 in average. Figure 7.28a and Figure 7.28b illustrate the $L^2$ errors of numerical solutions versus cell size and number of degrees of freedom, respectively, which again indicate that the WENO($P_1P_2$) method requires much fewer degrees of freedom than the underlying RDG($P_1P_1$) method for the same accuracy.

#### Table 7.5: Discretization errors and convergence rates of the RDG methods obtained on the four successively globally refined prismatic grids for an inviscid subsonic flow through a channel with a bump at $M_\infty = 0.5$.

<table>
<thead>
<tr>
<th>Grid</th>
<th>No. DOFs</th>
<th>$L^2$-error-$P_1P_1$</th>
<th>Order-$P_1P_1$</th>
<th>$L^2$-error-$P_1P_2$</th>
<th>Order-$P_1P_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>512</td>
<td>-0.28947E+01</td>
<td>-</td>
<td>-0.33689E+01</td>
<td>-</td>
</tr>
<tr>
<td>512</td>
<td>2,048</td>
<td>-0.35941E+01</td>
<td>2.323</td>
<td>-0.43314E+01</td>
<td>3.197</td>
</tr>
<tr>
<td>2,048</td>
<td>8,192</td>
<td>-0.43592E+01</td>
<td>2.542</td>
<td>-0.53343E+01</td>
<td>3.332</td>
</tr>
<tr>
<td>8,192</td>
<td>32,768</td>
<td>-0.51240E+01</td>
<td>2.541</td>
<td>-0.62793E+01</td>
<td>3.139</td>
</tr>
</tbody>
</table>

Figure 7.34a through Figure 7.34d display the surface triangular meshes of a sequence of four successively refined tetrahedral grids, the coarse grid (Level 1) shown in Figure 7.34a consists of 889 elements, 254 nodes and 171 boundary nodes; the medium grid (Level 2) shown in Figure 7.34b consists of 6,986 elements, 1,555 nodes and 691 boundary nodes; the fine grid (Level 3) shown in Figure 7.34c consists of 55,703 elements, 10,822 nodes and 2,711 boundary nodes, and the finest grid (Level 4) shown in Figure 7.34d consists of 449,522 elements, 81,567
Figure 7.28: $L^2$ errors of numerical solutions versus cell size (a) and number of degree of freedom (b) for inviscid subsonic flow through a channel with a smooth bump at $M_\infty = 0.5$ by RDG($P_1P_1$) and WENO($P_1P_2$) on prismatic grids.

Figure 7.29: A sequence of four successively globally refined prismatic grids for inviscid subsonic flow through a channel with a smooth bump at $M_\infty = 0.5$. 

91
Figure 7.30: Computed density contours in the flow field by the RDG($P_1P_1$) (a, c, e, g) and WENO($P_1P_2$) (b, d, f, h) solutions on the four successively refined prismatic grids for inviscid subsonic flow through a channel with a smooth bump at $M_\infty = 0.5$. 
Figure 7.31: Computed pressure contours in the flow field by the RDG(\textit{P}_1\textit{P}_1) (a, c, e, g) and WENO(\textit{P}_1\textit{P}_2) (b, d, f, h) solutions on the four successively refined prismatic grids for inviscid subsonic flow through a channel with a smooth bump at $M_\infty = 0.5$. 

93
Figure 7.32: Computed Mach number contours in the flow field by the RDG($P_1P_1$) (a, c, e, g) and WENO($P_1P_2$) (b, d, f, h) solutions on the four successively refined prismatic grids for inviscid subsonic flow through a channel with a smooth bump at $M_\infty = 0.5$. 
nodes and 10,999 boundary nodes. Computations are conducted on the first three grids (Level 1, 2 and 3) to obtain a quantitative measurement of the discretization errors and order of convergence, as shown in Table 7.6. The RDG(P₁P₁) and WENO(P₁P₂) methods achieved an averaged order of 1.89 and 2.80, respectively. Consider the fact that this is a 3D simulation of a 2D problem, and the unstructured tetrahedral grids are not symmetric by nature, thus causing errors in the z-direction. If the RDG(P₁P₁) method can be considered to deliver the designed second order of accuracy, the WENO(P₁P₂) method offers a full $O(h^3)$ order of convergence, adding one order of accuracy to the underlying RDG(P₁P₁) method, Figure 7.35a through Figure 7.35c and Figure 7.35d through Figure 7.35f display the computed density, pressure and Mach number contours in the flow field obtained by the RDG(P₁P₁) and WENO(P₁P₂) solutions on the fine grid, respectively. In comparison, these two sets of figures look very similar to each other. However, WENO(P₁P₂) yields a slightly more accurate solution than RDG(P₁P₁) at the same grid resolution of each level, which are shown in Figure 7.33a and Figure 7.33b.

Figure 7.36a and Figure 7.36b illustrate the WENO(P₁P₂) convergence history of logarithmic density residual with respect to timestep and CPU time on the grids Level 1, 2 and 3. In comparison, the implicit method is over four orders of magnitude faster than its explicit counterpart, indicating a significant reduction in computing time for internal flow problems. A gradually decreased convergence rate with respect to timestep is observed on grids of higher level for the implicit WENO(P₁P₂) method, due to the use of inexact Jacobians. Computations are also conducted on the grid Level 4, but only for the implicit methods. The convergence rate for UJ-GMRES is over three times slower than MJ-GMRES as shown in Figure 7.37, again indicating the limitation of the simplified, unmatching Jacobians used as the preconditioning matrix in implicit higher-order DG methods.

Table 7.6: Discretization errors and convergence rates of the RDG methods obtained on the three successively refined tetrahedral grids for an inviscid subsonic flow through a channel with a bump at $M_\infty = 0.5$.

<table>
<thead>
<tr>
<th>Grid</th>
<th>No. DOFs</th>
<th>$L^2$-error-P₁P₁</th>
<th>Order-P₁P₁</th>
<th>$L^2$-error-P₁P₂</th>
<th>Order-P₁P₂</th>
</tr>
</thead>
<tbody>
<tr>
<td>889</td>
<td>3,556</td>
<td>-0.26129E+01</td>
<td>-</td>
<td>-0.26709E+01</td>
<td>-</td>
</tr>
<tr>
<td>6,986</td>
<td>27,944</td>
<td>-0.31333E+01</td>
<td>1.730</td>
<td>-0.356436E+01</td>
<td>2.968</td>
</tr>
<tr>
<td>55,703</td>
<td>222,812</td>
<td>-0.37430E+01</td>
<td>2.026</td>
<td>-0.435115E+01</td>
<td>2.614</td>
</tr>
</tbody>
</table>
Figure 7.33: $L^2$ errors of numerical solutions versus cell size (a) and number of degree of freedom (b) for inviscid subsonic flow through a channel with a smooth bump at $M_\infty = 0.5$ by RDG(P$_1$P$_1$) and WENO(P$_1$P$_2$) on tetrahedral grids.

Figure 7.34: A sequence of four successively globally refined tetrahedral grids for inviscid subsonic flow through a channel with a smooth bump at $M_\infty = 0.5$. 
Figure 7.35: Computed density, pressure and Mach number contours in the flow field by the RDG(P₁P₁) (a, b, c) and WENO(P₁P₂) (e, f, g) solutions on the three successively refined tetrahedral grids for inviscid subsonic flow through a channel with a smooth bump at $M_\infty = 0.5$.

Figure 7.36: Convergence history for logarithmic density residual with respect to timestep (a) and CPU time (b) respectively for inviscid subsonic flow through a channel with a smooth bump at $M_\infty = 0.5$ on tetrahedral grids (Level 1, 2 and 3), by WENO(P₁P₂).
Figure 7.37: Convergence history for logarithmic density residual with respect to timestep (a) and CPU time (b) respectively for inviscid subsonic flow through a channel with a smooth bump at $M_\infty = 0.5$ on tetrahedral grids (Level 4), by WENO($P_1P_2$).

7.1.5 Transonic Flow over the ONERA M6 Wing

An inviscid transonic flow over the ONERA M6 wing is considered in this case. This test case is chosen in order to assess and validate the performance of WENO($P_1P_2$) and HWENO($P_1P_2$) methods at the presence of weak and strong discontinuities.

The DG method is not only linear stable but also has the ability to obtain a stable solution for weak discontinuities in spite of the over- and under-shots in the vicinity of shock waves. Firstly, a weak-shock condition is computed at a freestream Mach number of $M_\infty = 0.699$ and an attack angle of $\alpha = 3.06^\circ$ on a coarse tetrahedral grid consisting of 41,440 elements, 8,325 grid points and 5,146 boundary faces, as the surface triangular meshes are shown in Figure 7.38a. One can observe the coarseness of grid even in the vicinity of the leading edge as shown in Figure 7.38b. The computed pressure contours obtained by the WENO($P_1P_2$) solutions on the wing surface are shown in Figure 7.39. Figure 7.40 compares the pressure coefficient distributions at six span-wise locations on the wing surface between the numerical results and experimental data [115]. The pressure coefficients are computed at the two nodes of each triangle that intersect with the cut plane, and plotted by a straight line. This representation truly reflects the discontinuous nature of the DG solution. As expected, the spurious oscillations in the vicinity of shocks do appear in the WENO($P_1P_2$) solutions, as the over- and under-shoots in the vicinity of the shock waves are clearly visible. In comparison, the HWENO($P_1P_2$) reconstruction is indeed able to eliminate these oscillations. Overall, the numerical results show a good agreement with experiment data, which indicates that the RDG methods are able to maintain the linear stability of the underlying DG method.
Figure 7.38: Plot of the tetrahedral grid for inviscid transonic flow over the ONERA M6 wing at $M_\infty = 0.699$, $\alpha = 3.06^\circ$: (a) global view; (b) close-up at the leading edge.

Figure 7.41a and Figure 7.41b show the WENO($P_1P_2$) convergence history of logarithmic density residual with respect to timestep and CPU time, respectively. It can be observed that MJ-GMRES is three orders of magnitude faster than its explicit counterpart with respect to CPU time as shown in Figure 7.41b. In addition, the speed of convergence for UJ-GMRES is over three times slower than MJ-GMRES which is also shown in Figure 7.41b, again indicating the limitation of using a simplified, unmatching Jacobian matrix for preconditioning in implicit RDG methods.

Figure 7.39: Plot of the pressure contours on triangular boundary faces of the tetrahedral grid for inviscid transonic flow over the ONERA M6 wing at $M_\infty = 0.699$, $\alpha = 3.06^\circ$. (a) global view; (b) close-up at the leading edge.

Secondly, a strong-shock condition is computed at $M_\infty = 0.84$ and $\alpha = 3.06^\circ$ on a coarse tetrahedral grid consisting of 95,266 elements, 18,806 grid points and 10,579 boundary faces, as the surface triangular meshes are shown in Figure 7.42a. Due to the presence of strong
Figure 7.40: Plot of pressure coefficient distributions at six span-wise locations for inviscid transonic flow over the ONERA M6 wing at $M_{\infty} = 0.699$, $\alpha = 3.06^\circ$.
Figure 7.41: Logarithmic density residual versus timestep (a) and CPU time (b) respectively for inviscid transonic flow over the ONERA M6 wing at $M_\infty = 0.699$, $\alpha = 3.06^\circ$, by WENO($P_1P_2$).

For discontinuities, WENO reconstruction at $P_2$ level alone, WENO($P_1P_2$), is not able to maintain the non-linear stability required for stable convergence in this condition, thus resulting in a breakdown of computation. Therefore, the final results of this computation are presented only with HWENO($P_1P_2$) solutions. Figure 7.42b shows the computed pressure contours on the upper wing surface.

Figure 7.42: Plot of the (a) surface unstructured grid and (b) pressure contours for inviscid transonic flow over the ONERA M6 wing at $M_\infty = 0.84$, $\alpha = 3.06^\circ$.

The computed pressure coefficients are compared with experimental data [115] at six span-wise locations in Figure 7.43. The results conform closely with the experimental data, except at the root stations as shown in Figure 7.43a and Figure 7.43b, due to lack of
7.1.6 Transonic Flow over the Wing/Pylon/Finned-Store Configuration

The test case in this section is an inviscid transonic flow over the wing/pylon/finned-store configuration at a freestream Mach number of \( M_\infty = 0.95 \) and attack angle of \( \alpha = 0^\circ \), in which the detailed description of this configuration is reported in Reference [64]. The objective of this test case is to assess and verify the performance of the HWENO(P\(_1\)P\(_2\)) method for complex geometric configurations. The configuration consists of a clipped delta wing with a 45° sweep comprised from a constant NACA 64010 symmetric airfoil section. The wing has a root chord of 15 inches, a semispan of 13 inches, and a taper ratio of 0.134. The pylon is located at the midspan station and has a cross-section characterized by a flat plate closed at the leading and trailing edges by a symmetrical olive shape. The width of the pylon is 0.294 inches. The four fins on the store are defined by a constant NACA 0008 airfoil section with a leading-edge sweep of 45° and a truncated tip. A tetrahedral grid consisting of 319,134 elements, 147,289 grid points, and 28,738 boundary faces is used in this computation, as its surface triangular meshes are shown in Figure 7.44a and Figure 7.44c.

Figure 7.44b and Figure 7.44d show the computed pressure contours on the upper and lower wing surfaces, respectively. The computed pressure coefficient distributions are compared with experimental data at two span-wise stations in Figure 7.45. The comparison with experimental data is excellent on both upper and lower surfaces up to 70% chord. As expected from the Euler solution, the computation predicts a shock location that is downstream of that measured by the experiment due to the lack of viscous effect. Again, our third-order HWENO(P\(_1\)P\(_2\)) method captures the shock waves very sharply within one cell without visible under- and over-shoots.

Figure 7.46a and Figure 7.46b plot the convergence history of logarithmic density residual with respect to timestep and CPU time, respectively. In Figure 7.46b, one can see that the implicit method has achieved a speedup of over three orders of magnitude faster than the explicit method. The explicit method is expected to take a prohibitively large amount of CPU time to drive the flow to steady state for such a complex geometric configuration. This test case has again verified the superior performance of the developed implicit scheme for HWENO(P\(_1\)P\(_2\)).
Figure 7.43: Plot of pressure coefficient distributions at six span-wise locations for inviscid transonic flow over the ONERA M6 wing at $M_\infty = 0.84$, $\alpha = 3.06^\circ$
Figure 7.44: Computed pressure contours on the unstructured surface mesh of a tetrahedral grid ($Nelem = 319, 134$, $Npoin = 147, 289$, $Nbfac = 28, 738$) obtained by the HWENO($P_1P_2$) solution for a transonic flow over a wing/pylon/finned-store configuration at $M_\infty = 0.95$, $\alpha = 0^\circ$. 
Figure 7.45: Comparison of the computed pressure coefficient distributions with experimental data at two span-wise locations for a transonic flow over a wing/pylon/finned-store configuration at $M_\infty = 0.95$, $\alpha = 0^\circ$.

Figure 7.46: Logarithmic density residual versus timestep (a) and CPU time (b) for inviscid subsonic flow over the wing/pylon/finned-store configuration at $M_\infty = 0.95$, $\alpha = 0^\circ$.
7.1.7 Transonic Flow over a Boeing 747 Aircraft

Finally, a transonic flow past a complete Boeing 747 aircraft at a freestream of Mach number of $M_{\infty} = 0.85$ and an angle of attack of $\alpha = 2^\circ$ is presented in this test case. The B747 configuration includes the fuselage, wing, horizontal and vertical tails, under-wing pylons, and flow-through engine nacelle. The grid used in the computation contains 253,577 elements, 48,851 grid points, and 23,616 boundary faces for the half-span airplane. The computation was carried out using 8 compute nodes (equal to 128 CPUs) on NCSU’s ARC cluster, which took around 11 minutes for convergence, as Figure 7.48a and Figure 7.48b show the convergence history of logarithmic density residual with respect to timestep and CPU time, respectively. The computed Mach number contours on the surface of the airplane, along with the surface mesh, are shown in Figure 7.47. One can see that the shock waves on the upper surface of the wing are captured well within one cell, confirming the accuracy, robustness and efficiency of the HWENO($P_1P_2$) method for computing complicated flows of practical importance.

![Computed Mach number contours and unstructured surface mesh of a tetrahedral grid](image)

Figure 7.47: Computed Mach number contours and unstructured surface mesh of a tetrahedral grid ($N_{elem} = 253,577$, $N_{poin} = 48,851$, $N_{bfac} = 23,616$) for a transonic flow over a complete B747 aircraft at $M_{\infty} = 0.85$, $\alpha = 2^\circ$. 

106
Figure 7.48: Logarithmic density residual versus timestep (a) and CPU time (b) for a transonic flow over a complete B747 aircraft at $M_\infty = 0.85$, $\alpha = 2^\circ$, on 128 CPUs.

7.2 Viscous Cases

In this section, a number of numerical experiments for compressible viscous flow problems computed by the RDG methods are presented. Firstly, the discretization of the viscous and heat fluxes in the Navier-Stokes equations is assessed and validated via the test case of laminar flow past a flat plate. Secondly, the performance of the RDG methods is assessed through the test case of a laminar flow past a sphere with a low Reynolds number. Thirdly, the test case of a subsonic flow around a delta wing at $Re = 4000$ is conducted in order to assess if the RDG($P_1P_2$) method is able to effectively resolve the flow features in more complex conditions. Finally, the test case of a subsonic flow around a sharp-edged slender delta wing at $Re = 0.95 \times 10^6$ is carried out in order to assess and verify the performance of the RDG($P_1P_2$) method for high Reynolds number flow problems, as the experimental data is available for the geometric configuration considered in this test case.

7.2.1 Laminar Flow past a Flat Plate

The laminar boundary layer over an adiabatic flat plate at a free-stream Mach number of $M_\infty = 0.5$ and a Reynolds number of $Re = 100,000$ based on the free-stream velocity and the length of the flat plate is considered in this test case. This problem is chosen to illustrate the accuracy of the RDG($P_1P_2$) method and the underlying RDG($P_1P_1$) method for the discretization of the viscous and heat fluxes in the Navier-Stokes equations, as the classical Blasius solution can be used to measure the accuracy of the numerical solution. Computations are performed on four grids: two hexahedral grids, one prismatic/hexahedral hybrid grid, and one tetrahedral grid, as
shown in Figure 7.49, Figure 7.53, and Figure 7.56 to assess the accuracy and consistency of the reconstructed discontinuous Galerkin method on different types of grids for viscous flows.

The computational domain is bounded from -0.5 to 1.0 along the $x$-direction, from 0 to 1.0 along the $y$-direction, and from 0 to 0.1 along the $z$-direction, and the no-slip surface starts at point $(0, 0, z)$ and extends to $(1, 0, z)$. The first two hexahedral grids used in this computation have the same number of cells $(25+50) \times 30 \times 1$, with $25 \times 30 \times 1$ cells ahead of the flat plate and $50 \times 30 \times 1$ cells for the flat plate, the same distribution of the grid points in the $x$-direction, but a different distribution of grid points in the $y$-direction. In order to cluster points near the flat plate, the point distribution in the $y$-direction follows a geometric stretching. The stretching ratio (SR) is the ratio of the heights of the two successive elements. An SR value of 1.20 and 1.30 is used for the two grids in the computation, respectively. The grids are plotted in Figure 7.49. The computational grids were generated with Gridgen V15.17. The quantitative description is shown in Table 7.7. For example, for the grid with SR = 1.20, the height of the first element is $8.4611 \times 10^{-4}$, and is also characterized with the normalized height of $y^+ = 4.542$. The non-dimensional wall distance $y^+$ for a wall-bounded flow can be defined in the following way:

$$y^+ = \frac{u_* y}{\nu}$$

where $u_*$ is the friction velocity at the nearest wall, $y$ is the distance to the nearest wall, and $\nu$ is the local kinematic viscosity of the fluid. Dimensional analysis shows that at a wall a characteristic velocity, called the friction velocity, $u_*$, can be defined in the following way:

$$u_* = \sqrt{\frac{\tau_w}{\rho}}$$

where $\tau_w$ is the wall shear stress, and $\rho$ is the fluid density at the wall. As a result, the grid with SR = 1.30 provides a better grid resolution in the boundary layer region.

Table 7.7: Boundary layer grid resolution for laminar flow past a flat plate at $Re = 100,000$

<table>
<thead>
<tr>
<th>Stretching Ratio (SR)</th>
<th>Height of the first layer</th>
<th>$y^+$ of the first layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.20</td>
<td>$8.4611 \times 10^{-4}$</td>
<td>4.542</td>
</tr>
<tr>
<td>1.30</td>
<td>$1.1455 \times 10^{-4}$</td>
<td>0.615</td>
</tr>
</tbody>
</table>

No-slip and no-penetration conditions $\mathbf{V} = (0, 0, 0)$ are prescribed when $y = 0$ for $x \in [0, 1]$. A slip condition is prescribed along the bottom side of the domain for $x \in [-0.5, 0]$ with $v = 0$. 

108
Symmetry conditions are prescribed in the 1-cell thick region (front and back walls) with $w = 0$. The characteristic boundary is prescribed to the left side ($x = -0.5$), top side ($y = 1$), and right side ($x = 1$).

The numerical results obtained by RDG(P$_1$P$_1$) and RDG(P$_1$P$_2$) on these four grids are presented, and compared with the theoretical ones given by the Blasius solution. Figure 7.50a shows the logarithmic plot of the computed skin friction coefficient $c_f$ distributions along the flat plate $x \in [0, 1]$. The skin friction coefficient $c_f$ is defined by

$$c_f = \frac{\tau_w}{\frac{1}{2} \rho u_\infty^2}$$

where $\tau_w$ is the local wall shear stress, $\rho$ is the fluid density and $u_\infty$ is the free-stream velocity.

The wall shear stress $\tau_w$ is given by

$$\tau_w = \mu \left( \frac{\partial u}{\partial y} \right)_{y=0}$$

where $\mu$ is the dynamic viscosity. In comparison, the grid of SR = 1.30 provided a better overall prediction of $c_f$. Near the plate leading edge region, the $c_f$ distribution was overpredicted evidently on grid of SR = 1.20 due to the insufficient grid resolution. Furthermore, a close-up comparison of $c_f$ distributions obtained by RDG(P$_1$P$_1$) and RDG(P$_1$P$_2$) for $x \in [0.05, 0.1]$ is shown in Figure 7.50b, which has clearly demonstrated that the RDG(P$_1$P$_2$) method is also able to improve the accuracy of its underlying DG(P$_1$) method for viscous flows.

Figure 7.51a and Figure 7.52a show the plot of velocity profiles versus similarity variable $\eta$ for computed $x$-velocity $u$ and scaled $y$-velocity $v^+$ along the cells cut through by plane of $x = 0.1$, $x = 0.5$ and $x = 0.9$ in the boundary layer region, among which the cells cut by plane of $x = 0.9$ are adjacent to the outflow boundary. The similarity variable $\eta$ is defined as

$$\eta = \frac{y}{\delta(x)} = \frac{y}{\sqrt{\nu x / u_\infty}}$$

where $\delta(x)$ is the scaling argument, and $x$ is the downstream coordinate. In Figure 7.51a, it can be observed that the $u$ profiles at all the three downstream locations and the analytical solution are highly matched, which illustrates a consistent convergence and accuracy of numerical solution by RDG methods. On the other side, the $v^+$ profiles can also match the analytical solution very well as shown in Figure 7.52a, although not so good as the $u$ profiles, since the $y$-velocity is relatively a small quantity with respect to the $x$-velocity, and should require a much higher grid resolution if expected to match the analytical solution better. Moreover, a close-up comparison of $u$ versus $\eta \in [4, 8]$ and also $v^+$ versus $\eta \in [3, 6.5]$ obtained by RDG(P$_1$P$_1$)
and RDG(P₁P₂), are shown in Figure 7.51b and Figure 7.52b respectively, which again have reflected the improved accuracy by the RDG(P₁P₂) method over the underlying RDG(P₁P₁) method for viscous flows.

The hybrid grid shown in Figure 7.53a consists of 977 prismatic elements and 358 hexahedral elements. A close-up observation in Figure 7.53b shows that the boundary layer region mainly consists of hexahedral elements, except at the leading edge region of the plate, where prismatic elements are filled. Figure 7.54a shows the computed $c_f$ distributions obtained by RDG(P₁P₁) and RDG(P₁P₂). If we zoom in the figure and take a closer look, such as at $x \in [0.02, 0.1]$ shown in Figure 7.54b, clearly we can see that RDG(P₁P₂) predicts a more accurate numerical solution than the underlying RDG(P₁P₁) on this hybrid grid. Figure 7.55 shows the plot of computed velocity profiles obtained by RDG(P₁P₁) and RDG(P₁P₂) versus $\eta$ and compared with the analytical solutions at downstream coordinates $x = 0.1$, $x = 0.5$ and $x = 0.9$ (in cells adjacent to outflow boundary) in the boundary layer region. Again, the RDG methods demonstrate a highly consistent convergence of numerical solution if we look at the $x$-velocity $u$ profiles as shown in Figure 7.55a. The scaled $y$-velocity $v^+$ profiles are displayed in Figure 7.55b. If we focus on the solution quality at the cells adjacent to outflow boundary ($x = 0.9$), an evident improvement brought by RDG(P₁P₂) is observed, since the RDG(P₁P₂) method has presented the $v^+$ points which are closer to the analytical solution than the underlying RDG(P₁P₁) method.

Figure 7.56a shows the tetrahedral grid used in computation, which consists of 47,536 elements. Figure 7.56b renders a closer view of the grid at the near-wall region, which depicts the sparsity of the grid resolution in the boundary layer. Figure 7.57a shows the computed $c_f$ distributions obtained by RDG(P₁P₁) and RDG(P₁P₂). Both methods result in highly accurate $c_f$ distributions on this grid. The difference is only discernible if the figure is zoomed in, e.g., at $x \in [0.5, 0.6]$ as shown in Figure 7.57b, then one can see that RDG(P₁P₂) predicts a slightly better prediction than the RDG(P₁P₁) on this grid. Figure 7.58 shows the plot of computed velocity profiles obtained by RDG(P₁P₁) and RDG(P₁P₂) versus $\eta$ and compared with the analytical solutions at downstream coordinates $x = 0.1$, $x = 0.5$ and $x = 0.9$ (in cells adjacent to outflow boundary) in the boundary layer region. The RDG methods exhibit a consistent convergence of numerical solution if we examine both the $x$-velocity $u$ profiles shown in Figure 7.58a, and the scaled $y$-velocity $v^+$ profiles shown in Figure 7.58b.

The performance of convergence is assessed for RDG(P₁P₁) and RDG(P₁P₂) on this tetrahedral grid by using both the explicit RK3 and implicit GMRES time integration schemes. The CFL number for the implicit scheme is set to $1.0 \times 10^3$ for the initial 5 steps and increased to $1.0 \times 10^6$ afterwards. Figure 7.59a and Figure 7.59b show the convergence history of logarithmic density residuals with respect to timestep and CPU time respectively. For the explicit RK3 scheme, the time histories for both RDG(P₁P₁) and RDG(P₁P₂) are displayed, which
have shown only a limited amount of timesteps for the demonstration purpose. For the implicit GMRES scheme, the performance of three Jacobian matrix evaluation methods is assessed for RDG(P₁P₁): divided difference (DD-GMRES), unmatching Jacobian (UJ-GMRES), and automatic differentiation (AD-GMRES). One can see that AD-GMRES(P₁P₁) provides the best converging speed with respect to both timestep and CPU time. DD-GMRES(P₁P₁) provides a fairly good converging speed with respect to timestep, but becomes the slowest among the three with respect to CPU time, due to the fact that a huge amount of CPU time is required for computing the Jacobian matrix by divided difference.

For implicit RDG(P₁P₂), only the result of AD-GMRES(P₁P₂) is presented, since the other two Jacobian evaluation methods are either very unstable or rather slow for GMRES(P₁P₂) to drive the solution to steady state in this test case. To compare the explicit and implicit convergence for RDG(P₁P₂), one can observe that AD-GMRES(P₁P₂) can deliver a speedup of more than three orders of magnitude over RK3(P₁P₂), indicating the over-dominant advantage of using the implicit methods developed in this work for solving the Navier-Stokes equations. One can also see that with respect to timestep, the convergence of AD-GMRES(P₁P₂) is retarded and suffers some oscillation in comparison with that of AD-GMRES(P₁P₁) as shown in Figure 7.59a, due to the limitations of using the degenerate AD derived Jacobian matrix. Nevertheless, the latency brought by the P₁ Jacobian matrix for the RDG methods is limited as long as it is only used for preconditioning.

Table 7.8 lists the execution profile obtained for the implicit time integration for RDG(P₁P₂). The name, \textit{rhsbounvp1p2}, denotes the subroutine to obtain the interior face integrals contributed to the right-hand-side vector \(\mathbf{R}\). The \textit{lusgsp1} is the subroutine that executes the LU-SGS preconditioning. The \textit{rhsbvp1dv} is the AD generated subroutine that computes interior face integrals contributed to the elemental block diagonal Jacobian matrix, and its corresponding upper and lower block matrices. Note that it is called for the number of interior faces per timestep. The \textit{rhsdomnvp1p2} is the subroutine that computes the domain integrals contributed to the right-hand-side vector \(\mathbf{R}\). The \textit{rhsdp1dv} is the AD generated subroutine that computes the domain integrals contributed to the elemental block diagonal Jacobian matrix. Note that it is called for the number of elements per timestep. To sum up, the CPU time required to compute the Jacobian matrix (\textit{rhsbvp1dv} + \textit{rhsdomnvp1p2}) accounts for 21% of the total execution time, and is only about 54% of the one for evaluation of the right-hand-side vector (\textit{rhsbounvp1p2} + \textit{rhsdomnvp1p2}).

To conclude from this numerical experiment, the RDG methods have been verified to demonstrate a consistent performance with high accuracy of solution for the discretization of the viscous and heat fluxes in the Navier-Stokes equations on both single-type grids and hybrid grids.
Figure 7.49: Plot of the hexahedral grids for laminar flow past a flat plate at $Re = 100,000$: (a) $SR = 1.20$ in $y$-direction; (b) $SR = 1.30$ in $y$-direction.

Figure 7.50: (a) Logarithmic plot of the computed skin friction $c_f$ coefficient distribution obtained on the two hexahedral grids compared with the analytical solution along the flat plate $x \in [0, 1]$; (b) close-up comparison of $c_f$ distribution obtained by $\text{RDG}(P_1P_1)$ and $\text{RDG}(P_1P_2)$ for $x \in [0.05, 0.1]$. 

112
Figure 7.51: (a) Plot of $x$-velocity $u$ versus similarity variable $\eta$ obtained on the two hexahedral grids and compared with the analytical solutions at downstream coordinates $x = 0.1$, $x = 0.5$ and $x = 0.9$ (in cells adjacent to outflow boundary) in the boundary layer region; (b) close-up comparison of $u$ profiles obtained by RDG($P_1P_1$) and RDG($P_1P_2$) for $\eta \in [4,8]$.

Figure 7.52: (a) Plot of scaled $y$-velocity $v^+$ versus similarity variable $\eta$ obtained on the two hexahedral grids and compared with the analytical solutions at downstream coordinates $x = 0.1$, $x = 0.5$ and $x = 0.9$ (in cells adjacent to outflow boundary) in the boundary layer region; (b) close-up comparison of $v^+$ profiles obtained by RDG($P_1P_1$) and RDG($P_1P_2$) for $\eta \in [3,6.5]$. 

113
Figure 7.53: Plot of the prismatic/hexahedral grids for laminar flow past a flat plate at $Re = 100,000$: (a) global view of the the domain; (b) local view of boundary layer region.

Figure 7.54: (a) Logarithmic plot of the computed skin friction $c_f$ coefficient distribution obtained on the prismatic/hexahedral hybrid grid compared with the analytical solution along the flat plate $x \in [0, 1]$; (b) close-up comparison of $c_f$ distribution obtained by RDG($P_1P_1$) and RDG($P_1P_2$) for $x \in [0.02, 0.1]$. 

114
Figure 7.55: Plot of the computed velocity profiles versus similarity variable $\eta$ obtained on the prismatic/hexahedral hybrid grid and compared with the analytical solutions at downstream coordinates $x = 0.1$, $x = 0.5$ and $x = 0.9$ (in cells adjacent to outflow boundary) in the boundary layer region: (a) $x$-velocity $u$ versus $\eta$; (b) scaled $y$-velocity $v^+$ versus $\eta$.

Figure 7.56: Plot of the tetrahedral grids for laminar flow past a flat plate at $Re = 100,000$: (a) global view of the the domain; (b) local view of boundary layer region.
Figure 7.57: (a) Logarithmic plot of the computed skin friction $c_f$ coefficient distribution obtained on the tetrahedral grid compared with the analytical solution along the flat plate $x \in [0, 1]$; (b) close-up comparison of $c_f$ distribution obtained by RDG(P$_1$P$_1$) and RDG(P$_1$P$_2$) for $x \in [0.5, 0.6]$.

Figure 7.58: Plot of the computed velocity profiles versus similarity variable $\eta$ obtained on the tetrahedral grid and compared with the analytical solutions at downstream coordinates $x = 0.1$, $x = 0.5$ and $x = 0.9$ (in cells adjacent to outflow boundary) in the boundary layer region: (a) $x$-velocity $u$ versus $\eta$; (b) scaled $y$-velocity $v^+$ versus $\eta$. 
Table 7.8: Execution profile of the implicit time integration for RDG(P₁P₂) on tetrahedral grid for laminar flow past a flat plate at $Re = 100,000$.

<table>
<thead>
<tr>
<th>Time (%)</th>
<th>Cumulative (sec)</th>
<th>Self (sec)</th>
<th>Calls</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>31.80</td>
<td>178.32</td>
<td>178.32</td>
<td>462</td>
<td>rhsbounvp1p2_</td>
</tr>
<tr>
<td>17.23</td>
<td>274.90</td>
<td>96.58</td>
<td>462</td>
<td>lusgsp1_</td>
</tr>
<tr>
<td>16.21</td>
<td>365.76</td>
<td>90.86</td>
<td>3657720</td>
<td>rhsbvp1_dv_</td>
</tr>
<tr>
<td>6.97</td>
<td>404.86</td>
<td>39.10</td>
<td>462</td>
<td>rhsdomnvp1p2_</td>
</tr>
<tr>
<td>4.78</td>
<td>431.67</td>
<td>26.81</td>
<td>950720</td>
<td>rhsdvp1_dv_</td>
</tr>
</tbody>
</table>

Figure 7.59: Logarithmic density residual versus timestep (a) and CPU time (b) for laminar flow past a flat plate at $Re = 100,000$. 
7.2.2 Laminar Flow past a Sphere

A laminar flow past a sphere at a freestream Mach number of $M_\infty = 0.5$ and Reynolds number of $Re = 118$ based on the diameter of the sphere is considered in this test case. The objective for this numerical example is to assess and verify the performance of the RDG methods for low Reynolds number flow problems. The initial condition is a uniform freestream with non-slip boundary conditions on the solid wall. A coarse tetrahedral grid consisting of 119,390 elements, 22,530 grid nodes and 4,511 boundary nodes is used in this computation, as shown in Figure 7.60a. One can observe the coarseness of the triangular surface meshes near the sphere wall region. Note that only half of the configuration is modeled due to the symmetry of the problem.

Computations are carried out by using the explicit RK3 scheme and implicit GMRES scheme for both RDG($P_1P_1$) and RDG($P_1P_2$), respectively. The computed pressure contours and Mach number contours obtained by RDG($P_1P_2$) on the surface meshes in the flow field are displayed in Figure 7.61a and Figure 7.61b, respectively. The plot of computed velocity streamtraces on the symmetry plane is displayed in Figure 7.60b, in which two trailing vortices can be observed, and they are visually identical and symmetric to the centerline.

For implicit time integration, the $CFL$ number is set to $1.0 \times 10^3$ for the initial 5 steps and increased to $1.0 \times 10^6$ afterwards. Figure 7.62a and Figure 7.62b show the convergence histories of logarithmic density residuals with respect to timestep and CPU time respectively for RDG($P_1P_1$), which compare the converging speeds between RK3 and GMRES. The similar comparison is presented between RK3 and GMRES for RDG($P_1P_2$), as shown in Figure 7.63a and Figure 7.63b. The automatic differentiation derived Jacobian matrix is used for preconditioning in the GMRES solver. One can observe that the implicit GMRES scheme delivers a speedup of more than three orders of magnitude over the explicit RK3 scheme, again indicating the necessity of using an efficient implicit algorithm and well formulated Jacobian matrix for solving the Navier-Stokes equations. In addition, the latency of convergence for the implicit RDG($P_1P_2$) method is observed according to the timestep history of its underlying implicit RDG($P_1P_1$) method. However, such a latency due to the degenerate Jacobian matrix does not cause a significant slowdown for RDG($P_1P_2$) to converge to the steady state (usually a drop of 4~5 orders of magnitude for density residual) with respect to that of the implicit RDG($P_1P_1$) method.
Figure 7.60: (a) Plot of the surface meshes of the tetrahedral grid for laminar flow past a sphere at $M_\infty = 0.5$ and $Re = 118$; (b) plot of the computed velocity streamtraces on the symmetry plane.

Figure 7.61: Plot of the computed contours obtained by RDG($P_1P_2$) on the surface meshes in the flow field at $M_\infty = 0.5$ and $Re = 118$: (a) pressure contours; (b) Mach number contours.
Figure 7.62: Logarithmic density residual versus timestep (a) and CPU time (b) for laminar
flow past a sphere at $M_\infty = 0.5$ and $Re = 118$, by RDG($P_1P_1$).

Figure 7.63: Logarithmic density residual versus timestep (a) and CPU time (b) for laminar
flow past a sphere at $M_\infty = 0.5$ and $Re = 118$, by RDG($P_1P_2$).
7.2.3 Subsonic Flow around a Delta Wing at $Re = 4000$

A laminar flow at a high angle of attack around a delta wing with a sharp leading edge and a blunt trailing edge is considered in this test case. The flow conditions are freestream Mach number of $M_{\infty} = 0.3$, angle of attach $\alpha = 12.5^\circ$, and Reynolds number of $Re = 4000$ based on a mean cord length of 1. The objective of this test case is to assess if the RDG(P1P2) method is able to effectively resolve the flow features in more complex conditions. A tetrahedral grid consisting of 674,260 elements, 120,531 grid points and 12,991 boundary points is used in computation, as the triangular meshes of the delta wing surface are shown in Figure 7.65. The no-slip and adiabatic boundary conditions are prescribed to the wing surface. A complete description of the geometric configuration can be found in Reference [73, 136]. The computation is initialized with constant freestream values in the entire domain. A CFL number of $1.0 \times 10^3$ is prescribed for the initial 10 steps, and increased to $1.0 \times 10^6$ afterwards. The flow field is assumed to have reached full convergence with a drop of 8 orders of magnitude for the global density residual in 300 timesteps as shown in Figure 7.64a. The convergence history in terms of CPU time is shown in Figure 7.64b. The computed Mach number contours on the sliced planes of $x = 0.5$, $x = 0.75$, $x = 1.0$ and $x = 1.2$ are displayed in Figure 7.66 along with the streamtraces in the flow field. It is observed that as the flow passes the leading edge it rolls up and creates a vortex together with a secondary vortex. The vortex system remains over a distance behind the wing.

Figure 7.64: Logarithmic density residual versus timestep (a) and CPU time (b) respectively for subsonic flow over a delta wing at $M_{\infty} = 0.3$, $\alpha = 12.5^\circ$ and $Re = 4000$, by RDG(P1P2).
Figure 7.65: The surface meshes of a tetrahedral grid for subsonic flow over a delta wing at $M_\infty = 0.3$, $\alpha = 12.5^\circ$ and $Re = 4000$: (a) bottom view; (b) side view.

Figure 7.66: Computed Mach number contours on the sliced planes of $x = 0.5$, $x = 0.75$, $x = 1.0$ and $x = 1.2$ and streamtraces in the flow field for subsonic flow over a delta wing at $M_\infty = 0.3$, $\alpha = 12.5^\circ$ and $Re = 4000$, by RDG(P$_1$P$_2$).
7.2.4 Subsonic Flow around a Delta Wing at $Re = 0.95 \times 10^6$

The last test case is a laminar flow around a sharp-edged slender delta wing of aspect ratio equal to 1, which corresponds to a sweep angle of 75.9638°. The flow conditions are freestream Mach number of $M_\infty = 0.3$, angle of attack $\alpha = 20.5^\circ$, and Reynolds number of $Re = 0.95 \times 10^6$ based on a mean cord length of 1. The objective is to assess and verify the performance of the RDG(P1P2) method for high Reynolds number flow problems, as the experimental data is available for the geometric configuration considered in this test case. Figure D.1 in Appendix D is extracted from Reference [62] to describe the configuration: the upper surface of the wing is flat and the cross section is triangular ahead of $x/L = 0.9$, with the maximum thickness $0.021L$; the cross section downstream of $x/L = 0.9$ is trapezoidal, and the trailing edge is sharp. A tetrahedral grid consisting of 316,139 elements, 58,322 grid points and 21,712 boundary faces is used in computation, as the triangular meshes of the wing surface are shown in Figure 7.67a. The no-slip and adiabatic boundary conditions are prescribed to the wing surface. Figure 7.67b displays a sliced-plane of $x/L = 0.9$ to demonstrate the highly stretched elements piled on the wing surface. The thickness of the elements for the first layer is $0.5 \times 10^{-3}$, with a growth rate of 1.3 normal to the surface and 5 layers in total.

The computation is initialized with constant freestream values in the entire domain. A $CFL$ number of $1.0 \times 10^2$ is prescribed for the initial 10 steps, and increased to $1.0 \times 10^4$ afterwards. The flow filed is assumed to have reached the steady state after a decrease of 8 orders of magnitude for the global density residual in only 191 timesteps as shown in Figure 7.68a. The convergence history in terms of CPU time is shown in Figure 7.68b.

Figure 7.69a through Figure 7.69d show the computed surface pressure coefficient distributions compared with the experimental data at four stations along the chord of the geometry: $x/L = 0.3, 0.5, 0.7$ and 0.9. The pressure coefficients are computed at the two nodes of each surface triangular face that intersect with the sliced plane, and plotted by a straight line. At the forward and middle stations ($x/L = 0.3, 0.5$ and 0.7), the numerical results agree well with the experimental data, as shown in Figure 7.69a, Figure 7.69b, and Figure 7.69c. However the pressure coefficients are significantly over-predicted on the lower wing surface at station $x/L = 0.9$ as shown in Figure 7.69d. Similar results were previously reported by J. Thomas et.al [122] and T. Erwin et.al [48]. The reason for the disagreement is not clearly known, but it was surmised by J. Thomas et.al [122] that the disagreement might be caused by the presence of the pressure tubing that exits from the model in the lower surface trailing-edge region.
Figure 7.67: A tetrahedral grid for subsonic flow over a sharp-edged slender delta wing at $M_\infty = 0.3$, $\alpha = 20.5^\circ$ and $Re = 0.95 \times 10^6$: (a) triangular surface meshes; (b) extracted meshes on sliced plane of $x/L = 0.9$.

Figure 7.68: Logarithmic density residual versus (a) timestep and (b) CPU time respectively for subsonic flow over a delta wing at $M_\infty = 0.3$, $\alpha = 20.5^\circ$ and $Re = 0.95 \times 10^6$, by RDG(P$_1$P$_2$).
Figure 7.69: Surface pressure coefficient distributions at four typical downstream intersections for subsonic flow over a delta wing at $M_\infty = 0.3$, $\alpha = 20.5^\circ$ and $Re = 0.95 \times 10^6$, by RDG(P1P2).
7.3 Parallel Performance

In order to assess the scalability of the parallelization strategy implemented in the RDGFLO code, we used the finest grid which consists of 124,706 elements (equal to 498,824 DOFs) in the test case of subsonic flow past a sphere and split it into up to 128 subdomains. The computation is conducted on NCSU’s ARC Linux cluster [1] with the source code compiled by the PGI Accelerator Version 12.10. The hardware of ARC cluster consists of 1728 cores on 108 compute nodes integrated by Advanced HPC. All machines are 2-way SMPs with AMD Opteron 6128 (Magny Core) processors with 8 cores per socket (16 cores per node). A general description is presented in the following:

- Supermicro H8DGG-QF Motherboard with IPMI 2.0
- Transport 1022GG-TF with 2 GPU slots + 1 half-size PCI-E slot
- 32GB DRAM
- Mellanox ConnectX-2 VPI InfiniBand Adapter Card - Part ID: MHQH19B-XTR
- login node: login-0-0 (1TB HDD+nVidia C2050+ OCZ RevoDrive 80GB SSD under /mnt)
- nodes: compute-0-XXX, XXX=0..107 (1TB HDD, OCZ RevoDrive 120GB SSD under /mnt)
- 5 PFS (Lustre) nodes: mds-0-114 and oss-0-YYY, YYY=108-111 (each OSS has 10TB HDDs as RAID5)
- head node: arcs (has 12TB HDDs as RAID5) with Supermicro H8DGU-F Motherboard and Transport 2022G-URF

The scaling tests are carried out for the explicit time integration and implicit time integration, respectively. Figure 7.70a and Figure 7.70b display the parallel speedup and efficiency obtained on a single compute node (up to 16 CPUs). The parallel efficiency has arrived around 78% and 80% for the explicit and implicit methods respectively. Figure 7.71a and Figure 7.71b demonstrate the parallel speedup and efficiency obtained on multiple compute nodes (up to 8 nodes with 16 CPUs per node). As one can observe, the efficiency has a sharp decrease from 1 node (16 CPUs) to 2 nodes (32 CPUs) for the implicit method, but tends to drop slower with more nodes used. This large decrease of efficiency from single node to multiple nodes is probably due the deteriorated quality of inter-node communication. Finally, the explicit and implicit methods have achieved about 48% and 51% with 8 nodes, respectively. In future, we will test the parallel performance of the RDGFLO code with larger-scale problems in order to gain more experience, and see if further optimization can be explored.
Figure 7.70: Parallel speedup and efficiency on a single compute node (up to 16 CPUs): (a) explicit time integration; (b) implicit time integration.

Figure 7.71: Parallel speedup and efficiency on multiple compute nodes (up to 8 nodes with 16 CPUs per node): (a) explicit time integration; (b) implicit time integration.
Chapter 8

Conclusions

8.1 Summary of Completed Work

A parallel, implicit reconstructed discontinuous Galerkin method based on a Hierarchical WENO reconstruction: HWENO(P₁P₂), has been developed to solve the compressible Navier-Stokes equations on 3D hybrid grids. The HWENO(P₁P₂) method is designed not only to enhance the accuracy of the underlying second-order DG(P₁) method but also to avoid non-physical oscillations in the vicinity of strong discontinuities, without significant increase in computing costs and memory requirements. The implicit backward Euler method has been used for time advancement in the RDG method, where the three approaches: analytical differentiation, divided differencing (DD), and automatic differentiation (AD) are developed and implemented to obtain the resulting flux Jacobian matrices. The AD technique is especially attractive, being able to provide an inexact but relatively accurate inviscid and viscous flux Jacobians in a straightforward way. A notable feature of this implicit RDG(P₁P₂) method is that the Jacobian matrix is approximated by taking differentiation of the underlying second-order DG(P₁) right-hand-side (RHS) operator instead of the reconstruction-based RDG(P₁P₂) RHS operator, without the great computing effort to linearize the highly non-linear HWENO(P₁P₂) reconstruction process. In order to compute large-scale problems of practical importance, a parallel strategy based on domain partitioning and the Message Passing Interface (MPI) parallel programming environment has been developed to empower the RDGFLO solver to execute on high performance computing (HPC) platforms. The developed parallel implicit RDG(P₁P₂) method is assessed and validated through computing a variety of well-documented compressible inviscid and viscous flow test cases to demonstrate its accuracy, robustness, and non-oscillatory performance.

Three main contributions have been made in this work towards the development of a higher-order numerical method to simulate 3D aerodynamic flows on arbitrary grids:
• An HWENO reconstruction-based discontinuous Galerkin methodology, RDG(P₁P₂), is developed for the solution of compressible flows on 3D hybrid grids, and has been demonstrated as a potential candidate of superior DG method over the more mature and well-established second-order methods.

• A cost-effective implicit time integration scheme based on automatic differentiation has been developed for the RDG(P₁P₂) method, enabling the flow solver to deliver a significant speedup in terms of spatial convergence over the underlying explicit Runge-Kutta DG (RKDG) methods.

• A distributed-memory, MPI-based parallel infrastructure is established and implemented in order to empower the developed implicit RDG(P₁P₂) method to compute large-scale simulation tasks of practical importance.

The current RDGFLO flow solver must undergo more extensive verification and validation (V&V) process against benchmark test cases before it can be trusted and used for simulating a variety of compressible flow problems. The work accomplished in this PhD work only represents a first step towards the development of an accurate, efficient, and robust computer code to simulate a variety of 3D aerodynamic flows on arbitrary grids.

8.2 Outlook of Future Work

The future development in the timeline of the RDGFLO code may be focused on two main directions: one direction is the research-oriented development to design, add, and implement more capabilities and features for the study of more complex fluid flows, e.g., turbulent flows; the other can be the application-oriented development to adopt emerging software and hardware technologies to improve overall performance of the flow solver, e.g., GPU acceleration.

For the first, the fully implicit time integration methods [24, 134, 92], e.g., the second-order backward differentiation formular (BDF2) scheme and the third-order implicit Runge-Kutta (IRK3) scheme, is expected to be implemented so that the RDGFLO code would be able to compute the time-accurate flow problems. For the second, the features of direct numerical simulation (DNS) and large eddy simulation (LES) [61, 35, 36, 116, 41] are desired for the RDGFLO code in order to perform complex flow simulations, e.g., turbulent flows. For the third, a completely different approach to improve the solver’s robustness and efficiency would be the adaptive strategies [23, 111, 100, 68, 135], i.e., h-adaptivity, p-adaptivity or hp-adaptivity, with which the computing resource could be more rationally allocated and distributed, and thus the overall solution cost-effectiveness could be further improved. For the fourth, the dynamic & moving mesh capabilities [127, 78], if implemented, would enable the RDGFLO code to meet
the needs of challenging applications, including in-cylinder flows, flapping airfoils, etc. Finally, in order to tap the full potential of the DG methods with the emerging software and hardware technologies, i.e., general-purpose GPU computing (GPGPU) [99, 38, 69, 37, 39, 29] in which the computation-intensive portion of a codebase is offloaded from a host CPU to an attached GPU, a portable and multi-platform compatible GPU-acceleration technique should be included in the RDGFLO code in the near future.
REFERENCES


[41] C. C. de Wiart and K. Hillewaert. DNS and ILES of transitional flows around a SD7003 using a high order Discontinuous Galerkin Method. 7th International Conference on Computational Fluid Dynamics, ICCFD7-3604, 2012.


Appendix A

Various Types of Elements

In this section, the shape functions of the various reference elements frequently used in practice are presented. The geometrical types of elements are listed in Table A.1. From the geometrical information, we can construct the shape functions $\phi(\xi, \eta, \zeta)$ and their derivatives with respect to $\xi$, $\eta$ and $\zeta$. A more comprehensive introduction to the various types of elements are given in §2 of the book *Finite Element Method* by Dhatt et al. [42].

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRIA3</td>
<td>3-node triangle.</td>
</tr>
<tr>
<td>TRIA6</td>
<td>6-node second-order triangle (3 nodes associated with the vertices and 3 with the edges).</td>
</tr>
<tr>
<td>QUAD4</td>
<td>4-node quadrilateral.</td>
</tr>
<tr>
<td>QUAD8</td>
<td>8-node second-order quadrilateral (4 nodes associated with the vertices and 4 with the edges).</td>
</tr>
<tr>
<td>TETR4</td>
<td>4-node tetrahedron.</td>
</tr>
<tr>
<td>TETR10</td>
<td>10-node second-order tetrahedron (4 nodes associated with the vertices and 6 with the edges).</td>
</tr>
<tr>
<td>PYRA5</td>
<td>5-node pyramid.</td>
</tr>
<tr>
<td>PYRA13</td>
<td>13-node second-order pyramid (5 nodes associated with the vertices and 8 with the edges).</td>
</tr>
<tr>
<td>PRIS6</td>
<td>6-node prism.</td>
</tr>
<tr>
<td>PRIS15</td>
<td>15-node second-order prism (6 nodes associated with the vertices and 9 with the edges).</td>
</tr>
<tr>
<td>HEXA8</td>
<td>8-node hexahedron.</td>
</tr>
<tr>
<td>HEXA20</td>
<td>20-node second-order hexahedron (8 nodes associated with the vertices and 12 with the edges).</td>
</tr>
</tbody>
</table>
A.1 Triangular Elements (two dimensions)

A.1.1 Systems of Coordinates

For all triangular elements, the following reference element shown in Figure A.1 is used, where

\( \xi \leq 0, \eta \leq 0 \) and \( 1 - \xi - \eta \leq 0 \). The coordinates \((\xi, \eta)\) can be interpreted as curvilinear coordinates for the physical element. The barycentric coordinates \( L_1, L_2 \) and \( L_3 \) are often used to mark a point \( 0 \) of a straight-edged triangle as shown in Figure A.2.

\[
L_1 = \frac{A_1}{A} \; ; \; L_2 = \frac{A_2}{A} \; ; \; L_3 = \frac{A_3}{A} \\
A = A_1 + A_2 + A_3 \\
L_1 + L_2 + L_3 = 1
\]  

(A.1)
where $A_1$, $A_2$ and $A_3$ are the areas of the subtriangles 0-2-3, 0-3-1 and 0-1-2. $A$ is the area of the triangle 1-2-3. The coordinates $L_1$, $L_2$ and $L_3$ are linked to the coordinates $(\xi, \eta)$ by the following relation:

$$L_1 \equiv 1 - \xi - \eta, \quad L_2 \equiv \xi, \quad L_3 \equiv \eta \quad (A.2)$$

The reference element can be used to represent the space $L_1$, $L_2$, $L_3$ as shown in Figure A.3. The variables beneath the integral symbol are altered as shown in Eq. A.3.

![Figure A.3: Representation of barycentric coordinates for a reference triangle.](image)

$$\int \int f(x, y) \, dx \, dy = \int \int f(\xi, \eta) \left| J \right| \, d\xi \, d\eta \quad (A.3)$$

where $J$ is the Jacobian matrix of geometrical transformation, as described in Eq. 3.19. By convention, we number the nodes of the reference and physical elements in the positive sense of rotation on the oriented surface, starting with a vertex node.

### A.1.2 3-Node Linear Triangle

Figure A.1 shows the 3-node linear triangle in reference space and physical space, respectively. Accordingly, the shape functions of the 3-node triangle and their derivatives are listed in Table A.2.

### A.1.3 6-Node Curvilinear Triangle

Figure A.4 shows the 6-node curvilinear triangle in reference space and physical space, respectively. Accordingly, the shape functions and their derivatives are listed in Table A.3.
Table A.2: Shape functions for the 3-node triangle and their derivatives

<table>
<thead>
<tr>
<th>Node i</th>
<th>$\phi_i$</th>
<th>$\partial \phi_i / \partial \xi$</th>
<th>$\partial \phi_i / \partial \eta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$1 - \xi - \eta$</td>
<td>$-1$</td>
<td>$-1$</td>
</tr>
<tr>
<td>2</td>
<td>$\xi$</td>
<td>$1$</td>
<td>$0$</td>
</tr>
<tr>
<td>3</td>
<td>$\eta$</td>
<td>$0$</td>
<td>$1$</td>
</tr>
</tbody>
</table>

Figure A.4: Representation of the 6-node triangular element.

Table A.3: Shape functions for the 6-node triangle and their derivatives

<table>
<thead>
<tr>
<th>Node i</th>
<th>$\phi_i$</th>
<th>$\partial \phi_i / \partial \xi$</th>
<th>$\partial \phi_i / \partial \eta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$-\lambda(1 - 2\lambda)$</td>
<td>$1 - 4\lambda$</td>
<td>$1 - 4\lambda$</td>
</tr>
<tr>
<td>2</td>
<td>$-\xi(1 - 2\xi)$</td>
<td>$-1 + 4\xi$</td>
<td>$0$</td>
</tr>
<tr>
<td>3</td>
<td>$-\eta(1 - 2\eta)$</td>
<td>$0$</td>
<td>$-1 + 4\eta$</td>
</tr>
<tr>
<td>4</td>
<td>$4\xi\lambda$</td>
<td>$4(\lambda - \xi)$</td>
<td>$-4\xi$</td>
</tr>
<tr>
<td>5</td>
<td>$4\xi\eta$</td>
<td>$4\eta$</td>
<td>$4\xi$</td>
</tr>
<tr>
<td>6</td>
<td>$4\eta\lambda$</td>
<td>$-4\eta$</td>
<td>$4(\lambda - \eta)$</td>
</tr>
</tbody>
</table>

Note: $\lambda = 1 - \xi - \eta$
A.2 Quadrilateral Elements (two dimensions)

A.2.1 Systems of Coordinates

For all quadrilateral elements, the reference element shown in Figure A.5 is used, where the coordinates \((\xi, \eta)\) can be interpreted as curvilinear coordinates for the physical element. By convention, nodes are numbered in the positive sense of rotation on the oriented surface of the element, starting with a vertex node.

Figure A.5: Representation of the 4-node quadrilateral element.

The variables beneath the integral symbol are altered as shown in Eq. A.4.

\[
\int \int_{x \ y} f(x, y) \, dx \, dy = \int_{-1}^{1} \int_{-1}^{1} f(\xi, \eta) \, |J| \, d\xi \, d\eta \tag{A.4}
\]

A.2.2 4-Node Bilinear Quadrilateral

Figure A.5 shows the 4-node bilinear quadrilateral in reference space and physical space, respectively. Accordingly, the shape functions and their derivatives are listed in Table A.4.

A.2.3 8-Node Curvilinear Quadrilateral

Figure A.6 shows the 8-node curvilinear quadrilateral in reference space and physical space, respectively. Accordingly, the shape functions and their derivatives are listed in Table A.5.
Table A.4: Shape functions for the 4-node quadrilateral and their derivatives

<table>
<thead>
<tr>
<th>Node $i$</th>
<th>$\phi_i$</th>
<th>$\partial \phi_i / \partial \xi$</th>
<th>$\partial \phi_i / \partial \eta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$\frac{1}{4}(1 - \xi)(1 - \eta)$</td>
<td>$\frac{1}{4}(-1 + \eta)$</td>
<td>$-1 + \xi$</td>
</tr>
<tr>
<td>2</td>
<td>$\frac{1}{4}(1 + \xi)(1 - \eta)$</td>
<td>$\frac{1}{4}(1 - \eta)$</td>
<td>$-1 - \xi$</td>
</tr>
<tr>
<td>3</td>
<td>$\frac{1}{4}(1 + \xi)(1 + \eta)$</td>
<td>$\frac{1}{4}(1 + \eta)$</td>
<td>$1 + \xi$</td>
</tr>
<tr>
<td>4</td>
<td>$\frac{1}{4}(1 - \xi)(1 + \eta)$</td>
<td>$\frac{1}{4}(-1 - \eta)$</td>
<td>$1 - \xi$</td>
</tr>
</tbody>
</table>

Figure A.6: Representation of the 8-node quadrilateral element.

Table A.5: Shape functions for the 8-node quadrilateral and their derivatives

<table>
<thead>
<tr>
<th>Node $i$</th>
<th>$\phi_i$</th>
<th>$\partial \phi_i / \partial \xi$</th>
<th>$\partial \phi_i / \partial \eta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$\frac{1}{4}(1 - \xi)(1 - \eta)(-\eta - \xi - 1)$</td>
<td>$\frac{1}{4}(1 - \eta)(2\xi + \eta)$</td>
<td>$\frac{1}{4}(1 - \xi)(\xi + 2\eta)$</td>
</tr>
<tr>
<td>2</td>
<td>$\frac{1}{4}(1 + \xi)(1 - \eta)(-\eta + \xi - 1)$</td>
<td>$\frac{1}{4}(1 - \eta)(2\xi - \eta)$</td>
<td>$\frac{1}{4}(1 + \xi)(2\eta - \xi)$</td>
</tr>
<tr>
<td>3</td>
<td>$\frac{1}{4}(1 + \xi)(1 + \eta)(\eta + \xi - 1)$</td>
<td>$\frac{1}{4}(1 + \eta)(2\xi + \eta)$</td>
<td>$\frac{1}{4}(1 + \xi)(\xi + 2\eta)$</td>
</tr>
<tr>
<td>4</td>
<td>$\frac{1}{4}(1 - \xi)(1 + \eta)(\eta - \xi - 1)$</td>
<td>$\frac{1}{4}(1 + \eta)(2\xi - \eta)$</td>
<td>$\frac{1}{4}(1 - \xi)(2\eta - \xi)$</td>
</tr>
<tr>
<td>5</td>
<td>$\frac{1}{2}(1 - \xi^2)(1 - \eta)$</td>
<td>$-(1 - \eta)\xi$</td>
<td>$-\frac{1}{2}(1 - \xi^2)$</td>
</tr>
<tr>
<td>6</td>
<td>$\frac{1}{2}(1 - \eta^2)(1 + \xi)$</td>
<td>$\frac{1}{2}(1 - \eta^2)$</td>
<td>$-(1 + \xi)\eta$</td>
</tr>
<tr>
<td>7</td>
<td>$\frac{1}{2}(1 - \xi^2)(1 + \eta)$</td>
<td>$-(1 + \eta)\xi$</td>
<td>$\frac{1}{2}(1 - \xi^2)$</td>
</tr>
<tr>
<td>8</td>
<td>$\frac{1}{2}(1 - \eta^2)(1 - \xi)$</td>
<td>$-\frac{1}{2}(1 - \eta^2)$</td>
<td>$-(1 - \xi)\eta$</td>
</tr>
</tbody>
</table>
A.3 Tetrahedral Elements

A.3.1 Systems of Coordinates

For all tetrahedral-shaped elements, the reference element shown in Figure A.7 is used.

![Reference Tetrahedron](image)

(a) TETR4 in reference space
(b) TETR4 in physical space

Figure A.7: Representation of the 4-node tetrahedral element.

Just like for triangles, the coordinates \((\xi, \eta, \zeta)\) can be interpreted as curvilinear coordinates for the physical element. The surface \(\xi = \text{constant}\) (or \(\eta = \text{constant}\) or \(\zeta = \text{constant}\)) is plane, parallel to the faces of the element, in the case of an element with straight edges.

The barycentric coordinates \(L_1, L_2, L_3\) and \(L_4\) are sometimes used to represent a point \(0\) for a straight-edged tetrahedron as shown in Figure A.8.

![Barycentric Coordinates](image)

Figure A.8: Representation of barycentric coordinates for a tetrahedron in physical space.
\[ L_1 = \frac{A_1}{A}; L_2 = \frac{A_2}{A}; L_3 = \frac{A_3}{A}; L_4 = \frac{A_4}{A} \]

\[ A = A_1 + A_2 + A_3 + A_4 \]

\[ L_1 + L_2 + L_3 + L_4 = 1 \]  

(A.5)

where \( A_1, A_2, A_3 \) and \( A_4 \) are the volume of the tetrahedra 0-2-3-4, 0-1-3-4, 0-1-2-4 and 0-1-2-3, respectively. \( A \) is the volume of the whole tetrahedron. The barycentric coordinates are linked to the coordinates \((\xi, \eta, \zeta)\) by the following relation:

\[ L_1 \equiv 1 - \xi - \eta - \zeta, \quad L_2 \equiv \xi, \quad L_3 \equiv \eta, \quad L_4 \equiv \zeta \]

Note that the order of the numbering must be consistent between the reference element and the physical element. Here the first three nodes are taken in the positive sense of rotation about a unit normally oriented toward the interior of the element.

The variables beneath the integral symbol are altered as follows:

\[
\int_x \int_y \int_z f(x, y, z) \, dx dy dz = \int_0^1 \int_0^{1-\zeta} \int_0^{1-\eta-\zeta} f(\xi, \eta, \zeta) |J| \, d\xi d\eta d\zeta
\]

(A.7)

A.3.2 4-node Linear Tetrahedron

Figure A.7 shows the 4-node linear tetrahedron in reference space and physical space, respectively. Accordingly, the shape functions and their derivatives are listed in Table A.6.

Table A.6: Shape functions for the 4-node tetrahedron and their derivatives

<table>
<thead>
<tr>
<th>Node ( i )</th>
<th>( \phi_i )</th>
<th>( \partial \phi_i / \partial \xi )</th>
<th>( \partial \phi_i / \partial \eta )</th>
<th>( \partial \phi_i / \partial \zeta )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( 1 - \xi - \eta - \zeta )</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>2</td>
<td>( \xi )</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>( \eta )</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>( \zeta )</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

A.3.3 10-node Curvilinear Tetrahedron

Figure A.9 shows the 10-node curvilinear tetrahedron in reference space and physical space, respectively. Accordingly, the shape functions and their derivatives are listed in Table A.7.
Table A.7: Shape functions for the 10-node tetrahedron and their derivatives

<table>
<thead>
<tr>
<th>Node i</th>
<th>$\phi_i$</th>
<th>$\partial \phi_i / \partial \xi$</th>
<th>$\partial \phi_i / \partial \eta$</th>
<th>$\partial \phi_i / \partial \zeta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$-\lambda(1 - 2\lambda)$</td>
<td>$1 - 4\lambda$</td>
<td>$1 - 4\lambda$</td>
<td>$1 - 4\lambda$</td>
</tr>
<tr>
<td>2</td>
<td>$-\xi(1 - 2\xi)$</td>
<td>$-1 + 4\xi$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>$-\eta(1 - 2\eta)$</td>
<td>0</td>
<td>$-1 + 4\eta$</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>$-\zeta(1 - 2\zeta)$</td>
<td>0</td>
<td>0</td>
<td>$-1 + 4\zeta$</td>
</tr>
<tr>
<td>5</td>
<td>$4\xi \lambda$</td>
<td>$4(\lambda - \xi)$</td>
<td>$-4\xi$</td>
<td>$-4\xi$</td>
</tr>
<tr>
<td>6</td>
<td>$4\xi \eta$</td>
<td>$4\eta$</td>
<td>$4\xi$</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>$4\eta \lambda$</td>
<td>$-4\eta$</td>
<td>$4(\lambda - \eta)$</td>
<td>$-4\eta$</td>
</tr>
<tr>
<td>8</td>
<td>$4\zeta \lambda$</td>
<td>$-4\zeta$</td>
<td>$-4\zeta$</td>
<td>$4(\lambda - \zeta)$</td>
</tr>
<tr>
<td>9</td>
<td>$4\xi \zeta$</td>
<td>$4\zeta$</td>
<td>0</td>
<td>$4\xi$</td>
</tr>
<tr>
<td>10</td>
<td>$4\eta \zeta$</td>
<td>0</td>
<td>$4\zeta$</td>
<td>$4\eta$</td>
</tr>
</tbody>
</table>

Note: $\lambda = 1 - \xi - \eta - \zeta$
A.4 Hexahedral Elements

A.4.1 8-Node Trilinear Hexahedron

Figure A.10 shows the 8-node trilinear hexahedron in reference space and physical space, respectively. Accordingly, the shape functions and their derivatives are listed in Table A.8.

![Hexahedron](image)

(a) HEXA8 in reference space

(b) HEXA8 in physical space

Figure A.10: Representation of the 8-node hexahedral element.

Table A.8: Shape functions for the 8-node hexahedron and their derivatives

<table>
<thead>
<tr>
<th>Node i</th>
<th>$\phi_i$</th>
<th>$\partial \phi_i / \partial \xi$</th>
<th>$\partial \phi_i / \partial \eta$</th>
<th>$\partial \phi_i / \partial \zeta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$\frac{1}{8}(1 - \xi)(1 - \eta)(1 - \zeta)$</td>
<td>$-\frac{1}{8}(1 - \eta)(1 - \zeta)$</td>
<td>$-\frac{1}{8}(1 - \xi)(1 - \zeta)$</td>
<td>$-\frac{1}{8}(1 - \xi)(1 - \eta)$</td>
</tr>
<tr>
<td>2</td>
<td>$\frac{1}{8}(1 + \xi)(1 - \eta)(1 - \zeta)$</td>
<td>$\frac{1}{8}(1 - \eta)(1 - \zeta)$</td>
<td>$-\frac{1}{8}(1 + \xi)(1 - \zeta)$</td>
<td>$-\frac{1}{8}(1 + \xi)(1 - \eta)$</td>
</tr>
<tr>
<td>3</td>
<td>$\frac{1}{8}(1 + \xi)(1 + \eta)(1 - \zeta)$</td>
<td>$\frac{1}{8}(1 + \eta)(1 - \zeta)$</td>
<td>$\frac{1}{8}(1 + \xi)(1 - \zeta)$</td>
<td>$-\frac{1}{8}(1 + \xi)(1 + \eta)$</td>
</tr>
<tr>
<td>4</td>
<td>$\frac{1}{8}(1 - \xi)(1 + \eta)(1 - \zeta)$</td>
<td>$-\frac{1}{8}(1 + \eta)(1 - \zeta)$</td>
<td>$\frac{1}{8}(1 - \xi)(1 - \zeta)$</td>
<td>$-\frac{1}{8}(1 - \xi)(1 + \eta)$</td>
</tr>
<tr>
<td>5</td>
<td>$\frac{1}{8}(1 - \xi)(1 - \eta)(1 + \zeta)$</td>
<td>$-\frac{1}{8}(1 - \eta)(1 + \zeta)$</td>
<td>$\frac{1}{8}(1 - \xi)(1 + \zeta)$</td>
<td>$\frac{1}{8}(1 - \xi)(1 - \eta)$</td>
</tr>
<tr>
<td>6</td>
<td>$\frac{1}{8}(1 + \xi)(1 - \eta)(1 + \zeta)$</td>
<td>$\frac{1}{8}(1 - \eta)(1 + \zeta)$</td>
<td>$-\frac{1}{8}(1 + \xi)(1 + \zeta)$</td>
<td>$\frac{1}{8}(1 + \xi)(1 - \eta)$</td>
</tr>
<tr>
<td>7</td>
<td>$\frac{1}{8}(1 + \xi)(1 + \eta)(1 + \zeta)$</td>
<td>$\frac{1}{8}(1 + \eta)(1 + \zeta)$</td>
<td>$\frac{1}{8}(1 + \xi)(1 + \zeta)$</td>
<td>$\frac{1}{8}(1 + \xi)(1 + \eta)$</td>
</tr>
<tr>
<td>8</td>
<td>$\frac{1}{8}(1 - \xi)(1 + \eta)(1 + \zeta)$</td>
<td>$-\frac{1}{8}(1 + \eta)(1 + \zeta)$</td>
<td>$\frac{1}{8}(1 - \xi)(1 + \zeta)$</td>
<td>$\frac{1}{8}(1 - \xi)(1 + \eta)$</td>
</tr>
</tbody>
</table>
A.4.2 20-Node Curvilinear Hexahedron

Figure A.11 shows the 20-node curvilinear hexahedron in reference space and physical space, respectively.

The shape functions and their derivatives are presented as follows:
— Vertex nodes:

<table>
<thead>
<tr>
<th>Node $i$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\xi_i$</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>$\eta_i$</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>$\zeta_i$</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

\[
\phi_i = \frac{1}{8}(1 + \xi_i)(1 + \eta_i)(1 + \zeta_i)(-2 + \xi_i + \eta_i + \zeta_i)
\]

\[
\frac{\partial \phi_i}{\partial \xi} = \frac{1}{8}\xi_i(1 + \eta_i)(1 + \zeta_i)(-1 + 2\xi_i + \eta_i + \zeta_i)
\]

\[
\frac{\partial \phi_i}{\partial \eta} = \frac{1}{8}\eta_i(1 + \xi_i)(1 + \zeta_i)(-1 + \xi_i + 2\eta_i + \zeta_i)
\]

\[
\frac{\partial \phi_i}{\partial \zeta} = \frac{1}{8}\zeta_i(1 + \xi_i)(1 + \eta_i)(-1 + \xi_i + \eta_i + 2\zeta_i)
\]
— Nodes on the edges parallel to the $\xi$ axis:

<table>
<thead>
<tr>
<th>Node $i$</th>
<th>9</th>
<th>11</th>
<th>13</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\xi_i$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$\eta_i$</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>$\zeta_i$</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

\[
\phi_i = \frac{1}{4}(1 + \xi^2)(1 + \eta \eta_i)(1 + \zeta \zeta_i)
\]
\[
\frac{\partial \phi_i}{\partial \xi} = -\frac{1}{2} \xi(1 + \eta \eta_i)(1 + \zeta \zeta_i)
\]
\[
\frac{\partial \phi_i}{\partial \eta} = \frac{1}{4} \eta(1 - \xi^2)(1 + \zeta \zeta_i)
\]
\[
\frac{\partial \phi_i}{\partial \zeta} = \frac{1}{4} \zeta_i(1 - \xi^2)(1 + \eta \eta_i)
\]

— Nodes on the edges parallel to the $\eta$ axis:

<table>
<thead>
<tr>
<th>Node $i$</th>
<th>10</th>
<th>12</th>
<th>14</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\xi_i$</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
</tr>
<tr>
<td>$\eta_i$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$\zeta_i$</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

\[
\phi_i = \frac{1}{4}(1 + \xi \xi_i)(1 - \eta^2)(1 + \zeta \zeta_i)
\]
\[
\frac{\partial \phi_i}{\partial \xi} = \frac{1}{4} \xi_i(1 - \eta^2)(1 + \zeta \zeta_i)
\]
\[
\frac{\partial \phi_i}{\partial \eta} = -\frac{1}{2} \eta(1 + \xi \xi_i)(1 + \zeta \zeta_i)
\]
\[
\frac{\partial \phi_i}{\partial \zeta} = \frac{1}{4} \zeta_i(1 + \xi \xi_i)(1 - \eta^2)
\]

— Nodes on the edges parallel to the $\zeta$ axis:
### A.5 Prismatic Elements

#### A.5.1 6-Node Prism

Figure A.12 shows the 6-node prism in reference space and physical space, respectively. Accordingly, the shape functions and their derivatives are listed in Table A.9.

![Figure A.12](image)

(a) PRIS6 in reference space  
(b) PRIS6 in physical space

Figure A.12: Representation of the 6-node prismatic element.

#### A.5.2 15-node Prism

Figure A.13 shows the 15-node prism in reference space and physical space, respectively. Accordingly, the shape functions and their derivatives are listed in Table A.10.
Table A.9: Shape functions for the 6-node prism and their derivatives

<table>
<thead>
<tr>
<th>Node $i$</th>
<th>$\phi_i$</th>
<th>$\partial \phi_i / \partial \xi$</th>
<th>$\partial \phi_i / \partial \eta$</th>
<th>$\partial \phi_i / \partial \zeta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$\lambda a$</td>
<td>$-a$</td>
<td>$-a$</td>
<td>$-\frac{1}{2} \lambda$</td>
</tr>
<tr>
<td>2</td>
<td>$\xi a$</td>
<td>$a$</td>
<td>0</td>
<td>$-\frac{1}{2} \xi$</td>
</tr>
<tr>
<td>3</td>
<td>$\eta a$</td>
<td>0</td>
<td>$a$</td>
<td>$-\frac{1}{2} \eta$</td>
</tr>
<tr>
<td>4</td>
<td>$\lambda b$</td>
<td>$-b$</td>
<td>$-b$</td>
<td>$\frac{1}{2} \lambda$</td>
</tr>
<tr>
<td>5</td>
<td>$\xi b$</td>
<td>$b$</td>
<td>0</td>
<td>$\frac{1}{2} \xi$</td>
</tr>
<tr>
<td>6</td>
<td>$\eta b$</td>
<td>0</td>
<td>$b$</td>
<td>$\frac{1}{2} \eta$</td>
</tr>
</tbody>
</table>

Note: $\lambda = 1 - \xi - \eta$, $a = \frac{1}{2} (1 - \zeta)$, $b = \frac{1}{2} (1 + \zeta)$

Figure A.13: Representation of the 15-node prismatic element.
Table A.10: Shape functions for the 15-node prism and their derivatives

<table>
<thead>
<tr>
<th>Node $i$</th>
<th>$\phi_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$a\lambda(2\lambda - \zeta - 2)$</td>
</tr>
<tr>
<td>2</td>
<td>$a\xi(2\xi - \zeta - 2)$</td>
</tr>
<tr>
<td>3</td>
<td>$a\eta(2\eta - \zeta - 2)$</td>
</tr>
<tr>
<td>4</td>
<td>$b\lambda(2\lambda + \zeta - 2)$</td>
</tr>
<tr>
<td>5</td>
<td>$b\xi(2\xi + \zeta - 2)$</td>
</tr>
<tr>
<td>6</td>
<td>$b\eta(2\eta + \zeta - 2)$</td>
</tr>
<tr>
<td>7</td>
<td>$4a\lambda\xi$</td>
</tr>
<tr>
<td>8</td>
<td>$4a\eta\xi$</td>
</tr>
<tr>
<td>9</td>
<td>$4a\eta\lambda$</td>
</tr>
<tr>
<td>10</td>
<td>$4b\lambda\xi$</td>
</tr>
<tr>
<td>11</td>
<td>$4b\eta\xi$</td>
</tr>
<tr>
<td>12</td>
<td>$4b\eta\lambda$</td>
</tr>
<tr>
<td>13</td>
<td>$(1 - \zeta^2)\lambda$</td>
</tr>
<tr>
<td>14</td>
<td>$(1 - \zeta^2)\xi$</td>
</tr>
<tr>
<td>15</td>
<td>$(1 - \zeta^2)\eta$</td>
</tr>
</tbody>
</table>

Note: $\lambda = 1 - \xi - \eta - \zeta$, $a = \frac{1}{2}(1 - \xi)$, $b = \frac{1}{2}(1 + \xi)$

A.6 Pyramidal Elements

A.6.1 5-Node Pyramid

Figure A.14 shows the 5-node pyramid in reference space and physical space, respectively.

The shape functions of 5-node pyramid and their derivatives are listed in Table A.11

Table A.11: Shape functions for the 5-node pyramid and their derivatives

<table>
<thead>
<tr>
<th>Node $i$</th>
<th>$\phi_i$</th>
<th>$\partial\phi_i/\partial\xi$</th>
<th>$\partial\phi_i/\partial\eta$</th>
<th>$\partial\phi_i/\partial\zeta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$\frac{1}{8}(1 - \xi)(1 - \eta)(1 - \zeta)$</td>
<td>$-\frac{1}{8}(1 - \eta)(1 - \zeta)$</td>
<td>$-\frac{1}{8}(1 - \xi)(1 - \zeta)$</td>
<td>$-\frac{1}{8}(1 - \xi)(1 - \eta)$</td>
</tr>
<tr>
<td>2</td>
<td>$\frac{1}{8}(1 + \xi)(1 - \eta)(1 - \zeta)$</td>
<td>$\frac{1}{8}(1 - \eta)(1 - \zeta)$</td>
<td>$-\frac{1}{8}(1 + \xi)(1 - \zeta)$</td>
<td>$-\frac{1}{8}(1 + \xi)(1 - \eta)$</td>
</tr>
<tr>
<td>3</td>
<td>$\frac{1}{8}(1 + \xi)(1 + \eta)(1 - \zeta)$</td>
<td>$\frac{1}{8}(1 + \eta)(1 - \zeta)$</td>
<td>$\frac{1}{8}(1 + \xi)(1 - \zeta)$</td>
<td>$-\frac{1}{8}(1 + \xi)(1 + \eta)$</td>
</tr>
<tr>
<td>4</td>
<td>$\frac{1}{8}(1 - \xi)(1 + \eta)(1 - \zeta)$</td>
<td>$-\frac{1}{8}(1 + \eta)(1 - \zeta)$</td>
<td>$\frac{1}{8}(1 - \xi)(1 - \zeta)$</td>
<td>$-\frac{1}{8}(1 - \xi)(1 + \eta)$</td>
</tr>
<tr>
<td>5</td>
<td>$\frac{1}{2}(1 + \zeta)$</td>
<td>0</td>
<td>0</td>
<td>$\frac{1}{2}$</td>
</tr>
</tbody>
</table>
Figure A.14: Representation of the 5-node pyramidal element.

A.6.2 13-Node Pyramid

Figure A.15 shows the 13-node pyramid in reference space and physical space, respectively. Accordingly, the shape functions of 13-node pyramid are listed in Table A.12.

Figure A.15: Representation of the 13-node pyramidal element.
Table A.12: Shape functions for the 13-node pyramid

<table>
<thead>
<tr>
<th>Node $i$</th>
<th>$\phi_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$\frac{1}{8}(1 - \xi)(1 - \eta)(1 - \zeta)(-2 - \xi - \eta - \zeta)$</td>
</tr>
<tr>
<td>2</td>
<td>$\frac{1}{8}(1 + \xi)(1 - \eta)(1 - \zeta)(-2 + \xi - \eta - \zeta)$</td>
</tr>
<tr>
<td>3</td>
<td>$\frac{1}{8}(1 + \xi)(1 + \eta)(1 - \zeta)(-2 + \xi + \eta - \zeta)$</td>
</tr>
<tr>
<td>4</td>
<td>$\frac{1}{8}(1 - \xi)(1 + \eta)(1 - \zeta)(-2 - \xi + \eta - \zeta)$</td>
</tr>
<tr>
<td>5</td>
<td>$\frac{1}{8}(1 + \frac{\xi}{2})(1 + \zeta)$</td>
</tr>
<tr>
<td>6</td>
<td>$\frac{1}{8}(1 - \xi^2)(1 - \eta)(1 - \zeta)$</td>
</tr>
<tr>
<td>7</td>
<td>$\frac{1}{8}(1 + \xi)(1 - \eta^2)(1 - \zeta)$</td>
</tr>
<tr>
<td>8</td>
<td>$\frac{1}{8}(1 + \xi^2)(1 + \eta)(1 - \zeta)$</td>
</tr>
<tr>
<td>9</td>
<td>$\frac{1}{8}(1 - \xi)(1 + \eta^2)(1 - \zeta)$</td>
</tr>
<tr>
<td>10</td>
<td>$\frac{1}{8}(1 - \xi^2)(1 - \eta)(1 + \zeta)$</td>
</tr>
<tr>
<td>11</td>
<td>$\frac{1}{8}(1 + \xi)(1 - \eta^2)(1 + \zeta)$</td>
</tr>
<tr>
<td>12</td>
<td>$\frac{1}{8}(1 + \xi^2)(1 + \eta)(1 + \zeta)$</td>
</tr>
<tr>
<td>13</td>
<td>$\frac{1}{8}(1 - \xi)(1 + \eta^2)(1 + \zeta)$</td>
</tr>
</tbody>
</table>
Appendix B

Tables of Gaussian Quadrature Formulas

In this section all the used Gaussian quadrature points and weighting coefficients for various types of two-dimensional and three-dimensional elements are presented:

- §B.1 gives the integration constants of Gaussian quadrature rules for a triangle.
- §B.2 gives the integration constants of Gaussian quadrature rules for a quadrilateral.
- §B.3 gives the integration constants of Gaussian quadrature rules for a tetrahedron.
- §B.4 gives the integration constants of Gaussian quadrature rules for a hexahedron.
- §B.5 gives the integration constants of Gaussian quadrature rules for a prism.
- §B.6 gives the integration constants of Gaussian quadrature rules for a pyramid.

More detailed description is referred to Hesthaven [59], Dhatt [42] and Li [75].

B.1 Triangle

Formulas precisely integrating the \( m \)-th order polynomials:

\[
\int_0^1 \int_0^{1-\xi} f(\xi, \eta) \, d\xi d\eta \simeq \sum_i^r w_i f(\xi_i, \eta_i)
\]
Table B.1: Integration constants of Gaussian quadrature rules for a triangle.

<table>
<thead>
<tr>
<th>Order $p$</th>
<th>No. of point $r$</th>
<th>Coordinates $(\xi_i, \eta_i)$</th>
<th>Weights $w_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>(0.3333333333333333, 0.3333333333333333)</td>
<td>0.5000000000000000</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>(0.6666666666666667, 0.1666666666666667, 0.1666666666666667)</td>
<td>0.1666666666666667</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(0.1666666666666667, 0.6666666666666667, 0.1666666666666667)</td>
<td>0.1666666666666667</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(0.1666666666666667, 0.6666666666666667, 0.1666666666666667)</td>
<td>0.1666666666666667</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>(0.3333333333333333, 0.3333333333333333, 0.3333333333333333)</td>
<td>-0.2812500000000000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(0.2000000000000000, 0.2000000000000000, 0.2000000000000000)</td>
<td>0.2604166666666667</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(0.6000000000000000, 0.6000000000000000, 0.6000000000000000)</td>
<td>0.2604166666666667</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(0.1666666666666667, 0.1666666666666667, 0.1666666666666667)</td>
<td>0.2604166666666667</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>(0.3333333333333333, 0.3333333333333333, 0.3333333333333333)</td>
<td>0.1125000000000000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(0.09715871789770, 0.47014206105115, 0.47014206105115)</td>
<td>0.06619706394253</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(0.47014206105115, 0.05971587189770, 0.47014206105115)</td>
<td>0.06619706394253</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(0.47014206105115, 0.47014206105115, 0.47014206105115)</td>
<td>0.06619706394253</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(0.79742685353807, 0.101286507323456, 0.101286507323456)</td>
<td>0.062969590272414</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(0.101286507323456, 0.79742685353807, 0.101286507323456)</td>
<td>0.062969590272414</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(0.101286507323456, 0.101286507323456, 0.101286507323456)</td>
<td>0.062969590272414</td>
</tr>
</tbody>
</table>

Table B.2: Integration constants of Gaussian quadrature rules for a quadrilateral.

<table>
<thead>
<tr>
<th>Order $p$</th>
<th>No. of points $r$</th>
<th>Coordinates $(\xi_i, \eta_i)$</th>
<th>Weights $w_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>(0.0000000000000000, 0.0000000000000000)</td>
<td>4.0000000000000000</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>(0.577350269189625, -0.577350269189625, 0.577350269189625, 0.577350269189625)</td>
<td>1.0000000000000000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(0.577350269189625, 0.577350269189625, 0.577350269189625, 0.577350269189625)</td>
<td>1.0000000000000000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(0.577350269189625, 0.577350269189625, 0.577350269189625, 0.577350269189625)</td>
<td>1.0000000000000000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(0.577350269189625, 0.577350269189625, 0.577350269189625, 0.577350269189625)</td>
<td>1.0000000000000000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(0.577350269189625, 0.577350269189625, 0.577350269189625, 0.577350269189625)</td>
<td>1.0000000000000000</td>
</tr>
<tr>
<td>5</td>
<td>9</td>
<td>(0.77459669241483, 0.77459669241483, 0.77459669241483, 0.77459669241483)</td>
<td>0.308641975308642</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(0.77459669241483, 0.77459669241483, 0.77459669241483, 0.77459669241483)</td>
<td>0.493827160493827</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(0.77459669241483, 0.77459669241483, 0.77459669241483, 0.77459669241483)</td>
<td>0.308641975308642</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(0.77459669241483, 0.77459669241483, 0.77459669241483, 0.77459669241483)</td>
<td>0.493827160493827</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(0.77459669241483, 0.77459669241483, 0.77459669241483, 0.77459669241483)</td>
<td>0.308641975308642</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(0.77459669241483, 0.77459669241483, 0.77459669241483, 0.77459669241483)</td>
<td>0.493827160493827</td>
</tr>
</tbody>
</table>

B.2 Quadrilateral

Formulas integrating exactly the polynomials of order $m$:

$$\int_{-1}^{1} \int_{-1}^{1} f(\xi, \eta) \, d\xi d\eta \simeq \sum_{i} w_{i} f(\xi_{i}, \eta_{i})$$
Table B.3: Integration constants of Gaussian quadrature rules for a tetrahedron.

<table>
<thead>
<tr>
<th>Order $p$</th>
<th>No. of points $r$</th>
<th>Coordinates $(\xi_i, \eta_i, \zeta_i)$</th>
<th>Weights $w_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>$(0.250000000000000, 0.250000000000000, 0.250000000000000)$</td>
<td>$0.166666666666667$</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>$(0.138196601125011, 0.138196601125011, 0.138196601125011)$</td>
<td>$0.041666666666667$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$(0.585410196624969, 0.138196601125011, 0.138196601125011)$</td>
<td>$0.041666666666667$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$(0.138196601125011, 0.585410196624969, 0.138196601125011)$</td>
<td>$0.041666666666667$</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>$(0.250000000000000, 0.250000000000000)$</td>
<td>$-0.133333333333333$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$(0.166666666666667, 0.166666666666667, 0.166666666666667)$</td>
<td>$0.075000000000000$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$(0.500000000000000, 0.166666666666667, 0.166666666666667)$</td>
<td>$0.075000000000000$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$(0.166666666666667, 0.500000000000000, 0.166666666666667)$</td>
<td>$0.075000000000000$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$(0.166666666666667, 0.166666666666667, 0.500000000000000)$</td>
<td>$0.075000000000000$</td>
</tr>
</tbody>
</table>

B.3 Tetrahedron

Formulas integrating exactly the polynomials of order $m$:

$$
\int_{-1}^{1} \int_{-1}^{1} \int_{-1}^{1} f(\xi, \eta, \zeta) \, d\xi d\eta d\zeta \simeq \sum_{i} w_i f(\xi_i, \eta_i, \zeta_i)
$$

B.4 Hexahedron

Table B.4: Integration constants of Gaussian quadrature rules for a hexahedron.

<table>
<thead>
<tr>
<th>Order $p$</th>
<th>No. of points $r$</th>
<th>Coordinates $(\xi_i, \eta_i, \zeta_i)$</th>
<th>Weights $w_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>8</td>
<td>$( -0.577350269189625, -0.577350269189625, -0.577350269189625 )$</td>
<td>$1.000000000000000$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$( -0.577350269189625, -0.577350269189625, 0.577350269189625 )$</td>
<td>$1.000000000000000$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$( -0.577350269189625, 0.577350269189625, -0.577350269189625 )$</td>
<td>$1.000000000000000$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$( -0.577350269189625, 0.577350269189625, 0.577350269189625 )$</td>
<td>$1.000000000000000$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$( 0.577350269189625, -0.577350269189625, -0.577350269189625 )$</td>
<td>$1.000000000000000$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$( 0.577350269189625, -0.577350269189625, 0.577350269189625 )$</td>
<td>$1.000000000000000$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$( 0.577350269189625, 0.577350269189625, -0.577350269189625 )$</td>
<td>$1.000000000000000$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$( 0.577350269189625, 0.577350269189625, 0.577350269189625 )$</td>
<td>$1.000000000000000$</td>
</tr>
</tbody>
</table>

Formulas precisely integrating the $m$-th order polynomials:

$$
\int_{-1}^{1} \int_{-1}^{1} \int_{-1}^{1} f(\xi, \eta, \zeta) \, d\xi d\eta d\zeta \simeq \sum_{i} w_i f(\xi_i, \eta_i, \zeta_i)
$$

162
Table B.5: Integration constants of Gaussian quadrature rules for a prism.

<table>
<thead>
<tr>
<th>Order $p$</th>
<th>No. of points $r$</th>
<th>Coordinates $(\xi_i, \eta_i, \zeta_i)$</th>
<th>Weights $w_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>($0.333333333333333, 0.333333333333333, 0.000000000000000)$</td>
<td>1.000000000000000</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>($0.666666666666667, 0.166666666666667, -0.577350269189625, 0.166666666666667, 0.166666666666667, 0.166666666666667$)</td>
<td>0.166666666666667</td>
</tr>
</tbody>
</table>

B.5 Prism

Formulas precisely integrating the $m$-th order polynomials:

$$\int_{0}^{1} \int_{0}^{1-\xi} \int_{-1}^{1} f(\xi, \eta, \zeta) \, d\xi d\eta d\zeta \simeq \sum_{i} w_i f(\xi_i, \eta_i, \zeta_i)$$

B.6 Pyramid

Table B.6: Integration constants of Gaussian quadrature rules for a pyramid.

<table>
<thead>
<tr>
<th>Order $p$</th>
<th>No. of points $r$</th>
<th>Coordinates $(\xi_i, \eta_i, \zeta_i)$</th>
<th>Weights $w_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>($0.000000000000000, 0.000000000000000, -0.500000000000000)$</td>
<td>4.740740740740741</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>($-0.584237394672177, -0.584237394672177, -0.666666666666667, 0.584237394672177, -0.666666666666667$)</td>
<td>0.810000000000000</td>
</tr>
</tbody>
</table>

Formulas precisely integrating the $m$-th order polynomials:

$$\int_{\xi} \int_{\eta} \int_{\zeta} f(\xi, \eta, \zeta) \, d\xi d\eta d\zeta \simeq \sum_{i} w_i f(\xi_i, \eta_i, \zeta_i)$$

163
Appendix C

Flux Jacobians

In this section the inviscid flux Jacobian matrices are presented as follows:

\[
\frac{\partial F_x}{\partial U} = \begin{pmatrix}
0 & 1 & 0 & 0 & 0 \\
\varphi - u^2 & (3 - \gamma)u & (1 - \gamma)v & (1 - \gamma)w & \gamma - 1 \\
-uv & v & u & 0 & 0 \\
-uw & w & 0 & u & 0 \\
(\varphi - h)u & h - (\gamma - 1)u^2 & (1 - \gamma)uw & (1 - \gamma)uw & \gamma u
\end{pmatrix}
\] (C.1)

\[
\frac{\partial F_y}{\partial U} = \begin{pmatrix}
0 & 0 & 1 & 0 & 0 \\
-vu & v & u & 0 & 0 \\
\varphi - v^2 & (1 - \gamma)u & (3 - \gamma)v & (1 - \gamma)w & \gamma - 1 \\
-vw & 0 & w & v & 0 \\
(\varphi - h)v & (1 - \gamma)vw & h - (\gamma - 1)v^2 & (1 - \gamma)vw & \gamma v
\end{pmatrix}
\] (C.2)

\[
\frac{\partial F_z}{\partial U} = \begin{pmatrix}
0 & 0 & 0 & 1 & 0 \\
-wu & w & 0 & u & 0 \\
-wv & 0 & w & v & 0 \\
\varphi - w^2 & (1 - \gamma)u & (1 - \gamma)v & (3 - \gamma)w & \gamma - 1 \\
(\varphi - h)w & (1 - \gamma)wu & (1 - \gamma)wv & h - (\gamma - 1)w^2 & \gamma w
\end{pmatrix}
\] (C.3)

where \( \varphi = (\gamma - 1)(u^2 + v^2 + w^2)/2 \) and \( h = e + p/\rho \).
Appendix D

Configuration of a Delta Wing

An extracted figure from Reference [62] is presented in this section to illustrate the geometric configuration of a sharp-edged slender delta wing.

Figure D.1: Configuration of a sharp-edged slender delta wing

Figure D.2: Wind-tunnel model of delta wing X = 1
N = Quarter chord point of mean aerodynamic chord, situated at x/c = 0.5
Dimensions: Length: c = 750 mm
Span: b = 575 mm
Thickness: d = 16 mm