ABSTRACT

LIU, XIAODONG. A Parallel Implicit Reconstructed Discontinuous Galerkin Method for Compressible Turbulent Flows on 3D Hybrid Grids. (Under the direction of Dr. Hong Luo.)

In order to reduce the high computing costs associated with the Discontinuous Galerkin (DG) methods, the reconstructed discontinuous Galerkin (rDG) method using a Taylor basis has been developed for the solution of the compressible Euler/Navier-Stokes equations on arbitrary 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 objective of this PhD work is to develop a parallel, implicit rDG method for the solution of the compressible 3D turbulent flows on hybrid grids, e.g., steady cases (RANS), and unsteady cases (ILES and DNS).

For steady cases, in this work, a modified one-equation SA turbulence model is used to close the RANS system, guaranteeing the stability of high-order discretization for turbulence model equation. However, this modified SA model is discretized using the underlying DG(P₁) method. Furthermore, rDG(P₁P₂) using a Hermite WENO reconstruction is used to guarantee the stability of the developed rDG method. And it has also been validated that WENO reconstruction could still keep the accuracy of the solution for 3D turbulent flows. A number of benchmark test cases based on a set of uniformly refined quadratic curved meshes are presented to assess the performance of the resultant rDG(P₁P₂) method for turbulent flow problems. The numerical results demonstrate that the rDG(P₁P₂) method is able to obtain reliable and accurate solutions to 3D compressible turbulent flows at a cost slightly higher than its underlying second-order DG(P₁) method, outperforming the standard third-order DG(P₂) method in terms of both computing costs and storage requirements. Whereas, for ILES and DNS, the unsteady Navier-Stokes equations have to be solved. In our work, a class of linearly implicit Runge-Kutta (IRK) time integration methods, namely, Rosenbrock-Wanner methods (ROW) have been used to complete the time marching. ROW methods are derived by linearizing a diagonally IRK (DIRK) scheme, and replace the non-linear systems with a sequence of linear systems, in which some attractive properties in terms of stability are lost. However, as a trade-off, the computational costs per time step are reduced. A variety of test cases, ranging from inviscid flows to viscous flows, are presented to assess the performance of these schemes. Finally, Rosenbrock-Wanner methods have been successfully applied for the solution of implicit LES (ILES), e.g., "3D lid driven cavity problem" and DNS, e.g., "3D Taylor-Green Vortex problem". Numerical experiments demonstrate that the third-order ROW scheme can not only achieve the designed formal order of temporal discretization, but also require significantly less...
computing time than its counterpart, i.e., explicit first stage singly diagonally implicit Runge-Kutta (ESDIRK), to converge to the same level of discretization errors for all the flow simulations in our study, indicating that the ROW methods provide attractive alternatives for the higher-order time-accurate integration of the unsteady compressible Navier-Stokes equations.

Both steady and unsteady problems need to solve the resulting linear system of equations. For rDG(P₁P₂) method, the Jacobian matrix is evaluated based on the linearization of the underlying second-order DG(P₁) right hand side with respect to solution vector. In context of DG(P₁), Automatic differentiation (AD) has been implemented to obtain the exact Jacobian matrices without any approximation. 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. The parallelization in the rDG method is based on a message passing interface (MPI) programming paradigm, where the METIS library is used to partition a mesh.

In summary, the developed rDG method shows great potential to become a viable, attractive, competitive and ultimately superior high-order method for application to the real-world turbulent flows over complex geometries to the traditional finite volume method.
© Copyright 2016 by Xiaodong Liu
All Rights Reserved
A Parallel Implicit Reconstructed Discontinuous Galerkin Method
for Compressible Turbulent Flows on 3D Hybrid Grids

by
Xiaodong Liu

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
2016

APPROVED BY:

Dr. Jack R. Edwards Jr.          Dr. Hassan A. Hassan
Dr. Zhilin Li                  Dr. Hong Luo
Chair of Advisory Committee
DEDICATION

To my family.
Your love and support makes this work possible.
BIOGRAPHY

The author was born in one village in Pingdingshan, one city of Henan Province, China. He received the Bachelor’s degree of engineering in Thermal Energy and Power Engineering at Zhengzhou University (ZZU), Zhengzhou, Henan Province, China, in July 2010. He then attended the graduate school at University of Chinese Academy of Sciences in August 2010. He received the Master’s degree of engineering in refrigeration and cryogenic engineering in July 2013. The author was then admitted by the doctoral program of Department of Mechanical and Aerospace Engineering at North Carolina State University (NCSU), Raleigh, North Carolina, USA. His academic advisor is Dr. Hong Luo.
ACKNOWLEDGEMENTS

First and most important, I would like to thank my academic advisor Dr. Hong Luo for his great support and help with my research work during 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 also like to thank my former group member Dr. Yidong Xia. He gives me a lot of help to learn a lot of useful CFD techniques and helps me on my paper writing. I would also like to thank other group members for their help, Lijun Xuan, Lixiang Luo, Chuanjin Wang, Jialin Lou, Aditya K. Pandare, Aditya Kashi and Chad Rollins.

I would also like to thank Dr. Jian Cheng and Dr. Xiaoquan yang for their time to discuss with me on my research.

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

## LIST OF TABLES

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## Chapter 1 Introduction

1.1 Background of Turbulent Flows ................................. 1
1.2 Reconstructed Discontinuous Galerkin method .................... 4
1.3 Effective time integration schemes .............................. 10
1.4 Scope ........................................ 15

## Chapter 2 Governing Equations of Fluid Dynamics ................. 16

2.1 RANS-SA Equations ...................................... 16
2.2 Navier-Stokes Equations ................................... 20
2.3 Nondimensionalization ....................................... 20

## Chapter 3 Reconstructed Discontinuous Galerkin Spatial Discretization .... 24

3.1 Discontinuous Galerkin methods ................................ 24
  3.1.1 Weak Formulation ..................................... 24
  3.1.2 Basis Functions ....................................... 28
3.2 Reconstructed discontinuous Galerkin methods ................... 32
  3.2.1 Least-Squares Reconstruction .............................. 32
  3.2.2 WENO Reconstruction at P_2: WENO(P_1P_2) .................. 35
  3.2.3 WENO Reconstruction at P_1: HWENO(P_1P_2) ................. 37
3.3 Numerical Integration ....................................... 39
3.4 Implementation of Cufed Elements ............................... 40
  3.4.1 Transformation from physical to reference space .......... 43
  3.4.2 Wall Distance ....................................... 46
3.5 Boundary Conditions ....................................... 47
  3.5.1 Characteristic Boundary ................................ 48
  3.5.2 Slip Wall / Symmetry Boundary ............................ 49
  3.5.3 No-Slip Adiabatic Wall Boundary .......................... 50
  3.5.4 Periodic Boundary ..................................... 50

## Chapter 4 Temporal Integration Methods .......................... 53

4.1 Explicit Time Integration .................................... 54
4.2 Implicit Time Integration .................................... 55
4.3 p-multigrid solver ........................................ 57
4.4 Solution of the Linear System of Equations ..................... 59
  4.4.1 Jacobian matrix ...................................... 60
  4.4.2 Solver ........................................... 66

## Chapter 5 Parallel implementation ............................... 71

5.1 Domain Partition ........................................... 71
5.2 Interprocessor Communication ........................................... 72
5.3 Parallel Performance ......................................................... 76

Chapter 6 Numerical Results ..................................................... 78
6.1 Steady Cases ................................................................. 78
  6.1.1 Zero Pressure Gradient Flat Plate ................................. 79
  6.1.2 2D Turbulent Flow over a 2D Bump in a Channel ............. 84
  6.1.3 Turbulent Flow over a NACA0012 Airfoil ......................... 89
  6.1.4 Turbulent Flow over a 3D Sinusoidal Bump in a Channel .... 96
  6.1.5 3D Turbulent Flow over a 3D Hemisphere Cylinder .......... 102
  6.1.6 2D Transonic Turbulent Flow over RAE2822 airfoil ........... 106
6.2 Unsteady Cases ............................................................. 111
  6.2.1 Convection of an Isentropic Vortex in Inviscid Flows ......... 111
  6.2.2 Von-Karman Vortex Street ........................................... 115
  6.2.3 Flow past a SD7003 Airfoil ........................................... 119
  6.2.4 3D Lid Driven Cavity ................................................ 123
  6.2.5 Direct Numerical Simulation of the Taylor-Green Vortex .... 129

Chapter 7 Conclusions ............................................................ 140
  7.1 Summary of Completed Work ........................................... 140
  7.2 Outlook of Future Work ................................................ 141

References ................................................................. 143

Appendices .............................................................. 154
  Appendix A Various Types of Elements .................................. 155
    A.1 Triangular Elements (two dimensions) ......................... 155
        A.1.1 6-Node Curvilinear Triangle ............................... 155
    A.2 Quadrilateral Elements (two dimensions) .................... 157
        A.2.1 9-Node Curvilinear Quadrilateral ....................... 157
    A.3 Tetrahedral Elements ............................................ 158
        A.3.1 10-node Curvilinear Tetrahedron ......................... 158
    A.4 Hexahedral Elements ............................................ 159
        A.4.1 27-Node Curvilinear Hexahedron .......................... 159
    A.5 Prismatic Elements ............................................. 161
        A.5.1 18-node Prism .............................................. 161
  Appendix B Tables of coefficients of the Rosenbrock schemes considered in this thesis 163
LIST OF TABLES

Table 2.1 Reference variables for nondimensionalization of the governing equations  
Table 3.1 Cost analysis for different numerical methods on a quadratic curved hexahedral grid  
Table 3.2 Pseudo code: identifying the opposing one of one periodic boundary face  
Table 4.1 A simple source program  
Table 4.2 Source code generated by AD using tangent mode  
Table 4.3 Source code generated by AD using multi-directional mode  
Table 4.4 Pseudo code: contribution of domain integrals for the block diagonal matrix $D$, lower matrix $L$ and upper matrix $U$  
Table 4.5 Flowchart of the GMRES algorithm  
Table 5.1 Pseudo code: MPI communication for explicit method  
Table 5.2 The measurement of speed up and parallel efficiency using rDG($P_1P_2$)  
Table 6.1 Grid information for turbulent flow over a flat plate at $Ma_{\infty} = 0.2$, and $Re_{\infty} = 5 \times 10^6$  
Table 6.2 The quadratic meshes information for 2D turbulent flow over a 2D bump airfoil.  
Table 6.3 The quadratic meshes information for 2D turbulent flow over a 2D NACA0012 airfoil.  
Table 6.4 Computed lift and drag coefficients for the NACA0012 with $Ma_{\infty} = 0.15$, $AoA = 15^0$, and $Re_{\infty} = 6 \times 10^6$ using DG($P_1$) and rDG($P_1P_2$) on Level 3 hexahedral grid.  
Table 6.5 The quadratic meshes information for 3D turbulent flow over a 3D bump.  
Table 6.6 The quadratic meshes information for 3D turbulent flow over a 3D hemisphere cylinder.  
Table 6.7 The quadratic meshes information for 2D transonic turbulent flow over a RAE2822 airfoil.  
Table 6.8 Computed lift and drag coefficients for the RAE2822 with $Ma_{\infty} = 0.734$, $AoA = 2.79^0$, and $Re_{\infty} = 6.5 \times 10^6$ using DG($P_1$) and rDG($P_1P_2$) on quadratic curved hexahedral grid.  
Table 6.9 The investigation of relative error tolerances for the non-linear and linear iterations about the ESDIRK3.  
Table 6.10 The investigation of relative error tolerances for the linear iterations about the ROW.  
Table 6.11 Relative error tolerance for the non-linear and linear iterations.  
Table 6.12 Comparison of the CPU time (evaluated by running on 256 cores) between the ROW and ESDIRK3 for computing a viscous flow past a cylinder.  
Table 6.13 Comparison of the CPU time (evaluated by running on 256 cores) between the ROW and ESDIRK3 for computing a viscous flow past an SD7003 airfoil.
Table 6.14 Comparison of the CPU time (evaluated by running on 256 cores) between the ROW and ESDIRK3 for computing the LES of a lid driven cavity \((x:y:z = 1:1:0.5)\) at \(Re = 10,000\).

Table 6.15 Comparison of the CPU time (evaluated by running on 240 cores) between the ROW and ESDIRK3 for computing the Taylor-Green vortex at \(Re=1600\).

Table A.1 Abbreviation and description for various types of elements
Table A.2 Shape functions for the 6-node triangle and their derivatives
Table A.3 Shape functions for the 9-node quadrilateral and their derivatives
Table A.4 The coordinates of nodes in reference space for 10-node Curvilinear Tetrahedron
Table A.5 Shape functions for the 10-node tetrahedron and their derivatives
Table A.6 The coordinates of nodes in reference space for 27-node Curvilinear Hexahedron
Table A.7 The coordinates of nodes in reference space for 18-node Curvilinear Prism
Table B.1 Set of coefficients for ROSI2PW
Table B.2 Set of coefficients for ROS34PRW
Table B.3 Set of coefficients for ROS34PW2
### LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Representation of polynomial solutions using finite element shape functions: (a) $Q_1/P_1$; (b) $Q_2/P_2$.</td>
<td>28</td>
</tr>
<tr>
<td>3.2</td>
<td>Representation of polynomial solutions using a Taylor series expansion.</td>
<td>29</td>
</tr>
<tr>
<td>3.3</td>
<td>Representation of the four face-neighboring cells surrounding a tetrahedron (a) and the six face-neighboring cells surrounding a hexahedron (b).</td>
<td>33</td>
</tr>
<tr>
<td>3.4</td>
<td>Transformation of an element in $(x, y)$ physical space into a canonical element reference $(\xi, \eta)$ space.</td>
<td>41</td>
</tr>
<tr>
<td>3.5</td>
<td>Transformation of an element in $(x, y)$ physical space into a canonical element reference $(\xi, \eta)$ space.</td>
<td>41</td>
</tr>
<tr>
<td>3.6</td>
<td>Transformation of an element in $(x, y, z)$ physical space into a canonical element reference $(\xi, \eta, \zeta)$ space.</td>
<td>42</td>
</tr>
<tr>
<td>3.7</td>
<td>Transformation of an element in $(x, y)$ physical space into a canonical element reference $(\xi, \eta)$ space.</td>
<td>43</td>
</tr>
<tr>
<td>3.8</td>
<td>Transformation of an element in $(x, y, z)$ physical space into a canonical element reference $(\xi, \eta, \zeta)$ space.</td>
<td>46</td>
</tr>
<tr>
<td>5.1</td>
<td>The schematic diagram of domain partition.</td>
<td>73</td>
</tr>
<tr>
<td>6.1</td>
<td>The overview of the hexahedral grid (Level 3) for the case, turbulence flow over a flat plate.</td>
<td>81</td>
</tr>
<tr>
<td>6.2</td>
<td>The eddy viscosity contours (nondimensionalized by freestream laminar viscosity $\mu_\infty$) using rDG($P_1P_2$) method on Level 3 hexahedral grid.</td>
<td>81</td>
</tr>
<tr>
<td>6.3</td>
<td>The logarithmic plot of surface friction coefficient distribution over the flat plate obtained by DG($P_1$), rDG($P_1P_2$), WENO($P_1P_2$) methods on Level 3 hexahedral grid.</td>
<td>82</td>
</tr>
<tr>
<td>6.4</td>
<td>Eddy viscosity ratio distribution at $x=0.97$, obtained by DG($P_1$), rDG($P_1P_2$), and WENO($P_1P_2$) methods.</td>
<td>82</td>
</tr>
<tr>
<td>6.5</td>
<td>The grid convergence of drag coefficient obtained by DG($P_1$), rDG($P_1P_2$), and WENO($P_1P_2$) methods.</td>
<td>83</td>
</tr>
<tr>
<td>6.6</td>
<td>Convergence histories using DG($P_1$), rDG($P_1P_2$), and WENO($P_1P_2$) on Level 3 hexahedral grid.</td>
<td>83</td>
</tr>
<tr>
<td>6.7</td>
<td>Level 2 hexahedral curved mesh.</td>
<td>85</td>
</tr>
<tr>
<td>6.8</td>
<td>The eddy viscosity contours (nondimensionalized by freestream laminar viscosity $\mu_\infty$) using the rDG($P_1P_2$) method along the 2D wall bump.</td>
<td>86</td>
</tr>
<tr>
<td>6.9</td>
<td>The friction coefficient distribution along the 2D wall bump.</td>
<td>86</td>
</tr>
<tr>
<td>6.10</td>
<td>Grid convergence for lift coefficient $C_L$.</td>
<td>87</td>
</tr>
<tr>
<td>6.11</td>
<td>Grid convergence for drag coefficient $C_D$.</td>
<td>87</td>
</tr>
<tr>
<td>6.12</td>
<td>Convergence histories using DG($P_1$), and rDG($P_1P_2$) on Level 3 hexahedral grid.</td>
<td>88</td>
</tr>
<tr>
<td>6.13</td>
<td>The nondimensionalized turbulent eddy viscosity using rDG($P_1P_2$) on Level 2 hexahedral grid for $AoA = 10^0$.</td>
<td>90</td>
</tr>
</tbody>
</table>
Figure 6.14 Plot of the quadratic Level 2 hexahedral for 2D turbulent flow over the NACA0012 airfoil. ................................................................. 91
Figure 6.15 The grid convergence of drag coefficient $C_D$ for turbulent flow over 2D NACA0012 airfoil for $\alpha = 0^\circ$. ................................. 92
Figure 6.16 Convergence histories using DG(P1), rDG(P1P2) on Level 2 hexahedral grid for $\alpha = 0^\circ$. ................................................................. 92
Figure 6.17 The grid convergence of lift coefficient $C_L$ and drag coefficient $C_D$ for turbulent flow over 2D NACA0012 airfoil for $\alpha = 0^\circ$. 93
Figure 6.18 The friction coefficient $C_f$ distribution for turbulent flow over 2D NACA0012 airfoil for different $\alpha$ (a)$\alpha = 0^\circ$, (b)$\alpha = 10^\circ$. ............... 94
Figure 6.19 The friction coefficient $C_f$ distribution for turbulent flow over 2D NACA0012 airfoil for different $\alpha$ (c)$\alpha = 15^\circ$. ............................... 95
Figure 6.20 The Level 1 quadratic hexahedral mesh for turbulent flow over a 3D sinusoidal bump in a Channel. ...................................................... 97
Figure 6.21 The grid convergence of lift and drag coefficient using DG(P1) on the first 3 quadratic meshes for turbulent flow over a 3D sinusoidal bump in a Channel. ................................................................. 98
Figure 6.22 The grid convergence of pressure and viscous drag coefficients using DG(P1) on the first 3 quadratic meshes for turbulent flow over a 3D sinusoidal bump in a Channel. ................................................................. 98
Figure 6.23 The lift and drag coefficient using DG(P1) and rDG(P1P2) on the fourth quadratic mesh for turbulent flow over a 3D sinusoidal bump in a Channel. ................................................................. 99
Figure 6.24 The pressure and viscous drag coefficient using DG(P1) and rDG(P1P2) on the fourth quadratic mesh for turbulent flow over a 3D sinusoidal bump in a Channel. ................................................................. 99
Figure 6.25 The logarithmic residual using DG(P1) and rDG(P1P2) on level 4 quadratic mesh for turbulent flow over a 3D sinusoidal bump in a Channel. ................................................................. 100
Figure 6.26 The nondimensionalized turbulent eddy viscosity at different locations on Level 4 quadratic mesh using rDG(P1P2) .................................. 101
Figure 6.27 The Level 3 quadratic hexahedral mesh for turbulent flow over a 3D hemisphere cylinder. ................................................................. 103
Figure 6.28 The grid convergence of drag coefficient $C_D$ using DG(P1) and WENO(P1P2) ................................................................. 103
Figure 6.29 The pressure coefficient distribution using DG(P1) and WENO(P1P2) on the Level 3 quadratic mesh ................................................................. 103
Figure 6.30 The friction coefficient distribution using DG(P1) and WENO(P1P2) on the Level 3 quadratic mesh ................................................................. 104
Figure 6.31 The cell-averaged nondimensionalized turbulent eddy viscosity contours using WENO(P1P2) on the Level 3 quadratic mesh ................. 105
Figure 6.32 The logarithmic residual convergence histories using DG(P1) and WENO(P1P2) on the Level 3 quadratic mesh ................................................................. 105
Figure 6.33 Plot of the quadratic hexahedral mesh for 2D transonic turbulent flow over the RAE airfoil. ................................................................. 107
Figure 6.34 Pressure contours distribution for 2D transonic turbulent flow over the RAE airfoil with DG(P1) method. ........................................... 108
Figure 6.35 Pressure contours distribution for 2D transonic turbulent flow over the RAE airfoil with HWENO(P1P2) method. .................... 109
Figure 6.36 Pressure and friction coefficient distribution for 2D transonic turbulent flow over the RAE airfoil. ................................. 110
Figure 6.37 (a) The computational grid used for convection of an isentropic vortex in inviscid flow. (b) Density distribution for ESDIRK3 at t=10. .... 113
Figure 6.38 Temporal convergence of rDG(P1P2) with different time integration schemes obtained for convection of an isentropic vortex. (a)L2 norm versus dt and (b)L2 norm versus CPU time. ................................. 114
Figure 6.39 Mesh used for computing a viscous flow past a cylinder (nelem = 10204, npoin = 20800, nboun = 20800). ................................. 115
Figure 6.40 Time history of the computed lift and drag coefficients by ROS34PRW + rDG(P1P2) for flow past a cylinder at $M_\infty = 0.2$ and $Re_\infty = 200$. ................................. 116
Figure 6.41 Time history of the computed lift and drag coefficients by ROSI2PW + rDG(P1P2) for flow past a cylinder at $M_\infty = 0.2$ and $Re_\infty = 200$. ................................. 117
Figure 6.42 Grid used for computing the unsteady viscous flow past a SD7003 airfoil. ................................. 119
Figure 6.43 Computed pressure contours in the flow field for flow past a SD7003 airfoil at $M_\infty = 0.2$, $\alpha = 4^\circ$, and $Re_\infty = 10,000$. ................................. 120
Figure 6.44 Computed vorticity contours in the flow field for flow past a SD7003 airfoil at $M_\infty = 0.2$, $\alpha = 4^\circ$, and $Re_\infty = 10,000$. ................................. 120
Figure 6.45 Computed velocity vector in the flow field for flow past a SD7003 airfoil at $M_\infty = 0.1$, $\alpha = 4^\circ$, and $Re_\infty = 10,000$. ................................. 120
Figure 6.46 Time variation of lift (left) and drag (right) coefficients at $Re = 10,000$: comparison between ROSI2PW and ROS34PRW. ................. 122
Figure 6.47 The hexahedral grid (64 x 64 x 32 points) for the LES of a lid driven cavity ($x : y : z = 1 : 1 : 0.5$) at $Re = 10,000$. ................................. 123
Figure 6.48 Comparison of the normalized mean velocity components $< u > /u_B$ and $< v > /u_B$ in the span-wise mid-plane with the classical experimental data by Prasad & Koseff(1989) and numerical data by Zang et.al (1993), for the LES of a lid driven cavity ($x : y : z = 1 : 1 : 0.5$) at $Re = 10,000$. ................................. 126
Figure 6.49 Comparison of the scaled RMS velocity components $10 < u'u'>^{0.5}/u_B$ and $10 < v'v'>^{0.5}/u_B$ in the span-wise midplane with the classical experimental data by Prasad & Koseff (1989) and numerical data by Zang et.al (1993), for the LES of a lid driven cavity ($x : y : z = 1 : 1 : 0.5$) at $Re = 10,000$. ................................. 127
Figure 6.50 Comparison of the scaled mean Reynolds stress tensor components 500 $< u'u' > /u_B^2$ in the span-wise mid-plane with the classical experimental data by Prasad & Koseff (1989) and numerical data by Zang et.al (1993), for the LES of a lid driven cavity ($x : y : z = 1 : 1 : 0.5$) at $Re = 10,000$. ................................. 128
Figure 6.51 The isosurfaces of Q2 criterion colored by velocity magnitude at time $t = 8t_c$. ................................. 131

xi
Figure 6.52 Evolution of the dimensionless kinetic energy as a function of the dimensionless time using RK4. .................................................. 131
Figure 6.53 Evolution of the dimensionless kinetic energy dissipation rate as a function of the dimensionless time using RK4. ............................... 132
Figure 6.54 Evolution of the dimensionless enstrophy as a function of the dimensionless time using RK4. .................................................. 133
Figure 6.55 Kinetic energy dissipation rate balance for the Taylor-Green vortex problem using RK4. .................................................. 133
Figure 6.56 Biased pressure-dilatation term for the Taylor-Green vortex problem using RK4. .................................................. 134
Figure 6.57 The unstructured tetrahedral grid, $6 \times 44^3$. .................................................. 134
Figure 6.58 Evolution of the dimensionless kinetic energy as a function of the dimensionless time using RK4. .................................................. 135
Figure 6.59 Evolution of the dimensionless kinetic energy dissipation rate as a function of the dimensionless time using RK4. ............................... 136
Figure 6.60 Evolution of the dimensionless kinetic energy dissipation rate as a function of the dimensionless time on tetrahedral and hexahedral grids using RK4. 136
Figure 6.61 Evolution of the dimensionless kinetic energy as a function of the dimensionless time using ROW. .................................................. 137
Figure 6.62 Evolution of the dimensionless kinetic energy dissipation rate as a function of the dimensionless time using ROW. .................................................. 137
Figure 6.63 Evolution of the dimensionless enstrophy as a function of the dimensionless time using ROW. .................................................. 138
Figure 6.64 Biased pressure-dilatation term for the Taylor-Green vortex problem using ROW. .................................................. 138
Figure 6.65 Kinetic energy dissipation balance for the Taylor-Green vortex problem. 139

Figure A.1 Representation of the 6-node triangular element. ......................... 156
Figure A.2 Representation of the 9-node quadrilateral element. ...................... 157
Figure A.3 Representation of the 10-node tetrahedral element. ...................... 158
Figure A.4 Representation of the 27-node hexahedral element. ...................... 159
Figure A.5 Representation of the 18-node prismatic element. ...................... 161
Chapter 1

Introduction

1.1 Background of Turbulent Flows

It is well known that CFD has become very widespread and is utilized frequently by all kinds of users, ranging from government, academia and industry to analyze and design a lot of transport vehicles, e.g., aerospace vehicles, automobiles, etc. Also, it is worth pointing out that most flows around the transport vehicles are turbulent flows. It is well known, for aerospace vehicles, the drag is the heart of the aerodynamic design. Even small changes of the drag coefficient could be critical. On the Concorde, a one count drag increase ($\Delta C_D = .0001$) requires two passengers, out of the 90-100 passenger capacity, be taken off the North Atlantic run [118]. Therefore, it is especially important to predict the drag coefficient accurately when we design the aerospace vehicle. Then we need to know exactly the factors which affect the drag coefficients. The drag coefficient depends not only on the complex object shape and inclination, but also the effects of air viscosity and compressibility. To correctly predict the drag coefficient, we must guarantee that the viscosity and compressibility effects in our measured case are the same as that of the predicted case. Herein, we only focus on the viscosity effects. The important parameter associated with viscosity is the Reynolds number that denotes the ratio of inertial forces to viscous forces. It is well known that skin friction drag depends directly on the viscous interaction of the object and the air flow [4]. If the Reynolds number of the experiment is very close to that of the flight, then that means the effects of the viscous forces relative to the inertial forces have been modeled properly. However, if they are quite different, we do not correctly model the physics of the real problem, leading to meaningless prediction of the drag coefficient. Also we need to point out that, most practical engineering problems, e.g., the flow over aerospace vehicles, are high Reynolds number flow, namely turbulent flow. Therefore, in order to predict the drag coefficient accurately, the turbulent flow over the aerospace vehicles needs to be solved as accurately as possible.
Before we introduce CFD techniques for the turbulent flow, the first thing should be to understand it. The basic properties of turbulence is listed here [132].

- **Unsteady, irregular (aperiodic) motion**
  For turbulence, every quantity, e.g., mass, and momentum fluctuates in time and space. The striking feature of turbulent flow is the enhanced diffusivity since the turbulence enhances the transfer of mass, momentum and energy.

- **Statistical**
  Turbulence is characterized by random fluctuation thus necessitating the use of statistical methods to analyze it. This lays the underlying foundation for the RANS in the following chapter.

- **Turbulence scales and the cascade**
  Turbulence consists of continuous spectrum of scales ranging from largest to smallest, as opposed to a set of discrete set of scales. Usually, in order to describe the Cascade, the turbulence eddies are considered to be comprised of eddies of different size. Turbulence scales involves the turbulence length scales \((l)\), turbulence time scales \((t)\), thus the characteristic velocity could be calculated as follows, \(u = \frac{l}{t}\). First, the length scale (eddies size) is introduced in details. The largest eddy size is characterized by \(L\), comparable to the flow scale constrained by the geometry dimension. The characteristic velocity of the largest eddy size, \(U\), is on the order of magnitude as the flow velocity. Then the \(Re\) of the largest eddy could be arrived at, \(Re = \frac{UL}{\nu}\). Then turbulence time scale of the largest eddy size could be arrived at, \(t_L = \frac{L}{U}\), referred to as the large eddy turnover time.

  Obviously, for large eddy, the \(Re\) is is on the order of magnitude as the freestream \(Re_\infty\), which leads to the small viscosity effect on the large eddies. Then, it is about the smallest turbulence length scale, Kolmogorov length scale, denoted by \(\eta\) [132]. By the Kolmogorov’s universal equilibrium theory, the dissipation rate at which the smaller eddies dissipate energy to heat should be almost equal to the supplying rate at which the energy is transferred from the large eddies to the small ones. Thus, the smallest eddies should only depend on: the rate of supplying the energy from the large eddies to the smallest eddies, and the kinematic viscosity. Therefore, the turbulence length scale of smallest eddies could be determined as follows, \(\eta = (\frac{\nu^3}{\epsilon})^{1/4}\), where \(\epsilon\) means the rate at which the larger eddies supply energy, and \(\nu\) denotes the kinematic viscosity. Through the turbulence, the eddies overlap in space, large eddies carry the smaller ones. Therefore, turbulence is characterised by a cascade process, whereby the energy is transferred from the large eddies to the smaller ones. Finally, the smallest eddies dissipate the energy into heat due to viscous effect.
After we understand the properties of the turbulent flows, let’s come to the CFD techniques for turbulent flows. Indeed, due to the importance of the turbulent flow in the practical applications and physics, a lot of useful numerical methods have been developed to investigate the turbulent flows, e.g., RANS, URANS, LES, and DNS. Herein, firstly, some basic description about the aforementioned methods for solving the turbulent flows will be given. It is well known that turbulent flow is unsteady and three-dimensional in nature [132]. However, if the unsteady part is only associated to the turbulence, the mean flow could be viewed to be steady. Therefore, Reynold-averaging (time-averaging) could be used to deal with the Navier-Stokes equations. It should be noted that [132], the integral time length should be very long relative to the maximum period of the velocity fluctuations, but should be very short relative to the time scales characteristic of the slow variation in the flow. The instantaneous physical value is decomposed to mean value and turbulent fluctuation value. Following the decomposition, the time-averaging is used to deal with the Navier-Stokes equation to arrive at the RANS equations. The mean flow is solved using RANS equations, whereas all the scales are modeled by the turbulence model, e.g., Spalart-Allmaras model, $k-\omega$ SST model and so on. Generally, the Reynold-averaging of the Navier-Stokes equations should separate the coherent fluctuations from the turbulence fluctuations. Turbulence models are not designed to model the effects of coherent fluctuations. The RANS turbulence model could be used for calculation of unsteady turbulent flows, terms as unsteady RANS or URANS. For URANS [60], ensemble-averaging should be used to deal with the N-S equations. The ensemble-averaging is imposed on experimental data by running the same experiment a lot of times, and taking data at selected points and at selected times. Then the ensemble-averaging value is obtained by averaging the data at the same point and the specified time from each experiment. Similar to the time-averaging for steady RANS, the instantaneous value could be decomposed to ensemble-averaging one and the turbulence fluctuations. Therefore, this ensemble-averaging leads to the format similar to that obtained by time-averaging. But it should be mentioned that, the Reynold Stress obtained by ensemble-averaging still varies in time as does the ensemble-averaging velocity. For RANS and URANS, only the mean flow is solved. All the turbulence scales are modeled using RANS model. Indeed, this could lead to loss of detailed information of Navier-Stokes equations [132]. Different form the RANS and URANS, LES could solve large turbulence length scales whereas the smallest eddies are modeled using subgrid-scale SGS model. This means that, the underlying premise is that the largest eddies are directly affected by the boundary conditions, carry most of the Reynold stress, thus must be computed. The small-scale turbulence is weaker, contributing less to the Reynold stresses, and is therefore less critical. Also we need to mention that using LES method to deal with the turbulence, it is always unsteady and three-dimensional. Since LES involves modeling the smallest eddies, the smallest finite-difference cells could be much larger than the Kolmogorov length, and much larger time steps could be used to keep the time accuracy. In order to resolve
all the turbulence scales, DNS has to be used. DNS means a complete time dependent and three-dimensional solutions of Navier-Stokes equations. Theoretically, all turbulent flows can be simulated by numerically solving the full Navier-Stokes equations directly.

Over the last 15 years, the AIAA Applied Aerodynamics Technical Committee has sponsored 5 drag prediction workshops (DPW), aiming to assess the current state-of-the-art CFD solvers for predicting the forces and moments on industry-relevant geometries, e.g., absolute drag. For the 5th workshop [97], the level of code-to-code scatter was reduced compared with the 2nd through 4th workshop. And the one standard deviation level of scatter is below 2.5 drag counts, which is only 1% out of approximately 250 drag counts for total drag. Albeit this scatter has improved over the past 4 workshops, it is still substantially larger than that required by the airframe manufacturer, as mentioned above. From Marvriplis’ paper [95], the discretization error may be a great factor contributing to the numerical error of the current CFD methods since previously, for DPW, most solvers are based on second-order finite volume method. Different grid-converged results may be predicted by different families of meshes, since the inherent scales of such flow problems maybe easily vary by over about 6 orders of magnitude, it is not easy to solve all the computational domain through successively refinement of the initial mesh, which will have an effect on the solution of the computational domain. In order to solve the turbulence length scales exactly, there are two straightforward methods, i.e., finer meshes and high-order numerical methods. In our work, the high-order reconstructed DG is used to complete all the calculations ranging from RANS ,ILES [37] to DNS. ILES means LES without the use of an explicit sub-grid scale turbulence model. The high-order methods ”only dissipate the scales that the model is not able to capture correctly, thus acting like a sub-grid scale (SGS) model”.

It is obviously, except the steady RANS turbulence equations, in order to solve time scales, high-order time integration methods are also necessary for URANS, ILES and DNS.

1.2 Reconstructed Discontinuous Galerkin method

There exist some popular high-order CFD methods capable of solving the Navier-Stokes equations, e.g., discontinuous Galerkin (DG) method, k-exact finite-volume method, streamline-upwind Petrov-Galerkin method, spectral element type method, the residual distribution method, and spectral volume method. The discontinuous Galerkin method have recently become very popular for the solution of systems of conservation laws. In 1973, Reed and Hill [111] introduced a DG method to solve the hyperbolic neutron transport equation. In 1974, Le Saint and Raviart [71] 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 [120] 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. [31]. The discontinuous Galerkin methods have many attractive features, listed in reference [84, 90, 92].

In Luo’s work [79], 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, listed in §3.1.2. However, in comparison with 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 [138]. In order to reduce the high computing costs associated with the DG methods, Dumbser et al. [41, 40, 42] have introduced a new family of reconstructed DG methods, termed P

P

schemes and referred to as $\text{rDG}(P_nP_m)$ in this work, where $P_m$ indicates that a piecewise polynomial of degree of $n$ is used to compute the fluxes. The $\text{rDG}(P_nP_m)$ 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 $\text{rDG}(P_nP_m)$ 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 $\text{rDG}(P_nP_m)$ 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 $\text{rDG}(P_0P_m)$ 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 $\text{rDG}(P_nP_n)$ scheme yields a standard DG method.

Obviously, the construction of an accurate and efficient reconstruction operator is crucial to the success of the $\text{rDG}(P_nP_m)$ schemes [85, 86, 84, 90, 92]. In the reconstructed DG method
using a Taylor basis [85, 86, 84, 90, 92] 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. [148, 149] 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. Recently, Luo et al. [91, 93] have conducted a comparative study for these three reconstructed discontinuous Galerkin methods rDG(P1P2) 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(P0P1), the resultant rDG(P1P2) method is numerically unstable. Although the rDG(P0P1) 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 [49]. Unfortunately, the least-squares rDG(P1P2) 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 [51], WENO [74, 105] and Hermite-WENO [103, 104] 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 Hermite-WENO scheme has been developed on 1D and 2D structured grids for the DG methods by Balsara et al [8], where the Hermite-WENO reconstruction scheme is relatively simple and straightforward. In the work presented, a Taylor basis [90] reconstruction-based DG method, rDG(P1P2), based on a Hierarchical WENO reconstruction scheme, termed as HWENO(P1P2) [92], is developed for the solution of the compressible Euler and Navier-Stokes
equations on single-type and hybrid unstructured grids in 3D [133]. 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(P₁P₂) method, a quadratic solution is first reconstructed to enhance the accuracy of the underlying DG method in two steps: (1) all second derivatives in each cell are first reconstructed using the solution variables and their first derivatives from adjacent face-neighboring cells via a strong interpolation; (2) the final second derivatives on each cell are then obtained using a WENO strategy based on the reconstructed second derivatives on the cell itself and its adjacent face-neighboring cells. 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(P₁) method is second-order accurate, and the basis functions are at most linear functions, fewer integration points are then required for both domain and face integrals, and the number of unknowns (the number of degrees of freedom) remains the same as for the DG(P₁) method. Consequently, this rDG(P₁P₂) method is more efficient than its third-order DG(P₂) counterpart in terms of storage requirement and computational cost.

A lot of valuable research work with rDG(P₁P₂) method has been conducted, e.g., developing a reconstructed discontinuous Galerkin method, rDG(P₁P₂)), based on a Hermite WENO reconstruction for the solution of the compressible Euler equations on unstructured tetrahedral grids [90], a reconstructed discontinuous Galerkin method based on a hierarchical WENO reconstruction, HWENO (P₁P₂) for the solution of the compressible Euler equations on unstructured tetrahedral grids [92], an Implicit Reconstructed Discontinuous Galerkin method, IRDG (P₁P₂), based on a Hermite WENO reconstruction for the solution of the compressible Euler and Navier-Stokes equations on unstructured hybrid grids [136, 140]. All these great success has motivated us to extend rDG(P₁P₂) method to the 3D compressible turbulent flows.

It is well known that, to match well with the free-stream turbulence working variable, there indeed exists an abrupt change of the curvature for the turbulence working variable at the interface between turbulent/non-turbulent regions of the flow, namely the edge of the boundary layer [26]. For high-order discretization methods, if this region is under-resolved due to insufficient grid resolution, the abrupt change of the curvature for the turbulence working variable could produce Gibbs oscillation, easily causing solver to fail. This is the truth for high-lift configurations where especially negative values with large magnitude may occur. There have been some interesting methods to stabilize high-order solutions of the turbulence model equations [26, 96, 28]. These proposed methods attempt to remove the non-smooth behavior resulting from the discrete solution of the turbulence model equation in the same way that shock waves
are stabilized, e.g., limiters or artificial diffusion. However, all of these methods either allow for negative eddy viscosity values or compromise high-order solution accuracy. In order to prevent solver from blowing up originating from turbulence model’s non-smooth behavior and furthermore obtain the accurate high-order solution, reference[99] has introduced some modifications to the SA turbulence model equation to solve this equation with high-order DG discretizations. Modifications change the continuous SA turbulence model equation. These are quite different from stabilization methods because stabilization methods attempt to modify the non-smooth behavior of the discrete solution. Different from preventing negative values[99], the approach of reference [96] defines a new auxiliary variable which is always positive and turns off the production, destruction, and dissipation terms of the turbulence model equation once negative turbulent working variables occur. This strategy is particularly effective to guarantee the stability of high-order discretization for turbulence model equation because the source and diffusion terms are the causes of instability when the turbulence model working variable becomes negative. These additional modifications to SA turbulence model are the important components for a robust high-order RANS-SA solver.

Although the use of high-order accurate discretizations for RANS problems is not commonplace, there are several examples of successful high-order DG RANS solutions using standard turbulence models in the literature. In recent years, the application of high-order methods for computing high Reynolds number turbulent flows governed by the Reynolds-averaged Navier-Stokes (RANS) equations with turbulence model has been an active research topic. In the pioneering work by Bassi and Rebay [15], they successfully solved the RANS equations with a modified $k$-$\omega$ model based on a high-order DG framework. In his work, a non-standard implementation of the $k$-$\omega$, whereby the logarithm of $\omega$ rather than $\omega$ itself is used as the unknown, has been found very useful to enhance stability of the method. And by further enforcing realizability constraints of the turbulence model, significant improvement in the robustness and efficiency of this method has been obtained. Similarly, Hartmann et al. [53] developed a DG code for 3D turbulent flow computation with adaptive mesh refinement based on the $k$-$\omega$ model. In particular, some test cases with increasing complexity have been used to validate the solver. Compared to the use of two-equation $k$-$\omega$ model, more researchers chose to apply one-equation SA model due to its simplicity in implementation. Nicholas et al. [26] and Wang et al. [129] developed high-order DG methods for solving a fully coupled RANS-SA system respectively. The modified SA model is particularly designed to make the original SA model insensitive to negative values of turbulence working variables as numerical experiments show that turbulence working variable often drops several orders of magnitudes at the edge of the turbulent boundary layer. Marco et al. [29] applied a high-order output-based adaptive solution technique to the two-dimensional RANS equations closed with the modified negative SA model which originally proposed by Spalart and Allmaras [7]. They found that, compared with uniform refinement at
second-order finite volume method, high-order methods yield faster convergence. Nguyen et al. [98] implemented the SA model equation based on a DG framework with an artificial viscosity modification for SA equation. It is aimed to accommodate high-order RANS approximations on too coarse grids. Besides that, Crivellini et al. [36] analyzed and implemented a modified SA model based on high-order DG methods for incompressible flows. For modified SA model, they introduced an SA model implementation that dealt with negative $\tilde{\nu}$ values by modifying the source and diffusion terms in the SA model equation only when the working variable or one of the model closure functions became negative. Cheng et al. [150] successfully solved RANS-SA system in a high-order correction procedure via reconstruction (CPR) framework. In their work, a high-order solver based on CPR has been developed for the Eikonal equation to compute the nearest distance to the wall.

Consistent with high-order discretization methods, correct representation of physical boundary is crucial [12]. Errors can arise when a piecewise linear approximation is used to represent the physical geometry for high-order discretization methods. Large numerical errors may arise near the geometry surface and then pollute the solution inside the domain. This is especially important for high-order DG methods, where errors resulting from geometrical approximation may exceed the discretization error, rendering the use of a higher-order DG method meaningless. Therefore, this requires that the boundary faces of the computational meshes must be able to represent the physical boundary rather than just linear approximation of the boundary face. This strategy works fine for inviscid and low Reynolds number laminar flow, because the cells inside the boundary layer are almost isotropic, and curved boundary surface will not tangle with the faces inside the domain. However, when highly stretched cells are used in viscous boundary layer for high Reynolds numbers, e.g., turbulent flows, invalid cells are likely to occur near the boundary only if the boundary cells near the geometry surface are set to be curved. Thus, for viscous flow with high Reynolds numbers, high-order curved elements are employed not only for the boundary cells near the geometry surface but also for the interior cells. By doing this, the negative cells could be avoided. It is thus necessary to somehow propagate the curvature from the geometrical boundary to the interior domain. Several smoothing schemes have been proposed in the literature to fulfill this effect: linear smoothing techniques such as Laplacian smoothing [47], Winslow smoothing [66] or linear elasticity with varying stiffness. Such simple techniques have led to valuable results, however, there is no guarantee that using such a linear smoother could always arrive at a valid and excellent curved mesh. Nonlinear smoothing techniques [101] have also been investigated to generate the valid curvilinear mesh, in particular using a nonlinear elasticity analogy. However, computing a non-linear mechanics problem involving large deformations on a highly stretched high-order mesh is numerically very complex compared with solving the Navier-Stokes equations on the same grid. Therefore, the computational cost for the curved mesh generation may then be excessive compared with the
solution of the Navier-Stokes equation on the same grid. This is not what we expect to be. In addition to linear and nonlinear smoothing techniques, mesh adaptation techniques have been used to eliminate invalid elements by a combination of local mesh refinements, edge and face swaps, and node relocations [73, 94, 116].

1.3 Effective time integration schemes

The spatial discretization of the compressible Navier-Stokes equations or RANS-SA system leads to a system of Ordinary Differential Equations (ODEs). For unsteady problems, DG methods have generally been implemented by combining high-order accurate explicit time-integration methods, such as explicit Runge-Kutta DG (RKDG) methods [78]. While such methods are well suited for problems with similar spatial and temporal scales, they are notoriously inefficient for problems with disparate temporal and spatial scales, such as low-reduced frequency problems, and for steady-state problems [88]. There have been a lot of effective implicit time integration schemes, e.g., multi-step backward difference methods and multi-stage Runge-Kutta methods. The main disadvantage of high-order multi-step BDF schemes lies in that initialization of the first several time steps needs to be done via an integration scheme that does not require solution at several previous time steps, e.g., low-order BDF schemes and Runge-Kutta type schemes [146]. This disadvantage makes rigorous convergence research a little difficult. Furthermore [130], the highest order A-stable BDF scheme is BDF2, which is only second-order, which doesn’t match well with high-order spatial discretization with DG methods. Significant progress has also been made in developing efficient higher-order multi-stage implicit Runge-Kutta time integration methods for such system in order to reduce the temporal discretization error incurred from the use of lower-order time integration methods. In practice, one usually adopts singly diagonally implicit (SDIRK) methods, for which the diagonal entries are identical to each other, allowing us to perform a sole LU factorization per integration step. However, the SDIRK method is limited to first stage order [108]. This means that when solving DAEs of index 2, they only have the first convergence order. Therefore, this greatly limited the application of SDIRK methods. The stage order of diagonally implicit methods can be increased to two by introducing an explicit first stage. The resulting methods are called explicit first stage SDIRK methods (ESDIRK). Bijl et al. [20] introduced ESDIRK schemes for the finite volume solutions to the Navier–Stokes equations. Later on, Wang and Mavriplis [128] extended the ESDIRK schemes to solve the compressible Euler equations using a high-order p-multigrid DG method. Xia et al. [143] also used ESDIRK for the time accurate solutions of the 3D compressible Navier–Stokes equations in the context of the rDG(P1P2) methods. They all have concluded that the higher-order ESDIRK schemes are more efficient than those second-order time integration methods to achieve the solutions for the same level of temporal accuracy. However, ESDIRK
method has to solve nonlinear systems, increasing the computational cost. Therefore, recently, a family of Rosenbrock-type time integration schemes for the unsteady flow simulations has been extensively explored, replacing non-linear systems with a sequence of linear systems. Some effective Rosenbrock-type methods have been designed to solve the DAEs (Differential Algebraic Equations) of Index-1 and Index-2. The DAEs originate in the modeling of various physical or chemical phenomena. They are classified by their differential index, that is, the minimum number of times that a DAEs system must be differentiated in order to become an ODEs system. For instance, the discrete incompressible Stokes, Oseen, and Navier-Stokes equations belong to index-2 DAEs systems. Unlike the incompressible Navier–Stokes equations, there is no algebraic constraint for the compressible Navier-Stokes equations, indicating that the Rosenbrock-type methods, originally designed for the DAEs, could also be introduced to advance the unsteady compressible Navier–Stokes equations in time. However, a major disadvantage associated to the standard Rosenbrock method is that it requires the use of exact Jacobians in order to achieve formal order of temporal convergence, which is usually too demanding in practice. On the other hand, the Rosenbrock–Wanner method, abbreviated as ROW in this paper, does not have such prohibitive restriction, and would only need approximate Jacobians.

Bassi et al. [10] investigated the use of Rosenbrock-type schemes to integrate in time high order DG discretizations of Navier-stokes equations. They applied the best performing Rosenbrock schemes to the Implicit Large Eddy Simulation of the transitional flow around SD7003 airfoil. Blom et al. [24] compared the time adaptive ESDIRK and ROW method in the context of finite volume discretizations of the compressible Navier–Stokes equations. They concluded that the ROW scheme, ROS34PW2 (34 stands for order 3 with 4 stages, W for W-method and 2 is an internal number), were suitable candidates for engineering accuracies. Birken et al. [21] also compared the performance of the adaptive time integration in the context of discontinuous Galerkin methods for the 3D unsteady compressible Navier–Stokes equations.

One objective of this thesis is to present a set of efficient third-order ROW schemes for Index-2 DAEs [109, 107] to the solution of the ODEs in the context of a spatially third-order rDG (P1P2) method. And we also make comparative study between the ROW for index-1 ROW for index-2 and ESDIRK3 schemes in the context of rDG method through a variety of test problems.

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. 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., [117, 16, 127, 65, 131, 83, 9]). Luo et al. [76] 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 lies in 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. Such method was later applied to the computation of compressible turbulent flows on unstructured grids [89] successfully.

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 [14]. 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 [13], which has many impressive features: in the development of implicit integration schemes; 1) turbulence effects are accounted for by means of the low-Reynolds $k$-$\omega$ turbulence model; and 2) 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 [15], 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. Following Bassi, a lot of other meaningful attempts were implemented for solution of compressible Euler equations in the context of DG methods. In 2001, an implicit discontinuous spectral Galerkin method for the solution of the compressible Euler equations on 2D grids was developed by Rasetarinera and Hussaini [110]. In their method, a matrix-free LU-SGS preconditioned Newton-Krylov algorithm was used to solve the implicit system, where the memory requirement for storing the off-diagonal blocks of the Jacobian matrices was largely reduced by the matrix-free approach. The overall performance of the implicit version of the method was orders of magnitude better than an explicit method. In 2002 [54], an approach for the design of adaptive discontinuous Galerkin finite element methods was applied to physically relevant problems arising in inviscid compressible fluid flows governed by the Euler equations. The resulting system of nonlinear equations were solved using a damped Newton iteration method; the resulting linear iteration steps, together with the system of linear equations arising from the dual problem, were solved by employing the GMRES method with a block Gauss–Seidel preconditioner. In 2004, semi-implicit numerical schemes were proposed for 2D numerical solution for inviscid compressible Euler equations by M. Feistauer [39]. In their work, semi-implicit DGFE schemes were proposed, based on the homogeneity of the inviscid fluxes and properties of the Vijayasundaram numerical flux, leading in a natural way to a linear system on each time level. The linear algebraic systems were solved by the GMRES method. The described method was not based on the Newton linearization, didn’t not require
differentiating the numerical flux and can be applied on arbitrary meshes.

As to the compressible Navier-stokes and RANS equations, there also have been some successful implementation.

In 2006, the Interior Penalty discontinuous Galerkin method for the compressible Navier-Stokes equations was presented by Hartmann [52]. Furthermore, his work demonstrated the use of a Newton-GMRES algorithm for solving the nonlinear discrete problems. In 2006, a space-time discontinuous Galerkin method for the compressible Navier-Stokes equations on 3D Cartesian grids was developed by Klaij et al. [64]. This method didn’t distinguish between space and time, thereby providing optimal flexibility to accommodate time-dependent boundaries and element deformation.

In 2008, an implicit discontinuous Galerkin method for RANS simulation utilizing pointwise relaxation algorithm on 3D hybrid grids was introduced by Kanako Yasue at al. [145]. In the pointwise relaxation, those contributions from the degrees of freedom in own computational cell were accounted for in the implicit matrix inversion. The resulting scheme was shown to be stable with very large CFL numbers for both the Euler and the Navier-Stokes equations for typical test problems. A faster convergence was achieved by utilizing a $p$-multigrid scheme 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 was presented by Crivellini and Bassi [35], and compared with its matrix based counterpart. In 2012 [27], a robust discontinuous Galerkin solver for high-lift aerodynamic flows using one-equation SA turbulence model was developed by Burgess. In this work, the RANS-SA system was solved using a damped Newton solver that treated the mean flow and turbulence model equation in a coupled way. Furthermore, the linear solver used at each Newton step was based on a line-implicit colored Gauss-Seidel preconditioned GMRES solver.

In 2016 [29], a high-order output-based adaptive solution technique for RANS-SA system was presented by Marco. The system of nonlinear equations was solved using Newton’s method with pseudo-transient continuation for improved robustness. In his work, the linear systems at each Newton iteration were solved with an element-line preconditioned GMRES solver. In this work, pseudo-time continuation method was considered to solve a lot of problems ranging from intermediate to difficult in 2D and 3D. 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. The block diagonal matrix requires a storage of $(N_{degr} \times N_{tot}) \times (N_{degr} \times N_{tot}) \times N_{elem}$, where $N_{degr}$ 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), $Netot$ is the number of components in the solution vector (4 for 2D, and 5 for the 3D Navier-Stokes equations; 5 for 2D, and 6 for the 3D RANS-SA system), and $Nelem$ is the number of elements for the grid. For example, for a 4th-order DG($P_3$) method in 3D, the storage of this block diagonal matrix alone requires $120 \times 120$ words per element for RANS-SA system.

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 [134, 137, 135], and then extended to hybrid grids [136, 140], which just takes up only a small memory, yet retains the third-order accuracy compared with standard third-order DG($P_2$). A notable feature of this implicit rDG($P_1P_2$) method is that the Jacobian matrix $J = \partial R/\partial U$, i.e., linearization of the right-hand-side (RHS) vector $R$ in terms of the conserved state vector $U$, 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, the full linearization of the rDG($P_1P_2$) method is practically not attractive, especially when the highly non-linear HWENO($P_1P_2$) reconstruction is used. As a matter of fact, the property of implicit rDG($P_1P_2$), only using the Jacobian matrix of the underlying DG($P_1$), makes it a perfect candidate for the solution of unsteady problems with ROW method. As mentioned above, the exact Jacobian matrix is not necessary for ROW method to calculate the unsteady problems with the designed temporal order of accuracy. This means that even with inexact Jacobian matrix, the designed temporal order of accuracy could still be preserved. Therefore, for unsteady problems, it is very promising to combine the advantage of ROW method with implicit rDG($P_1P_2$) method to achieve third-order of accuracy for both temporal and spatial discretization. Finally, based on the success of the implicit rDG($P_1P_2$) for inviscid and viscous problems, it is desirable to extend this effective method to the solution of RANS-SA systems. For turbulent RANS-SA systems, the storage requirement of the block diagonal matrix for rDG($P_1P_2$) is only 16% of that for DG($P_2$). This provides a very promising and effective spatial third-order method for practical high $Re$ number engineering problems with fine meshes.

Several approaches were proposed and implemented in the previous work [134, 137, 135] to calculate the Jacobian matrix, 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 [3, 23, 22], 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. Therefore, in this work, the Jacobian matrix is only calculated by AD tool. Finally, the linear system arising from the Newton’s linearization of the ODEs is solved iteratively by a GMRES+LU-SGS algorithm.
1.4 Scope

The objective of this PhD work is to develop a parallel, implicit reconstructed discontinuous Galerkin (rDG) method using Taylor basis for the solution of the compressible turbulent flows on 3D hybrid grids. This third-order accurate rDG method is based on a hierarchical weighted 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.

The PhD work involves the research of 3D turbulent flows in the context of third-order rDG method, e.g., steady cases (RANS), and unsteady cases (ILES and DNS). As to RANS, the excellent website [6] has been referred to for a lot of benchmark test cases. Simple implicit backward Euler method is implemented for time advancement of steady cases with rDG method. For ILES and DNS, third-order accurate implicit ROW methods are implemented to advance the time integration, saving greatly the computational cost compared with the high-order implicit Runge-Kutta method. For the implicit method, analytical differentiation is developed and implemented to obtain the resulting flux Jacobian matrices. 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.

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 rDG method is discussed in detail. The temporal integration methods (both explicit and implicit) are discussed in Chapter 4. Chapter 5 deals with the parallel implementation strategy. Numerical results for a variety of compressible turbulent test cases are presented in Chapter 6. Finally the conclusion and future work is discussed in Chapter 7.
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, including both the Navier-Stokes equations (§2.2) and the Reynolds Averaged Navier-Stokes (RANS) equations with the modified one-equation Spalart-Allmaras (SA) turbulence model (§2.1). The nondimensionalization of the governing equations is described in the last section.

2.1 RANS-SA Equations

The conservation form of the compressible RANS equations with the modified one-equation Spalart-Allmaras (SA) turbulence model can be given as below,

$$\frac{\partial U}{\partial t} + \frac{\partial F_k(U)}{\partial x_k} = \frac{\partial G_k(U)}{\partial x_k} + S \quad (2.1)$$

where the summation convention ($k = 1, 2, 3$) has been used. In Eq. 2.1, the conservative variables $U$ is defined as

$$U = \begin{pmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \\ \rho e \\ \rho \tilde{\nu} \end{pmatrix} \quad (2.2)$$

where $\rho$, $p$, and $e$ denote the density, pressure, and specific total energy of the fluid, respectively, $u$, $v$, and $w$ are the velocity components of the flow in the coordinate direction $x$, $y$ and $z$, and
\( \tilde{\nu} \) represents the turbulence working variable in the modified SA model. The pressure can be computed from the equation of state

\[
p = (\gamma - 1)\rho \left( e - \frac{1}{2}(u^2 + v^2 + w^2) \right)
\]

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

\[
h = e + \frac{p}{\rho}
\]

The inviscid flux vector \( \mathbf{F} = (\mathbf{F}_x, \mathbf{F}_y, \mathbf{F}_z) \) is defined by

\[
\mathbf{F}_x = \begin{pmatrix}
\rho u \\
\rho u^2 + p \\
\rho uv \\
\rho uw \\
u(\rho e + p) \\
\rho u\tilde{\nu}
\end{pmatrix}, \quad
\mathbf{F}_y = \begin{pmatrix}
\rho v \\
\rho vu \\
\rho v^2 + p \\
\rho vw \\
v(\rho e + p) \\
\rho v\tilde{\nu}
\end{pmatrix}, \quad
\mathbf{F}_z = \begin{pmatrix}
\rho w \\
\rho wu \\
\rho w^2 + p \\
\rho vw \\
w(\rho e + p) \\
\rho w\tilde{\nu}
\end{pmatrix}
\]
, the viscous flux vector $\mathbf{G}$ is defined by

\[
\mathbf{G}_x = \begin{bmatrix}
0 \\
\tau_{xx} \\
\tau_{xy} \\
\tau_{xz} \\
u \tau_{xx} + v \tau_{xy} + w \tau_{xz} + q_x \\
\frac{1}{\sigma} \mu (1 + \psi) \frac{\partial \nu}{\partial x}
\end{bmatrix}
\]

\[
\mathbf{G}_y = \begin{bmatrix}
0 \\
\tau_{yx} \\
\tau_{yy} \\
\tau_{yz} \\
u \tau_{yx} + v \tau_{yy} + w \tau_{yz} + q_y \\
\frac{1}{\sigma} \mu (1 + \psi) \frac{\partial \nu}{\partial y}
\end{bmatrix}
\]

\[
\mathbf{G}_z = \begin{bmatrix}
0 \\
\tau_{zx} \\
\tau_{zy} \\
\tau_{zz} \\
u \tau_{zx} + v \tau_{zy} + w \tau_{zz} + q_z \\
\frac{1}{\sigma} \mu (1 + \psi) \frac{\partial \nu}{\partial y}
\end{bmatrix}
\]

(2.6)

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

\[
\begin{bmatrix}
\tau_{xx} & \tau_{xy} & \tau_{xz} \\
\tau_{yx} & \tau_{yy} & \tau_{yz} \\
\tau_{zx} & \tau_{zy} & \tau_{zz}
\end{bmatrix}
\]

and the source term $\mathbf{S}$ is defined by

\[
\mathbf{S} = \begin{bmatrix}
0 \\
0 \\
0 \\
0 \\
c_b \tilde{S} \mu \psi + \frac{c_a^2}{\sigma} \rho \nabla \nu \cdot \nabla \tilde{\nu} - c_w \rho f_w (\frac{\nu}{\sigma})^2 - \frac{1}{\sigma} \nu (1 + \psi) \nabla \rho \cdot \nabla \tilde{\nu}
\end{bmatrix}
\]

(2.7)

Next, we will look into the details of the viscous stress tensor $\tau$. The Newtonian fluid with the Stokes hypothesis is valid under the current framework, since only air is considered. Thus $\tau$ is
symmetric and it represents the viscous stress tensor, as defined by

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

(2.8)

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

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

(2.9)

where $\mu_0$ denotes the viscosity coefficient at the reference temperature $T_0$, and $S$ is a constant which is assumed the value $S = 110K$, and $\mu_t$ denotes the turbulence eddy viscosity, which could be achieved by:

$$
\mu_t = \begin{cases}
\rho \tilde{\nu} f_{v1} & \text{if } \tilde{\nu} \geq 0 \\
0 & \text{if } \tilde{\nu} < 0
\end{cases}
$$

(2.10)

The temperature of the fluid $T$ is determined by

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

(2.11)

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

$$
q_j = -c_p \left( \frac{\mu}{Pr} + \frac{\mu_t}{Pr_t} \right) \frac{\partial T}{\partial x_j}
$$

(2.12)

where $c_p$ is the specific heat capacity at constant pressure, $Pr$ is the nondimensional laminar Prandtl number, and $Pr_t$ is the turbulent Prandtl number.

Finally, the source term for the SA equation will be introduced in detail. The parameters for the production and destruction terms of the modified SA turbulence model are given as,

$$
\tilde{S} = \begin{cases}
S + \hat{S} & \text{if } \hat{S} \geq -c_{v2} S \\
S + \frac{S(\psi^2 + c_{v3} S)}{(c_{v3} - 2c_{v2})S - S} & \text{if } \hat{S} < -c_{v2} S
\end{cases}
$$

(2.13)

where

$$
S = \sqrt{\overline{\omega} \cdot \overline{\omega}}, \quad \hat{S} = \frac{\nu \psi}{\kappa T d^2} f_{v2}, \quad f_{v1} = \frac{\psi^3}{\psi^3 + c_{v1}}, \quad f_{v2} = 1 - \frac{\psi}{1 + \psi f_{v1}}
$$

(2.14)
and
\[ r = \min(10.0, \frac{\nu \psi}{S \kappa T^2 d^2}), \quad g = r + c_{w2}(r^6 - r), \quad f_w = g \left( \frac{1 + c_{w3}^6}{g^6 + c_{w3}^6} \right)^{\frac{1}{6}} \] (2.15)

The vorticity vector, \( \vec{\omega} \), is defined by
\[ \vec{\omega} = \nabla \times \mathbf{U} \] (2.16)

The parameter \( d \) denotes the distance from a specific position in the flow field to the nearest wall, which will be discussed in the chapter 3.4.2. The parameter \( \psi \) is designed for high-order discretization schemes to remove the effects of negative turbulence working variable to guarantee the robustness of the turbulence model. This parameter \( \psi \) is given by,
\[ \psi = \begin{cases} 0.015 \ln(1 + e^{20 \chi}) & \text{if } \chi \leq 10 \\ \chi & \text{if } \chi > 10 \end{cases} \] (2.17)

where \( \chi = \frac{\bar{c}}{\bar{P}} \). The parameter \( \psi \) is designed to become zero as the turbulence working variable goes negative, thereby turning off the production, destruction, and dissipation terms to guarantee stabilities. The constants in the modified SA model to close the main flow equations are given by,
\[ c_{b1} = 0.1355, \quad \sigma = \frac{2}{3}, \quad c_{b2} = 0.622, \quad \kappa_T = 0.41, \quad c_{w1} = \frac{c_{b1}}{\kappa_T^2} + \frac{1 + c_{b2}}{\sigma}, \quad c_{w2} = 0.3, \quad c_{w3} = 2, \quad c_{v1} = 7.1, \quad c_{v2} = 0.7, \quad c_{v3} = 0.9 \] (2.18)

### 2.2 Navier-Stokes Equations

Navier-Stokes equations could be arrived by turning off the SA turbulence equations, and setting the turbulent eddy viscosity in \( G_k \) equal to 0. For RANS-SA equations, the related variables, i.e., are all averaged variable, whereas the variables for NS equations are all instantaneous variables.

### 2.3 Nondimensionalization

The nondimensional form of governing equations are often used. 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, 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

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

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

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

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

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

When it comes to RANS-SA system, the reference turbulent working variable $\nu_{ref}$ should also be given. Since the Reynolds number of most practical cases in the aeronautical industry varies between $10^6$ and $10^7$, leading to that the SA working variable, $\bar{\nu}$, typically spans several orders of magnitude. Therefore, it is desirable to choose an appropriate reference value for $\bar{\nu}$. Marco [30] introduced a constant reference value for $\bar{\nu}$ to non-dimensionalize $\rho \bar{\nu}$ by a factor
larger than the physical viscosity, given by,

$$\frac{\rho \bar{v}}{\sqrt{Re \mu}} = \frac{\rho \bar{v}}{\sqrt{Re \mu}}$$

Accordingly, the turbulent eddy viscosity $\mu_t$ could also be non-dimensionalized as follows,

$$\mu_t = \begin{cases} \frac{\rho \bar{v}}{\sqrt{Re \mu}} f_{\nu} & \text{if } \bar{v} \geq 0 \\ 0 & \text{if } \bar{v} < 0 \end{cases}$$

In the normalized governing equations, the nondimensional viscous stress tensor is

$$\bar{\tau}_{ij} = \left( \bar{\mu} + \frac{Ma \sqrt{Re \mu}}{\mu_t} \right) \left( \frac{\partial \bar{u}_i}{\partial \bar{x}_j} + \frac{\partial \bar{u}_j}{\partial \bar{x}_i} - \frac{2}{3} \frac{\partial \bar{u}_k}{\partial \bar{x}_k} \delta_{ij} \right)$$

the nondimensional heat flux $\bar{q}_j$ vector is

$$\bar{q}_j = -\bar{c}_p \left( \frac{\bar{\mu}}{Pr} + \frac{Ma \sqrt{Re \mu}}{Pr_t} \right) \frac{\partial T}{\partial \bar{x}_j}$$

the nondimensional source term $\bar{S}$ is

$$\bar{S} = \sqrt{Re} c_{b1} \bar{S} \bar{\mu} \bar{\psi} + \sqrt{Re} c_{b2} \frac{\bar{c}_w}{\sigma} \bar{\rho} \nabla \bar{\nu} \cdot \nabla \bar{\nu} - \frac{Ma \sqrt{Re}}{c_{w1} \bar{\rho} f_w} \left( \frac{\bar{v}_w}{d} \right)^2 - \frac{\sigma}{2} \bar{\nu} (1 + \psi) \nabla \bar{\rho} \cdot \nabla \bar{\rho}$$

From the above three nondimensional formula, it could be observed that the term, $\sqrt{Re \mu}$ or its inverse, occurred there. This is because that the dynamic viscosity, $\mu$, and the turbulent eddy viscosity, $\mu_t$, are scaled by different reference values respectively.

For the setting of a specific flow problem, the nondimensional input parameters include two fixed-value quantities $\bar{\rho}_\infty = 1.0$ and $a_\infty = 1.0$, and five user-adjustable quantities: $M_\infty$, angle of attack $\alpha$, yaw angle $\beta$, $Re_\infty$ and $Pr$. For turbulent flow, the freestream turbulent variable working variable $\rho \bar{v}$ is set equal to $3 \mu_\infty$. With these inputs, a uniform flow field is prescribed for a steady-state problem at the initialization stage and the corresponding conservative variables are

$$\bar{\rho}_\infty = 1.0$$
$$\bar{\rho}u_\infty = M_\infty \cos \alpha \cos \beta$$
$$\bar{\rho}v_\infty = M_\infty \cos \alpha \sin \beta$$
$$\bar{\rho}w_\infty = M_\infty \sin \alpha$$
$$\bar{\rho}c_\infty = \frac{1}{\gamma(\gamma - 1)} + \frac{1}{2} M_\infty^2$$
\[ \rho \nu_\infty = \frac{3.0}{\sqrt{Re_\infty}} \]

The other derived dimensionless variables are

\[ \bar{p}_\infty = \frac{1}{\gamma} \]

\[ \bar{\mu}_\infty = \frac{M_\infty}{Re_\infty} \]

\[ \bar{c}_p = \frac{1}{\gamma - 1} \]

\[ \bar{\lambda} = \bar{\mu} \frac{1}{Pr \gamma - 1} \]

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

Reconstructed Discontinuous
Galerkin Spatial Discretization

In the present work, the governing equations are discretized in space by using the reconstructed
discontinuous Galerkin method. The outline of this chapter is organized in the following. In
section §3.1, the discontinuous Galerkin discretization for the RANS-SA equations is described
in detail, i.e., the weak formulation §3.1.1 and the choice of basis function §3.1.2. In section §3.2,
the reconstrcuted discontinuous Galerkin discretization is introduced in detail. The numerical
integration and the quadrature rules for DG discretization are stated in section §3.3. The
implementation of curved elements is given in section §3.4 in detail. Finally, the implementation
of boundary conditions are described in section §3.5.

3.1 Disconitnuous Galerkin methods

3.1.1 Weak Formulation

The RANS-SA 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,

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

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 $V^p_h$,

$$V^p_h = \{ v_h \in [L^2(\Omega)]^m : v_h|_{\Omega_e} \in [V^p_m] \forall \Omega_e \in \Omega \}$$  \hspace{1cm} (3.2)

which consists of discontinuous vector-values polynomial functions of degree $p$, and where $m$ is the dimension of the unknown vector and

$$V^m_p = \text{span} \{ \prod x_i^{\alpha_i} : 0 \leq \alpha_i \leq p, 0 \leq i \leq d \}$$  \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 $U \in V^p_h$ such as

$$\frac{d}{dt} \int_{\Omega_e} U_h W_h \, d\Omega + \int_{\Gamma_e} F_k(U_h)n_k W_h \, d\Gamma - \int_{\Omega_e} F_k(U_h) \frac{\partial W_h}{\partial x_k} \, d\Omega$$

$$= \int_{\Gamma_e} G_k(U_h, \nabla U_h)n_k W_h \, d\Gamma - \int_{\Omega_e} G_k(U_h, \nabla U_h) \frac{\partial W_h}{\partial x_k} \, d\Omega$$

$$+ \int_{\Omega_e} S(U_h, \nabla U_h) W_h \, d\Omega \quad \forall W_h \in V^p_h$$  \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.

In our work, the inviscid flux function is approximated using the HLLC approximate Riemann solver [121], which has been successfully used to compute compressible viscous and turbulent flows on both structured grids [18] and unstructured grids [77]. 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.

Furthermore, a fully coupled HLLC Riemann solver is employed for the convective numerical flux of the RANS-SA system for all test cases in this work. The explicit form of the HLLC flux evaluated at the integration point of face $ij$ is defined by

$$H_{\text{HLLC}}(U_l, U_r, n_{ij}) = \begin{cases} 
H_l(U_l) & \text{if } S_L > 0 \\
H(U_{l}^*) & \text{if } S_L \leq 0 < S_M \\
H(U_{r}^*) & \text{if } S_M \leq 0 \leq S_R \\
H_r(U_r) & \text{if } S_R < 0 
\end{cases}$$  \hspace{1cm} (3.5)

where the subscript $l$ and $r$ denote the elemental state vectors $U_{hi}$ from cell $i$ and $U_{hj}$ from
cell \( j \) evaluated at the face integration point respectively as follows

\[
U_l = \begin{pmatrix}
\rho_l \\
(pu)_l \\
(pv)_l \\
(pw)_l \\
(pe)_l \\
(p\tilde{v})_l
\end{pmatrix}, \quad U_r = \begin{pmatrix}
\rho_r \\
(pu)_r \\
(pv)_r \\
(pw)_r \\
(pe)_r \\
(p\tilde{v})_r
\end{pmatrix}
\] (3.6)

where the calculation of \( U_l \) and \( U_r \) involves what method we choose to represent the numerical solution in each element, which will be discussed in detailed in next section.

In Eq. 3.5, the symbols with superscript * are defined by

\[
U_l^* = \begin{pmatrix}
\rho_l^* \\
(pu)_l^* \\
(pv)_l^* \\
(pw)_l^* \\
(pe)_l^* \\
(p\tilde{v})_l^*
\end{pmatrix} = \Omega_l \begin{pmatrix}
\rho_l(S_L - q_l) \\
(S_L - q_l)(pu)_l + (p^* - p_l)n_x \\
(S_L - q_l)(pv)_l + (p^* - p_l)n_y \\
(S_L - q_l)(pw)_l + (p^* - p_l)n_z \\
(S_L - q_l)(pe)_l - p_lq_l + p^*S_M \\
(S_L - q_l)(p\tilde{v})_l
\end{pmatrix}
\]

\[
U_r^* = \begin{pmatrix}
\rho_r^* \\
(pu)_r^* \\
(pv)_r^* \\
(pw)_r^* \\
(pe)_r^* \\
(p\tilde{v})_r^*
\end{pmatrix} = \Omega_r \begin{pmatrix}
\rho_r(S_R - q_r) \\
(S_R - q_r)(pu)_r + (p^* - p_r)n_x \\
(S_R - q_r)(pv)_r + (p^* - p_r)n_y \\
(S_R - q_r)(pw)_r + (p^* - p_r)n_z \\
(S_R - q_r)(pe)_r - p_rq_r + p^*S_M \\
(S_R - q_r)(p\tilde{v})_r
\end{pmatrix}
\] (3.7) (3.8)

\[
H_l^* \equiv H(U_l^*) = \begin{pmatrix}
\rho_l^*S_M \\
((pu)_l^*S_M + p^*n_x) \\
((pv)_l^*S_M + p^*n_y) \\
((pw)_l^*S_M + p^*n_z) \\
((pe)_l^* + p^*)S_M \\
((p\tilde{v})_l^*S_M)
\end{pmatrix}, \quad H_r^* \equiv H(U_r^*) = \begin{pmatrix}
\rho_r^*S_M \\
((pu)_r^*S_M + p^*n_x) \\
((pv)_r^*S_M + p^*n_y) \\
((pw)_r^*S_M + p^*n_z) \\
((pe)_r^* + p^*)S_M \\
((p\tilde{v})_r^*S_M)
\end{pmatrix}
\]

\[
\Omega_l \equiv (S_L - S_M)^{-1}, \quad \Omega_r \equiv (S_R - S_M)^{-1},
\]

\[
p^* = \rho_l(q_l - S_L)(q_l - S_M) + p_l = \rho_r(q_r - S_R)(q_r - S_M) + p_r
\]

\[
q_l \equiv u_ln_x + v_ln_y + w_ln_z, \quad q_r \equiv u_rn_x + v_rn_y + w_rn_z
\]

\[
(3.10) \quad (3.11) \quad (3.12)
\]
with \((n_x, n_y, n_z)^T\) being the unit vector normal to face \(ij\). \(S_M\) is taken from Batten et al. [17]:

\[
S_M = \frac{\rho r q_r (S_R - q_r) - \rho q_l (S_L - q_l) + p_l - p_r}{\rho r (S_R - q_r) - p_l (S_L - q_l)}
\]

(3.13)

and \(S_L, S_R\) are taken from Einfeldt et al. [44]:

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

(3.14)

where \(\lambda_1(U^{Roe})\) and \(\lambda_m(U^{Roe})\) are the smallest and largest eigenvalues of the Roe matrix [113].

Among the many possible schemes [11, 15, 33, 19, 100, 46, 87, 125, 124, 106] 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) [15] for the discretization of the viscous fluxes.

Assuming that \(B\) is the basis of polynomial function of degrees \(p\), this is then equivalent to the following system of \(N\) equations,

\[
\frac{d}{dt} \int_{\Omega_e} U_h B_i \, d\Omega + \int_{\Gamma_e} H_{k}^{inv}(U_{hl}, U_{hr}, n_k) B_i \, d\Gamma - \int_{\Omega_e} F_k(U_h) \frac{\partial B_i}{\partial x_k} \, d\Omega = \int_{\Gamma_e} H_{k}^{vis}(U_{hl}, \nabla U_{hl} + \beta r_l, U_{hr}, \nabla U_{hr} + \beta r_r, n_k) B_i \, d\Gamma \\
- \int_{\Omega_e} G_k(U_h, \nabla U_h + R) \frac{\partial B_i}{\partial x_k} \, d\Omega + \int_{\Omega_e} S(U_h, \nabla U_h + R) B_i \, d\Omega \quad 1 \leq i \leq N
\]

(3.15)

where \(N\) is the dimension of the polynomial space. The inviscid and viscous flux function appearing in Eq. 3.15 is replaced by numerical flux function \(H_{k}^{inv}\) and \(H_{k}^{vis}\), respectively, where \(U_{hr}\) and \(U_{hl}\) are the solution polynomial at the left and right states of the cell interface. In the viscous flux, \(r\) in the fifth term is the local lifting operator, \(\beta\) is called penalty parameter, and \(R\) is the global lifting operator. We need to mention that, through our numerical experiment, there exists lower bounds of the parameter \(\beta\) to ensure the stability of the method, whereas lower \(\beta\) could give better convergences without spoiling the accuracy of the numerical solution. 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.
3.1.2 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:

$$U_h = \sum_{i=1}^{N} U_i B_i(x) \quad (3.16)$$

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

![Figure 3.1: Representation of polynomial solutions using finite element shape functions: (a) $Q_1/P_1$; (b) $Q_2/P_2$.](image)

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:

\[
U_h = U_c + \frac{\partial U}{\partial x} \bigg|_c (x - x_c) + \frac{\partial U}{\partial y} \bigg|_c (y - y_c) + \frac{\partial U}{\partial z} \bigg|_c (z - z_c) \\
+ \frac{\partial^2 U}{\partial x^2} \bigg|_c \left( \frac{(x - x_c)^2}{2} \right) + \frac{\partial^2 U}{\partial y^2} \bigg|_c \left( \frac{(y - y_c)^2}{2} \right) + \frac{\partial^2 U}{\partial z^2} \bigg|_c \left( \frac{(z - z_c)^2}{2} \right) \\
+ \frac{\partial^2 U}{\partial x \partial y} \bigg|_c (x - x_c)(y - y_c) + \frac{\partial^2 U}{\partial x \partial z} \bigg|_c (x - x_c)(z - z_c) + \frac{\partial^2 U}{\partial y \partial z} \bigg|_c (y - y_c)(z - z_c)
\] (3.17)

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

\[
U_h = \bar{U} + \frac{\partial \bar{U}}{\partial x} \bigg|_c (x - x_c) + \frac{\partial \bar{U}}{\partial y} \bigg|_c (y - y_c) + \frac{\partial \bar{U}}{\partial z} \bigg|_c (z - z_c) \\
+ \frac{\partial^2 \bar{U}}{\partial x^2} \bigg|_c \left( \frac{(x - x_c)^2}{2} - \frac{1}{\Omega_e} \int_{\Omega_e} (x - x_c)^2 \, d\Omega \right) \\
+ \frac{\partial^2 \bar{U}}{\partial y^2} \bigg|_c \left( \frac{(y - y_c)^2}{2} - \frac{1}{\Omega_e} \int_{\Omega_e} (y - y_c)^2 \, d\Omega \right) \\
+ \frac{\partial^2 \bar{U}}{\partial z^2} \bigg|_c \left( \frac{(z - z_c)^2}{2} - \frac{1}{\Omega_e} \int_{\Omega_e} (z - z_c)^2 \, d\Omega \right) \\
+ \frac{\partial^2 \bar{U}}{\partial x \partial y} \bigg|_c (x - x_c)(y - y_c) - \frac{1}{\Omega_e} \int_{\Omega_e} (x - x_c)(y - y_c) \, d\Omega \\
+ \frac{\partial^2 \bar{U}}{\partial x \partial z} \bigg|_c (x - x_c)(z - z_c) - \frac{1}{\Omega_e} \int_{\Omega_e} (x - x_c)(z - z_c) \, d\Omega \\
+ \frac{\partial^2 \bar{U}}{\partial y \partial z} \bigg|_c (y - y_c)(z - z_c) - \frac{1}{\Omega_e} \int_{\Omega_e} (y - y_c)(z - z_c) \, d\Omega
\] (3.18)

where \( \bar{U} \) 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. 3.15 then leads to the following ten equations

\[
\frac{d}{dt} \int_{\Omega_e} \bar{U} \, d\Omega + \int_{\Gamma_e} H_{k}^{inv}(U_{hl}, U_{hr}, n_k) \, d\Gamma \\
= \int_{\Gamma_e} H_{k}^{vis}(U_{hl}, \nabla U_{hl} + \beta_{re}, U_{hr}, \nabla U_{hr} + \beta_{re}, n_k) \, d\Gamma + \int_{\Omega_e} S(U_h, \nabla U_h + R) \, d\Omega \quad i = 1 \tag{3.20}
\]

\[
\sum_{j=2}^{10} \int_{\Omega_e} B_i B_j \, d\Omega \frac{d}{dt} \left( \begin{array}{c}
\frac{\partial U}{\partial x} | c \\
\frac{\partial U}{\partial y} | c \\
\frac{\partial U}{\partial z} | c \\
\frac{\partial^2 U}{\partial x^2} | c \\
\frac{\partial^2 U}{\partial x \partial y} | c \\
\frac{\partial^2 U}{\partial x \partial z} | c \\
\frac{\partial^2 U}{\partial y^2} | c \\
\frac{\partial^2 U}{\partial y \partial z} | c \\
\frac{\partial^2 U}{\partial z^2} | c \\
\end{array} \right) + \int_{\Gamma_e} H_{k}^{inv}(U_{hl}, U_{hr}, n_k) B_i \, d\Gamma - \int_{\Omega_e} F_k(U_h) \frac{\partial B_i}{\partial x_k} \, d\Omega \quad i = 1 \tag{3.21}
\]

\[
= \int_{\Gamma_e} H_{k}^{vis}(U_{hl}, \nabla U_{hl} + \beta_{re}, U_{hr}, \nabla U_{hr} + \beta_{re}, n_k) B_i \, d\Gamma \\
- \int_{\Omega_e} G_k(U_h, \nabla U_h + R) \frac{\partial B_i}{\partial x_k} \, d\Omega + \int_{\Omega_e} S(U_h, \nabla U_h + R) B_i \, d\Omega \quad 2 \leq i \leq 10
\]

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
\[ \int_{\Omega_e} B_i B_i \, d\Omega = 0 \quad 2 \leq i \leq 10 \quad (3.22) \]

In the implementation of this DG method, the basis functions are actually normalized in order to improve the conditioning of the system matrix Eq. 3.21 as follows:

\[ \tilde{B}_1 = 1 \quad \tilde{B}_2 = \frac{x - x_c}{\Delta x} \quad \tilde{B}_3 = \frac{y - y_c}{\Delta y} \quad \tilde{B}_4 = \frac{z - z_c}{\Delta z} \]

\[ \tilde{B}_5 = \frac{(x - x_c)^2}{2\Delta x^2} - \frac{1}{\Omega_e} \int_{\Omega_e} \frac{(x - x_c)^2}{2\Delta x^2} \, d\Omega \]

\[ \tilde{B}_6 = \frac{(y - y_c)^2}{2\Delta y^2} - \frac{1}{\Omega_e} \int_{\Omega_e} \frac{(y - y_c)^2}{2\Delta y^2} \, d\Omega \]

\[ \tilde{B}_7 = \frac{(z - z_c)^2}{2\Delta z^2} - \frac{1}{\Omega_e} \int_{\Omega_e} \frac{(z - z_c)^2}{2\Delta z^2} \, d\Omega \]

\[ \tilde{B}_8 = \frac{(x - x_c)(y - y_c)}{\Delta x \Delta y} - \frac{1}{\Omega_e} \int_{\Omega_e} \frac{(x - x_c)(y - y_c)}{\Delta x \Delta y} \, d\Omega \]

\[ \tilde{B}_9 = \frac{(x - x_c)(z - z_c)}{\Delta x \Delta z} - \frac{1}{\Omega_e} \int_{\Omega_e} \frac{(x - x_c)(z - z_c)}{\Delta x \Delta z} \, d\Omega \]

\[ \tilde{B}_{10} = \frac{(y - y_c)(z - z_c)}{\Delta y \Delta z} - \frac{1}{\Omega_e} \int_{\Omega_e} \frac{(y - y_c)(z - z_c)}{\Delta y \Delta z} \, d\Omega \]

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

\[ U_{x,c} = \frac{\partial U}{\partial x} \bigg|_{c} \Delta x \quad U_{y,c} = \frac{\partial U}{\partial y} \bigg|_{c} \Delta y \quad U_{z,c} = \frac{\partial U}{\partial z} \bigg|_{c} \Delta z \]

\[ U_{xx,c} = \frac{\partial^2 U}{\partial x^2} \bigg|_{c} \Delta x^2 \quad U_{yy,c} = \frac{\partial^2 U}{\partial y^2} \bigg|_{c} \Delta y^2 \quad U_{zz,c} = \frac{\partial^2 U}{\partial z^2} \bigg|_{c} \Delta z^2 \quad (3.24) \]

\[ U_{xy,c} = \frac{\partial^2 U}{\partial x \partial y} \bigg|_{c} \Delta x \Delta y \quad U_{xz,c} = \frac{\partial^2 U}{\partial x \partial z} \bigg|_{c} \Delta x \Delta z \quad U_{yz,c} = \frac{\partial^2 U}{\partial y \partial z} \bigg|_{c} \Delta y \Delta z \]

Finally, the quadratic polynomial solutions can be rewritten as

\[ U_h = \tilde{U} + U_{x,c} \tilde{B}_2 + U_{y,c} \tilde{B}_3 + U_{z,c} \tilde{B}_4 + U_{xx,c} \tilde{B}_5 + U_{yy,c} \tilde{B}_6 + U_{zz,c} \tilde{B}_7 + U_{xy,c} \tilde{B}_8 + U_{xz,c} \tilde{B}_9 + U_{yz,c} \tilde{B}_{10} \quad (3.25) \]

This is especially helpful and important to remove the stiffness of the system matrix for higher-order DG approximations.
This formulation has a number of distinct, desirable, and attractive features and advantages.[79]. First, the same numerical polynomial solutions are used for any shape of elements, e.g., triangle, quadrilateral and polygon in 2D, and tetrahedron, pyramid, prism and hexahedron in 3D. Therefore, 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 [74, 79, 105, 103, 104], 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 [78, 81] 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.

3.2 Reconstructed discontinuous Galerkin methods

Compared with 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, especially for implicit method. 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. Obviously, an accurate and efficient reconstruction is the crucial step to extend the underlying DG method to higher order accuracy. The discussion in this work is mainly focused on the third-order DG method $r$DG($P_1P_2$).

3.2.1 Least-Squares Reconstruction

In the case of DG($P_1$) method, a linear polynomial solution $\mathbf{U}_i$ in any cell $i$ is

$$\mathbf{U}_i = \hat{\mathbf{U}}_i + \mathbf{U}_{x,i}\hat{B}_2 + \mathbf{U}_{y,i}\hat{B}_3 + \mathbf{U}_{z,i}\hat{B}_4$$

(3.26)
Using this underlying linear polynomial DG solution in the neighboring cells, one can reconstruct a quadratic polynomial solution $U^R_i$ as follows:

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

(3.27)

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 3.3. 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.,

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

(3.28)

Figure 3.3: Representation of the four face-neighboring cells surrounding a tetrahedron (a) and the six face-neighboring cells surrounding a hexahedron (b).
As a result, only six second derivatives need to be determined. This can be accomplished as follows. For any neighboring cell \( j \), one requires

\[
\ddot{U}_j = \frac{1}{\Omega_j} \int_{\Omega_j} \left( \ddot{U}_i + \dot{U}_{x,i} \ddot{B}_2^j + \dot{U}_{y,i} \ddot{B}_3^j + \dot{U}_{z,i} \ddot{B}_4^j \\ + U_{x,x,i} \ddot{B}_5^j + U_{y,y,i} \ddot{B}_6^j + U_{z,z,i} \ddot{B}_7^j + U_{x,y,i} \ddot{B}_8^j + U_{x,z,i} \ddot{B}_9^j + U_{y,z,i} \ddot{B}_{10}^j \right) d\Omega_j
\]

\[
\frac{\partial U}{\partial x} \bigg|_{j} = U_{x,x,i} \frac{1}{\Delta x_i} + U_{x,y,i} \frac{\ddot{B}_2^j}{\Delta x_i} + U_{x,z,i} \frac{\ddot{B}_4^j}{\Delta x_i} + U_{y,x,i} \frac{\ddot{B}_2^j}{\Delta x_i} + U_{z,x,i} \frac{\ddot{B}_4^j}{\Delta x_i}
\]

\[
\frac{\partial U}{\partial y} \bigg|_{j} = U_{y,y,i} \frac{1}{\Delta y_i} + U_{y,z,i} \frac{\ddot{B}_3^j}{\Delta y_i} + U_{z,y,i} \frac{\ddot{B}_3^j}{\Delta y_i} + U_{x,y,i} \frac{\ddot{B}_3^j}{\Delta y_i} + U_{z,y,i} \frac{\ddot{B}_3^j}{\Delta y_i}
\]

\[
\frac{\partial U}{\partial z} \bigg|_{j} = U_{z,z,i} \frac{1}{\Delta z_i} + U_{z,x,i} \frac{\ddot{B}_5^j}{\Delta z_i} + U_{x,z,i} \frac{\ddot{B}_5^j}{\Delta z_i} + U_{y,z,i} \frac{\ddot{B}_5^j}{\Delta z_i} + U_{z,z,i} \frac{\ddot{B}_5^j}{\Delta z_i}
\]

where the basis functions \( B \) are evaluated at the center of cell \( j \), i.e., \( \ddot{B}^j = \ddot{B}(x_j, y_j, z_j) \). The explicit expressions about \( \ddot{B}(x, y, z) \) could be found by Eq. 3.23.

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

\[
A \times \left( \begin{array}{c} U_{x,i}^R \\ U_{y,i}^R \\ U_{z,i}^R \\ U_{x,y,i}^R \\ U_{x,z,i}^R \\ U_{y,z,i}^R \end{array} \right) = \left( \begin{array}{c} \ddot{U}_j - \left( \ddot{U}_i \int_{\Omega_j} \ddot{B}_2^j d\Omega_j + \dot{U}_{x,i} \int_{\Omega_j} \ddot{B}_4^j d\Omega_j + \dot{U}_{y,i} \int_{\Omega_j} \ddot{B}_3^j d\Omega_j + \dot{U}_{z,i} \int_{\Omega_j} \ddot{B}_4^j d\Omega_j \right) \\ \Delta x_i \ddot{U}_{x,j} - \ddot{U}_{x,i} \\ \Delta y_i \ddot{U}_{y,j} - \ddot{U}_{y,i} \\ \Delta z_i \ddot{U}_{z,j} - \ddot{U}_{z,i} \end{array} \right)
\]

\[
(3.30)
\]

where,

\[
A = \left( \begin{array}{cccccc} \int_{\Omega_j} \ddot{B}_2^j d\Omega_j & \int_{\Omega_j} \ddot{B}_3^j d\Omega_j & \int_{\Omega_j} \ddot{B}_4^j d\Omega_j & \int_{\Omega_j} \ddot{B}_5^j d\Omega_j & \int_{\Omega_j} \ddot{B}_6^j d\Omega_j & \int_{\Omega_j} \ddot{B}_{10}^j d\Omega_j \\ \ddot{B}_2^j & 0 & 0 & \ddot{B}_3^j & \ddot{B}_4^j & 0 \\ 0 & \ddot{B}_3^j & 0 & \ddot{B}_2^j & 0 & \ddot{B}_4^j \\ 0 & 0 & \ddot{B}_4^j & 0 & \ddot{B}_2^j & \ddot{B}_3^j \end{array} \right)
\]

\[
(3.31)
\]

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. The associated numerical details could be found in [138].

### 3.2.2 WENO Reconstruction at P₂: WENO(P₁P₂)

This linear reconstruction-based rDG(P₁P₂) 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 [86, 84, 149, 91]. 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(P₀P₁) [49, 139]. This linear instability is attributed to the fact that the reconstruction stencils only involve the von Neumann neighborhood, i.e., adjacent face-neighboring cells [49]. And it can be achieved using extended stencils, which will unfortunately sacrifice the compactness of the underlying DG methods. In the present work [90], 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,

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

(3.32)

where \( N_{es} \) denotes the number of its face-neighboring cells (\( N_{es} \) is 4 for tetrahedron, 5 for pyramid and prism, and 6 for hexahedron), and the normalized nonlinear weights \( w_k \) are computed as

\[
w_k = \frac{\tilde{w}_k}{1+N_{es} \sum_{i=1}^{N_{es}} \tilde{w}_i}
\]  

(3.33)

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

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

(3.34)

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

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

(3.35)
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. 3.15. The resulting DG method, termed a “reconstructed DG” method (rdDG(P_1P_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. Let’s take a quadratic curved hexahedral element into consideration, compared with DG(P_1) the extra costs are mainly due to the least-squares reconstruction; However, when it comes to DG(P_2), in addition to less degrees of freedom, many less gauss quadrature points are required to complete the face and domain integration. Especially for implicit method, implicit rdDG(P_1P_2) will outperform the implicit DG(P_2) in terms of storage requirement of the block diagonal Jacobian matrix, i.e., 24 \( \times \) 24 vs. 60 \( \times \) 60. A summary of cost analysis on a quadratic curved hexahedral grid is presented in Table 3.1, where the numbers of Gauss quadrature points for both the domain and boundary integrals required by the rdDG(P_1P_2) method are the same as the DG(P_1) method, but much less than the DG(P_2) method. The similar summary of cost analysis for the rdDG(P_0P_1) (FV(P_1)), DG(P_1), rdDG(P_1P_2) and DG(P_2) on a tetrahedral cell could refer to [138]. It also could be observed that, compared with rdDG method, the storage requirements for the implicit DG methods are extremely demanding, especially for higher-order DG methods. Note that this rdDG method is not compact anymore, as the neighbor’s neighbors are used in updating the solution. However, the stencils used in the reconstruction are compact, involving only von Neumann neighbors. Consequently, the resultant rdDG method can be implemented in a compact manner.

| Table 3.1: Cost analysis for different numerical methods on a quadratic curved hexahedral grid |
|-----------------------------------------|-----------------|-----------------|-----------------|-----------------|
|                                        | rDG(P_0P_1)     | DG(P_1)         | rDG(P_1P_2)     | DG(P_2)         |
| Number of quadrature points for boundary integrals | 1               | 9               | 9               | 16              |
| Number of quadrature points for domain integrals | 0               | 27              | 27              | 64              |
| Reconstruction                         | Yes             | No              | Yes             | No              |
| Order of accuracy                      | \( \mathcal{O}(h^2) \) | \( \mathcal{O}(h^2) \) | \( \mathcal{O}(h^3) \) | \( \mathcal{O}(h^3) \) |
| Storage for implicit diagonal matrix   | 36 words per element | 576             | 576             | 3600            |
3.2.3 WENO Reconstruction at $P_1$: HWENO($P_1P_2$)

The WENO($P_1P_2$) method cannot remove inherent oscillations in the underlying DG($P_1$) solutions. 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($P_1P_2$) [92] in this work, where a hierarchical reconstruction (successively from high order to low order) strategy [144] 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 $(i,j_2,j_3,j_4)$ designate the four adjacent face-neighboring cells of the cell $i$ are chosen to construct a Lagrange polynomial such that

$$
\tilde{U}_j = \frac{1}{\Omega_j} \int_{\Omega_j} \left( U_{i}^R + U_{x,i}^R \tilde{B}_j^2 + U_{y,i}^R \tilde{B}_j^3 + U_{z,i}^R \tilde{B}_j^4 \right) 
+ U_{x,i}^R \tilde{B}_j^5 + U_{y,i}^R \tilde{B}_j^6 + U_{z,i}^R \tilde{B}_j^7 + U_{x,z,i}^R \tilde{B}_j^8 + U_{y,z,i}^R \tilde{B}_j^9 + U_{z,y,i}^R \tilde{B}_j^{10} \right) d\Omega_j \tag{3.36}
$$

where $\tilde{U}_j$ refers to the cell-averaged value of the reconstructed polynomial solution at 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

$$
\frac{\partial U}{\partial x} \bigg|_j = U_{x,i}^R \frac{1}{\Delta x_i} + U_{x,i}^R \tilde{B}_j^2 \frac{1}{\Delta x_i} + U_{x,x,i}^R \tilde{B}_j^3 \frac{1}{\Delta x_i} + U_{x,z,i}^R \tilde{B}_j^4 \frac{1}{\Delta x_i} \tag{3.37}
$$

$$
\frac{\partial U}{\partial y} \bigg|_j = U_{y,i}^R \frac{1}{\Delta y_i} + U_{y,y,i}^R \tilde{B}_j^3 \frac{1}{\Delta y_i} + U_{y,x,i}^R \tilde{B}_j^4 \frac{1}{\Delta y_i} + U_{y,z,i}^R \tilde{B}_j^5 \frac{1}{\Delta y_i} \tag{3.37}
$$

$$
\frac{\partial U}{\partial z} \bigg|_j = U_{z,i}^R \frac{1}{\Delta z_i} + U_{z,z,i}^R \tilde{B}_j^4 \frac{1}{\Delta z_i} + U_{z,x,i}^R \tilde{B}_j^5 \frac{1}{\Delta z_i} + U_{z,y,i}^R \tilde{B}_j^6 \frac{1}{\Delta z_i} \tag{3.37}
$$

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

$$
\frac{\partial U}{\partial x_m} \bigg|_{WENO} = \sum_{k=1}^{N_{sten}} w_k \frac{\partial U}{\partial x_m} \bigg|_k \tag{3.38}
$$
where, $N_{sten}$ denotes the number of stencils. In general, $N_{sten}$ is equal to $C_{Nes}^3 + N_{es} + 1$ 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

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

(3.39)

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

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

(3.40)

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

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

(3.41)

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 only neighbour’s information is required to complete the reconstruction, which makes the implementation of parallelization easy. 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 [79]. 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. A number of numerical experiments for a variety of flow conditions have been conducted to demonstrate the accuracy, robustness, and non-oscillatory performance of the HWENO($P_1P_2$) method, as presented in Reference [92, 138].
3.3 Numerical Integration

The domain and boundary integrals in Eq. 3.15 are calculated using 2p and 2p + 1 order accurate Gauss quadrature formulas, respectively [80]. The number of gauss quadrature points necessary for a given order depends on the quadrature rule used. In the case of linear, quadratic, and cubic shape function, the domain integrals are evaluated using 3, 6, and 12 points respectively, for 2D. In 3D, integration over the elements for \( P_1 \) and \( P_2 \) approximation is performed using 4 and 11 quadrature points, respectively, and integration over the element boundaries for \( P_0 \), \( P_1 \), and \( P_2 \) is performed using 1, 4, and 7 quadrature points, respectively [80]. The detailed procedures of numerical integration, e.g., the principles and complete concepts, can refer to some well-accepted textbooks [58, 72].

Since different elements have different shapes, it is not a trivial work to develop the shape functions for individual elements. Instead, the reference element is used and the shape function is defined in the reference element. Therefore, numerical integration over a control face, i.e., inviscid flux integration over a triangle, can be performed with respect to the reference coordinates \((\xi, \eta)\):

\[
\int_{\Gamma_e} F_k(U_h) n_k B_i \, d\Gamma = \int_{\Gamma_e} H_k \left( \sum_{j=1}^{N} B_j^L u_j^L, \sum_{j=1}^{N} B_j^R u_j^R, n_k \right) B_i^L \, d\Gamma
\]

\[
= \sum_{l=1}^{M} H_k \left( \sum_{j=1}^{N} B_j^L u_j^L, \sum_{j=1}^{N} B_j^R u_j^R, n_k^l \right) B_i^L w_l |J_{\Gamma_e}(\xi_l, \eta_l)| \quad 1 \leq i \leq N \tag{3.42}
\]

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, and \(|J_{\Gamma_e}(\xi_l, \eta_l)|\) represents the Jacobian of the transformation from the physical face to reference face. In Eq. 3.42, \( w_l \) are the associated weighting factors, \( n_k^l \) are the unit vector normal to the differential face \( d\Gamma \) at the integration point, and \( M \) is the total number of integration points.

The associated weights and the gauss quadrature point coordinate in reference frame could refer to Appendix B of Xia’s PhD thesis [138].

Similarly, the domain integral, e.g., inviscid flux integration over a tetrahedron can be achieved as follows:

\[
\int_{\Omega_e} F_k(U_h) \frac{\partial B_i}{\partial x_k} \, d\Omega = \sum_{i=1}^{M} F_k \left( \sum_{j=1}^{N} B_j u_j \right) \frac{\partial B_i}{\partial x_k} w_l |J_{\Omega_e}(\xi_l, \eta_l, \zeta_l)| \quad 1 \leq i \leq N \tag{3.43}
\]

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 of Xia’s PhD thesis [138].

3.4 Implementation of Curved Elements

Correct representation of physical boundary is crucial for developing accurate numerical schemes [69, 12]. Difficulties can arise when a piecewise linear approximation is used to represent the physical geometry. Large errors may arise in the boundary layer and then pollute the solution inside the domain. This is especially important for DG methods, where errors due to geometrical approximation may exceed the discretization error, rendering the use of a higher-order scheme useless. As boundary elements are curved to conform with the physical boundary, invalid negative cells are likely to be generated, particularly when highly stretched elements are applied in viscous boundary layer for high Reynolds numbers, e.g., turbulent flows.

Thus, for viscous flow with high Reynolds numbers, high-order curved elements are employed not only for the physical boundary cells but also for the interior cells, especially in the boundary layer. By doing this, the negative cells could be avoided. It is thus necessary to somehow propagate the curvature inside the domain. Several smoothing schemes have been proposed in the literature to this effect: linear smoothing techniques such as Laplacian smoothing [47], Winslow smoothing [66], linear elasticity with varying stiffness, and nonlinear smoothing techniques [101], etc.

In this work, two methods are used to generate the quadratic curved meshes, i.e., GMSH library [48] and simple agglomeration [56].

Gmsh is a free 3D finite element grid generator with a build-in CAD engine and post-processor. The function, i.e., high order tools, could generate the curved meshes based on the linear mesh and CAD geometry. There exist 3 schemes for high order tools, i.e., Linear elasticity analogy, fast curving, optimization. Linear elasticity analogy assumes that the computational mesh obeys the isotropic linear elasticity relations. Here, in our work, usually, elasticity analogy, is used. A schematic diagram about how linear elasticity works is given in Figure 3.4. In addition to GMSH, also the simple, yet effective agglomeration has been used to obtain the curved meshes in our code. It is quite straightforward to generate the curved meshes based on block-structured meshes. There exist one block-structured mesh, of which the element number in every direction, i.e., i, j, k, is the multiple of p. Then the curved mesh could be formed as follows [56]. First, keep every \( p^{th} \) point in every direction; then, use the remaining points to define polynomial functions of degree \( p \), namely the high-order nodes for curved element. The simple schematic is given in Figure 3.5. This section takes the case, i.e., turbulent flow over a 2D bump-in-channel as an example to show the quadratic mesh, which is generated by keeping only second point in every direction in Figure 3.6. In this work, the implementation of quadratic curved elements
Figure 3.4: Transformation of an element in \((x, y)\) physical space into a canonical element reference \((\xi, \eta)\) space

Figure 3.5: Transformation of an element in \((x, y)\) physical space into a canonical element reference \((\xi, \eta)\) space
Figure 3.6: Transformation of an element in \((x, y, z)\) physical space into a canonical element reference \((\xi, \eta, \zeta)\) space.
for 3D domain integral and face integral for discontinuous Galerkin methods is introduced in brief, since of curved elements some textbooks [58, 72] could also be referred to for the detailed information. The numerical integration for the discretized weak formulation of the governing equations is also described concisely.

### 3.4.1 Transformation from physical to reference space

First, the quadratic curved face has been implemented in our framework. Since different elements have different shapes, it is not a trivial work to develop shape functions for individual elements on physical space. Instead, the concept of mapping to the reference element is used and the shape function is defined in the reference element to transform the gaussian quadrature from physical to reference space. Then all computational work could be done on the reference domain. This approach also is efficient from the point of view of computer memory, since the numerical quadrature data are stored on the reference domain only. The shape functions and their partial derivatives can be stored via their values at integration points in the reference domain.

An example is shown in Figure 3.7, where a curvilinear quadrilateral face (2D or 3D) is mapped to its corresponding 2D reference face in the normalized coordinate system. The physical element is defined in x-y-z system, whereas the reference element is defined in ξ-η system. According to the transformation from the reference system to the physical system, the co-

---

**Figure 3.7:** Transformation of an element in \((x, y)\) physical space into a canonical element reference \((\xi, \eta)\) space
ordinate variables $x$, $y$ and $z$ in the physical system could be expressed using the standard Lagrangian shape functions in the form

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

(3.44)

where $(x_i, y_i, z_i)$ represents the face-wise geometric mapping coefficients, which are given by the coordinates of the face vertices as well as additional high-order nodes on the surface, and $\phi_i$ denotes the nodal shape function of node $i$. The nodal shape function $\phi_i$ is defined as follows,

$$\begin{align*}
\phi_i(\xi, \eta) &= 1, \quad \text{at node } i \\
\phi_i(\xi, \eta) &= 0, \quad \text{at other nodes}
\end{align*}$$

(3.45)

Note that by this kind of transformation, the mapping can be constructed from every face in the physical space to the reference face. It is well known that a parametric surface is a surface in the Euclidean space $\mathbb{R}^3$ which is defined by a parametric equation with two parameters $\vec{r}: \mathbb{R}^2 \to \mathbb{R}^3$. Here, $\vec{r} = (x, y, z) = (x(\xi, \eta), y(\xi, \eta), z(\xi, \eta))$. Therefore, $\xi$ and $\eta$ are the two parameters. According to the parametric surface, the unit vector normal to the differential face $dA$ at some point, e.g., the Gaussian integration quadrature, can be calculated by

$$n_r = \frac{\vec{r}_\xi \times \vec{r}_\eta}{|\vec{r}_\xi \times \vec{r}_\eta|}$$

(3.46)

where $\vec{r}_\xi = (x_\xi, y_\xi, z_\xi)$ and $\vec{r}_\eta = (x_\eta, y_\eta, z_\eta)$. Obviously, for linear/bilinear face, $n_r$ is a constant vector. Note that the grid topology should guarantee that the direction of $n_r$ points outward of its host cell.

From Eq. 3.42, the differential area $d\Gamma$ needs to transformed from physical surface to reference face as follows,

$$d\Gamma = |\vec{r}_\xi \times \vec{r}_\eta| d\xi d\eta$$

(3.47)

where $|\vec{r}_\xi \times \vec{r}_\eta|$ is nothing but the Jacobian matrix $|J_\Gamma(\xi, \eta)|$. And the Jacobian of the transformation $|J_\Gamma(\xi, \eta)|$ is calculated by

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

(3.48)

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

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

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

\[
dz(\xi, \eta) = \sum_{i} z_i \frac{\partial \phi_i(\xi, \eta)}{\partial \xi} d\xi + \sum_{i} z_i \frac{\partial \phi_i(\xi, \eta)}{\partial \eta} d\eta
\] (3.49)

From Eq. 3.44 to Eq. 3.49, the parameters required by the numerical face integration in Eq. 3.42, e.g., \( n_l^k \) and \( |J_{\Gamma_e}(\xi_l, \eta_l)| \), could be specified.

Second, from Eq. 3.21, in addition to the face integral, DG method also needs to calculate the domain integral. Similarly, the shape function is defined in a reference volume. And the physical volume is defined using x-y-z, whereas the reference one is defined in \( \xi-\eta-\zeta \) system. For the hexahedral mesh, the reference volume is a cubic spanning between \( 1 \leq \xi, \eta, \zeta \leq 1 \). An example is shown in Figure 3.8, where a curvilinear hexahedral is mapped to its corresponding 3D reference volume in the normalized coordinate system. Similar to the transformation between the physical face and the reference face, the coordinate variables \( x, y \) and \( z \) in the physical domain could be expressed using the standard Lagrangian shape functions in the form

\[
x(\xi, \eta, \zeta) = \sum_{i} x_i \phi_i(\xi, \eta, \zeta), \quad y(\xi, \eta, \zeta) = \sum_{i} y_i \phi_i(\xi, \eta, \zeta), \quad z(\xi, \eta, \zeta) = \sum_{i} z_i \phi_i(\xi, \eta, \zeta)
\] (3.50)

where \( N_e \) is equal to the number of the shape functions for the element. The 27-node shape functions and derivatives for quadratic hexahedron are expressed in §A.4.1, respectively. The Jacobian matrix \( |J_{\Omega}(\xi, \eta)| \) is necessary to transform the integral in the physical domain to that in the reference domain, as shown in the following Eq. 3.51,

\[
dV(\xi, \eta, \zeta) = |J_{\Omega}(\xi, \eta)| d\xi d\eta d\zeta
\] (3.51)

And the Jacobian of the transformation \( |J_{\Omega}(\xi, \eta)| \) is calculated by

\[
|J_{\Omega}(\xi, \eta)| = \begin{vmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial x}{\partial \eta} & \frac{\partial x}{\partial \zeta} \\ \frac{\partial y}{\partial \xi} & \frac{\partial y}{\partial \eta} & \frac{\partial y}{\partial \zeta} \\ \frac{\partial z}{\partial \xi} & \frac{\partial z}{\partial \eta} & \frac{\partial z}{\partial \zeta} \end{vmatrix}
\] (3.52)

Note that the functions \( x, y \) and \( z \) are differentiable with respect to the local coordinates \( \xi \) and
Figure 3.8: Transformation of an element in \((x, y, z)\) physical space into a canonical element reference \((\xi, \eta, \zeta)\) space.

\[
\begin{align*}
\eta \text{ in reference domain:} &= dx(\xi, \eta, \zeta) = \frac{\partial x}{\partial \xi} d\xi + \frac{\partial x}{\partial \eta} d\eta + \frac{\partial x}{\partial \zeta} d\zeta = \sum_{i}^{N_e} x_i \frac{\partial \phi_i}{\partial \xi} d\xi + \sum_{i}^{N_e} x_i \frac{\partial \phi_i}{\partial \eta} d\eta + \sum_{i}^{N_e} x_i \frac{\partial \phi_i}{\partial \zeta} d\zeta \\
&= dy(\xi, \eta, \zeta) = \frac{\partial y}{\partial \xi} d\xi + \frac{\partial y}{\partial \eta} d\eta + \frac{\partial y}{\partial \zeta} d\zeta = \sum_{i}^{N_e} y_i \frac{\partial \phi_i}{\partial \xi} d\xi + \sum_{i}^{N_e} y_i \frac{\partial \phi_i}{\partial \eta} d\eta + \sum_{i}^{N_e} y_i \frac{\partial \phi_i}{\partial \zeta} d\zeta \\
&= dz(\xi, \eta, \zeta) = \frac{\partial z}{\partial \xi} d\xi + \frac{\partial z}{\partial \eta} d\eta + \frac{\partial z}{\partial \zeta} d\zeta = \sum_{i}^{N_e} z_i \frac{\partial \phi_i}{\partial \xi} d\xi + \sum_{i}^{N_e} z_i \frac{\partial \phi_i}{\partial \eta} d\eta + \sum_{i}^{N_e} z_i \frac{\partial \phi_i}{\partial \zeta} d\zeta \\
&= (3.53)
\end{align*}
\]

in which all the partial derivative entries of the matrices can be calculated according to Eq. 3.50.

3.4.2 Wall Distance

For RANS-SA system, in addition to the domain integrals arising from the discretizations of the Navier-Stokes equations, there also exists another domain integral resulting from the discretization of the source term of the turbulent working variable equation. It could be expressed
by,
\[
\int_{\Omega} SB_i \, d\Omega = \int_{\Omega} \left( \frac{\sqrt{Re_{\infty}}}{Ma_{\infty}} c_{b1} \tilde{S} \tilde{\mu} \psi + \frac{\sqrt{Re_{\infty}}}{\sigma} \tilde{\mu} \nabla \cdot \nabla \tilde{\nu} \right) \nabla \cdot \nabla \tilde{\nu} \cdot \nabla \tilde{\nu} \, d\Omega - \frac{Ma_{\infty}}{\sqrt{Re_{\infty}}} c_{w1} \tilde{p} f_w \left( \frac{\bar{\nu}}{\sigma} \right)^2 - \frac{1}{\sigma} \tilde{\nu} (1 + \psi) \nabla \tilde{\rho} \cdot \nabla \tilde{\nu} \Big) B_i \, d\Omega \tag{3.54}
\]

Similar to the viscous domain integral, it is straightforward to achieve the above domain integral, whereas, we need to note \( d \) existing in the domain integral. As described in §2.2, the parameter \( d \) denotes the distance from a Gaussian quadrature of the element \( \Omega \) to the nearest wall.

Distance value \( d \) is a primary parameter in a range of key turbulence models. Furthermore, the SA (Spalart-Allmaras) model requires \( d \) accurate or a third of the boundary layer. Distance evaluation methods can be broadly classified as follows [122]: (I) search procedures, (II) integral approaches, and (III) differential equation-based methods. Crude search procedures require \( O(n_v n_s) \) operations, where \( n_s \) and \( n_v \) correspond to the number of wall surface and internal node points. For curved surface, it is obvious that, more operations are required, since that it is not accurate to use the linear plane to represent the curved surface. Integral methods are a little difficult to apply for the complex geometries. Hence, a lot of meaningful research has been done on differential equation-based methods. Thus, some very impressive results have been achieved. However, for standard Navier-Stokes solvers, its implementation doesn’t seem to be as natural. Also note that, due to curved surface, special care has to be taken to calculate the normal distance from a point in flow field to the surface. It is well known that, for a point \( P(x, y, z) \), the closest point \( P_0(\xi_0, \eta_0) \) on a surface \( R(x(\xi, \eta), y(\xi, \eta), z(\xi, \eta)) \) is the nearest root of
\[
\begin{align*}
\left\{ \frac{\nabla \tilde{R}}{\nabla \xi} \cdot \frac{\partial \tilde{R}}{\partial \xi} &= 0 \\
\frac{\nabla \tilde{R}}{\nabla \eta} \cdot \frac{\partial \tilde{R}}{\partial \eta} &= 0
\end{align*}
\tag{3.55}
\]

where \( \frac{\partial \tilde{R}}{\partial \xi} = (\frac{\partial \tilde{R}}{\partial \xi}, \frac{\partial \tilde{R}}{\partial \xi}, \frac{\partial \tilde{R}}{\partial \xi}) \) and \( \frac{\partial \tilde{R}}{\partial \eta} = (\frac{\partial \tilde{R}}{\partial \eta}, \frac{\partial \tilde{R}}{\partial \eta}, \frac{\partial \tilde{R}}{\partial \eta}) \). The Eq. 3.55 lead to the non-linear sets of equations. Therefore, Newton method could be used to solve the non-linear equations, whereas, it requires a good initial estimate. In our work, some different points at different position \((\xi, \eta)\) are used as the initial estimates to drive the Newton method rather than only one initial good estimate. Then with different initial estimate, maybe different roots could be obtained, which is determined by the nature of Newton method. As a result, the minimum distance could be selected as the distance from the integration point to the curved surface.

### 3.5 Boundary Conditions

All the boundary conditions are weakly imposed in this PhD work. DG methods usually employ the ghost state vector \( U_b^h(U_b^-, U_\infty) \) for the boundary integration point. Here, \( U_b^- \) represents
the state vector of interior cell and \( \mathbf{U}_\infty \) denotes the known free-stream vector. Obviously, for some special boundary condition involving the curved boundary surface, i.e., the Slip Wall / Symmetry Boundary, the unit vector normal to the curved boundary surface at the integration point has to be taken into account to calculate the boundary ghost state vector, \( \mathbf{U}_b^j(\mathbf{U}_h^-, \mathbf{U}_\infty) \), at the the boundary integration point. This will be discussed in detail later.

3.5.1 Characteristic Boundary

The actual flow condition at the boundary face is determined by the Mach number \( M^b \) derived from the mean state vector \( \mathbf{P}_0 \) in the interior cell. Four conditions, i.e., "supersonic inflow", "supersonic outflow", "subsonic inflow", and "subsonic outflow", are identified based on the value of \( M^b \), and then the \( \mathbf{U}_b^h \) vector and the approach of flux evaluation are specified accordingly.

**Supersonic Inflow** \( (M^b \leq -1) \)

\[
\mathbf{H}^b = \mathbf{F}(\mathbf{U}_\infty)
\]

where the first-order free-stream values \( \mathbf{U}_\infty \) are prescribed at the integration point of the boundary face for flux evaluation.

**Subsonic Inflow** \( (-1 < M^b < 0) \)

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

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

**Subsonic Outflow** \( (0 \leq M^b < 1) \)

\[
\mathbf{U}_h^b = \begin{pmatrix}
\rho^- \\
\rho u^- \\
\rho v^- \\
\rho w^- \\
\frac{p^\infty}{\gamma - 1} - \frac{1}{2} \rho^- |\mathbf{V}^-|^2 \\
\rho \bar{\nu}^-
\end{pmatrix}, \quad \mathbf{H}^b = \mathbf{F}(\mathbf{U}_h^b)
\]
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 based on the interior cell.

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

\[
H^b = F(U^-_h)
\]

where the interior state vector is extrapolated at the integration point of the boundary face directly.

### 3.5.2 Slip Wall / Symmetry Boundary

For the slip wall / symmetry boundary, the flow tangency requirement \( V \cdot n = 0 \) needs to be satisfied at the gaussian integration point. Instead of using the constant unit normal vector \( n^b \) for linear surface, the unit vector \( n^g \) normal to the curved surface at the gaussian integration point is required for high-order DG discretization to obtain real high-order solution \([68]\). The vector \( n^g \) is obtained through a quadratic representation of the solid wall boundary, i.e., 6-node curvilinear triangle and 9-node curvilinear quadrilateral as shown in Figure A.1 and Figure A.2. And \( n^g \) at each integration point is approximated by Eq. 3.46. The associated description of various types of higher-order elements can be found in Appendix A. Then, the ghost state vector at the integration point could be arrived at,

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

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

\[
U^b_h = \begin{pmatrix}
\rho^- \\
\rho^- u^b \\
\rho^- v^b \\
\rho^- w^b \\
\rho e^- \\
\rho \tilde{\nu}^-
\end{pmatrix}, \quad H^b = H(U^-_h, U^b_h, n^g)
\]

where one can see that \( n^g \) is also used in flux evaluation.
3.5.3 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

\[ u^b = v^b = w^b = 0 \quad \left( \frac{\partial T}{\partial n^b} \right) = (\nabla T)^b \cdot n^b = 0 \]

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

\[ U^b_h = \begin{pmatrix} \rho^- \\ 0 \\ 0 \\ 0 \\ \rho e^- \\ -\rho \nu^- \end{pmatrix} \]

The numerical flux at the no-slip adiabatic wall boundary is computed as

\[ H^b = H(U^-_h, U^b_h, n^g) \]

3.5.4 Periodic Boundary

In practical applications [59], the applicability of periodic boundary condition is restricted to flow configurations that are indeed periodic owing to their geometry, such as flow around turbine blades, or flows through channel, pipe, and duct with one or more statistically homogeneous flow directions. For time-dependent DNS or LES, a periodic boundary is often encountered, e.g., "Taylor-Green Vortex", and "turbulent channel flow". A periodic boundary may be regarded as an overlapping inflow and outflow boundary or an internal face boundary. Indeed, an internal face boundary doesn’t bring in any challenge, since that the flow field through the ghost cell of one periodic boundary outside the domain is known quantitically from the computations on the interior cell adjacent to the opposing periodic boundary inside the domain. Therefore, the specification of periodic boundaries should be quite straightforward. However, care should be taken to ensure the correct implementation of periodic boundary for parallel computation. In the parallel mode, the most important step is about how to identify the opposing periodic boundary face of the given periodic boundary face, since the periodic boundary face and its opposing one may not locate in the same partition. Then the detailed information is given as follows. One crucial clue is that the global numbering of the boundary face is fixed in spite that there are more than one processors.

In our work, for every processor, the array bface(nbfai, nbfac) is used to store the inforam-
tion of the boundary face, where \( nbfai \) denotes the dimension for each boundary face, and \( nbfac \) represents the number of all the boundary faces. In this array, \( bface(1,:) \) denotes the boundary type, \( bface(15,:) \) denotes the global numbering of the opposing one for one periodic boundary face, and \( bface(16,:) \) refers to the global numbering of this periodic boundary face.

First, \( bface(16,1:nbfac) \) could be gathered from every processor to the root processor stored in a new array \( mapfg(nbfmax) \) using the collective communication routine \( MPI_{\text{gatherv}} \), where \( nbfmax \) denotes the number of all the boundary faces from all the processors.

Then the array \( mapfg(nbfmax) \) will be sent to every processor from the root processor using the collective communication routine \( MPI_{\text{bcast}} \).

Finally, for the periodic boundary face in every processor, we need to identify the storage location of the opposing periodic boundary face (stored in \( bface(15,:) \)) in the array \( mapfg(1:nbfmax) \) by simple conditional expressions in programming. Once the opposing periodic boundary face is determined, the flow field information of the ghost cell could also be specified.

The numerical flux at the periodic boundary is computed using the same way as the internal face.
Table 3.2: Pseudo code: identifying the opposing one of one periodic boundary face

```
subroutine getperiodic(... , bface)
implicit none
...
real*8, dimension(nbfai, nbfac) : bface
real*8, dimension(nbfac) : bperi
! local arrays
real*8, dimension(nbfmax) : mapfg
real*8, dimension(nbfac) : mapfl
!
!Step 1: Using the local array to store bface(16, :)
! Gathering the mapfl from every processor to root processor
do ifa = 1, Nbfac
 mapfl(ifa)=bface(16,ifa)
enddo
! call MPI_gatherv(mapsf, ...,mapfg, ...)
!
!Step 2: Sending the mapfg from root processor to every processor
call MPI_bcast(mapfg, ...)  
!
!Step 3: Identifying the opposing periodic boundary condition
do ifa = 1, Nbfac
     if(face ifa is a periodic boundary face) then
     do ip=1,nbfmax
     ! Record the storage location of the opposing periodic boundary condition
       if(bface(15,ifa)==mapfg(ip)) bperi(ifa)=ip;exit
     enddo
     endif
enddo
!
! end of this subroutine
!
return
end
```
Chapter 4

Temporal Integration Methods

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. 3.15 can be written in an elemental semi-discrete form as

\[
M \frac{dU}{dt} = R(U)
\]  \hspace{1cm} (4.1)

where \( U = (U_1, U_2, \ldots, U_i, \ldots, U_{N_{elem}})^T \) is the global solution vector of \( N_{degr} \times N_{tot} \times N_{elem} \) degrees of freedom to be evolved in time. Each \( U_i \) itself is a vector, which represents solution of \( N_{degr} \times N_{tot} \) degrees of freedom for the \( i \)-th element. For example, the normalized linear (P_1) solution vector can be written as follows:

\[
U_i = \left( \rho, \frac{\partial \rho}{\partial x} \Delta x, \frac{\partial \rho}{\partial y} \Delta y, \frac{\partial \rho}{\partial z} \Delta z, \rho u, \frac{\partial \rho u}{\partial x} \Delta x, \frac{\partial \rho u}{\partial y} \Delta y, \frac{\partial \rho u}{\partial z} \Delta z, \rho v, \frac{\partial \rho v}{\partial x} \Delta x, \frac{\partial \rho v}{\partial y} \Delta y, \frac{\partial \rho v}{\partial z} \Delta z, \rho w, \frac{\partial \rho w}{\partial x} \Delta x, \frac{\partial \rho w}{\partial y} \Delta y, \frac{\partial \rho w}{\partial z} \Delta z, \rho e, \frac{\partial \rho e}{\partial x} \Delta x, \frac{\partial \rho e}{\partial y} \Delta y, \frac{\partial \rho e}{\partial z} \Delta z, \rho \tilde{\nu}, \frac{\partial \rho \tilde{\nu}}{\partial x} \Delta x, \frac{\partial \rho \tilde{\nu}}{\partial y} \Delta y, \frac{\partial \rho \tilde{\nu}}{\partial z} \Delta z \right)
\]  \hspace{1cm} (4.2)

Likewise, \( R = (R_1, R_2, \ldots, R_i, \ldots, R_{N_{elem}})^T \) represents the global residual vector, where \( R_i \) is the elemental residual vector for the \( i \)-th element, and approaches zero for a steady-state solution. Since the shape functions \( B_i \big|_{\Omega_i} \) are nonzero within the \( i \)-th element only, the global mass matrix \( M \) 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 \( \Omega_i \). The \( i \)-th elemental mass
matrix for the linear DG(P_1) method can be written as follows

\[
M_i = \begin{pmatrix}
\int_{\Omega_i} \tilde{B}_1 \tilde{B}_1 \, d\Omega & \int_{\Omega_i} \tilde{B}_1 \tilde{B}_2 \, d\Omega & \int_{\Omega_i} \tilde{B}_1 \tilde{B}_3 \, d\Omega & \int_{\Omega_i} \tilde{B}_1 \tilde{B}_4 \, d\Omega \\
\int_{\Omega_i} \tilde{B}_2 \tilde{B}_1 \, d\Omega & \int_{\Omega_i} \tilde{B}_2 \tilde{B}_2 \, d\Omega & \int_{\Omega_i} \tilde{B}_2 \tilde{B}_3 \, d\Omega & \int_{\Omega_i} \tilde{B}_2 \tilde{B}_4 \, d\Omega \\
\int_{\Omega_i} \tilde{B}_3 \tilde{B}_1 \, d\Omega & \int_{\Omega_i} \tilde{B}_3 \tilde{B}_2 \, d\Omega & \int_{\Omega_i} \tilde{B}_3 \tilde{B}_3 \, d\Omega & \int_{\Omega_i} \tilde{B}_3 \tilde{B}_4 \, d\Omega \\
\int_{\Omega_i} \tilde{B}_4 \tilde{B}_1 \, d\Omega & \int_{\Omega_i} \tilde{B}_4 \tilde{B}_2 \, d\Omega & \int_{\Omega_i} \tilde{B}_4 \tilde{B}_3 \, d\Omega & \int_{\Omega_i} \tilde{B}_4 \tilde{B}_4 \, d\Omega
\end{pmatrix}
\] (4.3)

Due to its symmetry and Eq. 3.22, only the volume of the element and other six components \( \int_{\Omega_i} \tilde{B}_i \tilde{B}_j \, d\Omega \ (j \geq i) \) need to be actually pre-computed and stored, as shown in the following equation

\[
M_i = \begin{pmatrix}
V_i & 0 & 0 & 0 \\
0 & \int_{\Omega_i} \tilde{B}_2 \tilde{B}_2 \, d\Omega & \int_{\Omega_i} \tilde{B}_2 \tilde{B}_3 \, d\Omega & \int_{\Omega_i} \tilde{B}_2 \tilde{B}_4 \, d\Omega \\
0 & \int_{\Omega_i} \tilde{B}_3 \tilde{B}_2 \, d\Omega & \int_{\Omega_i} \tilde{B}_3 \tilde{B}_3 \, d\Omega & \int_{\Omega_i} \tilde{B}_3 \tilde{B}_4 \, d\Omega \\
0 & \int_{\Omega_i} \tilde{B}_4 \tilde{B}_2 \, d\Omega & \int_{\Omega_i} \tilde{B}_4 \tilde{B}_3 \, d\Omega & \int_{\Omega_i} \tilde{B}_4 \tilde{B}_4 \, d\Omega
\end{pmatrix}
\] (4.4)

In our work, both the explicit and implicit time integration scheme could be adopted to march Eq. 4.1, while the emphasis is put on the latter. For steady state problems, BDF1, which could only achieve first-order accuracy in time, is used to drive Eq. 4.1 to converge to steady state. For unsteady problems, either explicit three-stage third-order TVD Runge-Kutta scheme or implicit four-stage third-order implicit Rosenbrock-Wanner method is leveraged to advance the solution in time. The outline of this chapter is organized in the following. The explicit multi-stage Runge-Kutta scheme is briefly introduced in §4.1. A general procedure of the implicit time integration scheme is described in section §4.2. Finally, the formulation of Jacobian matrix is given in §4.4.1, and the solver for the solution of linear systems is presented in §4.4.2.

### 4.1 Explicit Time Integration

The following explicit three-stage third-order TVD Runge-Kutta scheme [31, 32]

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

\[
U^{(2)} = \frac{3}{4} U^n + \frac{3}{4} [U^{(1)} + \Delta t M^{-1} R(U^{(1)})]
\] (4.5)

\[
U^{(n+1)} = \frac{1}{3} U^n + \frac{2}{3} [U^{(1)} + \Delta t M^{-1} R(U^{(2)})]
\]

is widely used to advance the solution in time. This method is linearly stable for a CFL number less than or equal to 1/(2p + 1).
4.2 Implicit Time Integration

For steady state problems, the spatially discretized governing equations Eq. 4.1 could be integrated implicitly in time using BDF1, then we can arrive at

\[ M \frac{(U^{n+1} - U^n)}{\Delta t} = R(U^{n+1}) \]  \hspace{1cm} (4.6)

Since the governing equations are nonlinear, Eq. 4.6 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 time level \( n \)

\[ R(U^{n+1}) \approx R(U^n) + \left( \frac{\partial R}{\partial U} \right)^n (U^{n+1} - U^n) \]  \hspace{1cm} (4.7)

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

If the right-hand-side term in Eq. 4.6 is replaced by Eq. 4.7, and then 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

\[ A \Delta U^n = \left( M \frac{1}{\Delta t} - \left( \frac{\partial R}{\partial U} \right)^n \right) \Delta U^n = R(U^n) \]  \hspace{1cm} (4.8)

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.

For unsteady problems, the use of explicit time-integration schemes has been widespread for DG discretizations. While such methods are well suited for problems with similar spatial and temporal scales, they are notoriously inefficient for problems with disparate temporal and spatial scales, such as low-reduced frequency problems. In this work we focus on the use of implicit high-order time-integration schemes, which are not restricted by the CFL-stability limit of explicit methods, and thus are capable of using maximum time-steps determined by accuracy considerations, and are thus more suitable for stiff problems. In Chapter 1, different implicit high-order time integration schemes have been investigated. And it has been concluded that, the linearly implicit high-order Rosenbrock-Wanner is one very promising candidate for the solution of unsteady problems with high-order rDG discretization. In our work, four-stage third-order implicit Rosenbrock-Wanner method has been employed to integrate Eq. 4.1 in time.

Rosenbrock-type schemes are part of a class of linearly implicit Runge-Kutta (IRK) time
integration methods. In general, the formula of an s-stage Rosenbrock-type scheme to integrate Eq. 4.1 in time can be written as [130]

\[(I - \Delta t \gamma_{ii} J)U^{(i)} = \Delta t R(U^n + \sum_{j=1}^{i-1} \alpha_{ij} U^{(j)}) + \Delta t \sum_{j=1}^{i-1} \gamma_{ij} U^{(j)}, \quad i = 1, \ldots, s.\] (4.9)

\[U^{n+1} = U^n + \sum_{j=1}^{s} b_j U^{(j)},\] (4.10)

where \(\alpha_{ij}, \gamma_{ij}, b_i\) are the determining coefficients and \(J = \frac{\partial R}{\partial U}(U^n)\). As shown in Eq. 4.9, the ROW methods are derived by linearizing a diagonally IRK (DIRK) scheme, and replace the non-linear systems with a sequence of linear systems, in which some attractive properties in terms of stability are lost. However, as a trade-off, the computational costs per time step are reduced in Eq. 4.9: s linear equation systems with a constant coefficient matrix and different right-hand-sides need to be solved, instead of s non-linear systems. They can be easily used along with the variable time step sizes and constructed to be A- and L-stable to obtain formal order of temporal accuracy [109, 107].

In Eq. 4.9, the Jacobian matrix \(J\) is approximately computed using an automatic differentiation toolkit TAPENADE [135], which will be discussed in detail later in §4.4.1. In many systems, the exact Jacobian matrix can be both costly and difficult to obtain, e.g., due to the size of the application and the use of complex spatial discretization schemes. The class of Rosenbrock-Wanner methods have been derived for such situations, which is aimed to not only reduce the computational costs within each time step, but also alleviate the impact from how the Jacobian matrix is formulated. They have the same form in Eq. 4.9, but the coefficients are selected such that the overall discretization order is preserved for approximate Jacobian matrix.

For implementation purposes, Eq. 4.9 and Eq. 4.10 can be rewritten by introducing a new variable vector \(W^{(i)}\):

\[W^{(i)} = \sum_{j=1}^{i} \gamma_{ij} U^{(j)}, \quad i = 1, \ldots, s,\] (4.11)

in order to avoid the matrix–vector multiplication \(J \sum \gamma_{ij} U^{(j)}\), which could result from the direct implementation of Eq. 4.9 and Eq. 4.10, as it requires the solution of a linear system with the matrix \((I - \Delta t \gamma_{ii} J)\) as well as the matrix–vector multiplication \(J \sum \gamma_{ij} U^{(j)}\). Note that if \(\gamma_{ij} \neq 0\) for \(j \leq i\), then the matrix \(\Gamma = (\gamma_{ij})\) is invertible and \(U^{(i)}\) can be determined from \(W^{(i)}\) with

\[U^{(i)} = \frac{1}{\gamma_{ii}} W^{(i)} - \sum_{j=1}^{i-1} c_{ij} W^{(j)},\] (4.12)

where \(C\) is given by \(C = \text{diag}(\gamma_{11}^{-1}, \ldots, \gamma_{ss}^{-1}) - \Gamma^{-1}\). Thus the following formulation of the
ROW method is found in practical implementations,

\[
LW^{(i)} = F(U^n + \sum_{j=1}^{i-1} a_{ij} W^{(j)}) + \frac{1}{\Delta t} \sum_{j=1}^{i-1} c_{ij} W^{(j)}, \quad i = 1, \ldots, s, \tag{4.13}
\]

with \( L = (\frac{1}{\gamma \Delta t} I - J) \), and \( \gamma = \gamma_{ii} \), thus \( L \) is constant for consecutive stages of the Rosenbrock scheme. The solution at the next time step \( U^{n+1} \) is given by

\[
U^{n+1} = U^n + \sum_{j=1}^{s} m_j W^{(j)}, \tag{4.14}
\]

where the coefficients \( (a_{ij}) = (\alpha_{ij}) \Gamma^{-1}, \) and \( (m_1, \ldots, m_s) = (b_1, \ldots, b_s) \Gamma^{-1}. \)

In this work, we mainly focus on the study of 3 third-order ROW methods: 1) ROSI2PW method (I2 for index 2 problems, P for semi-discretized PDE problems, W for W-method), 2) third-order ROS34PRW method (an extension of ROS34PW2 from index-1 DAEs to index-2 DAEs), 3) third-order ROS34PW2. All these 3 schemes are L-stable \([130]\), which is quite desirable property for solution of numerical stiff problems. In addition to the aforementioned 3 ROW methods, another Rosenbrock-type method ROSI2Pw has also been tested for solution of unsteady problems. However, we need to point out that, ROSI2Pw is a standard Rosenbrock method, therefore exact Jacobian matrix has to be used to guarantee the designed order of accuracy. And, also the comparison of different Rosenbrock-type methods between index-1 (ROS34PW2) and index-2 (ROSI2PW and ROS34PRW) type has been made in Chapter 6. The details for all these 3 methods can be found in \([109, 107]\), and the related coefficients also have been listed at Appendix B. Also, for every stage, an efficient LU-SGS preconditioned GMRES solver \([76, 89]\), namely GMRES+LU-SGS, is then applied for solving the linearized system of equations in Eq. 4.13.

### 4.3 p-multigrid solver

In order to reduce the computational cost with high-order discretization DG method, \( p \)-multigrid techniques could also be adopted to drive the steady problem to convergence.

Nowadays, geometric multigrid methods are very popular among CFD field to accelerate the convergence of Euler and Navier-Stokes equations to a steady state on unstructured grids. Analysis of geometric multigrid indicates mesh that independent results are possible, which leads to drastically reduction of the computational cost. In standard \( h \)-multigrid, spatially coarser grids are used to correct the error on the fine grid. With the coarse grids, low-frequency error modes from the fine grid appear as high-frequency modes, and are therefore effectively eliminated by the smoothers. \( p \)-Multigrid method is a natural extension of geometric multigrid
methods to high-order finite element formulation, such as spectral-\(hp\) or discontinuous Galerkin methods, where lower-order approximations serve as the coarse levels and systems of equations are solved by recursively iterating on solution approximations of different polynomial order [45].

The basic idea of a \(p\)-multigrid method is to perform time steps on the lower order approximation levels to obtain the corrections to a solution on a higher order approximation level [78]. A two level V-cycle \(p\)-multigrid method has been used to drive the iterations. More specifically, this two level \(p\)-multigrid method consists of the following steps at each \(p\)-multigrid cycle:

1. Perform a time-step at the high approximation order, that is \(rDG(P^1P^2)\), which yields the initial solution \(U^0_{P^1}\).

2. Transfer the flow solution and residual to the low approximation level, that is, \(DG(P^0)\).

This can be readily obtained using the shape function as

\[
U_{P^0}(\Omega_e) = \sum_{i=1}^{N} U^{n+1}_{P^1} B_i(x_c) \tag{4.15}
\]

\[
R_{P^0}(\Omega_e) = \sum_{i=1}^{N} R^{n+1}_{P^1P^2} B_i(x_c) \tag{4.16}
\]

where \(x_c\) is the coordinates of the center of element \(\Omega_e\).

3. Compute the force terms on the lower approximation level

\[
F_{P^0} = R_{P^0} - R(U_{P^0}) \tag{4.17}
\]

4. Perform a time-step at the lower approximation level where the residual is given by

\[
R = R(U_{P^0}) + F_{P^0} \tag{4.18}
\]

which yields the solution at the lower level \(U^{n+1}_{P^0}\).

5. Interpolate the correction \(C_{P^0}\) back from the lower level to update the higher level solution

\[
C_{P^0} = U^{n+1}_{P^0} - U_{P^0} \tag{4.19}
\]

\[
\tilde{U}^{n+1}_{P^1} = U^{n+1}_{P^1} + C_{P^0} \tag{4.20}
\]

In our work, for \(P^0\) level, the BR2 discretization is used to complete the discretization of viscous term with the penalty parameter defined by Fidkowski [45].
### 4.4 Solution of the Linear System of Equations

As shown in §4.2, both steady (BDF1) and unsteady problems (ROW), need to solve the resulting linear system of equations. Therefore, in this section, we will give a detailed description as to how to solve the linear system of equations. We will continue this section based on the linear system of equations described by Eq. 4.8. In order to improve the efficiency and robustness of GMRES, preconditioning technique is used to attempt to cluster the eigenvalues of matrix $A$ at a single value. Therefore, by left preconditioning, we could arrive at

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

where $P$ is the preconditioning matrix. Obviously, the optimal choice of $P$ is $A$, in which case the underlying matrix problem for GMRES is trivially solved with one Krylov vector. In our work, the LU-SGS is used as a preconditioner, i.e.,

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

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 to $U$, $L$ and $D$ from face integrals are computed as below

$$ U_\tau = \int_{\Gamma_{ij}} \frac{\partial H_{\text{inv}}}{\partial U_j} \tilde{B}^i \ d\Gamma - \int_{\Gamma_{ij}} \frac{\partial H_{\text{vis}}}{\partial U_j} \tilde{B}^i \ d\Gamma $$

$$ L_\tau = \int_{\Gamma_{ij}} \frac{\partial H_{\text{inv}}}{\partial U_i} \tilde{B}^j \ d\Gamma + \int_{\Gamma_{ij}} \frac{\partial H_{\text{vis}}}{\partial U_i} \tilde{B}^j \ d\Gamma $$

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

where, $H_{\text{inv}} = H_{\text{inv}}(U_{hi}(U_i), U_{hj}(U_j), n_{ij})$, and $H_{\text{vis}} = H_{\text{vis}}(U_{hi}, \nabla U_{hi} + \beta r_i, U_{hj}, \nabla U_{hj} + \beta r_j, n_{ij})$. And, herein, $U_{hi}$ denotes the values at the gauss integration point in cell $i$, and $U_i$ represents the solution vector for cell $i$. The contributions to $L$ and $D$ from domain integrals are

$$ L_\Omega = -\int_{\Omega_i} \frac{\partial F_k(U_{hi})}{\partial U_j} \frac{\partial \tilde{B}^i}{\partial x_k} \ d\Omega + \int_{\Omega_i} \frac{\partial G_k(U_{hi}, \nabla U_{hi} + R_i)}{\partial U_j} \frac{\partial \tilde{B}^i}{\partial x_k} \ d\Omega $$

$$ D_\Omega = -\int_{\Omega_i} \frac{\partial F_k(U_{hi})}{\partial U_i} \frac{\partial \tilde{B}^i}{\partial x_k} \ d\Omega + \int_{\Omega_i} \frac{\partial G_k(U_{hi}, \nabla U_{hi} + R_i)}{\partial U_i} \frac{\partial \tilde{B}^i}{\partial x_k} \ d\Omega $$

In Eq. 4.23, Eq. 4.24, Eq. 4.25, Eq. 4.26, and Eq. 4.27, $i$ and $j$ denote cell $i$ and its adjacent face-neighboring cell $j$, respectively, assuming that $j > i$. Finally, the time derivative term $\frac{M}{\Delta t}$
is added to the $D$, and the elemental block diagonal matrix is as below

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

(4.28)

in which the crucial part is the assembly of the Jacobian matrix $J_i$, and we will devote an effort to it in §4.4.1. The global block diagonal matrix requires a storage of $Nelem \times (Ndegr \times Netot)^2$ units. Both the upper and lower matrices require a storage of $Nafac \times (Ndegr \times Netot)^2$ units, where $Nafac$ is the number of faces. It is well known that two key factors, i.e., Jacobian matrix and linear solver, are crucial to successfully solve the linear systems. Therefore, next two subsections will focus on the Jacobian matrix and linear solver.

### 4.4.1 Jacobian matrix

To formulate the Jacobian matrix, the exact linearization of flux functions has to be considered. But as to the Hermite-WENO reconstruction, the linearization process is quite complicated, since that the Hermite-WENO reconstruction is highly nonlinear in nature itself, and practically not accessible in a compact form. Thus, approximate Jacobians are usually formulated instead of the exact ones for the implicit finite volume / discontinuous Galerkin methods involving the non-compact reconstruction.

In order to make a balance between the speedup and extra computational cost by implementing an implicit algorithm, the Jacobian matrix is evaluated using $J_{P_1}$ resulting from the linearization of the second-order RHS operator $R_{P_1}$ with respect to $U_{P_1}$, instead of the reconstructed third-order RHS operator $R_{P_1P_2}$. Eq. 4.8 can be rewritten in the form of approximate linear system as follows

$$\left( \frac{M}{\Delta t} - J_{P_1} \right) \Delta U^0 = R_{P_1P_2}$$

(4.29)

Due to the inexact representation of the LHS matrix used in Eq. 4.29, the quadratic convergence of the Newtons method can no longer be achieved. However, as compared with DG(P_2), the number of DoFs is greatly reduced for rDG(P_1P_2), which is a substantial saving in storage. For example, the implicit rDG(P_1P_2) method only requires 16% of the memory of DG(P_2) for storing the block diagonal part of the Jacobians ($24 \times 24$ versus $60 \times 60$ per element), while the scheme still remains to be third-order accurate in space.

Finally, let’s come to the linearization process. Obviously, it is not a trivial work to linearize the RHS with respect to the unknown vector $U$, e.g., upwind numerical flux functions, viscous domain integral involving the global lift operator, and so on. In general, four approaches are widely adopted in a variety of applications, i.e., differentiation by hand, symbolic differentiation, numerical differentiation and automatic differentiation. As to the above four methods, you could refer to the PhD thesis for the details by Xia [138].
As described in §3.1.1, the approximate Riemann solver HLLC is used to compute the inviscid flux, and BR2 scheme is adopted to discretize the viscous flux.

About the linearization of inviscid flux, both **differentiation by hand** and **automatic differentiation** have been implemented exactly. For the finite volume methods, Batten et.al. [18] proposed two version. One is the frozen acoustic wavespeed version of the implicit HLLC flux, while another is fully linearized implicit HLLC flux, which has taken the differentiation of the acoustic wavespeed into account. There have been also some successful implementation of the linearization of BR2 scheme in various studies in the DG community recently. In Bassi and Rebay’s 2005 publication [15], 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. In 2005, Hartmann [55] formulated a DG discretization of the compressible Navier-Stokes equations, based on employing the generalization of the symmetric version of the interior penalty method for the numerical approximation of the viscous terms. The Fréchet derivative was analytically derived with suitable approximation. In 2008, Landmann [70] 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 was also analytically derived. In 2010, Yasue et.al [145] 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. In 2015, Hartmann [112] presented a DG method designed to improve the accuracy and efficiency of steady solutions of the compressible fully coupled RANS and $k$ – $\omega$ turbulence model equations for solving all-speed flows, where BR2 was used to deal with viscous flux. The Jacobian matrix of the DG discretization was computed analytically without any approximation, except for the dissipative part of the numerical flux for which the Jacobian was computed numerically. All these works discussed above indicate that the key to a robust and efficient implicit DG Navier-Stokes solver is the quality of the linearization of viscous terms.

In this work, only the AD method is adopted to calculate the Jacobian matrix. As a matter of fact, the linearization of the inviscid HLLC scheme using AD method is the same as the fully linearized implicit HLLC flux by Batten et.al. [18]. However, unlike the inviscid part, it is not straightforward to linearize the viscous flux resulting from BR2 scheme exactly. The implementation of AD in our original code just treated the global lift operator in the viscous domain integral as a constant. It is obvious that this approximate strategy could greatly reduce the complexity of linearizing the viscous domain integral with respect to the solution vector $\mathbf{U}$. Although this approximation worked well for a lot of test cases, the exact linearization is still quite attractive since what it requires additionally is the array transfer. Therefore, the next section will focus on the exact linearization of the viscous domain integral.
To apply the implicit BR2 scheme, the implicit form of the viscous flux evaluated at the integration point of face $ij$ is defined by

$$
\mathbf{H}^{n+1}_{vis}(U_{hi}, \nabla U_{hi} + r_i, U_{hj}, \nabla U_{hj} + r_j)
= \mathbf{H}^{n}_{vis}(U_{hi}, \nabla U_{hi} + r_i, U_{hj}, \nabla U_{hj} + r_j)
+ \frac{\partial \mathbf{H}_{vis}}{\partial U_{hi}} \Delta U_i + \frac{\partial \mathbf{H}_{vis}}{\partial U_{hj}} \Delta U_j
+ \frac{\partial \mathbf{H}_{vis}}{\partial (\nabla U_{hi} + r_i)} \Delta U_i + \frac{\partial \mathbf{H}_{vis}}{\partial (\nabla U_{hj} + r_j)} \Delta U_j
$$

(4.30)

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

$$
\int_{\Gamma_i} r_i \tilde{B}_m^i \, d\Gamma = \int_{\Gamma_{ij}} \frac{1}{2} (U_{hj} - U_{hi}) n_k \tilde{B}_m^i \, d\Gamma \quad 1 \leq m \leq N
$$

(4.31)

where, $N$ represents the dimension of the polynomial space. The locality means that the integration is evaluated on face $ij$ only.

Herein, the details about how to linearize the viscous term are described. Let’s take the term, $\frac{\partial U_{hi}}{\partial U_i}$ as an example. $U_{hi}$ could be expressed as follows,

$$
U_{hi} = U_1 \tilde{B}_1^i + U_2 \tilde{B}_2^i + U_3 \tilde{B}_3^i + U_4 \tilde{B}_4^i
$$

(4.32)

where, $U_1$, $U_2$, $U_3$, and $U_4$ denote the independent variables, shown in Eq. 4.33.

$$
\begin{pmatrix}
\rho_1, \rho_2, \rho_3, \rho_4 \\
(\rho u)_1, (\rho u)_2, (\rho u)_3, (\rho u)_4 \\
(\rho v)_1, (\rho v)_2, (\rho v)_3, (\rho v)_4 \\
(\rho w)_1, (\rho w)_2, (\rho w)_3, (\rho w)_4 \\
(\rho e)_1, (\rho e)_2, (\rho e)_3, (\rho e)_4 \\
(\rho v)_1, (\rho v)_2, (\rho v)_3, (\rho v)_4
\end{pmatrix}
= \begin{pmatrix}
\rho, \frac{\partial \rho}{\partial x} \Delta x, \frac{\partial \rho}{\partial y} \Delta y, \frac{\partial \rho}{\partial z} \Delta z \\
(\rho u), \frac{\partial (\rho u)}{\partial x} \Delta x, \frac{\partial (\rho u)}{\partial y} \Delta y, \frac{\partial (\rho u)}{\partial z} \Delta z \\
(\rho v), \frac{\partial (\rho v)}{\partial x} \Delta x, \frac{\partial (\rho v)}{\partial y} \Delta y, \frac{\partial (\rho v)}{\partial z} \Delta z \\
(\rho w), \frac{\partial (\rho w)}{\partial x} \Delta x, \frac{\partial (\rho w)}{\partial y} \Delta y, \frac{\partial (\rho w)}{\partial z} \Delta z \\
(\rho e), \frac{\partial (\rho e)}{\partial x} \Delta x, \frac{\partial (\rho e)}{\partial y} \Delta y, \frac{\partial (\rho e)}{\partial z} \Delta z \\
(\rho v), \frac{\partial (\rho v)}{\partial x} \Delta x, \frac{\partial (\rho v)}{\partial y} \Delta y, \frac{\partial (\rho v)}{\partial z} \Delta z
\end{pmatrix}
$$

(4.33)

We need to point out all the vraibles are independent. Assuming $U_i(1 : 24)$ and $U_j(1 : 24)$ represent the 24 independent variables for cell $i$ and $j$ seperately, therefore, we have the following,

$$
\frac{\partial U_i(k)}{\partial U_i(m)} = \begin{cases} 
1 & \text{if } k = m \\
0 & \text{if } k \neq m
\end{cases} \quad \frac{\partial U_j(k)}{\partial U_j(m)} = \begin{cases} 
0 & \text{if } k = m \\
0 & \text{if } k \neq m
\end{cases}
$$

(4.34)
Therefore, for $\rho$

$$\frac{\partial \rho}{\partial U_i} = \begin{pmatrix} \frac{\partial \rho}{\partial p_1}, \frac{\partial \rho}{\partial p_2}, \frac{\partial \rho}{\partial p_3}, \frac{\partial \rho}{\partial p_4}, \\ \frac{\partial \rho}{\partial p_{w_1}}, \frac{\partial \rho}{\partial p_{w_2}}, \frac{\partial \rho}{\partial p_{w_3}}, \frac{\partial \rho}{\partial p_{w_4}}, \\ \frac{\partial \rho}{\partial \rho_{u_1}}, \frac{\partial \rho}{\partial \rho_{u_2}}, \frac{\partial \rho}{\partial \rho_{u_3}}, \frac{\partial \rho}{\partial \rho_{u_4}}, \\ \frac{\partial \rho}{\partial \rho_{v_1}}, \frac{\partial \rho}{\partial \rho_{v_2}}, \frac{\partial \rho}{\partial \rho_{v_3}}, \frac{\partial \rho}{\partial \rho_{v_4}}, \\ \frac{\partial \rho}{\partial \rho_{e_1}}, \frac{\partial \rho}{\partial \rho_{e_2}}, \frac{\partial \rho}{\partial \rho_{e_3}}, \frac{\partial \rho}{\partial \rho_{e_4}}, \\ \frac{\partial \rho}{\partial \tilde{\nu}_1}, \frac{\partial \rho}{\partial \tilde{\nu}_2}, \frac{\partial \rho}{\partial \tilde{\nu}_3}, \frac{\partial \rho}{\partial \tilde{\nu}_4} \end{pmatrix} \quad (4.35)$$

Combining with Eq. 4.34, we have the following,

$$\frac{\partial \rho}{\partial U_i} = \begin{pmatrix} 0, 0, 0, 0, \\ 0, 0, 0, 0, \\ 0, 0, 0, 0, \\ 0, 0, 0, 0, \\ 0, 0, 0, 0 \end{pmatrix} \quad (4.36)$$

Combined with Eq. 4.32, Eq. 4.35 could be simplified as follows,

$$\frac{\partial \rho}{\partial U_i} = \begin{pmatrix} \tilde{B}_1^i, \tilde{B}_2^i, \tilde{B}_3^i, \tilde{B}_4^i, \\ 0, 0, 0, 0, \\ 0, 0, 0, 0, \\ 0, 0, 0, 0, \\ 0, 0, 0, 0 \end{pmatrix} \quad (4.37)$$

Similarly, the following could be obtained,

$$\frac{\partial \rho}{\partial U_j} = \begin{pmatrix} 0, 0, 0, 0, \\ 0, 0, 0, 0, \\ 0, 0, 0, 0, \\ 0, 0, 0, 0, \\ 0, 0, 0, 0 \end{pmatrix} \quad (4.38)$$

Eq. 4.35, Eq. 4.36, Eq. 4.37, and Eq. 4.38 provide crucial foundation to get the linearization of the right hand side. In order to make it straightforward to do the linearization, the domain integral $G_{BR2}$ evaluated at the integration point of cell $i$ could be expressed using a math
function \( f_{vd} \) explicitly,

\[
\mathbf{G}_{BR2}(U_h, \nabla U_{hi} + R_i) = f_{vd}(U_{hi}, \nabla U_{hi} + R_i, \mu, \mu_t, C) \quad (4.39)
\]

where subscript \( vd \) denotes the viscous domain integral. In the Eq. 4.39, \( \mu \) is the molecular viscosity, \( \mu_t \) is turbulent eddy viscosity, and \( C \) represents the constants required for the viscous domain integral, e.g., \( Pr \), \( Pr_t \), and \( \gamma \). Then the linearization with respect to \( U_i \) could be done using the function \( f_{vd} \) as follows,

\[
f_{vd}^{n+1} = f_{vd}^n + \frac{\partial f_{vd}}{\partial U_i} \Delta U_i + \frac{\partial f_{vd}}{\partial U_j} \Delta U_j \quad (4.40)
\]

\[
\frac{\partial f_{vd}}{\partial U_i} = \frac{\partial f_{vd}}{\partial U_{hi}} \frac{\partial U_{hi}}{\partial U_i} \Delta U_i + \frac{\partial f_{vd}}{\partial (\nabla U_{hi} + R_i)} \frac{\partial \nabla U_{hi}}{\partial U_i} \Delta U_i \\
+ \frac{\partial f_{vd}}{\partial (\nabla U_{hi} + R_i)} \frac{\partial R_i}{\partial U_i} \Delta U_i + \frac{\partial f_{vd}}{\partial \mu_i} \frac{\partial \mu_i}{\partial U_i} \Delta U_i + \frac{\partial f_{vd}}{\partial \mu_{ti}} \frac{\partial \mu_{ti}}{\partial U_i} \Delta U_i
\]

\[
\frac{\partial f_{vd}}{\partial U_j} = \frac{\partial f_{vd}}{\partial (\nabla U_{hi} + R_i)} \frac{\partial R_i}{\partial U_j} \Delta U_j \quad (4.41)
\]

where \( R_i \) is the so-called global lifting operator at gauss integration point for element \( i \), and defined by

\[
\int_{\Omega_i} R_i \tilde{B}_m^i \ d\Omega = \int_{\partial \Omega_i} \frac{1}{2} (U_{hj} - U_{hi}) n_k \tilde{B}_m^i \ d\Gamma \quad (4.43)
\]

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

\[
R = \sum_{\Gamma_{ij}} \mathbf{r} \quad (4.44)
\]

Previously, when AD is used to linearize the viscous domain integral, the global lift operators are treated as constant [141, 140]. In this work, however, in order to do the exact linearization, the global lift operators are also linearized with respect to the \( U \).

\[
\int_{\Omega_i} R_i \tilde{B}_m^i \ d\Omega = \int_{\partial \Omega_i} \frac{1}{2} (U_{hj} - U_{hi}) n_k \tilde{B}_m^i \ d\Gamma \\
= \int_{\partial \Gamma_{ij}} \frac{1}{2} (U_{hj} - U_{hi}) n_k \tilde{B}_m^i \ d\Gamma + \int_{\partial \Gamma_{ik}} \frac{1}{2} (U_{hi} - U_{hk}) n_k \tilde{B}_m^i \ d\Gamma + \int_{\partial \Gamma_{im}} \frac{1}{2} (U_{hi} - U_{hn}) n_k \tilde{B}_m^i \ d\Gamma \quad (4.45)
\]
\[
\int_{\Omega} \frac{\partial R_i}{\partial U_i} \tilde{B}_m \, d\Omega = \int_{\partial \Gamma} \frac{1}{2} \left( -\frac{\partial U_{hi}}{\partial U_i} \right) n_k \tilde{B}_m \, d\Gamma \\
+ \int_{\partial \Gamma} \frac{1}{2} \left( -\frac{\partial U_{hi}}{\partial U_i} \right) n_k \tilde{B}_m \, d\Gamma \\
+ \int_{\partial \Gamma} \frac{1}{2} \left( -\frac{\partial U_{hi}}{\partial U_i} \right) n_k \tilde{B}_m \, d\Gamma 
\] (4.46)

\[
\int_{\Omega} \frac{\partial m}{\partial U_j} \tilde{B}_m \, d\Omega = \int_{\partial \Gamma} \frac{1}{2} \left( -\frac{\partial U_{bj}}{\partial U_j} \right) n_k \tilde{B}_m \, d\Gamma 
\] (4.47)

In Eq. 4.46 and Eq. 4.47, the related terms \( \frac{\partial U_{hi}}{\partial U_i} \) and \( \frac{\partial U_{bj}}{\partial U_j} \) could be stored in advance, when we linearize the viscous face integral involving the local lifting operator. In our work, the arrays \( \text{upperglobd} \) and \( \text{lowerglobd} \) are used to store the above terms, where, the definition of arrays \( \text{upperglobd} \) and \( \text{lowerglobd} \) is similar to that of the upper and lower matrix, i.e., \( U \) and \( L \). The only difference lies in that, the new arrays result from the linearization of the global lift operator instead of the RHS. Then the contribution from viscous domain integration to \( U \), \( L \) and \( D \) could be achieved.

Next, let’s go into the details of 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. 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. 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 [23]. An incomplete list of AD tools can be found at Community Portal for Automatic Differentiation [3].

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 transformation change the semantics by explicitly rewriting the code. Each of these approaches has its advantages and disadvantages [22]. 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 [5]. 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. In our work, the web server is used to obtain the source code for AD. For TAPENADE, there exists 3 differentiation modes, i.e., tangent mode, tangent multidirectional mode, and reverse mode. The "tangent mode" will build a program that, given some small variations of the independent
variables, computes the resulting variations of the dependent variables. The "tangent multidirectional mode" also computes the variations of the output variables, but simultaneously for several directions in the input space [119].

In order to explain how these two modes work clearly, one simple example has been given here. First, a FORTRAN77 file test.f which contains the routine func as shown in Table 4.1 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. For "tangent mode", TAPENADE will generate a file test_d.f which contains the differentiation routine func_d, which is given in Table 4.2. For Tangent Multidirectional Mode, TAPENADE will generate a file test_dv.f which contains the differentiation routine func_dv, which is given in Table 4.3. There are two more inputs p_xd(nbdirsmx, 2) and nbdirs, and one more output p_yd(nbdirsmx, 2) for the new routine. The nbdirsmx is a user-defined integer in DIFFSIZES.inc as a header file. It is the number of directions that the user want to do derivative with respect to. 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 whole Jacobian matrix could be computed. In this case,

\[ y = f(x) \] (4.48)

where, \( y^T = (y_1, y_2) \), and \( x^T = (x_1, x_2) \). Then we could arrive at the following,

\[ f' = \frac{\partial y}{\partial x} = \begin{pmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} \end{pmatrix} \] (4.49)

\[ \frac{\partial y}{\partial x} = \begin{pmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} \end{pmatrix} = \begin{pmatrix} yd(1,1) & yd(2,1) \\ yd(1,2) & yd(2,2) \end{pmatrix} \] (4.50)

From the Table 4.3, the matrix yd could could be obtained by

\[ yd = \begin{pmatrix} yd(1,1) & yd(1,2) \\ yd(2,1) & yd(2,2) \end{pmatrix} \] (4.51)

Obviously, \( \frac{\partial y}{\partial x} = yd^T \). This means that by Tangent Multidirectional Mode, we could get the transpose of the Jacobian matrix.

### 4.4.2 Solver

In our code, 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 [76, 89], namely GMRES+LU-SGS.
Table 4.1: A simple source program

```fortran
!... This routine demonstrates a simple algebraic function
SUBROUTINE FUNC(x, y)
IMPLICIT NONE
REAL*8 x(2), y(2)
y(1) = 2.0*x(1) + 3.0*x(2)
y(2) = 4.0*x(1)*x(2)
RETURN
END
```

Table 4.2: Source code generated by AD using tangent mode

```fortran
!... Differentiation of func in forward (tangent) mode: (tangent mode)
!... variations of useful results: y
!... with respect to varying inputs: x
!... RW status of diff variables: x:in y:out
SUBROUTINE FUNC_DV(x, xd, y, yd)
IMPLICIT NONE
REAL*8 x(2), y(2)
REAL*8 xd(2), yd(2)
INTEGER ii
DO ii = 1, 2
    yd(ii) = 0.0
ENDDO
yd(1) = 2.0*xd(1) + 3.0*xd(2)
y(1) = 2.0*x(1) + 3.0*x(2)
yd(2) = 4.0*(xd(1)*x(2)+x(1)*xd(2))
y(2) = 4.0*x(1)*x(2)
RETURN
END
```

Since GMRES is among the most popular and efficient implicit algorithms in CFD [31, 14, 110, 57, 45, 80, 82, 78, 81, 15, 76, 89, 134, 13, 35], we always resort to using the GMRES+LU-SGS scheme to solve the linear system of equations arising from the linearization of the system of nonlinear equations the numerical test cases if implicit time advancement is required.

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 4.5 to solve the linear system of equations within each timestep. The GMRES algorithm can be partly matrix-free, for a finite difference approach can be used to compute the matrix-vector products instead of calculating and storing the full matrix as shown below:

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

(4.52)
Table 4.3: Source code generated by AD using multi-directional mode

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

where the residual vector $\mathbf{R}(\mathbf{U} + \varepsilon \cdot \Delta \mathbf{U})$ of the governing equations is computed with the state variable vector $\mathbf{U}$ added by a set of perturbed state quantities. The detailed information about how to implement such technique could refer to [138, 141].
Table 4.4: Pseudo code: contribution of domain integrals for the block diagonal matrix $D$, lower matrix $L$ and upper matrix $U$

```fortran
subroutine adlhsdomnvp1(..., unkno, diago, lower, upper, lowerglobd, upperglobd)
  implicit none
  ...
  real*8, dimension(Mdegr, Netot, Nsize) : unkno
  real*8, dimension(Ndofe, Ndofe, Nelem) : diago
  real*8, dimension(Ndofe, Ndofe, Nafac) : lower, upper
  real*8, dimension(Ndofe, Ndimn, Nq+1, Nafac) : lowerglobd, upperglobd
  ! 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
    nbdirs = Ndofe
    ! 1. Get the contribution to lower from domain integral
    ! differentiation of the domain integral
    call rhsdp1glob_dv(..., upperglob(:, :, :, :, ifa), p_unl, p_unld, p_rzl, p_rzld, nbdirs)
    upper(:, :, ifa) = upper(:, :, ifa) - transpose(p_rzld(:, :))
    !
    ! 2. Get the contribution to upper from domain integral
    ! differentiation of the domain integral
    call rhsdp1glob_dv(..., lowerglob(:, :, :, :, ifa), p_unr, p_unrd, p_rzr, p_rzrd, nbdirs)
    ! transpose the Jacobian matrix, scatter to lower
    lower(:, :, ifa) = lower(:, :, ifa) - transpose(p_rzrd(:, :))
    ! end of loop over the interior faces
  enddo
  ! return
end
```

69
Table 4.5: Flowchart of the GMRES algorithm

\begin{verbatim}
DO \( l = 1, m \)
    \( v_0 = R - A \Delta U_0 \) \( m \) restarted iterations
    \( r_0 := P^{-1}v_0 \) initial residual
    \( \beta := ||r_0||_2 \) preconditioning step
    \( v_1 := r_0/\beta \) initial residual norm
    DO \( j = 1, k \)
        \( y_j := \frac{\partial R(U)}{\partial U} \Delta U \) define initial Krylov
        \( w_j = P^{-1}y_j \) inner iteration
        DO \( i = 1, j \)
            \( h_{i,j} = (w_j, v_i) \) matrix-vector product
            \( w_j = w_j - h_{i,j}v_i \) preconditioning step
        END DO
        \( h_{j+1,j} := ||w_j||_2 \) Gram-Schmidt step
        \( v_{j+1} := w_j/h_{j+1,j} \) define Krylov vector
    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
\end{verbatim}
Chapter 5

Parallel implementation

The compactness of the DG and rDG methods makes them ideally suited for parallel implementation on regular clusters and supercomputers. The RDGFLO framework utilizes the Message Passing Interface (MPI) library for parallel communication between different threads. And METIS is used to partition the computational meshes. The parallel strategy is designed for architectures with distributed memory. The outline of this chapter is organized in the following. The grid partition is briefly introduced in section §5.1. The data structures and parallel communication are then described in section §5.2. Finally, the parallel performance analysis of our 3D RDGFLO has also been presented in section §5.3.

5.1 Domain Partition

Domain decomposition techniques seem a natural way to make good use of parallel computers. This technique divides a computation task into two parts, i.e., local part and inter-communication part. Thus, it is necessary to guarantee that each partition has roughly the same number of elements, and that "connections" between different partitions be limited, as much as possible, to reduce the workload of interprocess communications.

The RDGFLO code uses the open-source software package METIS [61], developed in the Kryrypis Lab at the University of Minnesota, for domain decomposition in the pre-processing stage. The algorithms implemented in METIS are based on the multilevel recursive-bisection, multilevel k-way, and multi-constraint partition schemes [62]. The k-way partitioning problem is most frequently solved by recursive bisection [63]. That is, we first obtain a 2-way partitioning of domain $V$, and then we recursively obtain a 2-way partitioning of each resulting partition. However, computing k-way refinement for $k > 8$ is prohibitively expensive. Multi-constraint partition schemes are suitable for many large-scale scientific simulations. For example [62], in multi-phase mesh-based computations, single constraint is not sufficient to effectively balance
the overall computations. Multi-phase computations involve \(m\) distinct phases, each separated by an explicit synchronization step. Generally speaking, the amount of computations performed in each element of the mesh for one phase is different from that for another phase. In order to effectively balance parallel computations associated with multi-phase, the global domain must be partitioned such that the computational workload in each phase is balanced, and the amount of interaction among different processors in each phase is minimized. This thesis mainly work on the single-phase aerodynamic calculation. Therefore, multilevel k-way partition schemes have been chosen to finish our domain decomposition. Multilevel k-way focuses on computing a k-way partition of a graph such that the edge-cut is minimized and each partition has an equal number of elements. The objective is to minimize the edge-cut and the requirement is that the partitions should be of the same size. Obviously, this is nothing but single-objective single-constraint graph partitioning problem, which has widely been used for static distribution of the mesh in parallel scientific calculation.

After the global domain is decomposed into user-specified subdomains. Then we could start the solver stage by reading at least one subdomain by one processor, instead of the global domain. Obviously, the workload of one processor could be reduced greatly, since that the workload of one processor is directly determined by the number of elements handled by this processor.

One simple example is shown in Figure 5.1 to introduce some basic concepts, e.g., "interior cell", "partition boundary", "cell along the partition boundary", "ghost cell along the partition boundary". Here, subdomain 1 and 2 share the same face, which is called partition boundary, marked in pink color. For subdomain, the interior cell along the partition boundary is called cell along the partition boundary, and the cell outside the subdomain along the partition is called the ghost cell along the partition boundary.

### 5.2 Interprocessor Communication

Obviously, for subdomain 1, the relevant information of the ghost cell along the partition boundary is necessary to compute the flux through the partition boundary face, e.g., geometry parameters, solution vectors. Thus, the exchange between different processors is necessary to ensure the correct advancement of the solution. As to the geometry parameters, if the mesh is fixed through computation, the exchange could be achieved in advance before the computation is executed. And for the solution vectors, the exchange should be done once the solution vectors are updated. This will be given detailed description in the following part.

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
Figure 5.1: The schematic diagram of domain partition

(a) The global computational domain

(b) The domain partition
explicit and implicit RDG methods.

First, the communication for explicit methods is quite straightforward. The basic procedures are listed in Table 5.1.

Table 5.1: Pseudo code: MPI communication for explicit method

```
subroutine exrkdgp1(..., unkno)

implicit none

real*8, dimension(Mdegr, Netot, Nsize) :: unkno

! loop over the stages
do istag = 1, Nstag

! Second derivatives reconstruction
call reconstruction_lsp1p2(..., unkno)

! exchange the reconstructed values between different processors
call exchange(..., unkno)

! initialize the local RHS vectors
rhsel(:, :, :) := 0.0 ;

! Complete the contribution to the rhsel from viscous terms
call getrhsvis(..., unkno, rhsel)

! HWENO reconstruction for unkno
call reconstruction_hwenop1p2(..., unkno)

! exchange the reconstructed values between different processors
call exchange(..., unkno)

! Complete the contribution to the rhsel from inviscid terms
call getrhsinv(..., unkno, rhsel)

! Complete the contribution to the rhsel from inviscid terms
call getrhsvis(..., unkno, rhsel)

! Advance the solution
do ie = 1, Nelem
unkno = TVDRK3(..., rhsel, unold)
enddo

! exchange the updated values between different processors
call exchange(..., unkno)
enddo
!
return
end
```

For the implicit rDG methods, since only the LHS of underlying DG(P₁) is adopted for rDG methods, thus the communication of the LHS resulting from the reconstruction stage is not necessary. Two kinds of solvers are available for solving the linear system of equations. One is SGS solver, of which the parallelization is similar to explicit method and is very simple.
to implement. As to the communication between different processors, the solution vectors of the elements along the partition boundary need only to be updated at the end of each time step. Another one is the GMRES+LU-SGS solver, of which the parallization is more complex than the previous one. The parallization of LU-SGS is not straightforward due to inherent data dependancies. Some strategies have been proposed about how to parallize the LU-SGS. The LU-SGS scheme is implemented directly on parallel non-overlapped blocks without any interprocessors’s communications. This means that LU-SGS is used locally on each processor without any contribution from interprocessor boundaries. However, researches have shown that this method only works well for moderate number of processors, compared with the Hybrid LU-SGS. This hybrid scheme uses the LU-SGS preconditioner on the interior elements but uses the Data-Parallel Lower-Upper Relaxation (DP-LUR) method for elements located along the partition boundaries.

Based on Eq. 4.8, when the left-hand side Jacobian matrix $A$ is stored in lower, upper, and diagonal forms, the resulting delta form of the equations could be recast as follows,

$$(L + U + D)\Delta U = R$$  \hspace{1cm} (5.1)

In order to introduce the Hybrid LU-SGS scheme[115], the SGS scheme is given first. First, we need to zero out $\Delta U$,

$$\Delta U^0 = 0$$  \hspace{1cm} (5.2)

Second, the $k_{max}$ are advanced using the forward sweep:

$$(D + L)\Delta U^{k+1/2} = R - U\Delta U^k$$  \hspace{1cm} (5.3)

and then a backward sweep:

$$(D + U)\Delta U^{k+1} = R - L\Delta U^{k+1/2}$$  \hspace{1cm} (5.4)

For one subiteration ($k_{max} = 1$), the SGS method is equivalent to the LU-SGS approximate factorization method.

Then let’s come to the scheme DP-LUR. This method has no inherent data dependencies, so it can be easily parallelized in the same way as an explicit scheme. The method can be described in the following way. The first subiteration is given as follows:

$$\Delta U^0 = D^{-1}R$$  \hspace{1cm} (5.5)
Second, the $k_{\text{max}}$ subiterations are made using:

$$
\Delta U^{k+1} = D^{-1}(R - (U + L)\Delta U^k)
$$

(5.6)

And the research has shown that this hybrid approach benefits for a large of blocks [115]. Note that, either the SGS iteration or the LU-SGS preconditioning 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.

### 5.3 Parallel Performance

This section is used to assess the scalability of the parallelization strategy implemented in the RDGFLO code. Ideally, a computational job that is split up among $N$ processors could be finished in $1/N$ time. However, usually, serial work by a single processor is necessary for parallelization strategy, e.g., finding the minimum value through the whole domain. This part does not run much faster on a parallel collection of processors (and might even run more slowly). The "speedup" of a parallel code is defined to be the ratio of the rate at which work is done when a job is run on $N$ processors to the rate at which it is done by just one processor. Based on Amdahl’s law [1], which only applies to cases where the problem size is fixed, the speedup is given as follows,

$$
S = \frac{t_1}{t_N},
$$

(5.7)

where, $t_N$ denotes the time required to finish the task on $N$ processors. Further, the parallel efficiency is defined as follows.

$$
\eta = \frac{S}{N}.
$$

(5.8)

Ideally, the speed up should be equal to $N$, and parallel efficiency $\eta$ should be always equal to 1. The finest grid of the case "turbulent flow over 2d bump" is used to complete the assessment of the parallel efficiency. As mentioned in, there are 14,960 hexahedral elements. The computation is conducted on NCSU’s ARC Linux cluster [2] 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).

The tests are only carried out for the implicit time integration. Table 5.2 display the parallel speedup and efficiency obtained on multiple CPUs. It could be observed that, with CPU No. increasing, the increasing velocity of speed up decreases, and also the parallel efficiency decreases. This is reasonable since that more CPUs means more inter communication between different processors. This maybe leads to that the computational cost for inter communication
Table 5.2: The measurement of speed up and parallel efficiency using rDG(P₁P₂)

<table>
<thead>
<tr>
<th>CPU No.</th>
<th>speed up $S$</th>
<th>parallel efficiency $\eta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>2</td>
<td>1.99</td>
<td>0.99</td>
</tr>
<tr>
<td>4</td>
<td>3.83</td>
<td>0.96</td>
</tr>
<tr>
<td>8</td>
<td>7.44</td>
<td>0.93</td>
</tr>
<tr>
<td>16</td>
<td>14.55</td>
<td>0.91</td>
</tr>
<tr>
<td>32</td>
<td>28.52</td>
<td>0.89</td>
</tr>
<tr>
<td>64</td>
<td>56.06</td>
<td>0.88</td>
</tr>
<tr>
<td>128</td>
<td>105.67</td>
<td>0.83</td>
</tr>
</tbody>
</table>

will take a quite large proportion of the whole computational cost. This definitely will bring the loss of the parallel efficiency.
Chapter 6

Numerical Results

The content of this chapter is split into numerical experiments of steady flow (§6.1) and unsteady flow (§6.2) problems. For all the test cases, 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.

We need to point out that, in essence, all the turbulent flows are unsteady and three-dimensional flow. However, some effective methods have been resorted to for solving the turbulent flows, e.g., RANS, ILES, and DNS. If the unsteady part is only associated to the turbulence, the mean flow could be viewed to be steady. Therefore, Reynold-averaging (time-averaging) could be used to deal with the Navier-Stokes equations to yield the steady RANS equations. Through our numerical experiments, for steady problems, BDF1 is selected to drive the resulting ODEs from the discretization to the steady state.

 Whereas, for ILES and DNS, the unsteady Navier-Stokes equations have to be solved. In our code, third-order ROW method is used to advance time integration for ILES and DNS.

6.1 Steady Cases

A number of numerical test cases with varying complexities have been presented in this section to validate and assess the performance of the modified SA turbulence model implemented in the rDG code. The test cases considered in this study are chosen from the NASA Langley Turbulence Modeling Resource Database [6], which provides multiple 2D and 3D test problems used for the validation and verification of turbulence models, as well as the reference results by several state-of-the-art CFD codes (e.g., FUN3D, and CFL3D). The grids provided by the database are suitable for use in many finite volume or finite difference solvers. Flow conditions and geometry are chosen to approximate those employed by Rumsey [67]. The quadratic meshes§3.4 used in
our simulation were generated by Gmsh or by simple agglomeration of the linear meshes from the database without any change of the high-order nodes. A parallel implicit solver with the LU-SGS preconditioned GMRES algorithm is used for time marching, and the computation is continued until a steady-state solution is reached, where the residual of the mass equation drops 7 orders of magnitude if possible. The initial solution for third-order rDG(P₁P₂) in each test case was taken from a fully converged solution of DG(P₁). In addition the freestream turbulent viscosity has been set as $\tilde{\nu} = 3\nu_\infty$ for all the computations in this section.

### 6.1.1 Zero Pressure Gradient Flat Plate

The first test case to validate our implementation of the turbulence model, is a turbulent flow over a flat plate. Flow conditions and geometry are chosen to approximate those employed by Rumsey [67]. For this turbulent flat plate case, $Ma_\infty = 0.2$, and $Re_\infty = 5 \times 10^6$ based on unit length of the grid. For the flat plate, the linear mesh is accurate enough to represent the geometry. Therefore, the linear meshes from the NASA database are directly used for this case with high-order $rDG$ method. The detailed information of such meshes, e.g., dimensions, is listed in Table 6.1. This is a 3D simulation of a 2D problem. Figure 6.1 shows the hexahedral grid, i.e., level 3, used for computation. This grid consists of 136 cells in the $x$-direction, 1 cell in $y$-direction and 96 cells in $z$-direction, which has been specified in Table 6.1. Figure 6.2 shows the contours of nondimensionalized turbulent eddy viscosity, i.e., $\mu_t/\mu_\infty$ using rDG(P₁P₂) on Level 3 hexahedral grid, which is almost indistinguishable from the contours distribution from NASA database. In addition to DG(P₁) and rDG(P₁P₂), the numerical results using WENO(P₁P₂) are also presented. It should be noted that, rDG(P₁P₂) without WENO reconstruction on P₂ works well for this case. The reason why the numerical results using WENO(P₁P₂) is presented is that we just want to verify the accuracy of WENO(P₁P₂) compared with rDG(P₁P₂). Figure 6.3 presents the logarithmic plot of friction coefficient distribution along the flat plate with

<table>
<thead>
<tr>
<th>Grid No.</th>
<th>Dimensions</th>
<th>Off-wall spacing</th>
<th>Leading-edge spacing</th>
<th>$y^+$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level 1</td>
<td>$2 \times 35 \times 25$</td>
<td>8.32e-6</td>
<td>1.62e-2</td>
<td>1.70</td>
</tr>
<tr>
<td>Level 2</td>
<td>$2 \times 69 \times 49$</td>
<td>4.04e-6</td>
<td>8.05e-3</td>
<td>0.83</td>
</tr>
<tr>
<td>Level 3</td>
<td>$2 \times 137 \times 97$</td>
<td>2.00e-6</td>
<td>4.01e-3</td>
<td>0.41</td>
</tr>
<tr>
<td>Level 4</td>
<td>$2 \times 273 \times 193$</td>
<td>1.00e-6</td>
<td>2.00e-3</td>
<td>0.21</td>
</tr>
</tbody>
</table>
DG(P₁), rDG(P₁P₂) and WENO(P₁P₂) on level 3 hexahedral grid. Also, Figure 6.4 presents the nondimensionalized turbulence eddy viscosity at \( x = 0.97 \) with DG(P₁), rDG(P₁P₂) and WENO(P₁P₂) on level 3 hexahedral grid. As expected, the eddy viscosity drops to zero at the wall and approaches nearly zero rapidly at the edge of the boundary layer. Notice however that the eddy viscosity remains non-negative across the entire domain even if \( \tilde{\nu} \) becomes negative.

It could be observed that, by comparison, the results using WENO(P₁P₂) is almost identical to those using rDG(P₁P₂). We need to mention that, the Hermite WENO reconstruction-based discontinuous Galerkin method WENO(P₁P₂), is designed not only to enhance the accuracy of discontinuous Galerkin method but also to ensure linear stability of the rDG method on unstructured grids. Therefore, using this case with unstructured hexahedral mesh, we have further demonstrated the WENO(P₁P₂) could keep the accuracy of discontinuous Galerkin method.

Figure 6.5 shows the grid convergence of the drag coefficient, i.e., \( C_D \). In the plot the x-axis is plotting the number of degrees of freedom, using logarithmic scale. From the Figure 6.5, we could observe some interesting phenomenon. First, DG(P₁), rDG(P₁P₂) and WENO(P₁P₂) all could converge to the steady state solution. And it is obvious that rDG(P₁P₂) is superior to DG(P₁) in terms of number of degrees of freedom. This means that, third-order rDG(P₁P₂) only requires less degrees of freedom to arrive at the grid-independent solution compared with DG(P₁). Furthermore, the grid convergence of the drag coefficient is almost indistinguishable between rDG(P₁P₂) and WENO(P₁P₂), which demonstrate the accuracy of WENO(P₁P₂). Second, as a matter of fact, there exists a little difference between our converged solution and the reference solution of CFL3D and FUN3D. Obviously, even for well-validated CFL3D and FUN3D, there exists a little difference for the converged solution. This may be caused by different implementation of turbulence model equation. Compared with CFD3D, our converged solution agrees well with the reference solution up to 4 decimals, and the difference is less than 0.42%, which is acceptable. Finally, the typical convergence histories using different numerical methods on Level 3 hexahedral grid are presented in Figure 6.6. It could be observed that, the residual of DG(P₁) and rDG(P₁P₂) could drop 7 orders of magnitude, whereas the residual of WENO(P₁P₂) keeps oscillating after it drops 5 orders of magnitude. This could be explained as follows. This phenomenon is sometimes called "convergence stall", especially when nonlinear approaches like WENO schemes are used.
Figure 6.1: The overview of the hexahedral grid (Level 3) for the case, turbulence flow over a flat plate.

Figure 6.2: The eddy viscosity contours (nondimensionalized by freestream laminar viscosity $\mu_{\infty}$) using rDG($P_1P_2$) method on Level 3 hexahedral grid.
Figure 6.3: The logarithmic plot of surface friction coefficient distribution over the flat plate obtained by DG(P₁), rDG(P₁P₂), WENO(P₁P₂) methods on Level 3 hexahedral grid.

Figure 6.4: Eddy viscosity ratio distribution at x=0.97, obtained by DG(P₁), rDG(P₁P₂), and WENO(P₁P₂) methods.
Figure 6.5: The grid convergence of drag coefficient obtained by DG(P₁), rDG(P₁P₂), and WENO(P₁P₂) methods.

Figure 6.6: Convergence histories using DG(P₁), rDG(P₁P₂), and WENO(P₁P₂) on Level 3 hexahedral grid.
6.1.2 2D Turbulent Flow over a 2D Bump in a Channel

This test case simulates the 2D turbulent flow over a 2D bump in a channel, where $Ma_\infty = 0.2$, and $Re_\infty = 3 \times 10^6$. This test case is different from the flat plate test case above as it involves wall curvature, thus, pressure gradients is no longer equal to zero. Symmetric boundary conditions are used for the top and bottom of the channel, with exception of a smooth bump extending from $x = 0.0$ to $x = 1.5$, where adiabatic wall boundary condition is applied. The whole computational domain is $[-25.0, 25.0] \times [0.0, 5.0]$. 3 hexahedral meshes used for this test case, are fully quadratic curved meshes generated by GMSH library. The details of the meshes are given in Table 6.2. The level 2 mesh is shown in Figure 6.7.

Table 6.2: The quadratic meshes information for 2D turbulent flow over a 2D bump.

<table>
<thead>
<tr>
<th>Grid No.</th>
<th>Dimensions</th>
<th>Wall Spacing($y$ plus)</th>
<th>No. of Points Defining Bump</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level 1</td>
<td>$2 \times 56 \times 31$</td>
<td>$4.00e-5(4.8)$</td>
<td>75</td>
</tr>
<tr>
<td>Level 2</td>
<td>$2 \times 111 \times 61$</td>
<td>$2.00e-5(2.4)$</td>
<td>149</td>
</tr>
<tr>
<td>Level 3</td>
<td>$2 \times 221 \times 121$</td>
<td>$1.00e-5(1.2)$</td>
<td>297</td>
</tr>
</tbody>
</table>

This is a 3D simulation of a 2D problem. Figure 6.8 presents the nondimensionalized turbulent eddy viscosity contours, i.e., $\mu_t/\mu_\infty$, using rDG($P_1P_2$) on Level 3 hexahedral grid. It is essentially indistinguishable from the reference solution of CFL3D and FUN3D from NASA database.

Figure 6.9 presents the friction coefficient distribution along the 2D wall bump. By comparison with the reference solution from CFL3D and FUN3D on the finest mesh, i.e., $2 \times 1409 \times 641$, it could be observed that, it is a little difficult to compute the skin friction accurately at the top of bump. Also, indeed there exists some oscillation about the skin friction coefficient distribution, which has been observed by other researchers. They claims that, DG($P_1$) combined with quadratic curved mesh leads to the oscillation, and higher-order scheme, e.g., DG($P_2$) combined with curved mesh could remove such oscillation. Figure 6.10 and Figure 6.11 show the grid convergence of lift coefficient, $C_L$, and drag coefficient $C_D$ for this case. It could be observed that, both DG($P_1$) and rDG($P_1P_2$) could converge to the steady state solution, which agrees well with the reference solution from CFL3D and FUN3D. Also, it is quite obvious that, compared with the CFD3D and FUN3D, high-order rDG schemes just requires much less degrees of freedom to converge to the steady state solution.

Finally, the logarithmic residual convergence histories using DG($P_1$) and rDG($P_1P_2$) on Level 2 grid are given in Figure 6.12, which shows that the residual of both schemes could almost
reduce 7 orders of magnitude, demonstrating that the steady state solution has been obtained.
Figure 6.8: The eddy viscosity contours (nondimensionalized by freestream laminar viscosity $\mu_\infty$) using the rDG($P_1P_2$) method along the 2D wall bump.

Figure 6.9: The friction coefficient distribution along the 2D wall bump.
Figure 6.10: Grid convergence for lift coefficient $C_L$

Figure 6.11: Grid convergence for drag coefficient $C_D$
Figure 6.12: Convergence histories using DG(P$^1$), and rDG(P$^1$P$^2$) on Level 3 hexahedral grid.
6.1.3 Turbulent Flow over a NACA0012 Airfoil

This case considered is the turbulent flow over a NACA0012 airfoil, where $Ma_{\infty} = 0.15$, and $Re_{\infty} = 5 \times 10^6$ based on length "1" of the grid. The computational meshes employed for this test case are given in the following Table 6.3, which are quadratic curved meshes generated by GMSH library. The far field boundary is located almost 480 chords away from the airfoil. The level 2 mesh is shown in Figure 6.14.

This test case could be run at different Angle of Attack, abbreviated as AoA. In our work, only the test cases for $AoA = 0^0$, $AoA = 10^0$, $AoA = 15^0$ will be calculated.

First, the numerical results for $AoA = 0^0$ are shown in this section. For $AoA = 0^0$, the computed lift coefficient $C_L$ is approximately equal to zero. The grid convergence of drag coefficient $C_D$ is shown in Figure 6.15. The converged values for the lift and drag coefficient are comparable to the reference value of CFL3D from NASA database. It is obvious that, with higher discretization order, the rDG(P₁P₂) requires much less degrees of freedom to get the grid independent solution compared with DG(P₁). Also, the logarithmic residual convergence histories using DG(P₁) and rDG(P₁P₂) on Level 2 hexahedral grid are presented in Figure 6.16, which shows that the residual of both schemes could almost reduce 7 orders of magnitude.

Second, the numerical results for $AoA = 10^0$ are shown herein. For $AoA = 10^0$, the lift coefficient $C_L$ and drag coefficient $C_D$ is shown in Table 6.4. The converged values for the lift and drag coefficient are comparable to the reference value of CFL3D from NASA database. It is obvious that, with higher discretization order, the rDG(P₁P₂) requires much less degrees of freedom to get grid independent solution compared with DG(P₁). In addition, for $AoA = 10^0$, the nondimensionalized turbulent eddy viscosity, i.e., $\mu_t/\mu_{\infty}$ using rDG(P₁P₂) on Level 3 hexahedral grid, is presented in Figure 6.13. The wake is resolved very well using rDG(P₁P₂).

Third, the numerical results for $AoA = 15^0$ are shown in this section. For $AoA = 15^0$, the lift coefficient $C_L$ and drag coefficient $C_D$ on Level 3 hexahedral grid is shown in Table 6.4. The
Table 6.4: Computed lift and drag coefficients for the NACA0012 with $Ma_\infty = 0.15$, $AoA = 15^0$, and $Re_\infty = 6 \times 10^6$ using DG($P_1$) and rDG($P_1P_2$) on Level 3 hexahedral grid.

<table>
<thead>
<tr>
<th>Schemes</th>
<th>$C_L$</th>
<th>$C_D$</th>
</tr>
</thead>
<tbody>
<tr>
<td>DG($P_1$)</td>
<td>1.5484</td>
<td>0.02130</td>
</tr>
<tr>
<td>rDG($P_1P_2$)</td>
<td>1.5496</td>
<td>0.02110</td>
</tr>
<tr>
<td>Reference value</td>
<td>1.5446 – 1.5642</td>
<td>0.02073 – 0.02159</td>
</tr>
</tbody>
</table>

Converged values for the lift and drag coefficient are comparable to the reference value range from NASA database. From NASA database, for this test case, the numerical results are from several independent CFD codes: CFL3D, FUN3D, NTS, JOE, SUMB and so on. Thus, there exists one reference solution range, which has been give in Table 6.4.

In addition to the lift and drag coefficient, the friction coefficient $C_f$ distribution at different $AoA$ on the third mesh is also presented in Figure 6.18. The solution of our methods agree well with the reference one from CFL3D.

Finally, the drag polar for this case is given in Figure 6.19. It could be observed that, the numerical results of our rDG schemes match well with the experimental data, demonstrating the accuracy of the developed rDG method for turbulent flows.

Figure 6.13: The nondimensionalized turbulent eddy viscosity using rDG($P_1P_2$) on Level 2 hexahedral grid for $AoA = 10^0$. 
Figure 6.14: Plot of the quadratic Level 2 hexahedral for 2D turbulent flow over the NACA0012 airfoil.
Figure 6.15: The grid convergence of drag coefficient $C_D$ for turbulent flow over 2D NACA0012 airfoil for $AoA = 0^\circ$.

Figure 6.16: Convergence histories using DG(P$_1$), rDG(P$_1$P$_2$) on Level 2 hexahedral grid for $AoA = 0^\circ$. 
Figure 6.17: The grid convergence of lift coefficient $C_L$ and drag coefficient $C_D$ for turbulent flow over 2D NACA0012 airfoil for $AoA = 10^\circ$. 

(a) The grid convergence of lift coefficient $C_L$

(b) The grid convergence of drag coefficient $C_D$
Figure 6.18: The friction coefficient $C_f$ distribution for turbulent flow over 2D NACA0012 airfoil for different AoA (a) AoA = 0°, (b) AoA = 10°.

94
Figure 6.18: The friction coefficient $C_f$ distribution for turbulent flow over 2D NACA0012 airfoil for different AoA (c)AoA = 15°.

Figure 6.19: Drag polar on Level 3 mesh for turbulent flow over 2D NACA0012 airfoil.

95
6.1.4 Turbulent Flow over a 3D Sinusoidal Bump in a Channel

This case is the 3D sinusoidal bump in a channel defined analytically in the NASA database. This is the simplest 3D test case, where the wall bump is smooth and it has no flow separation. The Reynolds number per unit of length is \( Re = 3 \times 10^6 \) and the Mach number is equal to 0.2. For this 3D case, the mesh is from official website. The first 3 quadratic curved meshes for DG(P_1) method are obtained just by simple agglomeration of the linear hexahedral grids of the NASA TMR website, without any change of the high-order nodes. The detailed information about the quadratic curved mesh and the original linear element grid are given in Table 6.5. Due to the spanwise variation added for this 3D sinusoidal bump test, in order to guarantee the stability of rDG(P_1P_2), it requires a little finer resolution along the spanwise direction. Thus, the 4\(^{th}\) quadratic mesh is obtained by agglomeration based on the linear mesh generated by the author. The Level 1 quadratic hexahedral mesh is presented in Figure 6.20.

First, the numerical results based on the first 3 quadratic meshes are presented here using DG(P_1). The grid convergence of the lift and drag coefficient, \( C_L \) and \( C_D \), is shown in Figure 6.21. From Figure 6.21a, it could be observed that, DG(P_1) scheme just requires less degrees of freedom to arrive at the converged \( C_L \). And from Figure 6.21b, DG(P_1) scheme is a little superior to CFL3D and FUN3D in terms of number of degrees of freedom. In addition, the converged lift and drag coefficients of our rDG schemes agree well with the reference solution from CFL3D and FUN3D. The convergence of the pressure and viscous drag coefficient, \( C_{Dp} \) and \( C_{Dv} \), using DG(P_1) is shown in Figure 6.22.

Second, the numerical results based on the fourth quadratic mesh are presented here using both DG(P_1) and rDG(P_1P_2). The calculation with this mesh is intended to make a direct comparison between DG(P_1) and rDG(P_1P_2). The lift and drag coefficients, \( C_L \) and \( C_D \), are shown in Figure 6.23. Figure 6.23a presents the lift coefficient, \( C_L \), which is almost identical for DG(P_1) and rDG(P_1P_2). Whereas, the drag coefficient, \( C_D \), using rDG(P_1P_2) which is shown in Figure 6.23b, is approaching closer to the reference solution from CFL3D and FUN3D. This indicates that, compared with the reference solution, on the same mesh, rDG(P_1P_2) could provide more accurate solution. In addition, the pressure and viscous drag coefficients of our rDG schemes are also presented in Figure 6.24.

Then the nondimensionalized turbulent eddy viscosity at different locations, e.g., \( x = 0.3 \), and \( x = 1.2 \) is shown in Figure 6.26. The distribution agrees well with the reference solution from CFL3D.
Table 6.5: The quadratic meshes information for 3D turbulent flow over a 3D bump.

<table>
<thead>
<tr>
<th>Grid No.</th>
<th>Dimensions</th>
<th>Original linear mesh</th>
<th>Wall Spacing((y \text{ plus}))</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level 1</td>
<td>5 × 45 × 21</td>
<td>9 × 89 × 41</td>
<td>1.60 (e^{-5}(2.0))</td>
</tr>
<tr>
<td>Level 2</td>
<td>9 × 89 × 41</td>
<td>17 × 177 × 81</td>
<td>8.00 (e^{-6}(1.0))</td>
</tr>
<tr>
<td>Level 3</td>
<td>17 × 177 × 81</td>
<td>33 × 353 × 161</td>
<td>4.00 (e^{-6}(0.5))</td>
</tr>
<tr>
<td>Level 4</td>
<td>31 × 181 × 41</td>
<td>61 × 361 × 81</td>
<td>2.15 (e^{-5}(2.6))</td>
</tr>
</tbody>
</table>

Figure 6.20: The Level 1 quadratic hexahedral mesh for turbulent flow over a 3D sinusoidal bump in a Channel.
Figure 6.21: The grid convergence of lift and drag coefficient using DG(P₁) on the first 3 quadratic meshes for turbulent flow over a 3D sinusoidal bump in a Channel.

Figure 6.22: The grid convergence of pressure and viscous drag coefficients using DG(P₁) on the first 3 quadratic meshes for turbulent flow over a 3D sinusoidal bump in a Channel.
Figure 6.23: The lift and drag coefficient using DG(P₁) and rDG(P₁P₂) on the fourth quadratic mesh for turbulent flow over a 3D sinusoidal bump in a Channel.

Figure 6.24: The pressure and viscous drag coefficient using DG(P₁) and rDG(P₁P₂) on the fourth quadratic mesh for turbulent flow over a 3D sinusoidal bump in a Channel.
Figure 6.25: The logarithmic residual using DG(P₁) and rDG(P₁P₂) on level 4 quadratic mesh for turbulent flow over a 3D sinusoidal bump in a Channel.
Figure 6.26: The nondimensionalized turbulent eddy viscosity at different locations on Level 4 quadratic mesh using rDG(P₁P₂)
6.1.5 3D Turbulent Flow over a 3D Hemisphere Cylinder

NASA TMR also provides a test case for a turbulent flow over a smooth body of revolution in 3D. This case is designed primarily for numerical analysis of turbulence model simulations; e.g., convergence properties, effect of order of accuracy, etc. The Reynolds number per unit of length is $Re = 1.3779 \times 10^7$ and the Mach number is equal to 0.6. For this 3D case, the quadratic curved meshes for high-order rDG method are obtained just by simple agglomeration of the linear hexahedral grids from the TMR website, without any change of the high-order nodes. The detailed information about the quadratic curved mesh and the original linear grid is given in Table 6.6. Then the Level 3 quadratic curved mesh is presented in Figure 6.27.

Figure 6.28 presents the grid convergence of drag coefficient, $C_D$ using DG($P_1$) and WENO-($P_1P_2$) respectively. It is WENO($P_1P_2$) rather than rDG($P_1P_2$) without any reconstruction is used to achieve the calculation, since that WENO reconstruction on $P_2$ has to be used to ensure the stability for this case. Due to stability problem, for WENO($P_1P_2$), only the numerical results on the Level 3 quadratic mesh are presented. First, it seems that for the coarse mesh, the second-order CFL3D is superior to high-order DG($P_1$) scheme, since it only requires a little less degrees of freedom to achieve the same accuracy compared with the high-order DG($P_1$) scheme. However, with mesh refined, it could be observed that, high-order DG($P_1$) scheme outperforms the second-order CFL3D. Also, on Level 3 quadratic curved mesh, the solution for WENO($P_1P_2$) is approaching closer to the converged reference value from CFL3D.

Figure 6.29 shows the pressure coefficient distribution axially. It is obvious that, our numerical solution agrees quite well with the experimental data and the reference solution from CFL3D. Figure 6.30 shows the friction coefficient distribution axially. Indeed, there exist a little difference between our numerical solution and and the reference solution from CFL3D. However, it is necessary to point out that, the reference solution from CFL3D is based on the finest mesh, i.e., $161 \times 289 \times 129$.

Figure 6.31 shows the cell-averaged nondimensionalized turbulent eddy viscosity contours using WENO($P_1P_2$) on the Level 3 quadratic mesh. It could be observed that, the turbulent eddy viscosity grows gradually along the flow direction near the hemisphere wall.

Finally, the logarithmic residual convergence histories using DG($P_1$) and WENO($P_1P_2$) on Level 3 grid are given in Figure 6.32, which shows that the residual of DG($P_1$) schemes could reduce 8 orders of magnitude. We also note that, similar to that in the first test case, the phenomenon 'convergence stall' also occurs for this case with WENO($P_1P_2$). And we need to mention that, the phenomenon 'convergence stall' is about the residual convergence, rather than about the results.
Table 6.6: The quadratic meshes information for 3D turbulent flow over a 3D hemisphere cylinder.

<table>
<thead>
<tr>
<th>Grid No.</th>
<th>Dimensions</th>
<th>Original linear mesh</th>
<th>Wall Spacing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level 1</td>
<td>11 × 19 × 21</td>
<td>21 × 37 × 17</td>
<td>2.76 e−5</td>
</tr>
<tr>
<td>Level 2</td>
<td>21 × 37 × 17</td>
<td>41 × 73 × 33</td>
<td>1.24 e−5</td>
</tr>
<tr>
<td>Level 3</td>
<td>41 × 73 × 33</td>
<td>81 × 145 × 65</td>
<td>5.90 e−6</td>
</tr>
</tbody>
</table>

Figure 6.27: The Level 3 quadratic hexahedral mesh for turbulent flow over a 3D hemisphere cylinder.

Figure 6.28: The grid convergence of drag coefficient $C_D$ using DG(P1) and WENO(P1P2).
Figure 6.29: The pressure coefficient distribution using DG(P₁) and WENO(P₁P₂) on the Level 3 quadratic mesh.

Figure 6.30: The friction coefficient distribution using DG(P₁) and WENO(P₁P₂) on the Level 3 quadratic mesh.
Figure 6.31: The cell-averaged nondimensionalized turbulent eddy viscosity contours using WENO(P₁P₂) on the Level 3 quadratic mesh.

Figure 6.32: The logarithmic residual convergence histories using DG(P₁) and WENO(P₁P₂) on the Level 3 quadratic mesh.
6.1.6 2D Transonic Turbulent Flow over RAE2822 airfoil

This test case is aimed to test high-order methods for a 2D turbulent flow under transonic conditions with weak shock-boundary layer interaction effects. The test case is the RAE2822 airfoil Case 9, for which an extensive experimental database exists [34]. The test case has also been investigated numerically by many authors using low order methods. The original flow conditions in the wind tunnel experiment are $Ma = 0.730$, angle of attack $AoA = 3.19^\circ$, Reynolds number (based on the reference chord) $Re = 6.5 \times 10^6$. However, in order to take into account the wind tunnel corrections for comparison with experimental data, the computations have been made with corrected flow conditions, namely $Ma = 0.734$, angle of attack $AoA = 2.79^\circ$, Reynolds number (based on the reference chord) $Re = 6.5 \times 10^6$. For this case, the high-order meshes from high-order workshop are used. However, one thing has to be mentioned that the meshes from high-order workshop themselves are quartic curved elements. Then, the quadratic curved elements used in this thesis could be obtained just by extracting the associated points from the quartic curved elements. The quadratic curved mesh is shown in Figure 6.33. And the information of quadratic curved mesh used in our thesis is listed in Table 6.7. Figure 6.34 presents the pressure contours distribution using DG($P_1$) method. This case could be run without limiter. Figure 6.34a presents the pressure contours without limiter. This demonstrates that our solver is robust. However, it is obvious that, from Figure 6.34b there indeed exists some obvious oscillation near the shock, which is expected. Then once the limiter is added, from Figure 6.34d, we could find that the oscillation almost disappears, demonstrating WENO limiter developed really could suppress the oscillation near the discontinuity. Then the pressure contours from HWENO($P_1P_2$) are presented in Figure 6.35. From Figure 6.35b, it could be observed that the oscillation is almost suppressed, demonstrating the good performance of HWENO($P_1P_2$) method. The lift and drag coefficient are also given in Table 6.8, which is comparable to the reference solution from high order workshop. We need to point out that, here, only the quadratic curved meshes are used rather than quartic ones.

Table 6.7: The quadratic meshes information for 2D transonic turbulent flow over a RAE2822 airfoil.

<table>
<thead>
<tr>
<th>Grid No.</th>
<th>No. of elements</th>
<th>Wall Spacing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level 5</td>
<td>129,536</td>
<td>$1.3e-6$</td>
</tr>
</tbody>
</table>
Figure 6.33: Plot of the quadratic hexahedral mesh for 2D transonic turbulent flow over the RAE airfoil.
Figure 6.34: Pressure contours distribution for 2D transonic turbulent flow over the RAE airfoil with DG(P₁) method.
Figure 6.35: Pressure contours distribution for 2D transonic turbulent flow over the RAE airfoil with HWENO(P₁P₂) method.

Table 6.8: Computed lift and drag coefficients for the RAE2822 with $Ma_\infty = 0.734$, $AoA = 2.79^0$, and $Re_\infty = 6.5 \times 10^6$ using DG(P₁) and rDG(P₁P₂) on quadratic curved hexahedral grid.

<table>
<thead>
<tr>
<th>Schemes</th>
<th>$C_L$</th>
<th>$C_D$</th>
</tr>
</thead>
<tbody>
<tr>
<td>DG(P₁)</td>
<td>0.7877</td>
<td>$1.8564 \times 10^{-2}$</td>
</tr>
<tr>
<td>rDG(P₁P₂)</td>
<td>0.7882</td>
<td>$1.8560 \times 10^{-2}$</td>
</tr>
<tr>
<td>MIT</td>
<td>0.823</td>
<td>0.0184</td>
</tr>
<tr>
<td>Umich</td>
<td>0.788</td>
<td>0.0193</td>
</tr>
</tbody>
</table>

Finally, the pressure and friction coefficient distribution are presented in Figure 6.36.
Figure 6.36: Pressure and friction coefficient distribution for 2D transonic turbulent flow over the RAE airfoil.
6.2 Unsteady Cases

In this section, the unsteady cases mainly focus on the ILES and DNS for 3D turbulent flow. Several challenging test cases, including 3D lid driven cavity and 3D Taylor-Green vortex has been presented in this section to assess the performance and accuracy of our rDG code. As mentioned in §4, implicit ROW method is mainly used to advance time integration. Whereas, in order to validate the implementation of ROW method, a lot of unsteady flows will be calculated prior to the calculation for ILES and DNS.

A number of test cases are presented in order to assess the performance of the Rosenbrock schemes for computing the unsteady flows. In addition, comparative studies between the third-order ROW and ESDIRK3 schemes are conducted using the fixed time step sizes, where the related results of a four-stage, third-order ESDIRK3 scheme are refered to Xia et al. [143]. We also make comparison between ROW for index-1 and ROW for index-2 in case §6.2.2 and case §6.2.3. Since the relative error tolerances for the linear and non-linear systems solution have great influence on the efficiency and accuracy of a scheme, we investigate such influence by the case of convection of an isotropic vortex. The $L_2$-norm by Eq. 6.3, is used as the criterion to determine whether the relative tolerance is sufficient enough to get the converged solution. Here, the relative error tolerance $1e^{-n}$ in the tables means that the residual of the iteration is dropped $n$ orders of magnitude. Based on the Table 6.9 and Table 6.10, the relative error tolerances for the non-linear and linear iterations are determined for all the test cases in this paper, shown in Table 6.11.

Table 6.9: The investigation of relative error tolerances for the non-linear and linear iterations about the ESDIRK3.

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Non-linear</th>
<th>Linear</th>
<th>log(L2-norm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESDIRK3 × GMRES+LU-SGS</td>
<td>1E-3</td>
<td>1E-2</td>
<td>-4.8400</td>
</tr>
<tr>
<td></td>
<td>1E-2</td>
<td>1E-2</td>
<td>-4.7073</td>
</tr>
</tbody>
</table>

6.2.1 Convection of an Isentropic Vortex in Inviscid Flows

In this test case, we consider the passive convection of a quasi-2D inviscid isentropic vortex [43, 75] in order to assess the temporal convergence for the Rosenbrock and ESDIRK methods. The spatial discretization is carried out using a third-order rDG(P1P2) method. The analytical solution to this problem at any time $t$ is simply the passive advection of the initial solution at $t = 0$, which provides a valuable reference for measuring the accuracy of a numerical solution.
Table 6.10: The investigation of relative error tolerances for the linear iterations about the ROW.

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Non-linear</th>
<th>Linear</th>
<th>log(L2-norm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROSI2PW × GMRES+LU-SGS</td>
<td>– 1E-4</td>
<td>-4.8190</td>
<td></td>
</tr>
<tr>
<td></td>
<td>– 1E-3</td>
<td>-4.8182</td>
<td></td>
</tr>
<tr>
<td></td>
<td>– 1E-2</td>
<td>-4.7970</td>
<td></td>
</tr>
<tr>
<td>ROS34PRW × GMRES+LU-SGS</td>
<td>– 1E-4</td>
<td>-4.8190</td>
<td></td>
</tr>
<tr>
<td></td>
<td>– 1E-3</td>
<td>-4.8150</td>
<td></td>
</tr>
<tr>
<td></td>
<td>– 1E-2</td>
<td>-4.38631</td>
<td></td>
</tr>
</tbody>
</table>

Table 6.11: Relative error tolerance for the non-linear and linear iterations.

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Non-linear</th>
<th>Linear</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESDIRK × GMRES+LU-SGS</td>
<td>1E-3</td>
<td>1E-2</td>
</tr>
<tr>
<td>Rosenbrock-Wanner × GMRES+LU-SGS</td>
<td>–</td>
<td>1E-3</td>
</tr>
</tbody>
</table>

The initial condition is a linear superposition of a mean uniform flow with some perturbations. The free stream flow conditions are \((ρ_∞, u_∞, v_∞, p_∞) = (1, 1, 0, 1)\). The perturbations of the velocity components \(u\) and \(v\), entropy \(S\), and temperature \(T\) for the vortex are given by

\[
\delta T = -\frac{(\gamma - 1)}{8\gamma \pi^2} e^{1-\gamma^2} \delta u \quad \text{and} \quad \delta T = -\frac{(\gamma - 1)}{8\gamma \pi^2} e^{1-\gamma^2} \delta v = \frac{\varepsilon}{2\pi} e^{1-\gamma^2}\left(-\frac{(y - y_0)}{(x - x_0)}\right), \delta S = 0. \tag{6.1}
\]

where \(r^2 = (x - x_0)^2 + (y - y_0)^2\), and \((x_0, y_0)\) is the coordinate of the vortex center, and \(\varepsilon\) is the vortex strength. From \(ρ = ρ_∞ + δρ, u = u_∞ + δu, v = v_∞ + δv, T = T_∞ + δT\), and the isentropic relation, other physical variables can be determined as follows:

\[
ρ = T^{\frac{1}{\gamma - 1}} = (T_∞ + δT)^{\frac{1}{\gamma - 1}} = T_∞ - \frac{(\gamma - 1)}{8\gamma \pi^2} e^{1-\gamma^2} \frac{1}{\gamma - 1},
\]

\[
p = \rho γ,
\]

\[
e = \frac{p}{ρ(\gamma - 1)} + \frac{1}{2}(u^2 + v^2).
\]

The vortex strength \(ε = 5\), and the coordinate of the vortex center \((x_0, y_0) = (5, 0)\). The computational domain \(Ω \) is \([0, 10] \times [-5, 5]\) and the periodic boundary conditions are imposed.

112
The numerical solutions are obtained after one period, \( t = 10 \), and compared with the analytical solution simply given by the initial condition. The following \( L_2 \)-norm

\[
\| \rho - \rho^r \|_{L_2(\Omega)} = \sqrt{\int_\Omega (\rho - \rho^r)^2 \, d\Omega},
\]

is used to measure the error between the numerical and analytical solutions, where \( \rho \) is the numerical solution for the density and \( \rho^r \) is the reference one. Because the overall error is due to both spatial and temporal errors, a reference solution is obtained using a small time-step size at \( \Delta t = 0.001 \) in order to eliminate the effect of spatial error and isolate the temporal error. Figure 6.37a shows the hexahedral grid used in the computation, which consists of 1600 hexahedral cells. A series of successively refined time-step sizes \( \Delta t = 0.4, 0.2, 0.1, 0.05, \) and 0.025 are used in calculation for the temporal convergence study. Figure 6.38a provides the details of the temporal convergence for this numerical experiment. The ESDIRK3, ROSI2PW, and ROS34PRW schemes exhibit a slope of 2.97, 2.97 and 2.96, respectively, indicating that these three methods are able to offer the nearly third-order accuracy of temporal convergence. However, the ROSI2Pw scheme only exhibits a slope of 2.0, which is consistent to the theory that

Figure 6.37: (a) The computational grid used for convection of an isentropic vortex in inviscid flow. (b) Density distribution for ESDIRK3 at \( t=10 \).
this scheme would require the exact Jacobians in order to achieve the formal order of temporal accuracy. Figure 6.38b indicates that third order ROW consumes much less computing time than ESDIRK3 to achieve the same level of temporal errors. For reference purpose, the results obtained by the explicit three-stage third-order TVD Runge-Kutta (TVDRK3) time integration scheme are also presented in Figure 6.38a and Figure 6.38b. Note that the explicit methods are usually more efficient than their implicit counterparts for the type of problems like in this test case, where the physical time scales would not allow the implicit methods to use very large time step sizes while trying to retain the wave propagation accurately.

![Figure 6.38](image)

Figure 6.38: Temporal convergence of rDG($P_1P_2$) with different time integration schemes obtained for convection of an isentropic vortex. (a)$L_2$ norm versus dt and (b)$L_2$ norm versus CPU time.
6.2.2 Von-Karman Vortex Street

The von-Karman vortex street is probably one of the most extensively studied cases both experimentally and numerically in fluid dynamics. In this test case, the flow conditions are chosen (Reynolds number of $Re_\infty = 200$ based on the cylinder diameter of $d = 1.0$) such that vortex shedding is expected to occur downstream of the cylinder. This simulation provides a geometrically simple, yet aerodynamically complex test for our solver. Figure 6.39 shows the grid used in the computations, which consists of 10,204 hexahedral elements, 20,800 grid points, and 20,800 boundary faces. Note that this 3D grid is extruded from a 2D grid used in [50]. The normal grid spacing near the cylinder surface is 0.001 (normalized by the cylinder diameter), as demonstrated in Figure 6.39b. The free-stream Mach number $Ma_\infty$ is set as 0.2. We employ the steady-state solution obtained for $Re_\infty = 50$ (for which vortex shedding is not expected to occur) and an angle of attack $\alpha = 3^\circ$ as the initial solution for vortex shedding. To start the simulation of shedding, the flow conditions at the characteristic boundaries are set to $Re_\infty = 200$ and $\alpha = 0^\circ$ (the asymmetry of the initial flow, or sudden change of the angle of attack, leads to the vortex shedding behavior). The solutions were advanced in time with a fixed time-step size of $\Delta t = 0.5$ for ROS34PRW + rDG ($P_1 P_2$), $\Delta t = 0.1$ for ROSI2PW + rDG ($P_1 P_2$) and $\Delta t = 0.02$ for ROS34PW2 + rDG ($P_1 P_2$), respectively, until 10 shedding cycles had passed to ensure that

Figure 6.39: Mesh used for computing a viscous flow past a cylinder ($nelem = 10204$, $npoin = 20800$, $nboun = 20800$).
Figure 6.40: Time history of the computed lift and drag coefficients by ROS34PRW + rDG(P₁P₂) for flow past a cylinder at \( M_\infty = 0.2 \) and \( Re_\infty = 200 \).
Figure 6.41: Time history of the computed lift and drag coefficients by ROSI2PW + rDG(P\textsubscript{1}P\textsubscript{2}) for flow past a cylinder at $M_\infty = 0.2$ and $Re_\infty = 200$. 
the solution was periodic in time. Note that in this test case, a much smaller allowable time-step size has to be used for ROSI2PW and ROS34PW2, which is determined considering the stability rather than the accuracy. Time histories of the computed surface drag and lift coefficients are presented in Figure 6.40 obtained by ROS34PRW + rDG (P₁P₂), and in Figure 6.41 obtained by ROSI2PW + rDG (P₁P₂). The Strouhal number is 1.923. These results agree well with experimental measurements and numerical results in the literature [50]. To assess the efficiency of the ROW methods, the computational costs for both ROW and ESDIRK3 solutions are listed in Table 6.12. Firstly, the comparison between ESDIRK3 and ROW methods is presented here. In comparison with ESDIRK3, ROS34PRW requires 45% CPU time less than ESDIRK3, using more strict criterion for GMRES. This is caused by that just linear systems need to be solved for ROW schemes, leading to that the computational costs per time step are reduced. Secondly, it could be observed that the ROW for index-2 DAEs outperforms the one for index-1 in terms of the computation efficiency. ROS34PRW and ROSI2PW requires 88% and 64% CPU time less than ROS34PW2, respectively. This could be explained as follows, some ROW schemes have to use small time step to preserve their stability. Above all, ROS34PRW demonstrated the best efficiency in this test case.

Table 6.12: Comparison of the CPU time (evaluated by running on 256 cores) between the ROW and ESDIRK3 for computing a viscous flow past a cylinder

<table>
<thead>
<tr>
<th>Time integration method</th>
<th>Time-step size</th>
<th>Time steps</th>
<th>CPU time</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESDIRK3</td>
<td>0.50</td>
<td>1000</td>
<td>965.1</td>
</tr>
<tr>
<td>ROSI2PW</td>
<td>0.10</td>
<td>5000</td>
<td>1555.8</td>
</tr>
<tr>
<td>ROS34PRW</td>
<td>0.50</td>
<td>1000</td>
<td>531.64</td>
</tr>
<tr>
<td>ROS34PW2</td>
<td>0.02</td>
<td>25000</td>
<td>4356.0</td>
</tr>
</tbody>
</table>

118
6.2.3 Flow past a SD7003 Airfoil

In this subsection, we consider a viscous flow past an SD7003 airfoil at $Ma_{\infty} = 0.2$, $\alpha = 4^\circ$, and $Re_{\infty} = 10,000$, as an abundance of numerical results can be found about this test case [123, 75]. The computation is initialized with constant values in the entire domain with no-slip, adiabatic boundary conditions on the solid wall, and terminated at $t = 125$. Figure 6.42 shows the grid used in the computation, which consists of 50,781 prismatic elements, 52,176 grid point, 101,562 triangular boundary faces, and 279 quadrilateral boundary faces.

![Grid used for computing the unsteady viscous flow past a SD7003 airfoil.](image)

(a) Global view

(b) Leading edge

(c) Trailing edge

Figure 6.42: Grid used for computing the unsteady viscous flow past a SD7003 airfoil.

Typical computed pressure contours in the flow field are displayed in Figure 6.43. The instantaneous vorticity contours are displayed in Figure 6.44, which capture the key flow features: separation of the flow on the upper surface of the airfoil and shedding of the tailing vortices.

Figure 6.45 shows the computed velocity vectors in the flow field, where the development of the boundary layers and flow separation on the upper surface of the airfoil are clearly visible.

From the reference [123], the initial transient is over by $t = 75$ as estimated from the forces on the wing. Therefore, unless otherwise specified, the instantaneous forces from $t = 100$ to 125 are presented in Figure 6.46, which could be comparable to the reference [123]. The computation
Figure 6.43: Computed pressure contours in the flow field for flow past a SD7003 airfoil at \( M_\infty = 0.2, \alpha = 4^\circ, \) and \( Re_\infty = 10,000. \)

Figure 6.44: Computed vorticity contours in the flow field for flow past a SD7003 airfoil at \( M_\infty = 0.2, \alpha = 4^\circ, \) and \( Re_\infty = 10,000. \)

Figure 6.45: Computed velocity vector in the flow field for flow past a SD7003 airfoil at \( M_\infty = 0.1, \alpha = 4^\circ, \) and \( Re_\infty = 10,000. \)
is performed using ROSI2PW, ROS34PRW, and ROS34PW2 with a fixed time-step size of $\Delta t = 0.05$, 0.05, and 0.05 respectively. This time-step size is consistent with that in the reference, so that we could make some comparison directly. The instantaneous solutions are written every 0.25 second for time-averaging calculations, starting from $t = 100$ when the flow separation on the upper surface of the airfoil is considered to have fully developed. It could be observed that, the lift and drag coefficient histories are very similar to that in reference [123].

Finally, in order to demonstrate the overall effectiveness of ROW + rDG ($P_1P_2$) on highly stretched grids, the computational costs for both the ROW and ESDIRK3 solutions are presented in Table 6.13. Compared to ESDIRK3, all ROW method require roughly 33% CPU time less; For ROW method, the CPU time for index-1 and index-2 is similar.

Table 6.13: Comparison of the CPU time (evaluated by running on 256 cores) between the ROW and ESDIRK3 for computing a viscous flow past an SD7003 airfoil.

<table>
<thead>
<tr>
<th>Time integration method</th>
<th>Time-step size</th>
<th>Time steps</th>
<th>CPU time</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESDIRK3</td>
<td>0.05</td>
<td>500</td>
<td>711.9</td>
</tr>
<tr>
<td>ROSI2PW</td>
<td>0.05</td>
<td>500</td>
<td>474.04</td>
</tr>
<tr>
<td>ROS34PRW</td>
<td>0.05</td>
<td>500</td>
<td>474.93</td>
</tr>
<tr>
<td>ROS34PW2</td>
<td>0.05</td>
<td>500</td>
<td>470.0</td>
</tr>
</tbody>
</table>
Figure 6.46: Time variation of lift (left) and drag (right) coefficients at Re = 10 000: comparison between ROSI2PW and ROS34PRW.
6.2.4 3D Lid Driven Cavity

The objective of this numerical experiment is to demonstrate the ability of the developed implicit rDG(P\textsubscript{1}P\textsubscript{2}) method for performing the implicit Large Eddy Simulation (ILES without the use of an explicit sub-grid scale turbulence model). With the ever-growing computer performance, the LES or even Direct Numerical Simulation (DNS) by using high-order methods has become more widespread. The DG methods "only dissipate the scales that the model is not able to capture correctly, thus acting like a sub-grid scale (SGS) model". This property makes the DG method an excellent candidate for ILES. A transitional flow in a 3D lid driven cavity at $Re_{\infty} = 10,000$ is considered in this test problem, as the accuracy of the numerical results obtained can be assessed through comparison with both the experimental data by Prasad and Koseff [102] and the LES data by Zang et al. [147]. The cavity dimensions are 1 unit in the

Figure 6.47: The hexahedral grid (64 × 64 × 32 points) for the LES of a lid driven cavity ($x : y : z = 1 : 1 : 0.5$) at $Re = 10,000$.

stream-wise $x$ direction and vertical $y$ direction, and 0.5 unit in the span-wise $z$ direction. A hexahedral grid consisting of $64 \times 64 \times 32$ grid points is used in computation, as shown in Figure 6.47. While being equally distributed in the $z$-direction, the grid points are clustered near the walls in the $x$-$y$ plane, with the grid spacing geometrically stretched away from the wall with the first element thickness being 0.005 ($y^+ = 3.535$). On the bottom and side walls, the no-slip, adiabatic boundary conditions are prescribed. Along the top "lid", the no-slip,
adiabatic boundary conditions with a lid velocity $v_b = (0.2, 0, 0)$ are prescribed as to ensure an essentially incompressible flow field. The computation is conducted in two stages. At stage I, the computation is started with a zero-velocity field, and sufficient steps are taken to evolve the field into a cyclically oscillating state by using BDF1 + rDG(P$_{1}$P$_{2}$) with CFL = 500. We used the solution obtained at the end of stage I as the initial solution for stage II, and run 30,000 time steps with a fixed $\Delta t = 0.1$, during which the instantaneous solutions are written every 300 time steps for time averaging calculations. For a comparative study, the following two options: 1) ROSI2PW + rDG (P$_{1}$P$_{2}$), 2) ROS34PRW + rDG (P$_{1}$P$_{2}$) are used respectively at stage II.

For large eddy simulation, statistic field output is required. In the present work [142], the statistics variable names adhere to the following conventions:

- **Instantaneous variables**
  Instantaneous variables are denoted by the variable name, e.g., velocity component $u$ in $x$-direction.

- **Reynolds means**
  Reynolds means are denoted by angled brackets: $\langle \cdot \rangle$. The mean (or mathematical expectation) is defined by
  \[
  \langle \phi \rangle = \int \phi f(\phi) d\phi 
  \] (6.4)
  where $f(\phi)$ is the probability density function of the fluctuating variable, $\phi$. Assuming the ergodic theorem holds, the expectation in Eq. 6.4 is numerically estimated by $\Delta t$-weighted time-averaged over $N$ time steps, given by the following formula,
  \[
  \langle \phi \rangle = \frac{\sum_{i=1}^{N} \phi_i \Delta t_i}{\sum_{i=1}^{N} \Delta t_i} 
  \] (6.5)

- **Fluctuation about the Reynolds mean**
  An apostrophe, $'$, denotes fluctuation about the Reynolds mean: $q' = q - \langle q \rangle$.

- **The covariance of N variables**
  The covariance of N variables, $p, q, ..., r$, is denoted by $\langle p', q', ..., r' \rangle$, defined as:
  \[
  \langle p', q', ..., r' \rangle = \langle (p - \langle p \rangle), (q - \langle q \rangle), ..., (r - \langle r \rangle) \rangle 
  \] (6.6)

- **TKE and RST**
  The variable names TKE and RST denote the turbulent kinetic energy and Reynolds stress tensor respectively, defined by the averages of the dot-, and tensor-products of the fluctuating velocity vector, $TKE = \langle v' \cdot v' \rangle / 2$, $RST = \langle v'v' \rangle / 2$, respectively.
• RMS

The RMS denotes the root mean square root of the variance of the given fluctuating variable. For example, RMS velocity components $\sqrt{\langle u' u' \rangle}$, $\sqrt{\langle v' v' \rangle}$, and $\sqrt{\langle w' w' \rangle}$.

The computed mean velocity and components of Reynolds stress along the center-lines on the span-wise mid-plane are presented in Figure 6.48-Figure 6.50 respectively. Those profiles were obtained by using a linear polynomial interpolation of the elemental solutions at the intersected nodes cut through by the center-lines. For example, in Figure 6.48, the mean $x$-velocity $u$ is plotted along the vertical center-line (vs. $y$-coordinate), and the mean $y$-velocity $v$ is plotted along the horizontal center-line (vs. $x$-coordinate). As clearly seen from Figure 6.48 to Figure 6.50, these results from the ROSI2PW and ROS34PRW agree well with experimental data and LES data. Furthermore, the high efficiency of the ROW methods is demonstrated in

Table 6.14: Comparison of the CPU time (evaluated by running on 256 cores) between the ROW and ESDIRK3 for computing the LES of a lid driven cavity ($x : y : z = 1 : 1 : 0.5$) at $Re = 10,000$.

<table>
<thead>
<tr>
<th>Time integration method</th>
<th>Time-step size</th>
<th>Time steps</th>
<th>CPU time</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESDIRK3</td>
<td>0.1</td>
<td>30000</td>
<td>164988.0</td>
</tr>
<tr>
<td>ROSI2PW</td>
<td>0.1</td>
<td>30000</td>
<td>103478.0</td>
</tr>
<tr>
<td>ROS34PRW</td>
<td>0.1</td>
<td>30000</td>
<td>100998.6</td>
</tr>
</tbody>
</table>

Table 6.14, where the costs for both the ROW and ESDIRK3 solutions are compared.

Compared to ESDIRK3, ROW method require roughly 38% CPU time less. We also could observe that, with the identical time step, the computational cost for ROW methods ROSI2PW and ROS34PRW are not the same. This is caused by that, for different methods, different iteration numbers are required to satisfy the tolerance criterion for GMRES.
Figure 6.48: Comparison of the normalized mean velocity components $<u>/u_B$ and $<v>/u_B$ in the span-wise mid-plane with the classical experimental data by Prasad & Koseff (1989) and numerical data by Zang et al. (1993), for the LES of a lid driven cavity ($x : y : z = 1 : 1 : 0.5$) at $Re = 10,000$. 


Figure 6.49: Comparison of the scaled RMS velocity components $10 \langle u'u' \rangle^{0.5}/u_B$ and $10 \langle v'v' \rangle^{0.5}/u_B$ in the span-wise midplane with the classical experimental data by Prasad & Koseff (1989) and numerical data by Zang et.al (1993), for the LES of a lid driven cavity ($x:y:z = 1:1:0.5$) at $Re = 10,000$. 
Figure 6.50: Comparison of the scaled mean Reynolds stress tensor components $500 < u'v' > /u_B^2$ in the span-wise mid-plane with the classical experimental data by Prasad & Koseff (1989) and numerical data by Zang et.al (1993), for the LES of a lid driven cavity ($x : y : z = 1 : 1 : 0.5$) at $Re = 10,000$. 
6.2.5 Direct Numerical Simulation of the Taylor-Green Vortex

The Taylor–Green vortex flow problem, one of the benchmark cases in the 3rd International Workshop on high order CFD methods, is chosen in this test case to assess the accuracy and performance of rDG(P₁P₂) for the Direct Numerical Simulation (DNS) of turbulent flows. This problem was originally designed to numerically study the dynamics of turbulence. The initial conditions are smooth, but the flow quickly transits to turbulence with the creation of small scales and begins to decay, mimicking homogeneous non-isotropic turbulence. The initial conditions are given by:

\[
\begin{align*}
    u_1 &= V_0\sin\left(\frac{x}{L}\right)\cos\left(\frac{y}{L}\right)\cos\left(\frac{z}{L}\right), \\
    u_2 &= -V_0\cos\left(\frac{x}{L}\right)\sin\left(\frac{y}{L}\right)\cos\left(\frac{z}{L}\right), \\
    u_3 &= 0, \\
    p &= p_0 + \rho_0 V_0^2 \left[ \frac{1}{16} (\cos\left(\frac{2x}{L}\right) + \cos\left(\frac{2y}{L}\right)) (\cos\left(\frac{2z}{L}\right) + 2) \right],
\end{align*}
\]

(6.7)

where \(\rho_0 = 1\), \(p_0 = 1/\gamma\), and \(u_1\), \(u_2\), and \(u_3\) are the components of the velocity in the \(x\)-, \(y\)- and \(z\)-directions respectively, and \(p\) is the pressure.

The flow is initialized to be isothermal \((p/\rho = p_0/\rho_0 = RT_0)\). To minimize the effects of compressibility, the free-stream Mach number is set to 0.1. The Reynolds number in this case is 1,600, which corresponds to a peak Taylor microscale Reynolds number of about 22. The flow is computed in a periodic and square box, which spans \([-\pi L, \pi L]\) in each coordinate direction. The physical duration of the computation is 20 based on the characteristic convective time defined as \(t_c = L/V_0\), i.e., \(t_{final} = 20t_c\). For this case, in order to guarantee the time accuracy, a lot of time integration schemes have been tested in our work, e.g., TVDRK3, RK4 and ROW.

Before we look into this case for details, some useful concepts should be introduced in advance. The temporal evolution of the kinetic energy integrated on the domain \(\Omega\) is defined,

\[
E_k = \frac{1}{\rho_0 \Omega} \int \rho \frac{\mathbf{v} \cdot \mathbf{v}}{2} d\Omega;
\]

The temporal evolution of the kinetic energy dissipation rate is given by,

\[
\epsilon = - \frac{dE_k}{dt};
\]

The temporal evolution of the enstrophy integrated on the domain \(\Omega\) is defined:

\[
\varepsilon = \frac{1}{\rho_0 \Omega} \int \rho \frac{\mathbf{\omega} \cdot \mathbf{\omega}}{2} d\Omega;
\]
The kinetic energy dissipation rate $e$ in compressible flows is given by the sum of three contributions,

$$e = e_1 + e_2 + e_3 = -\frac{dE_k}{dt}$$

where

$$e_1 = \frac{1}{\Omega} \int 2\mu s_{ij}s_{ij}d\Omega, \quad e_2 = \frac{1}{\Omega} \int 2\mu_v u_{kk}u_{kk}d\Omega, \quad e_3 = -\frac{1}{\Omega} \int p u_{kk}d\Omega,$$

where $s_{ij} = \frac{1}{2}(u_{ij} + u_{ji})$ is the strain-rate tensor. In this case, the gas is assumed to have zero bulk viscosity. Therefore, the dissipation due to the bulk viscosity is always equal to zero, meaning that $e_2 = 0$.

### 6.2.5.1 The grid convergence based on the explicit RK4

A grid convergence study to this problem is performed using the rDG($P_1$/$P_2$) method on 4 hexahedral meshes, which have $40^3$, $81^3$, $161^3$ and $256^3$ elements respectively. Figure 6.51 shows the computed vortex detection criterion $Q_2$ at $t = 8t_c$, on the finest mesh. One can observe that the vortex structure obtained by the rDG($P_1$/$P_2$) solution looks very similar to the one from Reference [25].

Figure 6.52 and Figure 6.53 compare the time history of the kinetic energy $E_k$ and the kinetic energy dissipation rate $e$ computed from the data at the space-time quadrature points, respectively, for all 4 grids, with the result from an incompressible simulation using a spectral code on a mesh of $512^3$ grid points[126]. There remains significant numerical dissipation on the coarse mesh and the point of peak dissipation is poorly captured. The higher the grid resolution is, the more accurate the numerical solutions are. The results from the rDG($P_1$/$P_2$) solution on the finest mesh agree very well with the ones from the spectral code solution.

The three contribution terms to the kinetic energy dissipation rate in compressible flows, i.e., $e_1$, $e_2$, and $e_3$, could also be used to assess the accuracy of the numerical solutions. As mentioned above, in this case, the gas is assumed to have zero bulk viscosity, $\mu_v = 0$. Thus, the dissipation due to the bulk viscosity is always equal to zero, meaning that $e_2 = 0$. Since the flow is nearly incompressible, the dissipation due to the pressure-dilatation term $e_3$ can be expected to be small. Therefore, approximately, $e = e_1$. Time histories of the computed $e$, $e_1$, and $e_3$ on the finest mesh are presented in Figure 6.55. The pressure-dilatation, $e_3$, has a significant bias, contributing some net positive kinetic energy dissipation. With increasing mesh refinement, the biased pressure-dilatation term decreases toward zero as shown in Figure 6.64.

Furthermore, tetrahedral grids are utilised to calculate this case. In our work, due to the simplicity of this computational domain, the tetrahedral grids are generated by splitting one hexahedral into 6 tetrahedrals. Therefore, in our work, the element number of the tetrahedral
Figure 6.51: The isosurfaces of Q2 criterion colored by velocity magnitude at time $t = 8t_c$.

Figure 6.52: Evolution of the dimensionless kinetic energy as a function of the dimensionless time using RK4.
Figure 6.53: Evolution of the dimensionless kinetic energy dissipation rate as a function of the dimensionless time using RK4.

grids is denoted by $6 \times \text{Nelem}^3$, where $\text{Nelem}$ refers to the element number of the original hexahedral grids. The calculation of tetrahedral grids are just used to demonstrate the capability of our code to deal with the unstructured meshes. Two sets of tetrahedral grids are adopted, i.e., $6 \times 22^3$ and $6 \times 44^3$.

Figure 6.57 presents the tetrahedral grid, $6 \times 44^3$. Figure 6.58 and Figure 6.59 show the evolution of kinetic energy $E_k$ and the kinetic energy dissipation rate $e$ using these 2 sets of tetrahedral grids. In order to save the computational cost, the numerical calculation on tetrahedral grids stopped until it arrived at $t = 16$. Also, the comparison of the numerical results, e.g., the kinetic energy $E_k$ and the kinetic energy dissipation rate $e$, between the hexahedral and tetrahedral grids is made in Figure 6.60.

We need to mention that, on the same picture, the number of degrees of freedom of the tetrahedral grid is almost identical to that of the hexahedral grid. It could be observed, with mesh refined, the numerical results agree better with each other for tetrahedral and hexahedral grids.
Figure 6.54: Evolution of the dimensionless enstrophy as a function of the dimensionless time using RK4.

Figure 6.55: Kinetic energy dissipation rate balance for the Taylor-Green vortex problem using RK4.
Figure 6.56: Biased pressure-dilatation term for the Taylor-Green vortex problem using RK4.

Figure 6.57: The unstructured tetrahedral grid, $6 \times 44^3$. 
6.2.5.2 The numerical results with the ROW

From the numerical experiment using explicit RK4, one hexahedral mesh, which have $204^3$ elements with degrees of freedom $324^3$, is used here to test the accuracy and performance of implicit ROW. The grid points are distributed equally in the three directions.

Figure 6.61 and Figure 6.62 compare the time history of the kinetic energy and the kinetic energy dissipation rate computed from the data at the space-time quadrature points, respectively, with the result from an incompressible simulation using a spectral code on a mesh of $512^3$ grid points [126]. Using ROW, the results from the rDG ($P_1P_2$) solution on this mesh agree very well with those from the spectral code solution. Figure 6.63 shows the results for the enstrophy over simulated time. As the figure shows, the enstrophy is more difficult to resolve numerically. The individual terms in the kinetic energy evolution equation can be used to assess the accuracy of the numerical solutions.

Similar to the explicit RK4, also the time histories of the computed $\mathbf{e}$, $\mathbf{e}_1$, and $\mathbf{e}_3$ on the this mesh are presented in Figure 6.65. The high effectiveness of the ROW methods is demonstrated in Table 6.15, where the costs for both the ROW and ESDIRK3 solutions are compared.

Compared to ESDIRK3, ROSI2PW and ROS34PRW require roughly 25% and 23% CPU time less respectively, although ESDIRK3 could use larger time step.
Figure 6.59: Evolution of the dimensionless kinetic energy dissipation rate as a function of the dimensionless time using RK4.

Table 6.15: Comparison of the CPU time (evaluated by running on 240 cores) between the ROW and ESDIRK3 for computing the Taylor-Green vortex at Re=1600.

<table>
<thead>
<tr>
<th>Time integration method</th>
<th>Time-step size</th>
<th>Time steps</th>
<th>CPU time</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESDIRK3</td>
<td>0.08</td>
<td>2500</td>
<td>360250.0</td>
</tr>
<tr>
<td>ROSI2PW</td>
<td>0.04</td>
<td>5000</td>
<td>269998.7</td>
</tr>
<tr>
<td>ROS34PRW</td>
<td>0.04</td>
<td>5000</td>
<td>274921.4</td>
</tr>
</tbody>
</table>

Figure 6.60: Evolution of the dimensionless kinetic energy dissipation rate as a function of the dimensionless time on tetrahedral and hexahedral grids using RK4.
Figure 6.61: Evolution of the dimensionless kinetic energy as a function of the dimensionless time using ROW.

Figure 6.62: Evolution of the dimensionless kinetic energy dissipation rate as a function of the dimensionless time using ROW.
Figure 6.63: Evolution of the dimensionless enstrophy as a function of the dimensionless time using ROW.

Figure 6.64: Biased pressure-dilatation term for the Taylor-Green vortex problem using ROW.
Figure 6.65: Kinetic energy dissipation balance for the Taylor-Green vortex problem.
Chapter 7

Conclusions

7.1 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 3D compressible turbulent flows, e.g., steady problems (RANS-SA), and unsteady problems (ILES and DNS).

For RANS-SA system, a modified one-equation SA turbulence model is coupled with the RANS system to guarantee the stability of high-order discretization for turbulence model equation. The implicit backward Euler method has been used for time advancement in the rDG method. Furthermore, rDG(P₁P₂) using a Hermite WENO reconstruction is used to guarantee the linear stability of the developed rDG method on unstructured grids, e.g., prism and tetrahedron. And it has also been validated that WENO reconstruction could keep the accuracy of the solution for 3D turbulent flows. A number of benchmark test cases based on a set of uniformly refined quadratic curved meshes are presented to assess the performance of the resultant rDG(P₁P₂) method for turbulent flow problems. In this work, two efficient methods, i.e., GMSH library and simple agglomeration, have been used to generate the quadratic curved meshes for high-order rDG(P₁P₂) method. The numerical experiments demonstrate that both methods could provide satisfactory quadratic curved meshes for high-order methods. Especially, simple agglomeration strategy based on linear meshes provide an easy, attractive yet reliable direction to generate the curved meshes. The numerical results, e.g., lift and drag coefficient, match well with the reference solution from FUN3D and CFL3D, demonstrating that the rDG(P₁P₂) method provides an attractive alternative for computing turbulent flows around complex geometries.

For unsteady problems, a linearly implicit Runge-Kutta (IRK) time integration method, namely, Rosenbrock-Wanner method, has been used to complete the time marching. A variety of test cases, ranging from inviscid flows to viscous flows, are presented to assess the perfor-
mance of these schemes. Finally, Rosenbrock-Wanner method has been successfully applied for
the solution of ILES, e.g., "3D lid driven caivity" and DNS, e.g., "3D Taylor-Green Vortex".
Numerical experiments demonstrate that the third-order ROW scheme for the DAEs of index-
2 can not only achieve the designed formal order of temporal convergence accuracy, but also
require significantly less computing time than its ESDIRK3 counterpart to obtain the same
level of discretization errors in our work. This indicates that the ROW method provides an
attractive alternative for the higher-order time-accurate integration of the unsteady compressible
Navier-Stokes equations.

Both steady and unsteady problems finally come to the solution of a resulting linear system
of equations. For rDG(P_1 P_2) method, the Jacobian matrix is evaluated based on the lineariza-
tion of the second-order DG(P_1) right hand side with respect to solution vector. Based on
DG(P_1), Automatic differentiation (AD) has been implemented to obtain the exact Jacobian
matrices without any approximation. 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.

In order to compute large-scale problems for real-world application, our RDGFLO code has
been parallelized using based on domain partitioning and the Message Passing Interface (MPI)
parallel programming environment. The analysis of parallel performance demonstrates that the
parallelization strategy is very promising for large-scale problems.

In summary, the developed parallel implicit rDG(P_1 P_2) method has been assessed and
validated through computing a variety of well-documented 3D turbulent flow benchmark test
cases, to demonstrate its accuracy, robustness and potential for the application to practical
turbulent flows over complex geometries.

7.2 Outlook of Future Work

The future development of the RDGFLO code may be focused on two main directions, i.e.,
application-oriented and research-oriented.

First it is about the application. RDGFLO code just has been verified by some simple
benchmark test cases. It is very necessary to apply RDGFLO for the other complex benchmark
test cases to demonstrate the rDG’s accuracy and robustness, e.g., Transonic turbulent flow
over Onera M6 wing, Transonic turbulent flow around the VFE-2 configuration, Turbulent flow
over DLR-F6 wing-body configuration, etc.

Second it is about the research. As described in our work, for a variety of benchmark
test cases, grid convergence study has been done based on a set of uniformly refined meshes.
Relative to uniform refinement at second order, adjoint-based mesh adaptation combined with
DG method is very promising to yield faster convergence with less degrees of freedom. Therefore,
one main future development will be focused on the adjoint-based $h$-$p$ adaptive rDG method for the compressible RANS-SA equations.
REFERENCES


143


[112] SM Renda, R Hartmann, C De Bartolo, and M Wallraff. A high-order discontinuous
   galerkin method for all-speed flows. *International Journal for Numerical Methods in


[114] Y. Saad and M. H. Schultz. GMRES: A generalized minimal residual algorithm for solving
   nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*,

[115] Dmitri Sharov, Hong Luo, Joseph D Baum, and Rainald Löhner. Implementation of un-
   structured grid gmres+ lu-sgs method on shared-memory, cache-based parallel computers.

[116] Mark S Shephard, Joseph E Flaherty, Kenneth E Jansen, Xiangrong Li, Xiaojuan Luo,
   Nicolas Chevaugeon, Jean-François Remacle, Mark W Beall, and Robert M O’Bara. Adaptive
   mesh generation for curved domains. *Applied Numerical Mathematics*, 52(2):251–271,
   2005.

   for the Euler Equations of Fluid Dynamics edited by F. Angrand, SIAM, Philadelphia*,
   1985.

   52, 1979.

   tropics/tapenade.html*.


[121] E. F. Toro, M. Spruce, and W. Speares. Restoration of the contact surface in the HLL-

[122] Paul G Tucker, Chris L Rumsey, Philippe R Spalart, Robert B Bartels, and Robert T

[123] A. Uranga, P.O. Persson, M. Drela, and J. Peraire. Implicit large eddy simulation of
   transition to turbulence at low reynolds numbers using a discontinuous galerkin method.


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. [38].

Table A.1: Abbreviation and description for various types of elements

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRIA6</td>
<td>6-node quadratic triangle.</td>
</tr>
<tr>
<td>QUAD9</td>
<td>9-node quadratic quadrilateral.</td>
</tr>
<tr>
<td>TETR10</td>
<td>10-node quadratic tetrahedron.</td>
</tr>
<tr>
<td>PYRA13</td>
<td>13-node quadratic pyramid.</td>
</tr>
<tr>
<td>PRIS18</td>
<td>18-node quadratic prism.</td>
</tr>
<tr>
<td>HEXA27</td>
<td>27-node quadratic hexahedron.</td>
</tr>
</tbody>
</table>

A.1 Triangular Elements (two dimensions)

A.1.1 6-Node Curvilinear Triangle

Figure A.1 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.2.
Figure A.1: Representation of the 6-node triangular element.

Table A.2: 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 9-Node Curvilinear Quadrilateral

Figure A.2 shows the 9-node curvilinear quadrilateral in reference space and physical space, respectively. Accordingly, the shape functions and their derivatives are listed in Table A.3.

![Figure A.2: Representation of the 9-node quadrilateral element.](image)

Table A.3: Shape functions for the 9-node quadrilateral and their derivatives

<table>
<thead>
<tr>
<th>Node</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>( \xi \eta(\xi - 1)(\eta - 1)/4 )</td>
<td>( (2\xi - 1)\eta(\eta - 1)/4 )</td>
<td>( \xi(\xi - 1)(2\eta - 1)/4 )</td>
</tr>
<tr>
<td>2</td>
<td>( \xi\eta(\xi + 1)(\eta - 1)/4 )</td>
<td>( (2\xi + 1)\eta(\eta - 1)/4 )</td>
<td>( \xi(\xi + 1)(2\eta - 1)/4 )</td>
</tr>
<tr>
<td>3</td>
<td>( \xi\eta(\xi + 1)(\eta + 1)/4 )</td>
<td>( (2\xi + 1)\eta(\eta + 1)/4 )</td>
<td>( \xi(\xi + 1)(2\eta + 1)/4 )</td>
</tr>
<tr>
<td>4</td>
<td>( \xi\eta(\xi - 1)(\eta + 1)/4 )</td>
<td>( (2\xi - 1)\eta(\eta + 1)/4 )</td>
<td>( \xi(\xi - 1)(2\eta + 1)/4 )</td>
</tr>
<tr>
<td>5</td>
<td>( (1 - \xi^2)\eta(\eta - 1)/2 )</td>
<td>(-\xi\eta(\eta - 1))</td>
<td>( (1 - \xi^2)(2\eta - 1)/2 )</td>
</tr>
<tr>
<td>6</td>
<td>( (1 - \eta^2)\xi(\xi + 1)/2 )</td>
<td>( (2\xi + 1)(1 - \eta^2)/2 )</td>
<td>(-\xi\eta(1 + \xi))</td>
</tr>
<tr>
<td>7</td>
<td>( (1 - \xi^2)\eta(\eta + 1)/2 )</td>
<td>(-\xi\eta(1 + \eta))</td>
<td>( (1 - \xi^2)(2\eta + 1)/2 )</td>
</tr>
<tr>
<td>8</td>
<td>( (1 - \eta^2)\xi(\xi - 1)/2 )</td>
<td>( (2\xi - 1)(1 - \eta^2)/2 )</td>
<td>(-\xi\eta(\xi - 1))</td>
</tr>
<tr>
<td>9</td>
<td>( (1 - \eta^2)(1 - \xi^2) )</td>
<td>(-2\xi(1 - \eta^2))</td>
<td>(-2\eta(1 - \xi^2))</td>
</tr>
</tbody>
</table>
A.3 Tetrahedral Elements

A.3.1 10-node Curvilinear Tetrahedron

Figure A.3 shows the 10-node curvilinear tetrahedron in reference space and physical space, respectively. The coordinates of the nodes in the reference space is given in Table A.4.

Table A.4: The coordinates of nodes in reference space for 10-node Curvilinear Tetrahedron

<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>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\xi_i$</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>1.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td>$\eta_i$</td>
<td>1.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.5</td>
<td>0.0</td>
<td>0.5</td>
<td>0.5</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>$\zeta_i$</td>
<td>0.0</td>
<td>1.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.5</td>
<td>0.5</td>
<td>0.0</td>
<td>0.0</td>
<td>0.5</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Accordingly, the shape functions and their derivatives are listed in Table A.5.

(a) TETR10 in reference space  (b) TETR10 in physical space

Figure A.3: Representation of the 10-node tetrahedral element.
Table A.5: 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 27-Node Curvilinear Hexahedron

Figure A.4 shows the 27-node curvilinear hexahedron in reference space and physical space, respectively. The coordinates of the nodes in the reference space is given in Table A.6.

(a) HEXA27 in reference space  
(b) HEXA27 in physical space

Figure A.4: Representation of the 27-node hexahedral element.
Table A.6: The coordinates of nodes in reference space for 27-node Curvilinear Hexahedron

<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>-1</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>

The shape functions of 27-node hexahedron are

\[
\begin{align*}
\phi_1(\xi, \eta, \zeta) &= \frac{1}{8} \xi (\xi + 1) \eta (\eta - 1) \zeta (\zeta - 1) \\
\phi_2(\xi, \eta, \zeta) &= \frac{1}{8} \xi (\xi + 1) \eta (\eta + 1) \zeta (\zeta - 1) \\
\phi_3(\xi, \eta, \zeta) &= \frac{1}{8} \xi (\xi - 1) \eta (\eta + 1) \zeta (\zeta - 1) \\
\phi_4(\xi, \eta, \zeta) &= \frac{1}{8} \xi (\xi - 1) \eta (\eta - 1) \zeta (\zeta - 1) \\
\phi_5(\xi, \eta, \zeta) &= \frac{1}{8} \xi (\xi + 1) \eta (\eta - 1) \zeta (\zeta + 1) \\
\phi_6(\xi, \eta, \zeta) &= \frac{1}{8} \xi (\xi + 1) \eta (\eta + 1) \zeta (\zeta + 1) \\
\phi_7(\xi, \eta, \zeta) &= \frac{1}{8} \xi (\xi - 1) \eta (\eta + 1) \zeta (\zeta + 1) \\
\phi_8(\xi, \eta, \zeta) &= \frac{1}{8} \xi (\xi - 1) \eta (\eta - 1) \zeta (\zeta + 1) \\
\phi_9(\xi, \eta, \zeta) &= \frac{1}{8} \xi (1 + \xi)(1 - \eta^2) \zeta (\zeta - 1) \\
\phi_{10}(\xi, \eta, \zeta) &= \frac{1}{4}(1 - \xi^2) \eta (\eta + 1) \zeta (\zeta - 1) \\
\phi_{11}(\xi, \eta, \zeta) &= \frac{1}{4}(1 - \xi^2) \eta (\eta - 1) \zeta (\zeta - 1) \\
\phi_{12}(\xi, \eta, \zeta) &= \frac{1}{4}(1 - \xi^2) \eta (\eta - 1) \zeta (\zeta + 1) \\
\phi_{27}(\xi, \eta, \zeta) &= (1 - \xi^2)(1 - \eta^2)(1 - \zeta^2)
\end{align*}
\]

(A.1)
A.5 Prismatic Elements

A.5.1 18-node Prism

Figure A.5 shows the 15-node prism in reference space and physical space, respectively. The coordinates of the nodes in the reference space is given in Table A.7. Accordingly, the shape functions are listed as follows,

Table A.7: The coordinates of nodes in reference space for 18-node Curvilinear Prism

<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>-1.0</td>
<td>-1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>-1.0</td>
<td>-1.0</td>
<td></td>
</tr>
<tr>
<td>$\eta_i$</td>
<td>1.0</td>
<td>0.0</td>
<td>0.0</td>
<td>1.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.5</td>
<td>0.0</td>
</tr>
<tr>
<td>$\zeta_i$</td>
<td>0.0</td>
<td>1.0</td>
<td>0.0</td>
<td>0.0</td>
<td>1.0</td>
<td>0.0</td>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td>( \xi_i )</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>-1.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>( \eta_i )</td>
<td>0.5</td>
<td>1.0</td>
<td>0.0</td>
<td>0</td>
<td>0.5</td>
<td>0.0</td>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td>( \zeta_i )</td>
<td>0.0</td>
<td>0.0</td>
<td>1.0</td>
<td>0</td>
<td>0.5</td>
<td>0.5</td>
<td>0.0</td>
<td>0.5</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
\phi_1(\xi, \eta, \zeta) &= \frac{1}{2} \xi \eta (\xi - 1)(2\eta - 1) \\
\phi_2(\xi, \eta, \zeta) &= \frac{1}{2} \xi \zeta (\xi - 1)(2\zeta - 1) \\
\phi_3(\xi, \eta, \zeta) &= \frac{1}{2} \xi (\xi - 1)(\zeta + \eta - 1)(2\zeta + 2\eta - 1) \\
\phi_4(\xi, \eta, \zeta) &= \frac{1}{2} \xi \eta (\xi + 1)(2\eta - 1) \\
\phi_5(\xi, \eta, \zeta) &= \frac{1}{2} \xi \zeta (\xi + 1)(2\zeta - 1) \\
\phi_6(\xi, \eta, \zeta) &= \frac{1}{2} \xi (\xi + 1)(\zeta + \eta - 1)(2\zeta + 2\eta - 1) \\
\phi_7(\xi, \eta, \zeta) &= 2\xi \eta \zeta (\xi - 1) \\
\phi_8(\xi, \eta, \zeta) &= -2\xi \zeta (\xi - 1)(\eta + \zeta - 1) \\
\phi_9(\xi, \eta, \zeta) &= -2\xi \eta (\xi - 1)(\eta + \zeta - 1) \\
\phi_{10}(\xi, \eta, \zeta) &= \eta(1 - \xi^2)(2\eta - 1) \\
\phi_{11}(\xi, \eta, \zeta) &= \zeta(1 - \xi^2)(2\zeta - 1) \\
\phi_{12}(\xi, \eta, \zeta) &= (1 - \xi^2)(\eta + \zeta - 1)(2\zeta + 2\eta - 1) \\
\phi_{13}(\xi, \eta, \zeta) &= 2\xi \eta \zeta (\xi + 1) \\
\phi_{14}(\xi, \eta, \zeta) &= -2\xi \zeta (\xi + 1)(\zeta + \eta - 1) \\
\phi_{15}(\xi, \eta, \zeta) &= -2\xi \eta (\xi + 1)(\zeta + \eta - 1) \\
\phi_{16}(\xi, \eta, \zeta) &= 4\eta \zeta (1 - \xi^2) \\
\phi_{17}(\xi, \eta, \zeta) &= 4\zeta (\xi^2 - 1)(\eta + \zeta - 1) \\
\phi_{18}(\xi, \eta, \zeta) &= 4\eta (\xi^2 - 1)(\eta + \zeta - 1)
\end{align*}
\]
Appendix B

Tables of coefficients of the
Rosenbrock schemes considered in
this thesis

In this section, the related coefficients for Rosenbrock methods used in this thesis have been presented as follows [109, 107].

Table B.1: Set of coefficients for ROSI2PW.

<table>
<thead>
<tr>
<th>Coefficients</th>
<th>Value</th>
<th>Coefficients</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\gamma_{21}$</td>
<td>$4.3586652150845900\times10^{-1}$</td>
<td>$\gamma_{21}$</td>
<td>$8.7173304301691801\times10^{-1}$</td>
</tr>
<tr>
<td>$\alpha_{21}$</td>
<td>$8.7173304301691801\times10^{-1}$</td>
<td>$\gamma_{31}$</td>
<td>$-8.7173304301691801\times10^{-1}$</td>
</tr>
<tr>
<td>$\alpha_{31}$</td>
<td>$-7.9937335839852708\times10^{-1}$</td>
<td>$\gamma_{31}$</td>
<td>$3.0647867418622479\times10^{0}$</td>
</tr>
<tr>
<td>$\alpha_{32}$</td>
<td>$-7.9937335839852708\times10^{-1}$</td>
<td>$\gamma_{32}$</td>
<td>$3.0647867418622479\times10^{0}$</td>
</tr>
<tr>
<td>$\alpha_{41}$</td>
<td>$7.0849664917601007\times10^{-1}$</td>
<td>$\gamma_{41}$</td>
<td>$-1.0424832458800504\times10^{-1}$</td>
</tr>
<tr>
<td>$\alpha_{42}$</td>
<td>$3.1746327955312481\times10^{-1}$</td>
<td>$\gamma_{42}$</td>
<td>$-3.1746327955312481\times10^{-1}$</td>
</tr>
<tr>
<td>$\alpha_{43}$</td>
<td>$-2.5959928729134892\times10^{-2}$</td>
<td>$\gamma_{43}$</td>
<td>$-1.4154917367329144\times10^{-2}$</td>
</tr>
<tr>
<td>$b_{1}$</td>
<td>$6.0424832458800504\times10^{-1}$</td>
<td>$b_{2}$</td>
<td>$-3.6210810811598324\times10^{-32}$</td>
</tr>
<tr>
<td>$b_{3}$</td>
<td>$-4.0114846096464034\times10^{-2}$</td>
<td>$b_{4}$</td>
<td>$4.3586652150845900\times10^{-1}$</td>
</tr>
</tbody>
</table>
Table B.2: Set of coefficients for ROS34PRW.

<table>
<thead>
<tr>
<th>( \gamma )</th>
<th>( \alpha_{21} )</th>
<th>( \alpha_{31} )</th>
<th>( \alpha_{32} )</th>
<th>( \alpha_{41} )</th>
<th>( \alpha_{42} )</th>
<th>( \alpha_{43} )</th>
<th>( b_1 )</th>
<th>( b_2 )</th>
<th>( b_3 )</th>
<th>( b_4 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \gamma = 4.3586652150845900e-01 )</td>
<td>( \alpha_{21} = 8.7173304301691801e-01 )</td>
<td>( \alpha_{31} = 1.4722022879435914e+00 )</td>
<td>( \alpha_{32} = -3.1840250568090289e-01 )</td>
<td>( \alpha_{41} = 8.1505192016694938e-01 )</td>
<td>( \alpha_{42} = 5.0000000000000000e-01 )</td>
<td>( \alpha_{43} = -3.1505192016694938e-01 )</td>
<td>( b_1 = 3.3303742833830591e-01 )</td>
<td>( b_2 = 7.1793326075422947e-01 )</td>
<td>( b_3 = -4.8683721060099439e-01 )</td>
<td>( b_4 = 4.3586652150845900e-01 )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( \gamma_1 )</th>
<th>( \gamma_2 )</th>
<th>( \gamma_3 )</th>
<th>( \gamma_4 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \gamma_1 = -8.7173304301691801e-01 )</td>
<td>( \gamma_2 = -8.7173304301691801e-01 )</td>
<td>( \gamma_3 = -1.2855347382089872e+00 )</td>
<td>( \gamma_4 = -4.8201449182864348e-01 )</td>
</tr>
</tbody>
</table>

Table B.3: Set of coefficients for ROS34PW2.

<table>
<thead>
<tr>
<th>( \gamma )</th>
<th>( \alpha_{21} )</th>
<th>( \alpha_{31} )</th>
<th>( \alpha_{32} )</th>
<th>( \alpha_{41} )</th>
<th>( \alpha_{42} )</th>
<th>( \alpha_{43} )</th>
<th>( b_1 )</th>
<th>( b_2 )</th>
<th>( b_3 )</th>
<th>( b_4 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \gamma = 4.3586652150845900e-01 )</td>
<td>( \alpha_{21} = 8.7173304301691801e-01 )</td>
<td>( \alpha_{31} = 8.4457060015369423e-01 )</td>
<td>( \alpha_{32} = -1.1299064236484185e-01 )</td>
<td>( \alpha_{41} = 0.0000000000000000e+00 )</td>
<td>( \alpha_{42} = 0.0000000000000000e+00 )</td>
<td>( \alpha_{43} = 1.0000000000000000e+00 )</td>
<td>( b_1 = 2.4212380706095346e-01 )</td>
<td>( b_2 = -1.223250589045147e+00 )</td>
<td>( b_3 = 1.5452602553351020e+00 )</td>
<td>( b_4 = 4.3586652150845900e-01 )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( \gamma_1 )</th>
<th>( \gamma_2 )</th>
<th>( \gamma_3 )</th>
<th>( \gamma_4 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \gamma_1 = -8.7173304301691801e-01 )</td>
<td>( \gamma_2 = 5.0507005541550687e-01 )</td>
<td>( \gamma_3 = 5.4180672388095326e-02 )</td>
<td>( \gamma_4 = 2.4212380706095346e-01 )</td>
</tr>
<tr>
<td>( \gamma_1 = -9.0338057013044082e-01 )</td>
<td>( \gamma_2 = 2.1793326075422950e-01 )</td>
<td>( \gamma_4 = -1.223250589045147e+00 )</td>
<td>( \gamma_4 = 5.4526025533510214e-01 )</td>
</tr>
</tbody>
</table>