PANDARE, ADITYA KIRAN. A Hybrid Reconstructed Discontinuous Galerkin and Continuous Galerkin finite element method for Incompressible Flows on Unstructured Grids. (Under the direction of Hong Luo.)

A hybrid reconstructed discontinuous Galerkin (rDG) and continuous Galerkin (CG), or simply \( rDG(P_n P_m) - CG(P_n) \) method, based on an incremental pressure-correction formulation is developed for solving the unsteady incompressible Navier-Stokes equations on unstructured grids. In the \( rDG(P_n P_m) - CG(P_n) \) method, an rDG finite element method is used for the discretization of the velocity, while the pressure is discretized using a CG finite element method. This \( rDG(P_n P_m) - CG(P_n) \) method is designed to increase the accuracy of the hybrid \( DG(P_n) - CG(P_n) \) method and yet still satisfy Ladyženskaya-Babuška-Brezzi (LBB) condition, thus avoiding the pressure checkerboard instability. A number of incompressible flow problems for a variety of flow conditions are computed to numerically assess the spatial order of convergence of the hybrid \( rDG(P_n P_m) - CG(P_n) \) method. The numerical experiments indicate that both \( rDG(P_0 P_1) - CG(P_1) \) and \( rDG(P_1 P_2) - CG(P_1) \) methods can attain the designed second order and third order of accuracy for velocity, respectively and the third order \( rDG(P_1 P_2) - CG(P_1) \) method outperforms its second order \( rDG(P_0 P_1) - CG(P_1) \) and \( rDG(P_1 P_1) - CG(P_1) \) counterparts: being able to not only significantly increase the accuracy of the velocity but also improve the accuracy of the pressure.
© Copyright 2015 by Aditya Kiran Pandare

All Rights Reserved
A Hybrid Reconstructed Discontinuous Galerkin and Continuous Galerkin finite element method for Incompressible Flows on Unstructured Grids

by
Aditya Kiran Pandare

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

Aerospace Engineering

Raleigh, North Carolina

2015

APPROVED BY:

Jack Edwards Jr.             Zhilin Li

Hong Luo
Chair of Advisory Committee
DEDICATION

To my parents.
BIOGRAPHY

The author was born on the 16th of August, 1990 in Pune, in the state of Maharashtra, India. He received his Bachelor’s degree in Mechanical Engineering from the University of Pune in May 2012. The author went on to study Aerospace Engineering in North Carolina State University (NCSU), Raleigh, North Carolina, USA in Fall 2013. Here he began working in the Computational Fluid Dynamics laboratory from January 2014. His academic and research advisor is Dr. Hong Luo.
ACKNOWLEDGEMENTS

I would like to thank my advisor Dr. Hong Luo for his great support and help with my research work in the past two years at North Carolina State University. I would also like to thank my committee members Dr. Jack R. Edwards from the Department of Mechanical and Aerospace Engineering, and Dr. Zhilin Li from the Department of Mathematics for their valuable suggestions. I would like to thank my group member Dr. Lijun Xuan in the CFD Laboratory for assistance and encouragement with my research work. I would also like to thank all my other group members for their help and support during my studies. Finally, I would like thank my family and friends for their love, support and blessings.
# TABLE OF CONTENTS

**LIST OF TABLES** .......................................................... vii

**LIST OF FIGURES** ...................................................... viii

**Chapter 1 Introduction** .............................................. 1

**Chapter 2 Projection method** ....................................... 5
  2.1 Governing equations .............................................. 5
  2.2 Numerical method to solve the incompressible NS equations .... 6
  2.3 The incremental projection method ............................. 6
  2.4 Boundary conditions ............................................ 7
  2.5 Algorithm ..................................................... 8

**Chapter 3 Temporal discretization** ............................... 10
  3.1 Explicit Euler forward difference ............................ 10
  3.2 Implicit Euler backward difference (BDF) ................... 11

**Chapter 4 Spatial discretization** ................................ 12
  4.1 Velocity ....................................................... 13
    4.1.1 Reconstruction to P2 .................................... 16
    4.1.2 Least-squares reconstruction ............................ 16
    4.1.3 P1 Mass matrix .......................................... 17
    4.1.4 P2 Mass matrix .......................................... 18
  4.2 Pressure ...................................................... 20
    4.2.1 Stiffness matrix for cG(P1) ............................. 21
    4.2.2 Stiffness matrix for cG(P2) ............................. 22
  4.3 Boundary conditions .......................................... 23

**Chapter 5 Fluxes** .................................................... 25
  5.1 Advective fluxes .............................................. 25
  5.2 Diffusive fluxes .............................................. 26
    5.2.1 Modified gradient method ............................... 26
    5.2.2 Direct discontinuous Galerkin method .................. 27
    5.2.3 Bassi-Rebay 2 method .................................. 28

**Chapter 6 Forming the Jacobian matrix** ........................ 31
  6.1 Structure of the Jacobian matrix ............................ 31
  6.2 Analytical differentiation .............................. 32
    6.2.1 Interface integral ...................................... 32
    6.2.2 Domain integral ...................................... 33
  6.3 Numerical finite difference ................................ 33

**Chapter 7 Linear system solvers** ................................ 35
LIST OF TABLES

Table 8.1  Total number of DOF’s for DG methods of different orders for 2D flows . . . . 40
Table 8.2  Errors for the Kovasznay flow test case . . . . . . . . . . . . . . . . . . . . . 41
Table 8.3  Errors for the Taylor vortex test case with $\nu = 0.01$ . . . . . . . . . . . . . . . 46
Table 8.4  Errors for the Taylor vortex test case with $\nu = 0.1$ . . . . . . . . . . . . . . . 46

Table A.1  Constants for Gaussian quadrature for a line . . . . . . . . . . . . . . . . . . . . . 64
Table A.2  Constants for Gaussian quadrature for a triangle . . . . . . . . . . . . . . . . . . . . 65
LIST OF FIGURES

Figure 4.1 Storage locations in the hybrid discretization ........................................ 12
Figure 4.2 Representation of DG polynomial solutions ............................................ 14
Figure 4.3 A P2 triangular element ........................................................................... 22
Figure 4.4 Ghost-state and ghost-cell ...................................................................... 24
Figure 5.1 Interface fluxes ....................................................................................... 26
Figure 8.1 Velocity magnitude and streamlines for the Kovasznay flow test case .... 41
Figure 8.2 Spatial convergence of velocity for Kovasznay flow test case ................. 42
Figure 8.3 Spatial convergence of pressure for Kovasznay flow test case ............... 42
Figure 8.4 Errors in velocity for Kovasznay flow plotted vs. DOF’s ....................... 43
Figure 8.5 Spatial convergence of velocity for Taylor vortex with $\nu = 0.01$ and $\nu = 0.1$ 45
Figure 8.6 Spatial convergence of pressure for Taylor vortex with $\nu = 0.01$ and $\nu = 0.1$ 45
Figure 8.7 Errors in velocity for Taylor vortex with $\nu = 0.01$ plotted vs. DOF’s .... 47
Figure 8.8 Velocity and Pressure contours, $\nu = 0.01$ ............................................ 47
Figure 8.9 Spatial convergence for velocity for forced-flow flow test case .......... 48
Figure 8.10 Spatial convergence for pressure for forced-flow flow test case .......... 49
Figure 8.11 Velocity contours in the boundary layer .............................................. 50
Figure 8.12 Coefficient of friction compared to Blasius’s solution for flat plate .... 51
Figure 8.13 Coefficient of friction compared to Blasius’s solution on a section of the plate 51
Figure 8.14 X-velocity at $x = 0.5$ ........................................................................ 52
Figure 8.15 Y-velocity at $x = 0.5$ ........................................................................ 52
Figure 8.16 X-velocity at $x = 0.8$ ........................................................................ 52
Figure 8.17 Y-velocity at $x = 0.8$ ........................................................................ 52
Figure 8.18 X-velocity at $x = 1.0$ ........................................................................ 52
Figure 8.19 Y-velocity at $x = 1.0$ ........................................................................ 52
Figure 8.20 Streamlines for 2D driven cavity flow .................................................. 53
Figure 8.21 Streamlines for 2D driven cavity flow .................................................. 54
Figure 8.22 Velocity profiles compared to Ghia et al. [16] ........................................ 54
Figure 8.23 Velocity profiles compared to Ghia et al. [16] ........................................ 55
Figure 8.24 Velocity profiles compared to Ghia et al. [16] ........................................ 55
Figure B.1 Information storage in edge-based data-structure .................................. 67
Figure B.2 A mesh for edge-based structure equivalence ....................................... 68
Figure C.1 Coarse mesh with 554 elements ........................................................... 69
Figure C.2 Fine mesh with 2216 elements .............................................................. 70
Figure C.3 Finest mesh with 8864 elements ............................................................ 71
Figure C.4 Mesh used to solve the flat plate problem ........................................... 72
Chapter 1

Introduction

The discontinuous Galerkin methods [4,9,10,13–15,19,29,32,40] (DGM) have recently become popular for the solution of systems of conservation laws, leading to applications in computational fluid dynamics, computational acoustics and magneto-hydrodynamics. The discontinuous Galerkin methods combine two advantageous features commonly associated with finite element and finite volume methods. As in classical finite element methods, accuracy is obtained by means of high-order polynomial approximation within an element rather than by usage of wider stencils as in the case of finite volume methods. The physics of wave propagation is, however, accounted for by solving the Riemann problems that arise from the discontinuous representation of the solution at element interfaces, which make them similar to finite volume methods. The discontinuous Galerkin methods have many attractive features:

- Their mathematical rigor implies useful mathematical properties with respect to conservation, stability and convergence
- The methods can be easily extended to higher-order (> 2nd) approximations
- They are well suited for complex geometries since they can be applied on unstructured grids. In addition, the methods can also handle non-conforming elements, where the grids are allowed to have hanging nodes
- The methods are highly parallelizable, as they are compact and each element is independent
- Since the elements are discontinuous, and the inter-element communications are minimal, domain decomposition can be efficiently employed. The compactness also allows for structured and simplified implementation and coding
• They can easily handle adaptive strategies, since refining or coarsening a grid can be achieved without considering the continuity restriction commonly associated with the conforming elements.

• The methods allow easy implementation of $hp$-refinement, for example, the order of accuracy, or shape, can vary from element to element. $p$-refinement can be achieved by simply increasing the order of the approximation polynomial.

However, the DGM have a number of weaknesses that have yet to be addressed, before they can be robustly used for flow problems of practical interest in a complex configuration environment. In particular, how to effectively control spurious oscillations in the presence of strong discontinuities, how to reduce the computing costs for the DGM, and how to efficiently discretize diffusion terms required for the Navier-Stokes equations remain the three most challenging and unresolved issues in the DGM. Indeed, compared to the finite element methods and finite volume methods, the DGM require solutions of systems of equations with more unknowns for the same grids. Consequently, these methods have been recognized as expensive in terms of both computational costs and storage requirements especially in the context of implicit methods, where the memory requirement for the Jacobian matrix grows quadratically with the order of the DG methods, thus leading to a significant increase in computational cost.

In order to reduce high costs associated with the DGM, Dumbser et al. [13–15] have introduced a new family of reconstructed DGM for solving compressible Euler equations, termed PnPm schemes and referred to as rDG(PnPm) in this paper, where Pn indicates that a piecewise polynomial of degree of n is used to represent a DG solution, and Pm represents a reconstructed polynomial solution of degree of m ($m \geq n$) that is used to compute the fluxes. This rDG(PnPm) method is used to discretize the velocity field. These schemes are designed to enhance the accuracy of the discontinuous Galerkin method by increasing the order of the underlying polynomial solution. The beauty of rDG(PnPm) schemes is that they provide a unified formulation for both finite volume and DGM, and contain both classical finite volume and standard DG methods as two special cases of rDG(PnPm) schemes, and thus allow for a direct efficiency comparison. When $n = 0$, i.e. a piecewise constant polynomial is used to represent a numerical solution, rDG(P0Pm) is nothing but classical high order finite volume schemes, 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 rDG(PnPn) scheme yields a standard DG method.

Obviously, the construction of an accurate and efficient reconstruction operator is crucial to the success of the rDG(PnPm) schemes. In Dumbser’s work, a higher order polynomial solution is reconstructed using a L2 projection, requiring it indistinguishable from the underlying DG solutions in the contributing cells in the weak sense. The resultant over-determined system is
then solved using a least-squares method that guarantees exact conservation, not only of the cell averages but also of all higher order moments in the reconstructed cell itself, such as slopes and curvatures. However, this conservative least-squares reconstruction approach is computationally expensive, as the L2 projection, i.e., the operation of integration, is required to obtain the resulting over-determined system. Furthermore, the reconstruction might be problematic for a boundary cell, where the number of the face-neighboring cells might not be enough to provide the necessary information to recover a polynomial solution of a desired order. Fortunately, the projection-based reconstruction is not the only way to obtain a polynomial solution of higher order from the underlying discontinuous Galerkin solutions. In a reconstructed DG method using a Taylor basis [29, 30] developed by Luo et al. for the solution of the compressible Euler and Navier-Stokes equations on arbitrary grids, a higher order polynomial solution is reconstructed by use of a strong interpolation, requiring point values and derivatives to be interpolated on the face-neighboring cells. The resulting over-determined linear system of equations is then solved in the least-squares sense. This reconstruction scheme only involves 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, Luo et al. [31, 33] have conducted a comparative study for these three reconstructed discontinuous Galerkin methods rDG(P1P2) by solving the compressible Euler equations on arbitrary grids. It is found that all three reconstructed discontinuous Galerkin methods can deliver the desired third order of 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.

The major difficulty in numerically solving the incompressible Navier-Stokes equations arises from the pressure-velocity coupling by the incompressibility (divergence-free) constraint. Chorin observed that the pressure does not carry any thermodynamic meaning for incompressible flows, due to the fact that there is no equation of state for an incompressible fluid. Pressure plays the role of a Lagrange multiplier which enforces the incompressibility constraint. This was followed by the development of an operator-splitting scheme, called as the projection or fractional-step method [7,8,37]. This class of methods are characterized by a two-step predictor-corrector approach, resulting in a decoupling of the velocity and pressure. In the first step, an intermediate velocity is computed from the momentum equations, using an approximation for the pressure gradient and ignoring the incompressibility constraint. In the second step, the Helmholtz-Hodge decomposition is used to project the velocity into a space of divergence-free vector fields, to obtain the pressure and a correction to the intermediate velocity that makes it satisfy the incompressibility constraint. This decoupling makes the projection methods more efficient than the coupled monolithic solvers, thus attracting a lot of attention in the past few
decades. As a result, many modifications and improvements have been made in the projection method [6,17,20,26,38].

Several DGM [2, 3, 5, 23] have been developed for numerically solving the incompressible Navier-Stokes equations. Bassi et al. [2,3] presented a DG method for the incompressible Navier-Stokes equations written in conservative form based on an artificial compressibility formulation. Botti, Di Pietro [5] and Kyriazis, Ekaterinaris [23] adopted the pressure-correction formulation [5–7, 18, 26, 39] to solve the incompressible Navier-Stokes equations. They use a continuous Galerkin CG(Pn) and CG(Pn-1) discretizations for the pressure field and DG(Pn) discretization for the velocity field in order to satisfy LBB condition. They report that the velocity converges with the order of n+1 and the pressure at a convergence rate of n for both DG(Pn)+CG(Pn) and DG(Pn)+CG(Pn-1) spatial discretization. In addition, they observe a 2nd order of convergence in time for a range of Re (10^2, 10^3, 10^4).

Based on the success of the rDG method for solving the compressible Euler and Navier-Stokes equations, the objective of this work is to develop an rDG method for solving the incompressible Euler and Navier-Stokes equations. A new pressure-correction method is thus developed for the incompressible flows using a hybrid formulation, where a reconstructed discontinuous Galerkin approximation (rDG(PnPm)) is used for the velocity field and a continuous Galerkin approximation (CG(Pm)) is used to discretize the pressure field. The developed rDG(PnPm)+CG(Pm) method inherently satisfies the so-called Ladyženskaja-Babuška-Brezzi (LBB) condition and thus can effectively avoid the pressure checkerboard instability. In order to suppress spurious oscillations in the velocity field for the inviscid or high Reynolds number viscous flows, the nonlinear convective fluxes in the momentum equations are computed using an upwind method. The developed projection method is used to compute a variety of incompressible flow problems on unstructured grids. The numerical experiments indicate that both rDG(P0P1)+CG(P1) and rDG(P1P2)+CG(P1) methods can attain the designed second order and third order of accuracy, respectively and that the third order rDG(P1P2)+CG(P1) method outperforms the second order rDG(P0P1)+CG(P1) method. The description of this newly developed rDG(PnPm)+CG(Pm) method for solving the incompressible Euler and Navier-Stokes equations and numerical results for computing a variety of incompressible flow problems on unstructured grids are presented in the following sections.
Chapter 2

Projection method

2.1 Governing equations

The incompressible Navier-Stokes equations are,

\[
\frac{\partial \mathbf{v}}{\partial t} + \nabla \cdot (\mathbf{v} \mathbf{v}) - \nu \nabla^2 \mathbf{v} + \frac{1}{\rho} \nabla p = \mathbf{f}, \quad \text{in } \Omega \times (0, T_f) \quad (2.1)
\]

\[
\nabla \cdot \mathbf{v} = 0, \quad \text{in } \Omega \times (0, T_f) \quad (2.2)
\]

with initial and boundary conditions,

\[
\mathbf{v}(\mathbf{x}, 0) = \mathbf{v}_0, \quad \text{in } \Omega \quad (2.3)
\]

\[
\mathbf{v} = \mathbf{v}_b, \quad \text{on } \Gamma_D \quad (2.4)
\]

\[
\frac{\partial \mathbf{v}}{\partial \mathbf{n}} = 0, \quad \text{on } \Gamma_N \quad (2.5)
\]

where, \(\Gamma = \partial \Omega\) is the boundary of the domain \(\Omega\); and subscripts \(D\) and \(N\) respresent Dirichlet and Neumann boundary conditions respectively. The solution of the incompressible Navier-Stokes equations is a challenge, due to the incompressibility constraint (divergence-free velocity field) which couples the velocity and pressure. As noted earlier, in the 1960’s Chorin observed that the pressure isn’t a thermodynamic variable since an equation of state doesn’t exist for an incompressible fluid. It is basically an implicit variable that adjusts its gradient (a Lagrange multiplier) so that the velocity stays divergence-free. This led to the development of operator-splitting approaches, called fractional-step or projection methods. The projection methods are more efficient than the coupled monolithic solvers due to their decoupled algorithm, thus making them one of the most famous methods for the solution of unsteady incompressible Navier-Stokes equations. We now briefly discuss the projection methods.
2.2 Numerical method to solve the incompressible NS equations

Out of the many methods exist to solve the incompressible Navier-Stokes equations, viz. monolithic solvers, artificial compressibility and so on, one of the most successful and widely used approach is the class of methods called the projection methods. Originally developed by Chorin [7] and Temam [37], these methods decouple the incompressibility constraint and the momentum equation, leading to a step-wise procedure to obtain the velocity and pressure at each time-step. The decoupling leads to a convection-diffusion equation for the velocity; and an elliptic (Poisson) equation for the pressure (or a related quantity). The projection methods are essentially of two types: fractional-step methods; and pressure-correction methods.

In the fractional-step methods, the pressure (incompressibility constraint) and diffusion are fully split in different substeps. As opposed to this, the pressure-correction methods involve a predictor-corrector procedure for the velocity. Using an approximate pressure field in the momentum equation, a prediction of a “non divergence-free” velocity is made, which satisfies the momentum equations. This is followed by the solution of a pressure-Poisson equation obtained by applying the incompressibility constraint to the momentum equation. The predicted velocity is then corrected using the gradient of the solution of this Poisson equation. This way, the final velocity is guaranteed to satisfy the incompressibility condition.

2.3 The incremental projection method

The projection method originally proposed by Chorin [7] and Temam [37] has undergone numerous improvements. We consider the projection method by Temam [37] and further modified by Timmermans in [38] in this work, with a few modifications from [5]. Consider the momentum equations (2.1). Using an approximate value of pressure (viz. a time-lagged value), say q, and considering a simple Euler backward difference formula for the time derivative, we get,

\[
\frac{v^* - v^n}{\Delta t} + \nabla \cdot (v v) - \nu \nabla^2 v + \frac{1}{\rho} \nabla q = f \tag{2.6}
\]

where \(v^*\) is the intermediate non divergence-free velocity. Subtracting (2.6) from (2.1), we get the equation for the velocity correction,

\[
\frac{v^{n+1} - v^n}{\Delta t} + \frac{1}{\rho} \nabla (p - q) = 0 \tag{2.7}
\]

\[
v^{n+1} = v^n - \frac{\Delta t}{\rho} \nabla (p - q) \tag{2.8}
\]
Here, note that \( p \) is the physical pressure at the current time-step and \( q \) is an approximation to it. There are various ways to treat this pressure difference. One of the most common ways is to assume \( q = 0 \) in the momentum equation. This gives rise to the non-incremental form of the pressure-correction methods. Another way is to approximate the \( q \) by some form of pressure, a time-lagged pressure or an extrapolated value of the pressure \([6, 26]\), known as the incremental form of the pressure-correction. We use the incremental method in this work. In doing so, we denote \( \phi = p - q \). Further, we take the divergence of (2.7). Noting that \( u^{n+1} \) is the actual physical velocity, and that it satisfies the divergence-free condition, gives us the \( \phi \)-Poisson equation,

\[
\nabla^2 \phi = \frac{\rho}{\Delta t} \nabla \cdot \mathbf{v}^* \tag{2.9}
\]

The relation between the physical divergence-free velocity and intermediate velocity can be summed up by considering a projection operator in the space of divergence-free functions. This relation is the classical Helmholtz-Hodge decomposition of the intermediate velocity,

\[
\mathbf{v}^* = \mathbf{v}^{n+1} + \frac{\Delta t}{\rho} \nabla (p^{n+1} - p^n) \tag{2.10}
\]

This multi-step algorithm leads to a strategy, which requires to solve a convection-diffusion equation for the velocity; and an elliptic (Poisson) equation for the pressure-correction. These, followed by an appropriate pressure update,

\[
p^{n+1} = q + \phi \tag{2.11}
\]

and velocity correction (2.7), constitute one time-step of the incremental projection method. This method obviates the need of a coupled monolithic solver.

### 2.4 Boundary conditions

The boundary conditions of the Poisson equation are a matter of quite some discussion in literature \([5, 6, 26]\). Usually, the Neumann boundary condition in the normal direction is applied to the boundaries, when the Dirichlet condition is to be applied to the velocity.

\[
\frac{\partial \phi}{\partial \mathbf{n}} = 0 \quad \text{on} \quad \partial \Omega
\]

\[
\frac{\partial (p^{n+1} - p^n)}{\partial \mathbf{n}} = 0
\]
Where $\phi = p^{n+1} - p^n$ represents the pressure update when $q = p^n$. But this represents an unphysical boundary condition on the pressure,

$$\frac{\partial p^{n+1}}{\partial n} = \frac{\partial p^n}{\partial n} = \cdots = \frac{\partial p^0}{\partial n} \quad \text{on} \quad \partial \Omega$$  \hspace{1cm} (2.12)

A spurious numerical boundary layer appears in the numerical solution due to this, which affects the temporal accuracy of the pressure [26]. Nevertheless, it has been observed in [6] that a modified pressure-update equation,

$$p^{n+1} = p^n + \phi - \nu \nabla \cdot \mathbf{v}^*$$  \hspace{1cm} (2.13)

alleviates this numerical boundary layer phenomenon. But it is observed that this boundary layer does not affect the spatial order of accuracy, making this update equation unnecessary for spatial accuracy tests. Since the focus of this work is to assess the spatial accuracy of the rDG-cG method, this additional term is not considered in the pressure-update equation.

The boundary conditions for the Poisson equation change when Neumann boundary condition needs to be enforced on the velocity field (viz. outflow BC). Dirichlet boundaries need to be set for the Poisson equation for this case. The details of enforcing these boundary conditions will be discussed in section §4.3 which pertains to spatial discretization of the pressure field.

\subsection*{2.5 Algorithm}

The incremental projection method is outlined in the following algorithm, for further clarity of implementation.

Initialize the solution $\mathbf{v}^0, p^0$;

\begin{algorithm}
\begin{algorithmic}
\State \textbf{while} $t_n < t_f$ \textbf{do}
\State \textbf{Set}: $t_n = t_n + \Delta t$;
\State \textbf{Set}: $q = p^n$;
\State Compute: $\text{rhs}_\text{mtm}(\mathbf{v}^n, q, t, BC'$s$)$;
\State Solve: Momentum equations (implicit/explicit) to get $\mathbf{v}^*$;
\State Compute: $\text{rhs}_\text{poisson}(\mathbf{v}^*, \Delta t, p, BC'$s$)$;
\State Solve: Poisson equation;
\State Correct velocity to get $\mathbf{v}^{n+1}$;
\State Update pressure: $p^{n+1} = p^n + \phi$;
\State \textbf{end}
\end{algorithmic}
\end{algorithm}

\textbf{Algorithm 1:} The incremental projection method
As already stated, the choice of $q$ and the equation for updating the pressure can be varied. These affect the temporal accuracy of the method [5, 6, 18, 26, 39], which is not the focus of this work.

In the discussion above, it was assumed that the discretization of the temporal derivative in (2.1) is by the explicit forward-Euler method. Although this is the easiest choice of temporal discretization, there are implicit methods which make the solution process more efficient. These details are discussed in the next section.
Chapter 3

Temporal discretization

The time-derivative in the momentum equation has been discretized using two temporal discretization schemes, the explicit Euler and the implicit Euler difference formulae in this work. Depending on which one is used, the algorithm becomes an explicit or implicit one.

3.1 Explicit Euler forward difference

The incremental projection method, using the forward Euler method for the temporal discretization is given by,

\[
\frac{\mathbf{v}^* - \mathbf{v}^n}{\Delta t} + \nabla \cdot (\mathbf{v}^n \mathbf{v}^n) - \nu \nabla^2 \mathbf{v}^n + \frac{1}{\rho} \nabla p^n = \mathbf{f}
\]

\[
\nabla^2 \phi = \frac{1}{\Delta t} \nabla \cdot \mathbf{v}^*, \quad \frac{\partial \phi}{\partial \mathbf{n}} \bigg|_\Gamma = 0
\]

\[
p^{n+1} = p^n + \phi^n - \frac{\nu}{2} \nabla \cdot \mathbf{v}^*
\]

\[
\mathbf{v}^{n+1} = \mathbf{v}^* - \Delta t \nabla \phi
\]

Since all the fluxes are calculated using the velocities at the previous time-step \( \mathbf{v}^n \), they get absorbed into the right-hand side and a single explicit equation is obtained. Although this formulation is easy to implement, and obviates the need of dealing with nonlinear implicit terms and iterative solvers; the time-step sizes which can be used by an explicit algorithm are extremely low. For time-independent flows, the steady-state is achieved after running the algorithm to a very long \( t_f \). Given the restriction on \( \Delta t \), this makes explicit algorithms prohibitive, especially for steady-state flows. Thus we resort to an implicit treatment of the velocity in the momentum equations, described in the following section.
3.2 Implicit Euler backward difference (BDF)

The incremental projection method, using the backward Euler method for the temporal discretization is given by,

\[
\frac{v^* - v^n}{\Delta t} + \nabla \cdot (v^* v^*) - \nu \nabla^2 v^* = -\frac{1}{\rho} \nabla p^n + f \tag{3.1}
\]

with the Poisson equation, pressure and velocity correction and boundary conditions same as the explicit method. Note that for the implicit method all the fluxes are computed using the velocities at the current time-step, \(v^n\). Also note that the \(x\) and \(y\) momentum equations are coupled via the convection speed. The system is now a nonlinear and coupled implicit system. The following methodology is used for the linearization: (3.1) can be written simply as,

\[
M \cdot \frac{v^* - v^n}{\Delta t} = R(v^*) \tag{3.2}
\]

where,

\[
R(v^*) = -\nabla \cdot (v^* v^*) + \nu \nabla^2 v^* - \frac{1}{\rho} \nabla p^n + f \tag{3.3}
\]

Linearization is carried out following the procedure used by [41]. The vector \(R(v^*)\) is linearized with respect to solution vector \(v^*\) at current time-step using a Taylor series expansion,

\[
R(v^*) = R(v^n) + \left( \frac{\partial R}{\partial v} \right)^n (v^* - v^n)
\]

where \(\left( \frac{\partial R}{\partial v} \right)^n\) is called the flux-Jacobian. Using this in (3.2), and denoting \(\Delta v^n = v^* - v^n\), the linearized form of (3.1) is obtained.

\[
A \cdot \Delta v^n = R(v^n) \tag{3.4}
\]

where,

\[
A = \frac{M}{\Delta t} - \left( \frac{\partial R}{\partial v} \right)^n
\]

is the system matrix, and \(R(v^n)\) is the right-hand side evaluated using \(v^n\). The challenge to solve this linearized system lies in the formation of the Jacobian matrix. Methods to obtain the Jacobian matrix and to solve the linearized system (3.4) are discussed in §6 and §7.
Chapter 4

Spatial discretization

This work implements a hybrid approach for the spatial discretization of the pressure and velocity, i.e. the discrete pressure and velocity are in different functional spaces. The pressure is in the continuous (or conforming) finite-element space, and is located at the nodal-points of the mesh. The velocity, on the other hand, is in the discontinuous (on non-conforming) finite-element space, and is located at the centroids of the elements of the mesh. See Fig. 4.1. The mathematical details such as the variational formulations of governing equations and reconstruction methods are discussed in this section.

Along with the standard DG-method for discretizing the velocity, we also consider the reconstructed-DG (or rDG) method. In this method, the velocity is reconstructed from a piecewise-linear (P1) space to a piecewise-quadratic (P2) space, to increase its order of accu-
reconstruction is done by using the underlying P1-solution, by deriving a P2 solution using the solution information from the neighborhood of each cell with a restriction that the scheme is conservative. This P1P2 reconstruction is also discussed here.

4.1 Velocity

The governing equations (3.1) are discretized using a discontinuous Galerkin finite element method for the velocity field and a standard continuous Galerkin finite element method for the pressure field. This leads to the momentum equation to be in the DG vector space and the Poisson equation for pressure to be in the CG vector space. The momentum equation can be rewritten in a simplified form as,

$$\frac{\partial \mathbf{v}}{\partial t} + \frac{\partial \mathbf{F}_k(v)}{\partial x_k} + \frac{1}{\rho} \nabla p = \mathbf{f},$$

(4.1)

where $\mathbf{F}$ is the flux vector given by,

$$\mathbf{F}_j = v_j \mathbf{v} - \nu \frac{\partial^2 \mathbf{v}}{\partial x_j \partial x_j}$$

(4.2)

The weak formulation for the momentum equation after applying integration by parts on the flux terms is given by,

$$\int_{\Omega} \frac{\partial \mathbf{v}}{\partial t} \mathbf{W} d\Omega + \int_{\Gamma} \mathbf{F}_k(v) n_k W_h d\Gamma - \int_{\Omega} \mathbf{F}_k(v) \frac{\partial W_h}{\partial x_k} d\Omega + \frac{1}{\rho} \int_{\Omega} \nabla p W_h d\Omega = \int_{\Omega} \mathbf{f} W_h d\Omega$$

(4.3)

where $\Omega$ denotes the domain and $\Gamma = \partial \Omega$, its boundary; and $\mathbf{n}$, the unit outward normal vector to the boundary. The domain $\Omega$ is subdivided into a collection of non-overlapping triangular elements $\Omega_e$. We introduce the following broken Sobolev space for the velocity field,

$$\mathbf{V}_h^p = \left\{ W_h \in [L_2(\Omega)]^m : W_h|_{\Omega_e} \in [V_m^p] \quad \forall \quad \Omega_e \in \Omega \right\}$$

(4.4)

which consists of discontinuous vector-valued polynomial functions of degree $p$, where $m$ is the dimension of the vector of unknowns $\mathbf{v}$ and,

$$V_m^p = \text{span}\left\{ \prod x_i^{\alpha_i} : 0 \leq \alpha_i \leq p, 0 \leq i \leq d \right\}$$

(4.5)
where $\alpha$ denotes a multi-index and $d$ is the dimension of space. Applying the weak formulation on each element $\Omega_e$, we get,

$$
\text{Find } v_h \in V_h \text{ such as }
\frac{\partial}{\partial t} \int_{\Omega_e} v_h W_h d\Omega + \int_{\Gamma_e} F_k(v_h) n_k W_h d\Gamma - \int_{\Omega_e} F_k(v_h) \frac{\partial W_h}{\partial x_k} d\Omega + \frac{1}{\rho} \int_{\Omega_e} \nabla p W_h d\Omega = \int_{\Omega_e} f W_h d\Omega,
$$

\forall \ W_h \in V_h

where $v_h$ and $W_h$ represent the finite element approximations to the analytical solution $v$ and test function $W$ respectively. These are approximated by piecewise polynomial functions of degree $p$, which are discontinuous between the cell interfaces. For the Galerkin type of finite element methods, the test function is chosen as the basis of these polynomial functions. This gives the following system of $N$ equations,

$$
\frac{\partial}{\partial t} \int_{\Omega_e} v_h B_i d\Omega + \int_{\Gamma_e} F_k(v_h) n_k B_i d\Gamma - \int_{\Omega_e} F_k(v_h) \frac{\partial B_i}{\partial x_k} d\Omega + \frac{1}{\rho} \int_{\Omega_e} \nabla p B_i d\Omega = \int_{\Omega_e} f B_i d\Omega,
$$

$1 \leq i \leq N$, \hspace{1cm} (4.6)

where $N$ is the dimension of the polynomial space. In the traditional DG methods, the numerical polynomial solutions $U_h$ in each element are expressed using standard Lagrange finite element or heirarchial nodal basis functions, as a result of which, the unknowns solved are the variables located at the nodes. Following [29], we express the numerical solutions in the form of a Taylor series expansion, with the unknown variables located at the cell center as shown in Fig. 4.2.

Figure 4.2: Representation of polynomial solutions using P1 finite element shape function (left) and using Taylor series approximation (right)

If we do a Taylor series expansion at the cell centroid, and take the cell averaged values of
the unknowns and their derivatives at the center of the cell, we get,

\[
\mathbf{V}_h = \tilde{\mathbf{V}} + \left. \frac{\partial \mathbf{V}}{\partial x} \right|_c (x - x_c) + \left. \frac{\partial \mathbf{V}}{\partial y} \right|_c (y - y_c) \\
+ \left. \frac{\partial^2 \mathbf{V}}{\partial x^2} \right|_c \left( \frac{(x - x_c)^2}{2} - \frac{1}{\Omega_e} \int_{\Omega_e} (x - x_c)^2 \, d\Omega \right) \\
+ \left. \frac{\partial^2 \mathbf{V}}{\partial y^2} \right|_c \left( \frac{(y - y_c)^2}{2} - \frac{1}{\Omega_e} \int_{\Omega_e} (y - y_c)^2 \, d\Omega \right) \\
+ \left. \frac{\partial^2 \mathbf{V}}{\partial x \partial y} \right|_c \left( (x - x_c)(y - y_c) - \frac{1}{\Omega_e} \int_{\Omega_e} (x - x_c)(y - y_c) \, d\Omega \right)
\]

(4.7)

where \( \tilde{\mathbf{V}} \) is the mean value of \( \mathbf{V} \) in the cell and \((x_c, y_c)\) are the coordinates of the centroid of the 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 the element shape. Using these expansions for the solution variable, (4.6) becomes,

\[
\left( \int_{\Omega_e} B_j B_i \, d\Omega \right) \frac{\partial v_j}{\partial t} + \int_{\Gamma_e} \mathbf{F}_k(\mathbf{v}_h) \mathbf{n}_k B_i \, d\Gamma - \int_{\Omega_e} \mathbf{F}_k(\mathbf{v}_h) \frac{\partial B_i}{\partial x_k} \, d\Omega + \frac{1}{\rho} \int_{\Omega_e} \nabla p B_i \, d\Omega = \int_{\Omega_e} \mathbf{f} B_i \, d\Omega, \quad 1 \leq i \leq N
\]

(4.8)

where \( \left( \int_{\Omega_e} B_j B_i \, d\Omega \right) \) is the mass-matrix of the system of equations.

We consider a three-dimensioned polynomial space, resulting in a piecewise linear solution. The corresponding three basis functions are

\[
B_1 = 1, \quad B_2 = \frac{(x - x_c)}{\Delta x}, \quad B_3 = \frac{(y - y_c)}{\Delta y}
\]

(4.9)

after a normalization as suggested by Luo et al. [33], where \( \Delta x = 0.5(x_{\text{max}} - x_{\text{min}}) \) and \( \Delta y = 0.5(y_{\text{max}} - y_{\text{min}}) \). \( x_{\text{max}}, x_{\text{min}}, y_{\text{max}} \) and \( y_{\text{min}} \) are the maximum and minimum \( x \) and \( y \) coordinates respectively in the cell \( \Omega_e \). The linear solution is then written as,

\[
\mathbf{V}_h = \tilde{\mathbf{V}} + \left. \frac{\partial \mathbf{V}}{\partial x} \right|_c \Delta x B_2 + \left. \frac{\partial \mathbf{V}}{\partial y} \right|_c \Delta y B_3
\]

(4.10)

In (4.8), the only term connecting the one element to the other is the interface flux term. The other terms are all uncoupled from the neighboring cells. This makes the DG system very compact, since the support extends only to the immediate neighborhood for each element. This is an important advantage of DG methods.
4.1.1 Reconstruction to P2

Further, we reconstruct the P1 polynomial solution to the P2 space to improve the accuracy following Luo et al. [33]. This is termed as the reconstructed-DG or rDG method. The underlying DG solution (4.10) is reconstructed to a quadratic polynomial solution, \( V_R^{h} = \tilde{V}_R + \frac{\partial V_R}{\partial x} \bigg|_c \Delta x B_2 + \frac{\partial V_R}{\partial y} \bigg|_c \Delta y B_3 + \frac{\partial^2 V_R}{\partial x^2} \bigg|_c \Delta x^2 B_4 + \frac{\partial^2 V_R}{\partial y^2} \bigg|_c \Delta y^2 B_5 + \frac{\partial^2 V_R}{\partial x \partial y} \bigg|_c \Delta x \Delta y B_6 \) (4.11)

where the basis functions in addition to \( B_1, B_2 \) and \( B_3 \) from (4.9) are,

\[
\begin{align*}
B_4 &= \frac{(x-x_c)^2}{2 \Delta x^2} - \frac{1}{\Omega_e} \int_{\Omega_e} \frac{(x-x_c)^2}{2 \Delta x^2} d\Omega, \\
B_5 &= \frac{(y-y_c)^2}{2 \Delta y^2} - \frac{1}{\Omega_e} \int_{\Omega_e} \frac{(y-y_c)^2}{2 \Delta y^2} d\Omega, \\
B_6 &= \frac{(x-x_c)(y-y_c)}{\Delta x \Delta y} - \frac{1}{\Omega_e} \int_{\Omega_e} \frac{(x-x_c)(y-y_c)}{\Delta x \Delta y} d\Omega
\end{align*}
\]

(4.12)

after a similar normalization. In order to maintain the compactness of the DG methods, the reconstruction is required to only involve information from the adjacent cells that share a face with the cell over which solution is being reconstructed. There are six degrees of freedom giving 6 unknowns. The first three can be determined trivially by requiring the reconstruction to be conservative, thus giving the first 3 DOF’s equal to the 3 DOF’s of the underlying DG solution at the cell centroid. The remaining DOF’s can be computed in a least-squares sense or using Green-Gauss theorem. Here, the least-squares method is used.

4.1.2 Least-squares reconstruction

An overdetermined system of equations can be obtained, by requiring that the reconstructed solution and its first derivatives be equal to the underlying DG solution and its first derivatives for all the adjacent face-neighboring cells. Consider a neighboring cell \( j \). For this cell, it would mean,

\[
\int_{\Omega_j} V_j d\Omega = \int_{\Omega_j} \left( \tilde{V}_i^R + \frac{\partial V_R}{\partial x} \bigg|_i \Delta x_i B_2 + \frac{\partial V_R}{\partial y} \bigg|_i \Delta y_i B_3 + \frac{\partial^2 V_R}{\partial x^2} \bigg|_i \Delta x_i^2 B_4 \\
+ \frac{\partial^2 V_R}{\partial y^2} \bigg|_i \Delta y_i^2 B_5 + \frac{\partial^2 V_R}{\partial x \partial y} \bigg|_i \Delta x_i \Delta y_i B_6 \right) d\Omega
\]

(4.13)
\[
\frac{\partial \mathbf{V}^R}{\partial x} |_j = \frac{\partial^2 \mathbf{V}^R}{\partial x^2} |_i \Delta x_i \frac{1}{\Delta x_i} + \frac{\partial^2 \mathbf{V}^R}{\partial x \partial y} |_i \Delta x_i \Delta y_i \frac{B_2}{\Delta x_i} + \frac{\partial^2 \mathbf{V}^R}{\partial y \partial x} |_i \Delta x_i \Delta y_i \frac{B_3}{\Delta y_i} \]
(4.14)

\[
\frac{\partial \mathbf{V}^R}{\partial y} |_j = \frac{\partial^2 \mathbf{V}^R}{\partial y^2} |_i \Delta y_i \frac{1}{\Delta y_i} + \frac{\partial^2 \mathbf{V}^R}{\partial x \partial y} |_i \Delta x_i \Delta y_i \frac{B_2}{\Delta y_i} + \frac{\partial^2 \mathbf{V}^R}{\partial y \partial x} |_i \Delta x_i \Delta y_i \frac{B_3}{\Delta x_i} \]

where the basis functions \(B\) are evaluated at the center of cell \(j\). In the first equation, the integral is over the neighboring cell \(i\). Let the integral of the basis function \(B\) over cell \(j\) be denoted by \(\tilde{B}\). Then, in matrix form,

\[
\begin{pmatrix}
\tilde{B}_4^j & \tilde{B}_5^j & \tilde{B}_6^j \\
\tilde{B}_2^j & 0 & \tilde{B}_3^j \\
0 & \tilde{B}_3^j & \tilde{B}_2^j
\end{pmatrix}
\cdot
\begin{pmatrix}
\frac{\partial^2 \mathbf{V}^R}{\partial x^2} |_i \Delta x_i^2 \\
\frac{\partial^2 \mathbf{V}^R}{\partial x \partial y} |_i \Delta x_i \Delta y_i \\
\frac{\partial^2 \mathbf{V}^R}{\partial y^2} |_i \Delta y_i^2
\end{pmatrix}
= \begin{pmatrix}
\mathbf{V}_j \Omega_j - (\mathbf{V}_i \tilde{B}_4^j + \frac{\partial \mathbf{V}^R}{\partial x} |_i \Delta x_i \tilde{B}_2^j + \frac{\partial \mathbf{V}^R}{\partial y} |_i \Delta y_i \tilde{B}_3^j)
\end{pmatrix}
\]
(4.15)

where \(\mathbf{R}\) is used to represent the right-hand side for simplicity. Such equations can be written for each of the face-neighboring cells of \(i\). This leads to a non-square matrix, \(9 \times 3\) for internal triangular cells; an overdetermined linear system. These can be solved in the least-squares sense.

This is done using the normal equation approach, in which the above system is premultiplied by the matrix transpose, yielding,

\[
\begin{pmatrix}
\sum_j (\tilde{B}_4^j \tilde{B}_4^j + \tilde{B}_4^j \tilde{B}_6^j) & \sum_j \tilde{B}_4^j \tilde{B}_5^j & \sum_j (\tilde{B}_4^j \tilde{B}_6^j + \tilde{B}_2^j \tilde{B}_3^j) \\
\sum_j \tilde{B}_4^j \tilde{B}_5^j & \sum_j (\tilde{B}_5^j \tilde{B}_5^j + \tilde{B}_5^j \tilde{B}_6^j) & \sum_j (\tilde{B}_5^j \tilde{B}_6^j + \tilde{B}_2^j \tilde{B}_3^j) \\
\sum_j (\tilde{B}_4^j \tilde{B}_6^j + \tilde{B}_2^j \tilde{B}_3^j) & \sum_j (\tilde{B}_5^j \tilde{B}_6^j + \tilde{B}_2^j \tilde{B}_3^j) & \sum_j (\tilde{B}_6^j \tilde{B}_6^j + \tilde{B}_2^j \tilde{B}_2^j + \tilde{B}_3^j \tilde{B}_3^j)
\end{pmatrix}
\cdot
\begin{pmatrix}
\frac{\partial^2 \mathbf{V}^R}{\partial x^2} |_i \Delta x_i^2 \\
\frac{\partial^2 \mathbf{V}^R}{\partial x \partial y} |_i \Delta x_i \Delta y_i \\
\frac{\partial^2 \mathbf{V}^R}{\partial y^2} |_i \Delta y_i^2
\end{pmatrix}
= \begin{pmatrix}
\sum_j (\tilde{B}_4^j \mathbf{R}_1^j + \tilde{B}_6^j \mathbf{R}_2^j) \\
\sum_j (\tilde{B}_5^j \mathbf{R}_1^j + \tilde{B}_6^j \mathbf{R}_3^j) \\
\sum_j (\tilde{B}_6^j \mathbf{R}_1^j + \tilde{B}_3^j \mathbf{R}_2^j + \tilde{B}_3^j \mathbf{R}_3^j)
\end{pmatrix}
\]
(4.16)

This \(3 \times 3\) system can be solved to get the second-order derivatives for the reconstruction. For the boundary cells, we can still obtain an overdetermined system, albeit of dimensions \(7 \times 3\), by including just the first equation from (4.14) for the boundary face, and all the 3 for the rest of the faces.

### 4.1.3 P1 Mass matrix

The mass matrix (and its inverse in the case of explicit methods) in (4.8) is precomputed and stored as a part of preprocessing. Unlike the conventional nodal DG methods, the mass matrices
are different for each element. The mass matrix for a 2-dimensional geometry is,

\[
\mathbf{M} = \begin{pmatrix}
B_1 B_1 & B_1 B_2 & B_1 B_3 \\
B_2 B_1 & B_2 B_2 & B_2 B_3 \\
B_3 B_1 & B_3 B_2 & B_3 B_3
\end{pmatrix}
\] (4.17)

Due to the choice of the basis functions (4.9), after simplification we get,

\[
\mathbf{M} = \begin{pmatrix}
\Omega_e & 0 & 0 \\
0 & \int_{\Omega_e} \frac{(x-x_c)^2}{\Delta x^2} d\Omega & \int_{\Omega_e} \frac{(x-x_c)(y-y_c)}{\Delta x \Delta y} d\Omega \\
0 & \int_{\Omega_e} \frac{(x-x_c)(y-y_c)}{\Delta x \Delta y} d\Omega & \int_{\Omega_e} \frac{(y-y_c)^2}{\Delta y^2} d\Omega
\end{pmatrix}
\] (4.18)

The first DOF is thus completely decoupled from the other two. Here, the similarity with finite-volume methods becomes obvious. We can clearly see that the finite-volume discretization is a lower order DG method. Thus, FVM’s can be considered as a special case of DG methods, with piecewise constant \((p = 0)\) approximations.

The mass matrix, in essence, becomes a \(2 \times 2\) symmetric matrix for each element, the inverse of which is easy to compute.

### 4.1.4 P2 Mass matrix

The P2 mass matrix is required for the Bassi-Rebay 2 diffusive fluxes outlined in §5.2.3 when the rDG(P1P2) discretization is used for the velocity. The mass matrix (and its inverse) for each element is precomputed and stored as a part of preprocessing. The block \(3 \times 3\) matrix \(M(3 \times 3)\) is equal to the P1 mass matrix. Also, similar to the P1 mass matrix, it is symmetric. Let the constants after integration in (4.12) be,

\[
C_4 = \frac{1}{\Omega_e} \int_{\Omega_e} \frac{(x-x_c)^2}{2 \Delta x^2} d\Omega,
\]

\[
C_5 = \frac{1}{\Omega_e} \int_{\Omega_e} \frac{(y-y_c)^2}{2 \Delta y^2} d\Omega,
\]

\[
C_6 = \frac{1}{\Omega_e} \int_{\Omega_e} \frac{(x-x_c)(y-y_c)}{\Delta x \Delta y} d\Omega.
\]
Then, the other elements of the mass matrix for a 2-dimensional geometry after simplification is,

\[ M(1, 4) = 0 \]
\[ M(1, 5) = 0 \]
\[ M(1, 6) = 0 \]

\[ M(2, 4) = \int_{\Omega_e} \left( \frac{(x - x_c)^3}{2\Delta x^3} - \frac{(x - x_c)C_4}{\Delta x} \right) \, d\Omega = \int_{\Omega_e} \frac{(x - x_c)^3}{2\Delta x^3} \, d\Omega \]
\[ M(2, 5) = \int_{\Omega_e} \left( \frac{(x - x_c)^3}{2\Delta x^3} - \frac{C_5}{\Delta x} \right) \, d\Omega = \int_{\Omega_e} \frac{(x - x_c)^3}{2\Delta x^3} \, d\Omega \]
\[ M(2, 6) = \int_{\Omega_e} \left( \frac{(x - x_c)^3}{2\Delta x^3} - \frac{C_6}{\Delta x} \right) \, d\Omega = \int_{\Omega_e} \frac{(x - x_c)^3}{2\Delta x^3} \, d\Omega \]

\[ M(3, 4) = \int_{\Omega_e} \left( \frac{(x - x_c)^3}{2\Delta x^3} - \frac{C_4}{\Delta x} \right) \, d\Omega = \int_{\Omega_e} \frac{(x - x_c)^3}{2\Delta x^3} \, d\Omega \]
\[ M(3, 5) = \int_{\Omega_e} \left( \frac{(y - y_c)^3}{2\Delta y^3} - \frac{C_5}{\Delta y} \right) \, d\Omega = \int_{\Omega_e} \frac{(y - y_c)^3}{2\Delta y^3} \, d\Omega \]
\[ M(3, 6) = \int_{\Omega_e} \left( \frac{(x - x_c)^3}{2\Delta x^3} - \frac{C_6}{\Delta x} \right) \, d\Omega = \int_{\Omega_e} \frac{(x - x_c)^3}{2\Delta x^3} \, d\Omega \]

\[ M(4, 4) = \int_{\Omega_e} \left( \frac{(x - x_c)^4}{4\Delta x^4} - \frac{(x - x_c)^2}{2\Delta x^2} \right) \, d\Omega = \int_{\Omega_e} \frac{(x - x_c)^4}{4\Delta x^4} \, d\Omega - C_4^2 \Omega_e \]
\[ M(4, 5) = \int_{\Omega_e} \left( \frac{(y - y_c)^4}{4\Delta y^4} - \frac{(y - y_c)^2}{2\Delta y^2} \right) \, d\Omega = \int_{\Omega_e} \frac{(y - y_c)^4}{4\Delta y^4} \, d\Omega - C_5^2 \Omega_e \]
\[ M(4, 6) = \int_{\Omega_e} \left( \frac{(x - x_c)^3}{2\Delta x^3} - \frac{(x - x_c)^2}{2\Delta x^2} \right) \, d\Omega = \int_{\Omega_e} \frac{(x - x_c)^3}{2\Delta x^3} \, d\Omega - C_6^2 \Omega_e \]
\[
\mathbf{M}(5, 5) = \int_{\Omega_e} \left( \frac{(y - y_c)^4}{4\Delta y^4} - 2 \frac{(y - y_c)^2}{2\Delta y^2} C_5 + C_5^2 \right) d\Omega = \int_{\Omega_e} \frac{(y - y_c)^4}{4\Delta y^4} d\Omega - C_5^2 \Omega_e
\]

\[
\mathbf{M}(5, 6) = \int_{\Omega_e} \left( \frac{(x - x_c)(y - y_c)^3}{2\Delta x\Delta y^3} - \frac{(y - y_c)^2}{2\Delta y^2} C_6 - \frac{(x - x_c)(y - y_c)}{\Delta y\Delta x} C_5 + C_5 C_6 \right) d\Omega
= \int_{\Omega_e} \frac{(x - x_c)(y - y_c)^3}{2\Delta x\Delta y^3} d\Omega - C_5 C_6 \Omega_e
\]

\[
\mathbf{M}(6, 6) = \int_{\Omega_e} \left( \frac{(x - x_c)^2(y - y_c)^2}{\Delta x^2\Delta y^2} - 2 \frac{(x - x_c)(y - y_c)}{\Delta x\Delta y} C_6 + C_6^2 \right) d\Omega
= \int_{\Omega_e} \frac{(x - x_c)^2(y - y_c)^2}{\Delta x^2\Delta y^2} d\Omega - C_6^2 \Omega_e
\]

where \(\Omega_e\) is the volume of the element under consideration, with the remaining elements being symmetric. The first DOF is again completely decoupled from the other five. The mass matrix, in essence, becomes a \(5 \times 5\) matrix for each element. The computation is done using Gaussian Quadrature with sufficient quadrature-points.

### 4.2 Pressure

The standard (continuous Galerkin) finite element space is used for pressure discretization. The weak form of the Poisson equation can be obtained following a similar procedure as described above for the momentum equations, except, using a piecewise polynomial space which satisfies \(C^0(\Omega_h)\), i.e.

\[
\mathbf{V}_p^m = \left\{ W_h \in \left[ H_0^1(\Omega) \right]^m : W_h|_{\Omega_e} \in \left[ V_p^m \right] \quad \forall \quad \Omega_e \in \Omega \right\} \tag{4.19}
\]

In the weak form, the pressure Poisson equation is,

\[
\phi_i \int_{\Omega} \nabla B_i \cdot \nabla B_j d\Omega = \int_{\Gamma} \nabla \phi_i \cdot \mathbf{n} B_j d\Gamma - \int_{\Omega} \frac{\nabla \cdot \mathbf{v}^*}{\Delta t} B_j d\Omega, \quad \forall \quad B \in \mathbf{V}_h^p \tag{4.20}
\]

Note that the \(B_i\) used here are different from the Taylor basis functions used for the DG velocity; they are the standard finite-element basis functions derived from the barycentric coordinates. The calculation of the stiffness matrix \(\left( \int_{\Omega} \nabla B_i \cdot \nabla B_j d\Omega \right)\) is discussed in §4.2.1 and §4.2.2. Since the predicted velocity \(\mathbf{v}^*\) is discontinuous at element interfaces, the domain integral has to take the interfacial “jumps” of the velocity into consideration. For this, we use integration by parts on the velocity-divergence term. This (including the Neumann boundary condition on \(\phi\)) results in,

\[
\phi_i \int_{\Omega} \nabla B_i \cdot \nabla B_j d\Omega = -\frac{1}{\Delta t} \left( \int_{\Gamma_e} \mathbf{v}^* \cdot \mathbf{n} B_j d\Gamma - \int_{\Omega_e} \nabla B_j \cdot \mathbf{v}^* d\Omega \right) \tag{4.21}
\]
As stated earlier, for the DG method, the value of unknown velocity at the interface is not unique. Thus, using the average of \( v|_{\Gamma_i} \) and \( v|_{\Gamma_j} \) as the value on the interface, \( v_{\Gamma} = v|_{\Gamma_i} + \frac{1}{2}(v|_{\Gamma_j} - v|_{\Gamma_i}) \) and denoting \([v] \equiv v|_{\Gamma_j} - v|_{\Gamma_i}; \) (4.21) becomes,

\[
\phi_i \int_{\Omega} \nabla B_i \cdot \nabla B_j d\Omega = -\frac{1}{\Delta t} \left( \int_{\Gamma_e} (v^*_{\Gamma_i} + \frac{1}{2}[v^*]) \cdot n B_j d\Gamma - \int_{\Omega_e} \nabla B_j \cdot v^* d\Omega \right) \tag{4.22}
\]

Returning \( v_{\Gamma_i} \) and the domain integral on the right-hand side to its original form using integration by parts in reverse, we get,

\[
\phi_i \int_{\Omega} \nabla B_i \cdot \nabla B_j d\Omega = -\frac{1}{\Delta t} \left( \int_{\Gamma} [v^*] \cdot n B_j d\Gamma + \int_{\Omega} \nabla \cdot v^* B_j d\Omega \right) \tag{4.23}
\]

While computing the right-hand side of (4.23), it is easier to loop over the faces to compute the boundary integral. Thus, due to the direction of one of the normal vectors being opposite to the outward normal, the jump terms add up to give,

\[
\phi_i \int_{\Omega} \nabla B_i \cdot \nabla B_j d\Omega = -\frac{1}{\Delta t} \left( \int_{\Gamma} [v^*] \cdot n B_j d\Gamma + \int_{\Omega} \nabla \cdot v^* B_j d\Omega \right) \tag{4.24}
\]

and the domain integral is trivial for \( DG(P1) \) space for velocity when Taylor bases are used. The computation of the stiffness matrix \( \int_{\Omega} \nabla B_i \cdot \nabla B_j d\Omega \) depends on the type of elements used. This is discussed in the following subsections.

### 4.2.1 Stiffness matrix for cG(P1)

The global stiffness matrix for piecewise-linear (P1) elements is assembled from the local stiffness matrices of all the elements. The local stiffness matrix is given as,

\[
A_{\Omega_e} = \begin{pmatrix}
\int_{\Omega_e} \nabla B_1 \cdot \nabla B_1 & \int_{\Omega_e} \nabla B_1 \cdot \nabla B_2 & \int_{\Omega_e} \nabla B_1 \cdot \nabla B_3 \\
\int_{\Omega_e} \nabla B_2 \cdot \nabla B_1 & \int_{\Omega_e} \nabla B_2 \cdot \nabla B_2 & \int_{\Omega_e} \nabla B_2 \cdot \nabla B_3 \\
\int_{\Omega_e} \nabla B_3 \cdot \nabla B_1 & \int_{\Omega_e} \nabla B_3 \cdot \nabla B_2 & \int_{\Omega_e} \nabla B_3 \cdot \nabla B_3
\end{pmatrix}
\]

where the \( B_i \)'s are standard finite-element basis functions derived from barycentric coordinates \( \lambda_i \). For triangular P1 elements, they are,

\( B_1 = \lambda_1, \quad B_2 = \lambda_2, \quad B_2 = \lambda_3 \)

and the barycentric coordinates are defined as,

\[
\lambda_i = \frac{a_i x + b_i y + c_i}{2\Omega_e}, \quad \text{where} \quad a_i = y_j - y_k, \quad b_i = -(x_j - x_k), \quad c_i = \begin{vmatrix} x_j & x_k \\ y_j & y_k \end{vmatrix}
\]
with $i, j,$ and $k$ as the three vertices of the triangular element. After the local elemental stiffness matrices have been computed, the assembly to the global stiffness matrix can be carried out. The global stiffness matrix has the dimensions $npoin \times npoin$, where $npoin$ is the total number of nodal points in the mesh. The assembly works by a gather-scatter algorithm, where, as already discussed, the elemental stiffness matrices are generated first, and then the elemental stiffness information is scattered to its nodes (vertices for P1). For large meshes, this might be too much memory. Also, the solver might end up being too slow, since it is looping over all the elements of a very-sparse matrix, which is unnecessary. This can be made computationally more efficient by using an edge-based data-storage similar to the one used by Luo et al. in [27].

The edge-based data structure is described in detail in Appendix B.

### 4.2.2 Stiffness matrix for cG(P2)

P2 elements are different from the P1 elements, in that, they approximate the solution by piecewise quadratic polynomials. To do so, they require additional (auxiliary) nodes. These nodes are placed on the centers of each of the 3 edges of the triangular elements. The elemental stiffness matrix for P2-triangular elements, thus, is of dimensions $6 \times 6$. A P2 element is shown in Fig. 4.3.

![Figure 4.3: A P2 triangular element](image)
The basis functions for P2 triangular elements are given as,

\[ B_i = \lambda_i(2\lambda_i - 1), \quad k = 1, 2, 3 \]
\[ B_4 = 4\lambda_1\lambda_2, \]
\[ B_5 = 4\lambda_2\lambda_3, \]
\[ B_6 = 4\lambda_3\lambda_1 \]

As an example, let us consider a few elements of the elemental stiffness matrix.

\[ a_{i,i} = \int_{\Omega_e} \nabla(\lambda_i(2\lambda_i - 1)) \cdot \nabla(\lambda_i(2\lambda_i - 1)) d\Omega, \quad i = 1, 2, 3 \]
\[ = \int_{\Omega_e} (4\lambda_i - 1)^2 \left( \frac{\partial \lambda_i}{\partial x} \right)^2 + \left( \frac{\partial \lambda_i}{\partial y} \right)^2 d\Omega \]
\[ = \left( \frac{\partial \lambda_i}{\partial x} \right)^2 + \left( \frac{\partial \lambda_i}{\partial y} \right)^2 \right) \Omega_e \]
\[ a_{1,4} = \int_{\Omega_e} \nabla(\lambda_1(2\lambda_1 - 1)) \cdot \nabla(4\lambda_1\lambda_2) d\Omega \]
\[ = 4 \int_{\Omega_e} \left( \frac{\partial \lambda_1}{\partial x} (4\lambda_1\lambda_2 \frac{\partial \lambda_1}{\partial x} + 4\lambda_1^2 \frac{\partial \lambda_2}{\partial x} - \lambda_2 \frac{\partial \lambda_1}{\partial x} - \lambda_1 \frac{\partial \lambda_2}{\partial x} + \frac{\partial \lambda_1}{\partial y} (4\lambda_1\lambda_2 \frac{\partial \lambda_1}{\partial y} + \cdots) \right) d\Omega \]

The integrals of the basis functions \( \lambda_i \) are from the standard result,

\[ \int_{\Omega_e} \lambda_i^p \lambda_j^q \lambda_k^r d\Omega = \frac{2p!q!r!}{(2 + p + q + r)!} \Omega_e \quad (4.25) \]

One should be cautious while deriving the elemental stiffness matrix for P2, in that, each of the 36 entries should be derived individually due to the complicated basis functions used. The edge-based data-storage cannot be used for the P2 stiffness matrix.

4.3 Boundary conditions

The boundary conditions for the velocity are weakly enforced, i.e. they are applied via the boundary flux integral computation in (4.8). Either the ghost-state, or ghost-cell approach can be used (Fig. 4.4) for this purpose. The ghost-state approach involves applying the BC at the quadrature points when computing the fluxes. For example, if the exact solution \( u_{exact} \) is to be used as a Dirichlet BC, and it is a function of \( \mathbf{x} \), the BC is applied as, \( u_{gs} = u_{exact}(x_{gs}) \), where the subscript \( gs \) denotes the ghost-state. The ghost cell approach, on the other hand, requires the storage of an imaginary cell, reflected across the boundary, with values of all the DOFs stored at its cell center. This means, the BC needs information about the coordinates of the
cell-center of the ghost-cell, the solution, and both (1st and 2nd) derivatives. This treatment is the same, whether the boundary being considered is a Dirichlet or Neumann boundary. In this work, we use the ghost-state approach, since it is easier to implement, and does not require the first and second derivatives at the boundary cells.

For the pressure, the boundary conditions are applied to the $\phi$-Poisson equation. As outlined in the previous section, the Neumann condition ($\frac{\partial \phi}{\partial n} = 0$) is quite naturally enforced on the weak form of the Poisson equation, by reducing $\int_{\Gamma} \nabla \phi_i \cdot n B_j d\Gamma$ to 0. On the other hand, enforcement of the Dirichlet condition requires some modifications, and can be done in two ways: 1) Reducing the number of equations (removing the equations for the Dirichlet boundary-points); 2) Modifying the equations so that the number of equations (and hence the size of the system stiffness matrix) stay the same. We use the second approach in our work, i.e. the existing equations (and the elements of the stiffness matrix) are modified, keeping its size the same. This is done by multiplying the diagonal entry in the rows of the Dirichlet-points by a large number (of the order $10^8$) and modifying the load-vector accordingly.

\[
A'(i,i)_{\partial \Omega_D} = C \times A(i,i)_{\partial \Omega_D}, \quad \text{where,} \quad C > O(10^6)
\]
\[
F'(i)_{\partial \Omega_D} = C \times A(i,i)_{\partial \Omega_D} \times \phi_D
\]

where, $\phi_D$ is the Dirichlet value desired at the boundary-point. Generally for incompressible flows, Dirichlet BC’s are used when the pressure at that boundary is specified and remains unchanged through the computation (e.g. outflow boundary). This will require $\phi_D = 0$.

This ends the description of the spatial discretization used in this work. It is now followed by a description of the advective and diffusive flux formulation.
Chapter 5

Fluxes

Choice of suitable fluxes is one of the most important parameters in numerical simulations. It decides the stability and accuracy of the numerical scheme. The flux vector (4.2) consists of two types of fluxes, advective and diffusive. It is well known that the DG methods are advantageous for discretizing hyperbolic (advection-type) equations, whether linear (linear advection equation) or nonlinear (Burgers equation, Euler equation, etc.) [9, 10]. But this advantage is lost when considering elliptic (diffusion-type) problems. This section is devoted to the treatment of these fluxes.

5.1 Advective fluxes

Due to the nature of the DG method, the value of the conservative variable is not unique on the element interfaces. Due to this, the interface integral \( \int_{\Gamma} F_{Ak}(v_h)n_k B_i d\Gamma \) comprises of a Riemann problem. An upwind flux is used to approximate the value of the flux at the interface. Consider the interface between elements \(-\) and \(+\) as shown in Fig. 5.1. The advection velocity resulting from an upwind scheme is,

\[
\hat{v}_i = v_i^+, \quad \text{if} \quad v^+ \cdot n^+ \geq 0 \quad (5.1)
\]

\[
= v_i^-, \quad \text{if} \quad v^- \cdot n^- > 0 \quad (5.2)
\]

Gaussian quadrature (Appendix A) is used to compute the integration \( \int_{\Gamma} F_k(v_h) n_k B_i d\Gamma \) over each interface. Note that the velocities \( v_i^{+/-} \) are at the quadrature point on the interface, not at the cell centre. Thus, either (4.10) or (4.11) is used to obtain this value.

The domain integral for the advective fluxes is given by,

\[
\int_{\Omega_c} F_{Ak}(v_h) \frac{\partial B_i}{\partial x_k} d\Omega = \int_{\Omega_c} v_h v_k \frac{\partial B_i}{\partial x_k} d\Omega
\]
Because of the choice of the basis functions $B_i$, the domain integral is non-zero only for the second and third degrees of freedom. It is also numerically evaluated over each cell using Gaussian quadrature.

5.2 Diffusive fluxes

Similar to the value of the unknowns, the derivative of the unknown is also multi-valued at cell interfaces in the DG methods. As opposed to the advective fluxes, there is no preferred direction for diffusive fluxes due to their elliptic nature. Resorting to a simple arithmetic mean of derivatives at cell-centers leads to inconsistencies. Thus, special treatment is required for the discretization of diffusive fluxes. Many methods have been developed over the years to treat the diffusive fluxes. For finite-volume (DG(P0) or rDG(P0P1)) methods the modified gradient method from Mathur and Murthy [34] has been used successfully [39]. For high order DG methods: interior-penalty [1], Bassi-Rebay 2 [4], local-DG [11], recovery [40], and direct-DG [25] are famous. In this work, the diffusive fluxes are handled using three of these approaches.

5.2.1 Modified gradient method

The modified gradient method used in [34,39] computes the gradient at the faces by approximating their normal and tangential components. The gradient at face $ij$ is obtained from,

$$\nabla v_{ij} = \frac{v_j - v_i}{\Delta r_{ij} \cdot n} n + \left( \nabla v_{ij} - \frac{\nabla v_{ij} \cdot \Delta r_{ij}}{\Delta r_{ij} \cdot n} n \right)$$

(5.3)
where \( \nabla v_{ij} = \frac{1}{2}(\nabla v_i + \nabla v_j) \) and \( \Delta r_{ij} \) is the displacement vector between the cell centroids \( i \) and \( j \). The first term in (5.3) is the component normal to the face \( ij \) and the second term is the tangential component computed by removing the normal component from the average gradient at the face. This method is used to compute the gradient for the rDG(P0P1) method in this work.

### 5.2.2 Direct discontinuous Galerkin method

A direct discontinuous Galerkin (DDG) approach developed by Liu and Yan [25] is used to compute the solution derivatives at the cell interfaces. As opposed to the famous Bassi-Rebay II scheme which involves writing the diffusion equation as a 1st order system, this approach uses the weak formulation directly. As a result, the cells interact with one-another via the numerical flux \( \hat{h}_x \) only, similar to interfacial advective fluxes. DDG applied to the diffusive flux term in the incompressible Navier-Stokes equations is shown here:

\[
\int_\Omega \partial_i \partial_j v B_k d\Omega = - \int_{\Omega_e} \partial_j v \partial_j B_k d\Omega + \int_{\Gamma_e} \hat{h}_x B_k d\Gamma \quad (5.4)
\]

Where, \( \hat{h}_x \) is the numerical flux across the interfaces. For \( P0 \) and \( P1 \) velocity polynomial space, it is given by,

\[
\hat{h}_x = \sum_{i=1}^{d} (\partial_i \hat{v})
\]

\[
\partial_i \hat{v} = \beta_0 \left[ \frac{u}{\Delta} n_i + \overline{\partial_i v} \right]
\]

whereas for \( P2 \) polynomial space, another term is added for the computation of \( \partial_i \hat{v} \),

\[
\partial_i \hat{v} = \beta_0 \left[ \frac{v}{\Delta} n_i + \overline{\partial_i v} + \beta_1 [\partial_i v] n_i \right]
\]

where,

\[
[v] = v^+ - v^-, \quad \overline{\partial_i v} = \frac{\partial v^+ + \partial v^-}{2} \quad \text{and,} \quad \Delta = \frac{\text{diam}\{K^+\} + \text{diam}\{K^-\}}{2}
\]

The \( \beta_0 \) in the numerical flux is arbitrary and can be adjusted, as long as it is of order 1, and \( \beta_1 = \frac{1}{12} \). It was observed by [25] that \( \beta_0 \) affects the error magnitude, but leaves the order of accuracy unchanged.

The advantage of this method lies in its easy implementation, as opposed to the BR2 method. But its disadvantage is that the first derivatives at the ghost-state (or ghost-cell) need to be specified for \( P1 \) discretization, and both first and second derivatives need to be specified for
the $P2$ discretization. This information is generally unavailable for real problems.

### 5.2.3 Bassi-Rebay 2 method

The second approach is the method developed by Bassi and Rebay [4] which is usually referred to as Bassi-Rebay 2 (BR2). The BR2 method is well established to compute the diffusive fluxes in DG spaces. We use this method as a standard to compare the previously explained DDG method.

As opposed to the DDG method, BR2 involves writing the diffusion equation as a 1st order system of equations. Consider the laplace equation:

$$-\nabla^2 u = f$$  \hspace{1cm} (5.5)

Introducing an auxiliary vector variable $q$, we get,

$$q = \nabla u$$

$$-\nabla \cdot q = f$$

thus reducing the second order equation (5.5) to a system of first order equations. Applying the DG formulation, and denoting $\Omega_{ni}$ as the set of adjacent neighbours of $\Omega_i$, we get,

$$\int_{\Omega_i} q_h \cdot \nabla v_h \, d\Omega - \int_{\Gamma_i} q_h \cdot n v_h \, d\Gamma = \int_{\Omega_i} f v_h \, d\Omega$$

$$\int_{\Omega_i} q_h \cdot w_h \, d\Omega + \int_{\Omega_i} u_h \nabla \cdot w_h \, d\Omega - \sum_{j \in \Omega_{ni}} \int_{\Gamma_{ij}} u_h w_h \cdot n \, d\Gamma = 0$$

where $v_h \in V^p_h$ from (4.4) and $w_h \in [V^p_h]^d$. The value of $u_h$ at the interface is then chosen to be the average of the left and right states. If $u|_{\Gamma_i}$ and $u|_{\Gamma_j}$ are the values on the left and the right of the interface, the value on the interface is, $u_{\Gamma_i} = u_{\Gamma_j} + \frac{1}{2}(u_{\Gamma_j} - u_{\Gamma_i})$. Denoting $[u] \equiv u_{\Gamma_j} - u_{\Gamma_i}$ as the jump-operator, performing integration by parts, the second equation above becomes,

$$\int_{\Omega_i} q_h \cdot w_h \, d\Omega + \int_{\Omega_i} u_h \nabla \cdot w_h \, d\Omega - \sum_{j \in \Omega_{ni}} \int_{\Gamma_{ij}} u_h w_h \cdot n \, d\Gamma = 0$$  \hspace{1cm} (5.6)

Here we define global and local lifting operators $\delta$ and $\delta_l$ respectively as,

$$\int_{\Omega_i} \delta \cdot w_h \, d\Omega + \sum_{j \in \Omega_{ni}} \frac{1}{2} \int_{\Gamma_{ij}} [u_h] \partial_{\Omega_j} w_h \cdot n \, d\Gamma = 0, \quad \forall \quad w_h \in [V^p_h]^d$$

$$\int_{\Omega_i} \delta_l \cdot w_h \, d\Omega + \frac{1}{2} \int_{\Gamma_{ij}} [u_h] \partial_{\Omega_j} w_h \cdot n \, d\Gamma = 0, \quad \forall \quad w_h \in [V^p_h]^d$$  \hspace{1cm} (5.7)
This gives us the primal form of the diffusion equation (5.5) as,

$$\int_{\Omega_i} (\nabla u_h - \delta) \cdot \nabla v_h \, d\Omega - \sum_{j \in \Omega_{wn}} \int_{\Gamma_{ij}} \langle \langle (\nabla u_h - \delta_l) \cdot n \rangle \rangle v_h \, d\Gamma = \int_{\Omega_i} f v_h \, d\Omega, \quad \forall \ v_h \in V_h^P$$  \hspace{1cm} (5.8)

where, $\langle \langle \psi \rangle \rangle_{\Gamma_{ij}} = \frac{\psi_{\Gamma_i} + \psi_{\Gamma_j}}{2}$ is the average operator.

First, the local lifting operator $\delta_l$ for each cells is computed by looping over the edges. Eq. (5.7) can be used for this purpose. Simultaneously, the right-hand side of the weak-form of the global lifting operator equation is computed by summing up the right-hand sides of the local lift operator. The global lifting operator is then computed, using this right-hand side and the inverse of the mass matrix. Note that when the rDG(P1P2) spatial discretization is used for the velocity field, the lift-operator is in the P2 space. Consequentially, the solution of the local and global lift operator equations requires the inverse of the P2 mass-matrix, which can be computed and stored as a part of preprocessing as discussed in §4.1.4. The computed local and global matrices are then used in Eq. 5.8 to get the diffusive-fluxes for the Navier-Stokes equations.

**Computing $\nabla v_h$**

In equation (5.8), $\nabla v_h$ is basically the second and third DOF ($V^{(2)}, V^{(3)}$) when a $P1$-polynomial with Taylor bases is used.

$$\left. \frac{\partial V}{\partial x} \right|_{ig} = V^{(2)} \big|_i$$

$$\left. \frac{\partial V}{\partial y} \right|_{ig} = V^{(3)} \big|_i$$

But for the reconstructed P1P2 discretization, $\nabla u_h$ is reconstructed at the quadrature-point $ig$ from the following equations:

$$\left. \frac{\partial V}{\partial x} \right|_{ig} = V^{(2)} \big|_i + \frac{\partial^2 V_R}{\partial x^2} \bigg|_i \Delta x_i \frac{B_{2ig}}{\Delta x_i} + \frac{\partial^2 V_R}{\partial x \partial y} \bigg|_i \Delta x_i \Delta y_i \frac{B_{3ig}}{\Delta x_i}$$

$$\left. \frac{\partial V}{\partial y} \right|_{ig} = V^{(3)} \big|_i + \frac{\partial^2 V_R}{\partial y^2} \bigg|_i \Delta y_i \frac{B_{3ig}}{\Delta y_i} + \frac{\partial^2 V_R}{\partial x \partial y} \bigg|_i \Delta x_i \Delta y_i \frac{B_{2ig}}{\Delta y_i}$$

Now the fluxes in (5.8) can be computed. The above formulation can be applied to the Navier-Stokes equations directly to compute the diffusive fluxes.

We have discussed the formulation of the convective and viscous fluxes in the momentum equations until now. If the explicit time-discretization is used, these fluxes are computed using the velocities at the previous timestep, thus getting absorbed into the right-hand side of the
momentum equation. In this case, given a linear-system solver for the Poisson equation, we are equipped to implement the projection method.

However, as discussed before, an explicit solver limits the size of the time-steps being used to a very low value. This results in a very slow time-marching code, especially for steady-state flows, where the steady-state solution is obtained when \( t_f \to \infty \). Hundreds of thousands of such small time-steps are needed to reach desired state. Here it is convenient to use an implicit time-discretization for the momentum equations, which would allow the use of higher time-step sizes. The linearized system (3.4), resulting from the implicit time-discretization is required to be solved. In this case, the fluxes discussed in this section are computed at the current time-step, not the previous one. This will lead to the fluxes remaining on the left-hand side of the system equation for \( \mathbf{v}^* \). Such implementation details for the solution of the implicit fully-coupled system are now discussed.

As mentioned earlier, computation of the flux-Jacobian matrix is one of the challenges faced in this. The following section deals with this.
Chapter 6

Forming the Jacobian matrix

The Jacobian matrix in (3.4) is required both, for forming the system matrix $A$ and for the preconditioning. The exact Jacobian matrix can be computed, but the process is quite complicated due to the nature of the reconstruction and the viscous fluxes, and the computing time and storage costs would be prohibitive. Thus, instead of the exact Jacobian, approximated Jacobians are usually computed. The details of their computation are discussed in this section.

6.1 Structure of the Jacobian matrix

The Jacobian matrix has a block-structure. This can be understood, if we consider the fact that the number of unknowns are $\text{ndegr} \times \text{neqns} \times \text{nelem}$. The unknown vector for each element in a 2-dimensional problem in P1 polynomial space is,

$$\mathbf{v} = \{v_1^{(1)}, v_1^{(2)}, v_1^{(3)}, v_2^{(1)}, v_2^{(2)}, v_2^{(3)}\}$$

where $v_j^{(i)}$ represents the $i$th degree of freedom of the unknown in equation $j$. Similarly, the residual vector $\mathbf{R}$ has 6 components. Thus, each component of the Jacobian matrix,

$$\frac{\partial \mathbf{R}}{\partial \mathbf{v}} = \begin{pmatrix}
\frac{\partial R_1}{\partial v_1} & \frac{\partial R_1}{\partial v_2} & \cdots & \frac{\partial R_1}{\partial v_{\text{nelem}}} \\
\frac{\partial R_2}{\partial v_1} & \frac{\partial R_2}{\partial v_2} & \cdots & \frac{\partial R_2}{\partial v_{\text{nelem}}} \\
\vdots & \vdots & \ddots & \vdots \\
\frac{\partial R_n}{\partial v_1} & \frac{\partial R_n}{\partial v_2} & \cdots & \frac{\partial R_n}{\partial v_{\text{nelem}}} 
\end{pmatrix}$$
is in itself a $6 \times 6$ matrix,

\[
\frac{\partial R_i}{\partial v_j} = \begin{pmatrix}
\frac{\partial R^{(1)}_i}{\partial v^{(1)}_j} & \frac{\partial R^{(1)}_i}{\partial v^{(2)}_j} & \cdots & \frac{\partial R^{(1)}_i}{\partial v^{(n\text{degr})}_j} \\
\frac{\partial R^{(2)}_i}{\partial v^{(1)}_j} & \frac{\partial R^{(2)}_i}{\partial v^{(2)}_j} & \cdots & \frac{\partial R^{(2)}_i}{\partial v^{(n\text{degr})}_j} \\
\vdots & \vdots & \ddots & \vdots \\
\frac{\partial R^{(n\text{degr})}_i}{\partial v^{(1)}_j} & \frac{\partial R^{(n\text{degr})}_i}{\partial v^{(2)}_j} & \cdots & \frac{\partial R^{(n\text{degr})}_i}{\partial v^{(n\text{degr})}_j}
\end{pmatrix}
\]

The Jacobian matrix can be computed in three ways:

1. Analytical differentiation
2. Numerical finite difference
3. Automatic differentiation toolkits

We focus on the first and second methods.

### 6.2 Analytical differentiation

In this approach, the Jacobian is computed manually. The benefit of this approach is that the code is computationally very efficient. However, implementing the differentiation can prove to be extremely challenging, if not impossible, for the reconstruction and viscous fluxes in the Navier-Stokes equations.

Here, the differentiation is manually performed to obtain the exact Jacobian matrix. For example, let us consider the inviscid fluxes, made up of the interface fluxes and domain integrals. Let us first consider the contribution of the interface fluxes to the Jacobian.

#### 6.2.1 Interface integral

In case of a 2-dimensional unstructured triangular mesh, each element has 3 direct neighbours. For a high-order DG discretization, Gauss quadrature is used for numerical integration (Please refer appendix for quadrature rules). Thus, the interface flux on element $i$, assuming $ng$-point quadrature, is,

\[
R_i = \int_{\Gamma_e} F_k(v_h) n_k B_i d\Gamma = \sum_{j=1}^{3} \sum_{g=1}^{ng} \hat{H}(v_i(x_g), v_j(x_g)) B_i(x_g)
\]
where $\hat{H}(v_i,v_j)$ is the interface flux function between elements $i$ and $j$. Its contribution to the Jacobian is computed as,

\[
J(v)_{ik} = \frac{\partial R_i}{\partial v_k} = \sum_{j=1}^{ng} \sum_{g=1}^{ng} \frac{\partial \hat{H}(v_i(x_g),u_j(x_g))}{\partial v_k} B_i(x_g) \quad \forall \ k \in N_i \text{ or } k = i
\]

\[
= 0, \quad \text{otherwise;}
\]

where $N_i$ is the set of elements neighbouring element $i$. The specific form of the Jacobian depends upon the choice of Riemann flux function used. Now let us consider the contribution of the domain integral.

### 6.2.2 Domain integral

The contribution of the domain integral is relatively simpler to derive, since it does not involve any neighbors, only the cell under consideration. The domain integral is given as,

\[
R_i = \int_\Omega F(v_i) \cdot \nabla B_i \, d\Omega = \int_\Omega v v_i \cdot \nabla B_i \, d\Omega = \sum_{g=1}^{ng} v(x_g) v_i(x_g) \cdot \nabla B_i(x_g)
\]

again using $ng$-point Gauss-quadrature. Its contribution to the Jacobian is computed as,

\[
J(v)_{ik} = \frac{\partial R_i}{\partial v_k} = \sum_{g=1}^{ng} \frac{\partial (v(x_g) v_i(x_g) \cdot \nabla B_i(x_g))}{\partial v_k} B_i(x_g) \quad \text{when } k = i
\]

\[
= 0, \quad \text{otherwise;}
\]

It can be seen that the Jacobian matrix for the inviscid fluxes is easy to derive, unless reconstruction is used. Even after reconstruction, a good approximation of the Jacobian matrix can still be derived. This task becomes difficult, if not impossible for the viscous fluxes. Thus for full Navier-Stokes equations, other methods are used to compute the flux-Jacobian.

### 6.3 Numerical finite difference

This approach is easier to implement, especially in the Jacobian-free solvers. It computes the matrix-vector product, eliminating the need to explicitly store the matrix itself. Although this method is easy to implement, it is extremely computationally intensive.

In this method, the Jacobian is numerically evaluated, by using the first principles of deriva-
tives,
\[ \frac{\partial R}{\partial v_j} \bigg|_n = \frac{R(v + \epsilon \cdot e_j) - R(v)}{\epsilon}, \quad \epsilon \in \mathbb{R} \]

where, \( e_j \) is the \( j^{th} \) component of the unit vector associated with the component of variable \( v \) (i.e. \( v_j \)) with respect to which the derivative is required. The scalar parameter \( \epsilon \) is problem dependent and is a very small number of the order \( 10^{-7} - 10^{-8} \).

The numerical difference approach can be naturally extended to the rDG methods, even for the viscous fluxes. It can be used irrespective of the type of flux-treatment implemented. This is the reason why it is widely accepted for computing the Jacobian matrix. However, there are two disadvantages of this approach of computing the Jacobian matrix:

- The quality of the approximation of the Jacobian greatly depends on \( \epsilon \), which is problem dependent, and unknown a priori. The value should be small enough to minimize the truncation errors that results from the polynomial approximation; but just large enough to avoid the roundoff errors of the finite precision of the computer (machine-error).

- The computational cost is very high, especially for high-order methods, considering that the fluxes (\( R \)) have to computed twice (which has all the loops over the quadrature-points) for each iteration of the linear solver, since they have more DOF’s.

Considering these disadvantages, the numerical differencing is used strictly when needed, viz. for the viscous fluxes where the exact Jacobian is difficult to derive.

Now that the Jacobian matrix, and hence \( A \) from (3.4) have been discussed, we move on to the linear solvers.
Chapter 7

Linear system solvers

Solvers for linear systems of equations have evolved from stationary to non-stationary iterations, to multi-grid methods. Due to their attractive convergence properties, the non-stationary iterative solvers are being widely used for numerical solutions to linear systems of equations. The conjugate gradient method and the generalized minimum residual method are the most well-known and efficient solvers of these. The stationary iterative solvers (such as Jacobi, symmetric Gauss-Seidel, etc.) are used only as preconditioners for the nonstationary solvers. We consider both, the conjugate gradient and the generalized minimum residual solvers to solve the linear systems generated in the algorithm.

We encounter two systems of equations, one linear, (4.24); and the other linearized, (3.4) in the DG-cG method. The solver requirements of these two systems are quite different. We briefly discuss the solvers implemented for each of these systems.

7.1 Poisson equation

The equation (4.24) leads to a linear system of equation. The system matrix, known as the stiffness matrix, is symmetric positive definite (SPD), i.e., for the stiffness matrix $A$, its elements satisfy,

$$\begin{align*}
a_{ij} &= a_{ji} \\
z^T A z &> 0, \quad \forall \quad z \in \mathbb{R}, \quad z \neq 0 \\
z^T A z &= 0, \quad \text{iff} \quad z = 0.
\end{align*}$$

This allows the use of the conjugate gradient non-stationary iterative solver. The conjugate gradient (CG) solver can be used to obtain numerical solutions for SPD systems. It performs extremely well too, provided, an adequately good preconditioning is used. The algorithm of
the preconditioned CG, henceforth called pCG, is shown in Alg. (2). We use the symmetric Gauss-Seidel (SGS) preconditioner in this work.

Initial guess $x_0$;
Compute residual $r_0 = b - Ax_0$;
Preconditioning step: solve $Mz_0 = r_0$;
Search direction: $p_0 = z_0$;
$k = 0$;
while $r_k > \varepsilon$ do
  Compute: $\theta_k = \frac{r_k^T z_k}{p_k^T Ap_k}$;
  Compute $k + 1$th iterate: $x_{k+1} = x_k + \theta_k p_k$;
  Compute $k + 1$th residual: $r_{k+1} = r_k - \theta_k Ap_k$;
  Preconditioning step: solve $Mz_{k+1} = r_{k+1}$;
  Compute: $\beta_k = \frac{r_{k+1}^T z_{k+1}}{r_k^T z_k}$;
  Compute $k + 1$th search direction: $p_{k+1} = z_{k+1} + \beta_k p_k$;
end
Algorithm 2: The preconditioned-CG algorithm

Observe that throughout the pCG iteration, nowhere is the system matrix $A$ explicitly required. Its product with a vector is all that’s needed by the solver. This enables us to use a “matrix-free” algorithm, where $A$, which is extremely large in size ($ndof \times ndof$); is not explicitly stored. A function which computes the product of $A$ with an input vector suffices. This procedure saves a lot of memory and computation time.

7.2 Momentum equations

The X and Y momentum equations form a set of coupled nonlinear system of equations, owing to the advection term. After performing the linearization outlined in §3.2, this leads to a non-symmetric system matrix. Thus the pCG solver cannot be used. We use the pGMRES (preconditioned Generalized Minimum RESidual) solver instead for this system.

Similar to pCG, pGMRES does not require the system matrix $A$ to be stored explicitly, saving a lot of memory (size of the system matrix is $ndof \times neqns \times nelem$) and computation time. A function which computes the product of $A$ with an input vector suffices. The “matrix-free” pGMRES is frequently used in numerical computations [28]. To implement the matrix-free pGMRES, consider (3.4),

$$
\frac{M}{\Delta t} \Delta v^n - \frac{\partial R}{\partial v} \bigg|_n \Delta v^n = R(v^n) \tag{7.1}
$$

The mass matrix $M$ is conveniently stored beforehand for each element, since it is geometry-
dependent, and it is just a \( ndof \times ndof \) matrix per element. The Jacobian \( \frac{\partial \mathbf{R}}{\partial \mathbf{u}} |_{n} \), on the other hand, is a much larger, albeit sparse matrix which is dependent on \( \mathbf{v}^n \). As mentioned earlier, the explicit Jacobian is never required, only its product with a vector. If the numerical differencing approach from §6.3 is used, this becomes very convenient to compute. Considering the vector with which the product is required is \( \mathbf{w} \),

\[
\frac{\partial \mathbf{R}}{\partial \mathbf{v}} \mathbf{w} = \frac{\mathbf{R}(\mathbf{v} + \epsilon \cdot \mathbf{w}) - \mathbf{R}(\mathbf{v})}{\epsilon}, \quad \epsilon \in \mathbb{R}
\]
gives the matrix-vector product required.

The GMRES solver is efficient because of the significantly less amount of iterations it takes to converge. Considering that each GMRES iteration is computationally very expensive; if the solver doesn’t converge in a certain economical number of iterations, the GMRES turns out to be very slow. To ensure convergence in least possible iterations, clustering eigen-values of the system matrix by an effective preconditioning is a prerequisite.

**Preconditioning for GMRES**

A preconditioner improves the condition number \( K \) of a matrix by clustering its eigen-values. The basic idea of preconditioning is outlined in the following equations.

\[
\begin{align*}
\mathbf{Ax} &= \mathbf{b} \\
\mathbf{M}^{-1} \mathbf{Ax} &= \mathbf{M}^{-1} \mathbf{b}
\end{align*}
\]

If the preconditioner is well chosen, the new system of equations is easier to solve than the original one. An ideal preconditioner would be the inverse of the matrix itself, but computing the inverse of the matrix is las hard as solving the system itself. An optimum is reached by choosing an approximate of the system matrix \( \mathbf{A} \) as the preconditioner, which clusters its eigen-values.

We use the block-Jacobi preconditioning for this system of equations. For generating the block-Jacobi preconditioning matrix, we follow the same steps as §6.2 or §6.3; just that we compute the derivatives with respect to variables at the element in consideration only, so that we are left with the block-diagonal terms. The system matrix is,

\[
\mathbf{A} = \mathbf{M} \frac{\Delta t}{\Delta t} - \frac{\partial \mathbf{R}}{\partial \mathbf{v}} |_{n}
\]

where \( \frac{\partial \mathbf{R}}{\partial \mathbf{v}} |_{n} \) is a \( 6 \times 6 \) matrix per element for a 2-dimensional P1 spatial discretization, whereas, \( \mathbf{M} \) has been written as a \( 3 \times 3 \) matrix in the previous sections. To derive the preconditioner,
the $6 \times 6$ matrix,

\[
\begin{pmatrix}
\Omega_e & 0 & 0 & 0 & 0 & 0 \\
0 & \int_{\Omega_e} B_2 B_2 & \int_{\Omega_e} B_2 B_3 & 0 & 0 & 0 \\
0 & \int_{\Omega_e} B_3 B_2 & \int_{\Omega_e} B_3 B_3 & 0 & 0 & 0 \\
0 & 0 & 0 & \Omega_e & 0 & 0 \\
0 & 0 & 0 & 0 & \int_{\Omega_e} B_2 B_2 & \int_{\Omega_e} B_2 B_3 \\
0 & 0 & 0 & 0 & \int_{\Omega_e} B_3 B_2 & \int_{\Omega_e} B_3 B_3
\end{pmatrix}
\]

\[
\frac{1}{\Delta t} \begin{pmatrix}
\frac{\partial R_1^{(1)}}{\partial v_1^{(1)}} & \frac{\partial R_1^{(2)}}{\partial v_1^{(2)}} & \frac{\partial R_1^{(3)}}{\partial v_1^{(3)}} & \frac{\partial R_1^{(1)}}{\partial v_2^{(1)}} & \frac{\partial R_1^{(2)}}{\partial v_2^{(2)}} & \frac{\partial R_1^{(3)}}{\partial v_2^{(3)}} \\
\frac{\partial R_2^{(1)}}{\partial v_1^{(1)}} & \frac{\partial R_2^{(2)}}{\partial v_1^{(2)}} & \frac{\partial R_2^{(3)}}{\partial v_1^{(3)}} & \frac{\partial R_2^{(1)}}{\partial v_2^{(1)}} & \frac{\partial R_2^{(2)}}{\partial v_2^{(2)}} & \frac{\partial R_2^{(3)}}{\partial v_2^{(3)}} \\
\frac{\partial R_3^{(1)}}{\partial v_1^{(1)}} & \frac{\partial R_3^{(2)}}{\partial v_1^{(2)}} & \frac{\partial R_3^{(3)}}{\partial v_1^{(3)}} & \frac{\partial R_3^{(1)}}{\partial v_2^{(1)}} & \frac{\partial R_3^{(2)}}{\partial v_2^{(2)}} & \frac{\partial R_3^{(3)}}{\partial v_2^{(3)}} \\
\frac{\partial R_1^{(1)}}{\partial v_1^{(1)}} & \frac{\partial R_1^{(2)}}{\partial v_1^{(2)}} & \frac{\partial R_1^{(3)}}{\partial v_1^{(3)}} & \frac{\partial R_1^{(1)}}{\partial v_2^{(1)}} & \frac{\partial R_1^{(2)}}{\partial v_2^{(2)}} & \frac{\partial R_1^{(3)}}{\partial v_2^{(3)}} \\
\frac{\partial R_2^{(1)}}{\partial v_1^{(1)}} & \frac{\partial R_2^{(2)}}{\partial v_1^{(2)}} & \frac{\partial R_2^{(3)}}{\partial v_1^{(3)}} & \frac{\partial R_2^{(1)}}{\partial v_2^{(1)}} & \frac{\partial R_2^{(2)}}{\partial v_2^{(2)}} & \frac{\partial R_2^{(3)}}{\partial v_2^{(3)}} \\
\frac{\partial R_3^{(1)}}{\partial v_1^{(1)}} & \frac{\partial R_3^{(2)}}{\partial v_1^{(2)}} & \frac{\partial R_3^{(3)}}{\partial v_1^{(3)}} & \frac{\partial R_3^{(1)}}{\partial v_2^{(1)}} & \frac{\partial R_3^{(2)}}{\partial v_2^{(2)}} & \frac{\partial R_3^{(3)}}{\partial v_2^{(3)}}
\end{pmatrix}
\]

is derived per element and then inverted. After the inversion, the preconditioning can be applied to the pGMRES solver. The Jacobian is analytically derived wherever possible, for computational efficiency. It was observed that this preconditioner reduced the number of iterations to converge the pGMRES solver by $\frac{2}{3}$rds.

Numerous test-cases were used to judge the performance of the method described in this work. We discuss a few results obtained from these tests in the next chapter.
Chapter 8

Numerical results

A few numerical results obtained from the method described in this work are now presented. The GNU FORTRAN compiler was used on Linux operating system to obtain the results. The test cases are broadly split into two sections: cases with analytical solutions to the governing equations, and cases without such solutions. The flows in the first category are useful to determine the order of accuracy of the method used in the algorithm, since errors can be easily computed from the exact solutions. A large majority of real flows fall in the second category. These test cases show the capability of the method to simulate real flows accurately.

8.1 Error and order of accuracy calculations

The errors with respect to exact solution are measured in the $L^2$-norm as follows,

$$||e_u||_{L^2(\Omega)} = \sqrt{\int_{\Omega} |u_h - u_e|^2 d\Omega} = \sqrt{\sum_{i=1}^{nelem} \int_{\Omega_i} |u_h - u_e|^2 d\Omega} \quad (8.1)$$

The integral over the domain is computed using Gaussian quadrature. The spatial order of accuracy of a discretization scheme can be assessed by computing the errors for a test case on successively refined meshes, with characteristic mesh-sizes $h$ and $h/2$ respectively. If the errors are $e_h$ and $e_{h/2}$, the order $\mathcal{O}(h^m)$ is given by the slope,

$$m = \frac{\log_{10}(e_h) - \log_{10}(e_{h/2})}{\log_{10} h - \log_{10} \frac{h}{2}} = \frac{\log_{10}(\frac{e_h}{e_{h/2}})}{\log_{10} 2} \quad (8.2)$$

such that the design orders for rDG(P0P1), DG(P1) and rDG(P1P2) are $\mathcal{O}(h^2)$, $\mathcal{O}(h^2)$ and $\mathcal{O}(h^3)$. The number of DOF’s is indicative of the computational cost of the method ($N_{dof} \rightarrow N_{eqns}$ to be solved). Thus, in addition to order of accuracy, the effectiveness of the reconstruc-
tion is also discussed by plotting the errors against the total number of DOF’s. The number of DOF’s for DG methods is shown in Table 8.1. The design-order for the reconstructed methods is one higher than the order of the underlying DG solution which is reconstructed.

Table 8.1: Total number of DOF’s for DG methods of different orders for 2D flows

<table>
<thead>
<tr>
<th>Method</th>
<th>Design-order</th>
<th>DOF’s</th>
</tr>
</thead>
<tbody>
<tr>
<td>DG(P0) (or FVM)</td>
<td>1</td>
<td>nelem</td>
</tr>
<tr>
<td>rDG(P0P1)</td>
<td>2</td>
<td>nelem</td>
</tr>
<tr>
<td>DG(P1)</td>
<td>2</td>
<td>3\times nelem</td>
</tr>
<tr>
<td>rDG(P1P2)</td>
<td>3</td>
<td>3\times nelem</td>
</tr>
<tr>
<td>DG(P2)</td>
<td>3</td>
<td>5\times nelem</td>
</tr>
</tbody>
</table>

Liu and Yan [25] report $O(h^3)$ using DDG for the diffusion fluxes for DG(P2) discretization on structured grids, yet we observed that it did not achieve $O(h^3)$ for rDG(P1P2) in unstructured grids. Thus, in all the reported test-cases involving viscous fluxes, the BR2 method outlined in §5.2.3 is used.

8.2 Kovasznay flow

The Kovasznay flow [22] is an exact solution to the stationary incompressible Navier-Stokes equations. It represents flow past a grid or an array of cylinders at low Reynolds numbers. This flow is considered in the two-dimensional domain $(-0.5, 1.5) \times (0, 2)$ with Dirichlet boundary conditions obtained from the exact solution and initial conditions corresponding to a uniform flow field. The velocity and pressure fields are given by,

$$V_x = 1 - e^{\lambda x} \cos(2\pi y)$$
$$V_y = \frac{\lambda}{2\pi} e^{\lambda x} \sin(2\pi y)$$
$$p = \frac{1}{2} (1 - e^{2\lambda x})$$

where, $\lambda = \frac{1}{2\nu} - \left(\frac{1}{4\nu^2} + 4\pi^2\right)^{1/2}$. We use $\nu = 0.025$ ($Re = 40$). Due to its steady nature, temporal-discretization errors do not affect the order of accuracy. Thus, it is a good case to assess the spatial order-of-convergence. We use unstructured triangular meshes with 554, 2216, 8864 elements (Figs. C.1 and C.3 respectively) for these tests. The errors are listed in Table 8.2. Fig. 8.2 shows the comparison of $L^2$-errors of the rDG(P0P1), DG(P1) and rDG(P1P2)
methods. The DG(P1) method is clearly more accurate than the rDG(P0P1) method, but the real advantage of the DG method shows after its reconstruction to P2-polynomial. The reconstruction clearly improves the order of accuracy of the 2nd order DG method to make it 3rd order accurate. It also significantly improves the accuracy of the method by reducing the errors by more than an order of magnitude. Fig. 8.4 shows the errors plotted against the total number of DOF’s in the mesh. Here it is observed that the rDG(P0P1) is more effective than DG(P1). The true advantage of the reconstruction from P1 to P2 can be seen in the significant drop of errors for the same number of DOF’s of rDG(P1P2).

Table 8.2: Errors for the Kovasznay flow test case

| Method       | Nelem | $||\epsilon||_2$ | $\mathcal{O}(h)$ |
|--------------|-------|-----------------|------------------|
| rDG(P0P1)    | 554   | -1.12095        |                  |
|              | 2216  | -1.82147        | 2.327            |
|              | 8864  | -2.44120        | 2.059            |
| DG(P1)       | 554   | -1.47241        |                  |
|              | 2216  | -2.12307        | 2.162            |
|              | 8864  | -2.76388        | 2.129            |
| rDG(P1P2)    | 554   | -2.23718        |                  |
|              | 2216  | -3.24622        | 3.352            |
|              | 8864  | -4.17433        | 3.083            |

Figure 8.1: Velocity magnitude and streamlines for the Kovasznay flow test case
Figure 8.2: Spatial convergence of velocity for Kovasznay flow test case

Figure 8.3: Spatial convergence of pressure for Kovasznay flow test case
Figure 8.4: Errors in velocity for Kovasznay flow plotted against the total number of DOF's
8.3 Taylor vortex

The Taylor vortex flow is an exact solution to the unsteady incompressible Navier-Stokes equations. Similar to the previous test case, the Dirichlet boundary conditions are obtained from the exact solution,

\[ V_x = \frac{1}{10} \pi \cos(0.1\pi x) \sin(0.1\pi y)e^{-0.02\pi^2\nu t} \]
\[ V_y = -\frac{1}{10} \pi \sin(0.1\pi x) \cos(0.1\pi y)e^{-0.02\pi^2\nu t} \]
\[ p = -\frac{1}{400} \pi^2(e^{-0.02\pi^2\nu t})^2(\cos(0.2\pi x) + \cos(0.2\pi y) - 2). \]

Unlike the previous test case, however, since this is an unsteady flow, the initial conditions are obtained from the exact solution as well. This flow is considered on the two-dimensional domain \((0, 10) \times (0, 10)\). We use \(\nu's\) corresponding to \(Re = 10\) and \(Re = 100\) to ensure that the spatial convergence is independent of the viscosity. The same set of meshes used for the Kovasznay flow test were used for this test case. The errors are tabulated in 8.3 and 8.4. From the convergence results of velocity in Figs. 8.5 and 8.7, it is seen that reconstruction of the 2nd order DG(P1) method makes the method 3rd order accurate, along with a considerable improvement in accuracy (magnitude of errors).

To assess the effect of the reconstruction of DG(P1) to P2 on pressure, the convergence of pressure was checked. It is shown in Fig. 8.6. The reconstruction slightly improves the accuracy of the pressure as well, but its order of convergence remains unaffected, as we should expect. Improvement of the order of convergence of pressure will depend on the discretization used for the pressure field. Since we use the second order CG(P1) space for pressure, the second order in its convergence is expected. Nevertheless, the accuracy of pressure is improved due to the improved accuracy of the velocity field, which affects the pressure computation. The velocity and pressure contours are shown in Figs. 8.8 respectively.
Figure 8.5: Spatial convergence of velocity for Taylor vortex with \( \nu = 0.01 \) and \( \nu = 0.1 \)

Figure 8.6: Spatial convergence of pressure for Taylor vortex with \( \nu = 0.01 \) and \( \nu = 0.1 \)
The errors for the Taylor vortex with \( \nu = 0.01 \) and \( \nu = 0.1 \) are tabulated in Tables 8.3 and 8.4 below. The improved spatial order of accuracy is clear.

Table 8.3: Errors for the Taylor vortex test case with \( \nu = 0.01 \)

| Method     | Nelem | \( ||e||_2 \) | \( O(h) \) |
|------------|-------|---------------|------------|
| rDG(P0P1)  | 554   | -2.10625      |            |
|            | 2216  | -2.68475      | 1.922      |
|            | 8864  | -3.24343      | 1.856      |
| DG(P1)     | 554   | -2.20621      |            |
|            | 2216  | -2.84913      | 2.136      |
|            | 8864  | -3.55224      | 2.336      |
| rDG(P1P2)  | 554   | -2.77672      |            |
|            | 2216  | -3.64570      | 2.887      |
|            | 8864  | -4.53029      | 2.939      |

Table 8.4: Errors for the Taylor vortex test case with \( \nu = 0.1 \)

| Method     | Nelem | \( ||e||_2 \) | \( O(h) \) |
|------------|-------|---------------|------------|
| rDG(P0P1)  | 554   | -2.16711      |            |
|            | 2216  | -2.76800      | 1.996      |
|            | 8864  | -3.32454      | 1.849      |
| DG(P1)     | 554   | -2.47761      |            |
|            | 2216  | -3.13027      | 2.168      |
|            | 8864  | -3.76045      | 2.093      |
| rDG(P1P2)  | 554   | -3.42070      |            |
|            | 2216  | -4.36191      | 3.127      |
|            | 8864  | -5.27800      | 3.043      |
Figure 8.7: Errors in velocity plotted against the total number of DOF’s for Taylor vortex with $\nu = 0.01$.

Figure 8.8: Velocity and Pressure contours, $\nu = 0.01$. 
8.4 Forced flow problem

We consider a manufactured solution of the unsteady incompressible Navier-Stokes equations from Liu et al. [26] as another test case. This is a forced flow problem with the exact solution,

\[
\begin{align*}
V_x &= \pi \sin(t) \sin^2(\pi x) \sin(2\pi y) \\
V_y &= -\pi \sin(t) \sin(2\pi x) \sin^2(\pi y) \\
p &= \sin(t) \cos(\pi x) \cos(\pi y)
\end{align*}
\]

and the forcing terms can be computed from the momentum equations. The initial and boundary conditions can be obtained from the exact solution. We use a two-dimensional domain \((0,1) \times (0,1)\) for this test case. We consider \(\nu = 0.01\) and 0.1 for this problem. From Fig. 8.9, it can be seen that the rDG(P1P2) method is 3\(^{rd}\) order accurate, whereas the DG(P1) method is only 2\(^{nd}\) order accurate.

![Figure 8.9: Spatial convergence for velocity for forced-flow flow test case](image-url)
Fig. 8.10 shows the spatial convergence of pressure. It can be seen that the pressure errors behave in an unexpected way for this test case. This might be an effect of the time-step chosen for this case. If very low time-steps are used, the right-hand side of the $\phi$-Poisson equation becomes too large. This might cause errors in the pressure computation.
8.5 Flow over a flat plate

The laminar boundary layer flow across a flat plate is considered in this test case, over a domain bounded by \((-0.5, 1) \times (0, 1)\), and the flat-plate starts at point \((0, 0)\) and extends to \((1, 0)\). This test is carried out at \(Re = 100,000\), with a mesh suitably generated to capture the boundary-layer. The height of the first element is \(0.3464E-03\) and \(0.8265E-03\) at the leading and trailing edge of the plate respectively. The mesh is shown in Fig. C.4 in the appendix. Numerical results obtained from the rDG(P0P1), DG(P1) and rDG(P1P2) using BR2 are compared with the theoretical Blasius solution.

The skin-friction coefficient (Fig. 8.12) and velocity profiles (Figs. 8.14, 8.15, 8.16, 8.17, 8.18, 8.19) across the boundary layer at \(x=0.5, x=0.8\) and \(x=1\) along the flat plate are compared with Blasius’s exact solution. Fig. 8.13 shows a blown up image of the coefficients of friction computed by the 3 methods. It is clear from these images that the higher-order rDG method predicts the coefficient of friction more accurately that the lower-order methods after the boundary layer growth. At the initial stages of the plate, however, the DG(P1) seems to be less accurate that the rDG (P0P1) method. Yet, in this region as well, the rDG(P1P2) is more accurate than the other two. The accuracy of the higher-order reconstruction is evident in the resolution of the y-component of velocities, which is extremely difficult to compute accurately (Figs. 8.15, 8.17, 8.19). Again, we observe that the DG(P1) method has inaccuracies, in that, the velocity profile seems to have some oscillations. This is rectified by the reconstruction, which makes it more accurate than the lower order methods.

Figure 8.11: Velocity contours in the boundary layer
Figure 8.12: Coefficient of friction computed by $rDG(P0P1)$, $DG(P1)$ and $rDG(P1P2)$ compared to Blasius’s solution

Figure 8.13: Coefficient of friction computed by $rDG(P0P1)$, $DG(P1)$ and $rDG(P1P2)$ on a section of the plate $x = 0.01$ to 1
Figure 8.14: X-velocity at $x = 0.5$

Figure 8.15: Y-velocity at $x = 0.5$

Figure 8.16: X-velocity at $x = 0.8$

Figure 8.17: Y-velocity at $x = 0.8$

Figure 8.18: X-velocity at $x = 1.0$

Figure 8.19: Y-velocity at $x = 1.0$
8.6 Driven cavity

The 2-dimensional driven cavity is a benchmark test case used to validate CFD codes. The computational domain is a square with edges of unit length. The top surface (edge in 2D) is imposed with a constant horizontal speed. This drives the fluid inside the square cavity into a vortical flow, resulting in complex flow fields consisting of a large central vortex accompanied by small recirculating zones in the corners. The number of these tiny corner vortices varies with $Re$. This test case therefore provides a means to test the solver at a range of $Re$’s.

The mesh shown in Fig. C.3 was used for these tests. An implicit time-integration with a constant time-step $\Delta t = 0.05$ is used to march the solution in time until a steady-state is reached. The streamlines for $Re = 400$, $1000$, $3200$ and $Re = 10000$ are shown in Fig. 8.21. The velocity profiles are compared with the data published by Ghia et al. [16] to ensure that the obtained results are accurate. The $x$-component of velocity along the vertical centerline and the $y$-component along the horizontal centerline are both shown in Fig. 8.22, 8.23 and 8.24. These results demonstrate the high accuracy attainable for flows at high Reynolds numbers by the rDG-CG discretization.

![Figure 8.20: Streamlines for 2D driven cavity flow at $Re = 400$ (left) $Re = 1000$ (right)]](image)
Figure 8.21: Streamlines for 2D driven cavity flow at $Re = 3200$ (left) $Re = 10000$ (right).

Figure 8.22: $x$-component of velocity along the vertical centerline (left); and $y$-component of velocity along the horizontal centerline (right) compared with results from Ghia et al. [16] represented by ◦'s. $Re = 400$. 
Figure 8.23: $x$-component of velocity along the vertical centerline (left); and $y$-component of velocity along the horizontal centerline (right) compared with results from Ghia et al. [16] represented by ◦’s. $Re = 1000$.

Figure 8.24: $x$-component of velocity along the vertical centerline (left); and $y$-component of velocity along the horizontal centerline (right) compared with results from Ghia et al. [16] represented by ◦’s. $Re = 3200$. 
Chapter 9

Conclusions

9.1 Summary of completed work

A hybrid reconstructed discontinuous Galerkin (rDG) and continuous Galerkin (CG) spatial discretization for the projection method has been developed to solve the 2D incompressible Navier-Stokes equations; where velocity is in the discontinuous finite element space, and pressure is in the continuous finite element space. The projection method by Temam [37] and Timmermans [38] in its incremental form has been used in this work.

The reconstruction of the velocity is done in a least-squares sense to improve the spatial order of accuracy in space from $P_n$ to $P_m$ ($m \geq n$), where the reconstruction reduces to the identity operator when $m = n$; leading to the name rDG$(P_nP_m)$. Taylor-bases are used to express the solution polynomial in the discontinuous Galerkin space. The advective fluxes are computed using an upwind-scheme, whereas the diffusive fluxes are computed using BR2. Both P1 and P2 polynomial spaces have been used for the pressure field. However, due to the excessive computation costs for P2 without any significant improvement in the accuracy of the velocity field, the P1 polynomial-space was used to obtain most of the results. An edge-based data-storage is used to store the Laplacian for the $\phi$-Poisson equation, significantly reducing the memory requirements computation time.

The explicit forward-Euler formula is used for discretizing the time-derivative for solving unsteady flows. For steady flows on the other hand, the implicit backward-Euler formula is used for discretizing the time-derivative to accelerate the convergence. A linearization is carried out for this purpose. The flux-Jacobian in the linearized system is computed by the finite-difference (divided differencing) approach. This system is solved using GMRES using block-Jacobi preprocessing. The $\phi$-Poisson equation is solved using the Conjugate-Gradient (CG) solver preconditioned using a symmetric Gauss-Seidel sweep.

This work makes significant contributions towards the development of a fully-implicit high-
order DG solver for incompressible flows on unstructured triangular grids:

- A hybrid reconstructed-DG and continuous FEM method is developed for the numerical solution of incompressible flows on unstructured grids. The reconstruction improves the usual $O(h^2)$ of the DG(P1) method to $O(h^3)$, making it a 3rd order method.

- This improvement is achieved with the same number of degrees of freedom, with a very small increase in computation time.

- A large speedup is observed when the linearized implicit formulation is used for obtaining converged solution for steady flows.

9.2 Future work

The work presented in this thesis serves as a beginning to further more complex flow studies. Preliminary developments in this work would be the inclusion of hybrid-mesh followed by 3D flow simulation capabilities. Even after these additions, this work is just the foundation for the study of more complex incompressible flow physics such as multiple phases, turbulent flows, moving-mesh, etc.

A large number of models have been developed for numerical simulation of turbulent flows. Beginning with the one-equation model by Spalart and Allmaras [36], turbulence modeling has progressed to seven-equation Reynolds stress models [24]. Direct Numerical Simulation (DNS) [21, 35] is a simulation which a whole range of spatial and temporal scales of turbulence are resolved. As expected, the computational costs of DNS are prohibitive. As a optimum between modeling and full resolution, the Large-Eddy Simulations (LES) is also used, where eddies up to a particular length-scale are resolved, and modeled below that. These models have been incorporated into discontinuous Galerkin codes [12]. When implemented, these will allow the code to simulate complex turbulent flows. In addition to these, to improve overall performance of the flow solver by reducing the time require to obtain a converged solution, parallel computing is necessary.

These additions would enable the solver to handle more complex and challenging problems based on real applications, and should be included in the code in the near future.
REFERENCES


APPENDICES
Appendix A

Gaussian quadrature formulae

The Gaussian quadrature formulae for 1D line element and 2D triangular element are included here.

A.1 Line

The quadrature formula is:

\[ \int_{\Gamma_e} f d\Gamma = \frac{\Gamma_e}{2} \sum_{i=1}^{r} w_i f (\xi_i, \eta_i) \]  (A.1)

<table>
<thead>
<tr>
<th>Order ( p )</th>
<th>No.of points ( r )</th>
<th>Coordinates ( (\xi_i) )</th>
<th>Weights ( (w_i) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0.000000000000000</td>
<td>2.000000000000000</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>-0.577350269189626</td>
<td>1.000000000000000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.577350269189626</td>
<td>1.000000000000000</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>-0.774596669241483</td>
<td>0.555555555555556</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.000000000000000</td>
<td>0.888888888888889</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.774596669241483</td>
<td>0.555555555555556</td>
</tr>
</tbody>
</table>

Table A.1: Constants for Gaussian quadrature for a line
A.2 Triangle

The quadrature formula is:

$$\int_{\Omega_e} f d\Omega = \Omega_e \sum_{i=1}^{r} w_i f(\xi_i, \eta_i) \quad (A.2)$$

Table A.2: Constants for Gaussian quadrature for a triangle

<table>
<thead>
<tr>
<th>Order $p$</th>
<th>No.of points $r$</th>
<th>Coordinates ($\xi_i, \eta_i$)</th>
<th>Weights ($w_i$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>(0.333333333333333, 0.333333333333333)</td>
<td>1.000000000000000</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>(0.666666666666667, 0.166666666666667)</td>
<td>0.166666666666667</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(0.666666666666667, 0.666666666666667)</td>
<td>0.166666666666667</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(0.166666666666667, 0.666666666666667)</td>
<td>0.166666666666667</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>(0.333333333333333, 0.333333333333333)</td>
<td>-0.281250000000000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(0.200000000000000, 0.200000000000000)</td>
<td>0.260416666666667</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(0.600000000000000, 0.200000000000000)</td>
<td>0.260416666666667</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(0.200000000000000, 0.600000000000000)</td>
<td>0.260416666666667</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>(0.333333333333333, 0.333333333333333)</td>
<td>0.112500000000000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(0.059715871789770, 0.470142064105115)</td>
<td>0.066197076394253</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(0.470142064105115, 0.059715871789770)</td>
<td>0.066197076394253</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(0.470142064105115, 0.470142064105115)</td>
<td>0.066197076394253</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(0.797426985353087, 0.101286507323456)</td>
<td>0.062969590272414</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(0.101286507323456, 0.797426985353087)</td>
<td>0.062969590272414</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(0.101286507323456, 0.101286507323456)</td>
<td>0.062969590272414</td>
</tr>
</tbody>
</table>
Appendix B

Edge-based data-storage for the stiffness matrix of the Poisson equation in P1-polynomial space

The stiffness matrix is sparse due to the compact support of the FEM. Only the non-zero parts of the stiffness matrix are stored instead of the full npoin × npoin matrix. The edge-based data structure is used for this purpose.

This is done using two arrays: diago, which stores the diagonal entries; and upper, which stores the strictly-upper-triangular entries. The lower-triangular entries need not be stored since the stiffness matrix is symmetric. The upper array has number of entries equal to the number of edges nedge, whereas the diago array has number of entries equal to npoin. The data-transfer from the elemental stiffness matrix is shown in Fig. B.1 by means of arrows. For example the upper(IJ1) can be written as,

\[ \text{upper}(IJ1) = (\nabla B_1 \cdot \nabla B_2)_{i1} + (\nabla B_2 \cdot \nabla B_3)_{i2}, \]

and the diago(I) can be written as,

\[ \text{diago}(I) = - \sum_{k \in \text{edgsu}(I)} \text{upper}(Ik) \]
\[ = -\left( \text{upper}(IJ1) + \text{upper}(IJ2) + \text{upper}(IJ3) + \cdots \right), \]

where edgsu(I) is the set of edges surrounding point I.

Consider the mesh shown in Fig. B.2 as an example. The node-based system would give the
stiffness matrix as,

\[ A(I, I) = a^{ie_1}_{(1,1)} + a^{ie_2}_{(2,2)} + a^{ie_3}_{(3,3)}, \]
\[ A(I, J_1) = a^{ie_1}_{(1,2)} + a^{ie_3}_{(3,2)}, \]
\[ \ldots \]

where, \( A \) is the global stiffness matrix, \( a^{ie} \) is the elemental stiffness matrix for element \( ie \) whose subscripts indicate the row and column based on the local node numbering in that element. The edge-based system will express the stiffness matrix as,

\[ upper(IJ1) = a^{ie_1}_{(2,1)} + a^{ie_3}_{(3,2)}, \]
\[ upper(IJ2) = a^{ie_1}_{(3,1)} + a^{ie_2}_{(2,3)}, \]
\[ upper(IJ3) = a^{ie_2}_{(1,2)} + a^{ie_3}_{(1,3)}, \]
\[ diago(I) = - \left( upper(IJ1) + upper(IJ2) + upper(IJ3) \right) \]
Both $\text{diago}$ with $A(I, I)$ are equal, this can be proved as follows:

$$
\text{diago}(I) = -\left( (a_{(2,1)}^{ie1} + a_{(2,3)}^{ie3}) + (a_{(3,1)}^{ie1} + a_{(3,2)}^{ie2}) + (a_{(1,2)}^{ie2} + a_{(1,3)}^{ie3}) \right)
$$

but,

$$
- a_{(i,j)}^e - a_{(i,k)}^e = a_{(i,i)}^e
$$

∴ $\text{diago}(I) = a_{(1,1)}^{ie1} + a_{(2,2)}^{ie2} + a_{(3,3)}^{ie3}$

∴ $\text{diago}(I) = A(I, I)$

The equivalence of $\text{upper}(IJ1)$ and $A(I, J1)$ is also quite obvious,

$$
\text{upper}(IJ1) = a_{(1,2)}^{ie1} + a_{(3,2)}^{ie3}
$$

∴ $\text{upper}(IJ1) = A(I, J1)$

This kind of storage saves a lot of memory and computation, making the code very efficient. Unfortunately, this cannot be used for P2 elements, where the computation time is approximately doubled because of the double number of nodes.
Appendix C

Meshes used for numerical tests

The coarsest mesh used for the spatial accuracy tests in §8.2, §8.3 and §8.4.

Figure C.1: Coarse mesh with 554 elements
The finer mesh used for the spatial accuracy tests in §8.2, §8.3 and §8.4.

Figure C.2: Fine mesh with 2216 elements
The finest mesh used for the spatial accuracy tests in §8.2, §8.3 and §8.4.

Figure C.3: Finest mesh with 8864 elements
Figure C.4: Mesh used to solve the flat plate problem