Imperial College London

Implicit Time Integration for High-order Compressible Flow Solvers

Yu Pan

Department of Aeronautics
Imperial College London

Submitted in part fulfilment of the requirements
for the degree of Doctor of Philosophy at Imperial College London
June 2, 2022

Declaration of originality

This is to certify that the work presented in this thesis has been carried out at Imperial College London and has not been previously submitted to any other university or technical institution for a degree or award. The thesis comprises only my original work, except where due acknowledgement is made in the text.

Yu Pan (2022)

Copyright declaration

Acknowledgements

I want to thank my supervisors, Profs. Spencer Sherwin and Joaquim Peiró, who guide me to CFD field in a deeper sight. They not only care about the progress of my research but also care about my living in London. I am also impressed by their profound knowledge and strict attitude to scientific researches.

I have to give special thanks for my darling parents. They gave me selfless and endless support when I felt confused in study. I have to thank my warm-hearted friends, Chen Xu and Tai-hang Zhu, who gave me support in life. I also need to give my gratitudes to other colleagues and professors during my learning, Dr. David Moxey, Dr. Chris Cantwell, Prof. Koen Hillewaert, Prof. Mike Kirby, Dr. Hui Xu, Dr. Yan Zhang, Dr. Giacomo Castiglioni and Dr. Gianmarco Mengaldo. They have given me solid support on academic studies. Lastly and most importantly, great thanks go to my senior, my friend Zhen-guo Yan, who worked as my third supervisor and gave me endless academic help. This work would not have been possible without their kind help.

It has been a wonderful experience studying at Imperial College London. I thank my colleagues, friends, and everyone who have been part of the journey in my life.

Abstract

The application of high-order spectral/hp element discontinuous Galerkin (DG) methods to unsteady compressible flow simulations has gained increasing popularity. However, the time step is seriously restricted when high-order methods are applied to an explicit solver. To eliminate this restriction, an implicit high-order compressible flow solver is developed using DG methods for spatial discretization, diagonally implicit Runge-Kutta methods for temporal discretization, and the Jacobian-free Newton-Krylov method as its nonlinear solver. To accelerate convergence, a block relaxed Jacobi preconditioner is partially matrix-free implementation with a hybrid calculation of analytical and numerical Jacobian.

The problem of too many user-defined parameters within the implicit solver is then studied. A systematic framework of adaptive strategies is designed to relax the difficulty of parameter choices. The adaptive time-stepping strategy is based on the observation that in a fixed mesh simulation, when the total error is dominated by the spatial error, further decreasing of temporal error through decreasing the time step cannot help increase accuracy but only slow down the solver. Based on a similar error analysis, an adaptive Newton tolerance is proposed based on the idea that the iterative error should be smaller than the temporal error to guarantee temporal accuracy. An adaptive strategy to update the preconditioner based on the Krylov solver's convergence state is also discussed. Finally, an adaptive implicit solver is developed that eliminates the need for repeated tests of tunning parameters, whose accuracy and efficiency are verified in various steady/unsteady simulations.

An improved shock-capturing strategy is also proposed when the implicit solver is applied to high-speed simulations. Through comparisons among the forms of three popular artificial viscosities, we identify the importance of the density term and add density-related terms on the original bulk-stress based artificial viscosity. To stabilize the simulations involving strong shear layers, we design an extra shear-stress based artificial viscosity. The new shock-capturing strategy helps dissipate oscillations at shocks but has negligible dissipation in smooth regions.

List of Work

[1]This work is related to Chapters 2 and 3, which introduces how to develop an efficient implicit compressible flow solver within Nektar++.

Zhen-Guo Yan, **Yu Pan**, Giacomo Castiglioni, Koen Hillewaert, Joaquim Peiró, David Moxey, and Spencer Sherwin. Nektar++: Design and implementation of an implicit, spectral/hp element, compressible flow solver using a Jacobian-free Newton-Krylov approach. Computers Mathematics with Applications, 81:351-372, January 2021.

[2] This work is related to Chapter 4, which provides a framework of adaptive strategies to choose parameters within the implicit solver.

Yu Pan, Zhen-Guo Yan, Joaquim Peiró, and Spencer Sherwin. Development of a balanced adaptive time-stepping strategy based on an implicit JFNK-DG compressible flow solver. Communications on Applied Mathematics and Computation, DOI : 10.1007/s42967-021-00138-1, 2021.

[3] **Yu Pan**, Zhen-Guo Yan, Joaquim Peiró, and Spencer Sherwin. Analysis of accuracy and efficiency of implicit time integration schemes. In European workshop on high order numerical methods for evolutionary PDEs: Theory and applications (HONOM 2019), Madrid, Spain, April 2019.

[4] Zhen-Guo Yan, **Yu Pan**, Joaquim Peiró, and Spencer Sherwin. Accelerating spectral/hp element DG simulations using implicit time integration methods. In European workshop on high order numerical methods for evolutionary PDEs: Theory and applications (HONOM 2019), Madrid, Spain, April 2019.9.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The project aims to develop a solver for simulating unsteady compressible flows efficiently and accurately. To achieve this goal, Section 1.1 discusses why a solver that combines spectral/hp methods and implicit time-integration methods is a good candidate. Sections 1.2 and 1.3 review the underlying foundations of the implicit solver: spatial discretization, temporal discretization, nonlinear solver, linear solver, and preconditioner. Section 1.4 points out three challenges likely encountered in the design of an efficient implicit solver: (a) lack of an efficient preconditioner, (b) complex parameter choices within the solver, and (c) lack of an effective shock-capturing strategy. The objective of this thesis' work is to address these difficulties, in the manner presented in Section 1.5. Lastly, the outline of the thesis is given in Section 1.6.

1.1 Why an implicit high-order solver?

The development of computational fluid dynamics (CFD) has focused on steady flow simulations for a long time. During the last several decades, the dramatical increase in computational power has made it practical to simulate complicated unsteady flows. To predict noises accurately in aircraft design (Lo et al., 2010; Zaporozhets et al., 2019), to understand the complicated unsteadiness and acoustic

waves in turbo-machinery flows (Roy, 2005; Tyacke et al., 2019), to catch small-scale physics within high Reynolds number boundary layer flows (Mengaldo, 2015), high-order methods are more and more appealing to simulate these phenomena, because they exhibit low dispersion and dissipation errors (Wang et al., 2013; Marty et al., 2015; De Laborderie et al., 2018). We focus on the studies of high-order spectral/hp element discontinuous Galerkin (DG) methods. The bases and variables are independently introduced in each element. The variables and their derivatives are only exchanged at elemental interfaces, and there is no requirement to reconstruct the variables using neighboring cells' information, which makes DG relatively compact. The compactness brings advantages such as easy extension to higher-order, convenient application using unstructured meshes, large-scale problem parallelization, hp-adaptivity, etc, (Karniadakis and Sherwin, 2013; Cassinelli et al., 2018). However, additional problems arise from the application of high-order DG methods to an explicit solver. The time step size of explicit time integration schemes is severely constrained by the CFL stability condition, especially in high order methods (Persson and Peraire, 2006a; Cherednichenko et al., 2012). The time step is more constrained with the increase of the order of spatial discretization P, which roughly decreases with P^2 (Karniadakis and Sherwin, 2013). Therefore, implicit schemes are preferable since they are not bound by stability constraints as explicit schemes.

The implementation of an implicit time integration scheme combined with high-order DG methods to a compressible flow solver also meets several practical challenges. The large storage requirement arising from the linearization of the compressible Navier-Stokes equations restricts the usage in large-scale problems. Additionally, an implicit solver generally is much more complex than an explicit solver. A number of user-defined parameters exist within the loops of an implicit solver, which both influence its efficiency and accuracy. An efficient implicit solver should overcome these problems.

1.2 Spectral/hp element methods

Finite element methods (FEM) are popular both in structural and fluid mechanics. In FEM, the computational domain is partitioned into a set of sub-elements, within each element the local solutions are represented by specific shape functions. The procedure for solving FEM problems is to find the unknown coefficients of the shape functions that can minimize the error norm between the local approximation and the original problem.

A high-order FEM utilizes higher-order polynomial bases as shape functions. The elemental size is denoted h and the order of the polynomial is denoted P. Spectral/hp element methods can be summarized as follows:

a. **h-type finite element method:** h-type FEM is to keep polynomial order P fixed and decrease h to get higher accuracy. Classical FEM using linear equations can be classified as a first-order h-type FEM when solving a smooth problem. h-type refinement can be used around complicated geometry to better describe the geometry shape.

b. **p-type finite element method:** In contrast, p-type FEM keeps h fixed and uses higher P^{th} order polynomial bases to do refinement and get accurate solutions.

c. **Spectral element method:** Spectral discretization is featured that multi-dimensional discretization can be formulated as a tensor product of one-dimensional discretization. The spectral method originally uses a single representation of function throughout the domain (Canuto, 2006). Typically, this method uses orthogonal bases as shape functions such as Fourier, Chebyshev, or Legendre series. Patera (1984) proposed the definition of the spectral elemental method that combines the spectral method and FEM. The spectral element method can be seen as a higher-order FEM.

d. **Spectral/hp element method:** Spectral/hp element method can be seen as

the combination of all the methods mentioned above (Karniadakis and Sherwin, 2013). The flexible choice of bases enables spectral/hp element method competitive in various fields.

1.2.1 Development of DG methods

Among the various spectral/hp element methods, one widely-adopted family of methods, discontinuous Galerkin (DG) methods, converts a continuous problem into a discrete formulation. Specifically, DG methods are based on the weak formulation and represented by piecewise polynomials, where the polynomials are discontinuous at the element interfaces. The flow solutions and derivatives are only exchanged at the element interfaces, which makes DG schemes very compact. The compactness of DG brings advantages in extension to higher-order, large-scale problem parallelization, *hp*-adaptivity, etc.

The DG method was first proposed in (Reed and Hill, 1973) for solving the neutron transport equation. Then the first numerical analysis of the DG method was carried out in (Lesaint and Raviart, 1974). Ever since DG methods have achieved a booming development. However, the DG methods in hyperbolic, elliptic, and parabolic problems were developed almost independently.

In the 1990s, Cockburn and his coworkers proposed a series of DG methods combined with explicit Runge-Kutta temporal discretization methods (RKDG) and applied them to nonlinear conservation laws (Cockburn et al., 1989, 1990; Cockburn and Shu, 1989, 1991). RKDG was further developed for solving hyperbolic problems, such as (DeCougny et al., 1994; Bassi and Rebay, 1997b; Cockburn and Shu, 2001). Among these schemes, the Bassi-Rebay (BR) scheme proposed in (Bassi and Rebay, 1997a) was a milestone that solves compressible Navier-Stokes equation by treating the solution itself and its gradient as independent variables. However, BR scheme's simple choice for treating the double-valued solutions and gradients at element interfaces are identical was found to be problematic in solving the Poisson problem (Brezzi et al., 1999), in which the existence of the approximate solution can

not be guaranteed. A modified version of BR scheme with additional stabilization terms was proposed in (Bassi et al., 1997), which is referred to as BR2 scheme. Similarly, Cockburn and Shu introduced a generalization of local discontinuous Galerkin (LDG) methods adding different versions of stability terms to the fluxes (Cockburn, 1998; Cockburn and Shu, 1998a; Cockburn and Dawson, 1999). The 'local' property is because the numerical solution flux does not depend on the gradient flux through global lifting. Therefore, second-order derivatives can be separated into two decoupled first-order derivatives. LDG can be extended to solving higher-order equations in a similar way. However, global lifting leads to a larger stencil in multi-dimensional problems. A variation of LDG, named compact DG (CDG) proposed in (Peraire and Persson, 2008) overcomes this issue and keeps the compactness through local lifting. Besides, Van Leer et al. (2007) proposed a recovery-based DG (RDG) method for diffusion equations that recovers smooth continuous solutions from discontinuous discrete solutions in the weak sense. Recently, a wide group of hybridisable DG (HDG) has also been developed (Nguyen et al., 2009; Qiu and Shi, 2016; Sevilla and Huerta, 2018) in which the globally coupled degrees of freedom are only approximated by the solutions defined on the boundaries of the elements, thus reducing the number of degrees of freedom.

Another group of independently developed DG methods to solve second-order elliptic and parabolic problems is the interior penalty (IP) methods. Inspired by the observation that Dirichlet boundaries can be weakly imposed instead of being built into finite element space (Arnold et al., 2002), the requirement of continuity of approximate solutions is satisfied through the addition of boundary and interior penalty terms. In addition to the favorable properties of DG methods, IP methods have a smaller stencil and are relatively easier to implement within an implicit solver. Various versions of IP are developed, such as symmetric interior penalty Galerkin (SIPG) method (Hartmann and Houston, 2006b), non-symmetric interior penalty Galerkin (NIPG) method (Rivière et al., 1999, 2001) and incomplete interior penalty Galerkin (IIPG) method (Sun, 2003). SIPG adds symmetry properties to

satisfy adjoint consistency (Arnold et al., 2002), which is critical to guarantee the optimal order of convergence using different orders of DG approximations. Standard IP formulations are not adjoint consistent even for symmetric problems, which leads to optimal convergence if the polynomial degree is odd and only suboptimal convergence if the polynomial degree is even (Rivire, 2008).

The fundamental review of DG by (Arnold et al., 2002) summarized most of the DG schemes mentioned above and proposed a uniform analysis.

1.2.2 Applications of DG in compressible flow simulations

The application of DG methods to the solution of compressible flows has gained increasing popularity since the 1990s. The application of BR or BR2 schemes in solving the compressible Navier-Stokes equations were proposed in (Bassi and Rebay, 1997a; Bassi et al., 1997). The diffusion part of Navier-Stokes equations was solved in a similar way in (Lomtev et al., 1998), but the solution and its gradient were solved in a coupled way within a large system. The extensions of IP methods to compressible Navier-Stokes were proposed in papers such as (Hartmann and Houston, 2006a,b).

The increasing popularity of DG schemes used in compressible problems is due to their inherent advantages. As defined in the previous section, DG methods approximate solutions using piecewise polynomials, which are similar to finite element methods (FEM). When solving the convection term, DG methods construct the numerical fluxes, which are similar to finite volume methods (FVM). Therefore, DG methods can be seen as a combination of FEM and FVM. Compared with DG methods, FVM encounter increasing difficulty to reconstruct a higher-order scheme, especially when using unstructured meshes. The accuracy order of the approximations using DG methods can be easily increased. Standard conforming FEM are suitable for solving elliptic, parabolic problems, and incompressible flows, where the solutions are regular. The resolutions of discontinuous solutions in compressible flows become complicated (Feistauer et al., 2003). In contrast, DG methods allow achieving high-order accuracy in a natural way with the aid of stabilization methods,

such as the shock-capturing method proposed in (Persson and Peraire, 2006b).

These advantages promote the usage of the DG methods in a wide range of practical compressible problems. The accuracy and efficiency of DG methods are investigated in (Bassi and Rebay, 2000; Bassi et al., 2005). The applications of DG to turbo-machinery simulations are popular due to DG methods' high accuracy. The ability of DG methods to simulate small-scale physics within high Reynolds number boundary layer flows is also discussed in (Landmann et al., 2008; Bolemann et al., 2015). Other applications of DG methods to compressible flow simulations can be found in (Kroll et al., 2010; Hirsch et al., 2021).

1.2.3 Shock-capturing methods

When high-order numerical methods are applied to simulate high-speed flows with shock waves and contact discontinuities, Gibbs phenomena manifested by oscillations at discontinuities are observed. These phenomena produce inaccurate even unstable numerical solutions that can ruin the computational process. Most numerical schemes to deal with shocks are based on numerical dissipation. Such techniques are called shock-capturing methods. The alternative is to use shock fitting methods, where the shock shapes, positions, and velocity are explicitly determined (Johnsen et al., 2010; Rawat and Zhong, 2011; Bonfiglioli and Paciorri, 2014).

A strategy used commonly in shock-capturing methods is limiting, where the shape of the solution is modified to retain properties such as positivity and freedom from spurious oscillations to retain various orders of accuracy. The computational cost will increase with the increase of the order of approximating polynomial (Cockburn and Shu, 1998b; Krivodonova, 2007; Klöckner et al., 2011). Therefore, these methods are more widely used in low-order simulations.

On the other hand, artificial viscosity methods are widely adopted in high-order DG methods. A locally-varying viscosity is added, which should ideally retain high-order accuracy in the presence of smooth solutions (Discacciati et al., 2020). These methods use sensors to identify discontinuities by detecting the entropy production

or estimating the decay rate of modal coefficients. Artificial diffusion is then added where it should dissipate. Artificial viscosity methods can date back to the vanishing viscosity solution of discontinuous flows (LeVeque, 1992), but can be generalized to the discontinuity regularization of various equations. Ever since the pioneering work of (VonNeumann and Richtmyer, 1950), significant progress has been made in their development and application. A renowned example is the artificial viscosity method developed by Jameson, Schmidt, and Turkel (Jameson and Mavriplis, 1986; Jameson, 2017) for low-order finite difference and finite volume methods.

The work in (Hartmann and Houston, 2002) and (Aliabadi et al., 2004) proposed artificial viscosity methods for the DG method, that only for first-order polynomial solutions. Persson and Peraire (2006b) exploited the use of artificial viscosity for higher-order DG schemes, considering the amount of artificial viscosity approximated by a P^{th} order polynomial to resolve a shock profile is only $\mathrm{O}(h/P)$ rather than the scale of element size h. Further, Fernandez et al. (2018) suggested smoothing the artificial viscosity field to ensure C^0 continuity is critical for robustness.

The artificial viscosity methods mainly include two components: a sensor and a consistent PDE-based dissipation term. The former is designed to detect if discontinuity exists in certain regions. Meanwhile, the artificial viscous flux determines the form and amount of artificial viscosity to each equation of the governing system.

Sensors detect when and where to add the artificial viscosity, thus very important for an efficient shock-capturing scheme. The construction of sensors can be based on physical and mathematical discontinuities. Shock sensors can take advantage of the compression properties of a shock wave such as the dilatation-based sensors (Barter and Darmofal, 2010; Nguyen and Peraire, 2011; Moro et al., 2016). Thermal sensors that detect the irregularities arising from thermal gradients (Cook, 2013; Fernandez et al., 2018). Alternatively, a number of methods rely on detecting the oscillations of the numerical solution itself. As mentioned in (Klöckner et al., 2011), the coefficients should decay sufficiently quickly for a smooth function. Based on this mathematical property of smooth flows, the highest modal decay (MDH) model (Persson and

Peraire, 2006b) detects the decay rate of coefficients in polynomial modal space. An entropy inequality can also be an indicator of shocks. The entropy satisfies a conservation equation only when solutions are smooth and satisfies an inequality in shocks. A discussion of these entropy viscosity methods can be found in (Guermond et al., 2011; Tonicello et al., 2020).

Various forms of artificial viscous fluxes can be constructed based on shear-stress tensors, bulk-stress tensors, or Laplacian operators. The shear-stress based artificial viscosity is simple to implement because the artificial viscous flux is consistent with the Navier-Stokes system and additional artificial viscosity coefficient can be directly added to the physical viscosity. The bulk-stress based artificial viscosity was firstly used to accelerate the convergence of flow solvers in steady incompressible flow simulations (Ramshaw and Mousseau, 1990). This term has no effect on the steady solution based on the fact that it is proportional to the divergence of velocity $\nabla \cdot \mathbf{u}$ that vanishes at steady state. This idea was then extended to the compressible Euler (Mazaheri and Roe, 2003) and Navier-Stokes equations (Alzaeili and Mazaheri, 2006), where the bulk-stress based artificial viscosity is formulated in $\nabla \cdot (\rho \mathbf{u})$ so that it vanishes at steady state. The Laplacian type artificial viscosity performs well in stabilizing flows with shocks in practice, as illustrated by (Nguyen and Peraire, 2011; Hartmann, 2013). However, Persson and Peraire (2006b) pointed out this approach is not very effective in discriminating between shocks and contact discontinuities in which the density might be discontinuous but the pressure and normal velocities are continuous. Based on the comparisons of different forms of artificial viscous fluxes, we develop a new shock-capturing strategy by selecting their best features in Section 5.

1.3 Implicit time integration methods

An implicit compressible Navier-Stokes solver not only includes the time integration discretization, but also consists of several sub-iterations such as a nonlinear equation

system, a linear equation system, and a preconditioning process. We will discuss the temporal discretization in Section 1.3.1 and the Jacobian-free Newton-Krylov (JFNK) method as well as the preconditioning in Section 1.3.2.

1.3.1 Implicit time integration schemes

The use of the method of lines (Berezin and Zhidkov, 1962) allows the application and analysis of spatial discretization and temporal discretization independently. Regarding time integration, both explicit and implicit schemes have been widely adopted. The solvers using explicit schemes are convenient to implement and take up relatively less storage. The most appealing explicit time integration discretization is the family of explicit Runge-Kutta (RK) schemes (Gustafsson, 1991; Owren and Zennaro, 1992; Kennedy et al., 2000). Higher-order RK schemes are easy to construct under some specific rules, and their stability region can also be enlarged. However, all the explicit schemes suffer from a Courant-Friedrichs-Lewy (CFL) stability condition (Bücker et al., 2009). Thus the time step size is restricted, especially in high-Reynolds number simulations or when highly-stretched meshes are used. In the contrast, implicit time integration schemes can relax or even overcome this stability restriction (Alexander, 1977; Kennedy and Carpenter, 2016). The benefits of implicit schemes in steady flow simulations are more obvious where time has no physical meaning. The accuracy of the time integration does not influence the accuracy of the steady-state solution (van Buuren René, 1999). The applications of implicit schemes in steady flows can achieve a considerable speed up.

Among implicit schemes, multi-step (Nigro et al., 2014; Wang and Rahmani, 2021) and multi-stage (Blom et al., 2016; Kennedy and Carpenter, 2016) schemes are popular choices with wide application.

Among multi-step schemes, the first-order backward differentiation formulation (BDF1) is appealing in engineering problems when the accuracy requirement is not strict while the second-order scheme (BDF2) is also widely-adopted in the industry due to its efficiency and relatively higher accuracy. However, the BDF schemes

of higher than second-order have limited stability properties, for instance, BDF3 is problematic in the presence of convection (Bijl et al., 2002) and BDF4 seldom remains stable in large-scale problems (Melson et al., 1993). The most attractive property of BDF schemes is that only one nonlinear equation system needs to be resolved per time step.

Among multi-stage schemes, the most popular is the family of implicit Runge-Kutta (IRK) schemes (Butcher, 2016; Jörgensen et al., 2018; Kennedy and Carpenter, 2019). The diagonally implicit Runge-Kutta (DIRK) family of methods is widely used in practical applications due to its relatively easy implementation. As the coefficients of Runge-Kutta schemes are usually aligned in a Butcher array (Butcher, 2016), DIRK schemes are characterized by a lower triangular Butcher array with at least one nonzero diagonal entry. This permits solving each stage's nonlinear system individually rather than simultaneously for all the stages. We focus on the use of singly diagonally implicit Runge-Kutta (SDIRK) schemes (Butcher, 1964; Butcher and Diamantakis, 1998; Ferracina and Spijker, 2008). The diagonal coefficients are identical, which permits the reuse of the similar linearization in the preconditioner (Benzi, 2002) over all sub-stages. Explicit singly diagonally implicit Runge-Kutta (ESDIRK) schemes (Jörgensen et al., 2018) are also widely-used since the first stage is free and saves computational cost. In practical experience, ESDIRK schemes not only retain the stability properties of IRK schemes but at significantly lower computational costs. There has been comprehensive research on the modified versions of DIRK schemes, such as the stiffly accurate DIRK schemes that can avoid order reduction in stiff problems (Prothero and Robinson, 1974), low storage versions which do not store the flux derivatives of all the stages (Higueras and Roldán, 2018), embedded versions used in time step adaptivity (Gustafsson, 1992, 1994; Söderlind, 2002), etc. We pay special attention to the embedded versions of ESDIRK proposed in (Kvrn, 2004). Firstly, the error term of some time integration schemes depends on the stiffness of the problem and results in the reduction of convergence order in stiff problems (Burrage and Petzold, 1990). ESDIRK schemes are often constructed

in a stiffly accurate formula to avoid order reduction. Secondly, embedded schemes estimate the temporal errors at a low cost by comparing the solutions approximated using two different orders of schemes. The time step is then adjusted through an error controller. Unlike most embedded ESDIRK schemes where the solution approximated by the embedded scheme is one order lower than the solution approximated by the advanced scheme, the schemes in (Kvrn, 2004) utilize a higher-order embedded scheme, and thus can be used to estimate the temporal errors accurately. Based on this family of schemes, we propose a new adaptive time-stepping strategy, which will be introduced in Sections 2.3 and 4.1.

1.3.2 Jacobian-free Newton-Krylov method

After the implicit temporal discretization (BDF/IRK) of the compressible Navier-Stokes system, the solution of the resulting nonlinear equation introduces a nonlinear iteration. There are two widely used methods: nonlinear multigrid methods (Brandt, 1977; Wesseling, 1995; Trottenberg et al., 2000), and Netwon-Krylov methods (Chan and Jackson, 1984; Brown and Saad, 1990; Kelley, 1995). When the ideas of multigrid, 'use coarse grid approximation to accelerate convergence in the fine grid', are directly applied to the nonlinear system, these methods are also called full approximation scheme (FAS) (Brandt, 1977). Firstly, the errors on the fine grid need to be pre-smoothed. Then the governing equation is projected to the coarse grid. After the governing equation is solved on the coarse grid, the coarse grid corrections are projected back to the solutions on the fine grid. Compared to the global linearization required by Newton-type nonlinear solvers, the process of FAS involves the projection operator, the smoothing operator, and the solving process on the coarse grid. The drawbacks of FAS are the need for forming a set of hierarchical meshes, which is not available by most CFD solvers. Furthermore, the optimal convergence of FAS cannot always be guaranteed (Wesseling, 1995; Mavriplis, 1998). In Newton-Krylov methods, there are at least two loops: the primal loop is the Newton-type nonlinear iteration, and the inner loop is using a Krylov linear solver to calculate the Newton

corrections. Generally, the outer Newton iteration is always inexact (Kelley, 2003) and quadratic convergence cannot be guaranteed. The asymptotic quadratic convergence is achievable and is still faster than other nonlinear solvers such as dichotomy, chord method, etc. The link between the nonlinear Newton solver and the Krylov linear solver is the globally linearized matrix, namely the Jacobian matrix, takes up much storage. Fortunately, the Jacobian matrix is only needed in matrix-vector products during solving the linear equation in Krylov space. Jacobian-free technique stores the products using a finite difference approximation, which avoids the saving of the whole Jacobian matrix. The whole nonlinear solving process using the Newton-type solver, the Krylov solver and the Jacobian-free method is called the Jacobian-free Newton-Krylov (JFNK) method. A more detailed introduction to the JFNK method can be found in the review (Knoll and Keyes, 2004).

As long as the Jacobian matrix is nonsingular, the calculation of Newton corrections is a standard linear equation process. If the matrix is order n, the computational cost is $O(n^3)$ and the storage requirement is $O(n^2)$ for a direct method such as the Gaussian elimination. It is prohibitive for a large-scale sparse system arising from a practical CFD problem. Alternatively, when using Krylov solvers, the solution to the linear system lies in a Krylov space whose dimension is the degree of the minimal polynomial of the Jacobian matrix (Campbell et al., 1996; Ipsen and Meyer, 1998). Therefore, if the minimal polynomial has low degrees then the space in which the Krylov method searches for a solution that could be small. A relatively accurate solution can be achieved at a low computational cost and storage. Another benefit of using Krylov solvers is that they require only matrix-vector products and there is no need to store a total matrix.

Krylov solvers include for instance conjugate gradient (CG) (Glowinski et al., 1985) and generalized minimal residual methods (GMRES) (Saad, 2003; Titley-Peloquin et al., 2014), etc. CG is only used in the symmetric and positive-definite system while GMRES can be applied to more general systems. Thus GMRES is adopted as the linear solver for the implicit compressible Navier-Stokes solver be-

cause the Jacobian matrix of the system is not symmetric or positive definite.

Krylov linear solvers can provide acceptable solutions within a few number of iterations much smaller than the order of the matrix. As mentioned in many references (Burrage and Erhel, 1998; Ipsen and Meyer, 1998), the number of iterations depends on the eigenvalues or pseudo-eigenvalues of the matrix (Ke et al., 2005; Trefethen and Embree, 2005). For symmetric systems using CG methods, it can be proved, for example, see (Benzi, 2002; Hogben, 2006), that the convergence rate is linked to the eigenvalue distributions. Unfortunately, for nonsymmetric systems solved by GMRES, there is not a general theory to assess its convergence (Nachtigal et al., 1992). However, if the matrix is not far from normal, the eigenvalue distributions are convinced to be a good indicator of convergence. Preconditioning plays an important role in accelerating the convergence of Krylov solvers. Generally speaking, a preconditioner attempts to improve the spectral properties of the Jacobian matrix by clustering the eigenvalues (Benzi, 2002). There are comprehensive studies on preconditioners, such as efficiency comparisons (Diosady and Darmofal, 2007; Gholami et al., 2016), parallel performance (Toselli and Widlund, 2005; Yang, 2006), and storage saving (Luo et al., 2006; Diosady and Darmofal, 2007). Section 3.1 discusses a series of eigenvalue analyses of current widely-used preconditioners. Section 3.2 introduces the block relaxed Jacobi (BRJ) preconditioner implemented in Nektar++, and discusses how to balance the storage requirement and efficiency in large-scale problems.

1.4 Challenges of developing an efficient implicit solver

The development of an implicit solver combined with high-order spectral element methods for unsteady compressible flow simulations is still limited in the open-source community. Hartmann and Houston (2006a) developed an implicit high-order DG solver for the compressible Navier-Stokes equations within the deal.II

framework. However, their solver was only tested on steady-state problems and only supports quadrilateral and hexahedral meshes. Within Nek5000 (Chudanov et al., 2014), an implicit solver also makes use of Jacobian-free Newton Krylov method, but only supports weakly compressible simulations through modifications of the incompressible Navier-Stokes equations. An implicit solver for incompressible Navier-Stokes equations has recently developed in MOOSE (Peterson et al., 2018). In summary, only a few implicit high-order solvers with limited capabilities are available in the open-source community.

Except for the complexity of an implicit high-order compressible flow solver, we point out the following challenges that likely encounter its promotion. These are: (a) lack of an efficient preconditioner, (b) complex parameter choices, and (c) lack of an effective shock-capturing strategy. The third point also exists within an explicit solver. We emphasize the importance of the third point for high-order solvers, both implicit and explicit.

1.4.1 An efficient preconditioner

The preconditioner plays the most significant role in the implementation of an efficient implicit solver for challenging problems (Trefethen and Bau III, 1997; Knoll and Keyes, 2004). An ideally efficient preconditioner should be equipped with two properties: (a) it is cheap to construct and implement, which takes up low storage and spends little computational cost, and (b) it performs well in accelerating the linear solver.

Since the Jacobian matrix is not explicitly stored in the JFNK method, the solver needs additional storage and computational time to fully or partially reconstruct the whole/part of the Jacobian matrix for preconditioning. It is challenging to balance the storage requirement and computational cost to construct a practical preconditioner. For instance, using the expression in (Yan et al., 2020) to approximate the storage of the Jacobian matrix in a 3D simulation, one Gbyte of memory only allows a 3D simulation using $P = 4$ DG polynomial and 45 hexahedral elements. Incom-

plete LU (ILU) preconditioners have been shown to be efficient in many problems (Chapman et al., 2000; Persson and Peraire, 2008; Diosady and Darmofal, 2009). Most implementations of ILU both store the Jacobian matrix and the factorization matrix. But this is not affordable for large-scale problems. Even implementing ILU using optimized algorithms, such as the in-place storage mentioned in (Diosady and Darmofal, 2007), the storage of ILU preconditioning still takes up more than the size of a Jacobian matrix. Another family of widely-used iterative preconditioners such as block Jacobi and block SOR preconditioners (Diosady and Darmofal, 2009; Edalatpour et al., 2015) only needs part of the Jacobian matrix. Bastian et al. (2019) compared the matrix-based, partially matrix-based and matrix-free implementations, where the matrix-free preconditioner is obtained by iteratively inverting the diagonal blocks of the Jacobian matrix. In Nektar++, the designed iterative preconditioner utilizes a hybrid method that stores the diagonal blocks of the Jacobian and calculates the off-diagonal blocks on the fly, which will be introduced in Section 3.2.

The second aspect to assess is that, there is not an optimal preconditioner for general problems (Benzi, 2002). Preconditioners specifically designed based on the physics of problems can speed up solvers in specific problems. For example, diffusion synthetic acceleration (DSA), which is widely used in the transport community can be viewed as physics-based preconditioning (Larsen, 1982; Ashby et al., 1995). These methods need a deep understanding of the physics of the target problems, are not always available and are typically very sensitive. In contrast, general-purpose preconditioners based on algebra are more widely-used in practical applications. As the convergence rate of a Krylov solver is highly-related to the eigenvalue distributions of the system (Ipsen and Meyer, 1998), spectral analysis is an effective tool to assess the performance of an algebraic preconditioner. For symmetric systems using a CG solver, it can be proved that the convergence rate is linked to the eigenvalue distributions (Benzi, 2002), such as the cluster of eigenvalues. For a nonsymmetric system using GMRES, if the preconditioned matrix is not far from normal, the

cluster of eigenvalues could still be a good indicator of an efficient preconditioning strategy. However, as demonstrated later in Section 3.1, there seem no obvious links between the eigenvalue distributions and the convergence rate of Krylov solver for different types of preconditioners such as ILU and the iterative preconditioners. There is another factor that influences the assessment of an effective preconditioner. As the flux derivatives of different conservative variables are coupled, the scales of the eigenvalues of the preconditioned matrix are quite different, thus influencing the desired performance. It is challenging to design an efficient preconditioner for general problems (Benzi, 2002; Il'in, 2021).

1.4.2 Complex parameter choices

Implicit solvers have the potential to increase the efficiency of high Reynolds number simulations by relaxing the challenging time step restriction, and have been successfully adopted in a variety of steady state and unsteady simulations (Bassi et al., 2016; Vandenhoeck and Lani, 2019; Noventa et al., 2016, 2020; Yan et al., 2020). However, implicit solvers are generally more complex to design and implement and introduce many parameters, some of which have significant influence on the performance of the solver. Take an implicit solver combined use of the Runge-Kutta (RK) temporal discretization scheme and the Jacobian-free Newton-Krylov (JFNK) method as an example, there are parameters such as the time step size, the Newton iteration convergence tolerance, the linear iteration convergence tolerance, and possibly other parameters introduced during preconditioning (Knoll and Keyes, 2004; Kennedy and Carpenter, 2016; Yan et al., 2020). These parameters are usually problem-dependent and have large influence on the accuracy, efficiency and robustness in specific simulations. Their selection could become a complex multi-objective optimization problem. Therefore, a reliable approach for determining these parameters is essential especially for computational fluid dynamics (CFD) software for automatic large-scale simulations in a design pipeline with a wide range of application areas.

In steady state simulations, there have been many studies on this topic, most of which mainly focus on accelerating the convergence to steady state. In (Vanderstraeten, 2001), an expert system for choosing an efficient CFL was developed, in which the convergence history is separated into three different stages and different adaptive strategies are used in each stage. In (Lian et al., 2009), a solution-limiting method is developed to determine the CFL. Bucker et al. (2009) compared some of the existing adaptive methods and concluded that there is no clear winner. Kalkote et al. (2019) developed an error-based adaptive CFL method to accelerate the late stage of convergence but still needs to manually divide the convergence history into different stages. Compared with studies on efficiency, much fewer studies have focused on robustness. Lian et al. (2009) addressed this problem with a solution-limiting method and claimed that the robustness is largely improved. Other attempts on this topic are mainly straightforward techniques such as rolling back and recomputing with a smaller time step if the solution is not satisfactory (Yildirim et al., 2019). These studies have highlighted the importance of these parameter choices in different aspects of the solver and research areas. Although significant progress has been made, it is concluded in (Bucker et al., 2009) that '*optimal CFL evolution is still an open problem*'.

For unsteady simulations, there are additional parameters and the concern of temporal accuracy. The idea of using adaptive time-stepping in unsteady simulations can date back to time adaptation studies of ODEs (ordinary differential equations) (Söderlind, 2002; Söderlind and Wang, 2006; Arévalo et al., 2021). Various methods for choosing the step size (or step size controllers) and other topics such as the requirement of the temporal error estimator and the stability properties in stiff problems are discussed in the (Söderlind, 2002; Söderlind and Wang, 2006). Birken adopted similar ideas used in ODE systems and developed a time step adaptation method for the simulation of compressible Navier-Stokes equations using an embedded scheme to estimate the local temporal error. This method was adopted in the comparison of Rosenbrock methods and explicit first stage singly

diagonally implicit Runge-Kutta (ESDIRK) time integration methods (Blom et al., 2016). Noventa et al. (2016) developed a similar time step adaptation strategy in unsteady incompressible turbulent flow simulations, which was further tested in compressible simulations focusing on the comparison of different step size controllers (Noventa et al., 2020). The adaptive method based on local temporal error was further developed for goal-oriented time step adaptation and compared with a method based on global error estimate (Meisrimel and Birken, 2020). However most of them, such as the studies in (Birken et al., 2013; Blom et al., 2016; Noventa et al., 2016, 2020), consider the temporal discretization methods separately as an ODE solver using the method of lines but did not take into account specific physical properties of the problem on the specific spatial discretization adopted. These methods rely on a user-defined temporal error tolerance, the choice of which is highly problem dependent and needs a deep understanding of the simulation settings. However, a naive choice of temporal error tolerance may lead to a large increase in CPU time without obviously improving the results as has been demonstrated in (Noventa et al., 2016).

1.4.3 An effective shock-capturing strategy

An effective shock-capturing strategy should properly add numerically dissipation around strong discontinuities but should have a negligible influence away from these. Shock-capturing strategies such as artificial viscosity firstly detect the discontinuities through sensors and then explicitly add locally varying viscosity to damp the oscillations typical of Gibbs phenomena in the representation of discontinuities by smooth functions. Due to the low dispersion and dissipation errors of DG schemes, the applications have increased attention to under-resolved computations such as large-eddy simulations (LES). However, the small-magnitude features in turbulent flows could be mistakenly identified as irregularities and diffused by artificial viscosity. Therefore the simulation of turbulent shock flows is still a challenging problem (Mani et al., 2009; Slotnick et al., 2014; Ferrand et al., 2020).

Nonlinear instabilities not only arise from sharp shocks, but also from contact discontinuities, shear layers, great thermal gradients, etc. Most methods focus on the stabilization at shocks but ignore other sharp features. Some methods use a physics-based sensor to detect the physical phenomenon of shocks such as the compression across a shock (Fernandez et al., 1994; Moro et al., 2016; Yu et al., 2018). Other methods directly rely on the detection of the irregularities of numerical solutions, which target all the sharp features (Cook and Cabot, 2005; Larsson et al., 2007; Johnsen et al., 2010). Generally speaking, these methods can provide non-oscillatory shock profiles for steady flows, but fail in unsteady turbulent flows (Fernandez et al., 2018).

Other challenges of effective shock-capturing are the cost of constructing a well-performing sensor. Some approaches rely on high-order derivatives for achieving accurate results (Cook and Cabot, 2005; Olson and Lele, 2013). These methods are computationally costly, sensitive to the accurate calculation of derivatives, and can be only applied to simple geometries. Additionally, in common practice, shock-capturing strategies applied in high-order DG add piecewise constant viscosity elementally (Persson and Peraire, 2006b). However, this strategy cannot work in some situations such as shock layers within a single element. Methods such as those mentioned by (Barter and Darmofal, 2007, 2010) reconstruct high-order viscosity using several elements' information. Again, the computational cost is high, especially in large-scale problems using unstructured meshes.

1.5 Objectives

The objective of this thesis is to develop an efficient implicit high-order spectral/hp element compressible flow solver within the Nektar++ framework. The implicit solver mainly takes advantage of the spectral/hp element DG spatial discretization, diagonally implicit Runge-Kutta (DIRK) temporal discretization, and the Jacobian-free Newton-Krylov (JFNK) method. Even though the implicit solver achieves an

obvious speed-up, it still encounters several challenges to be extended to large-scale practical applications. We target at optimizing the solver in three challenging aspects: (a) designing an efficient preconditioner, (b) optimizing the parameter choices, and (c) constructing an effective shock-capturing strategy.

A preconditioner plays an important role in accelerating the convergence of the solver. In large-scale practical problems, we need to balance the storage requirement and computational cost to implement the preconditioner. Based on the understanding of spectral properties of different popular preconditioners through spectral analysis, we adopt a partially matrix-free iterative preconditioner. The storage is reduced and the efficiency is improved through the hybrid calculation of analytical Jacobian and numerical Jacobian.

After the implementation of the preconditioner, the implicit compressible flow solver now includes four loops: (a) time integration iteration, (b) Newton-type nonlinear system iteration, (c) Krylov linear system iteration, and (d) the iteration of the iterative preconditioner. A number of user-defined parameters within these loops are coupled and all influence the efficiency or accuracy of the solver, including the time-step size, the convergence tolerance of the nonlinear solver, the convergence tolerance of the linear solver and the update frequency of preconditioner, etc. It is not surprising that a naive parameter choice slows down the solver several times inferior to the optimal one. Based on a theoretical analysis of error and detection of stiffness, a systematic framework is then designed to automatically choose these parameters

Lastly, the applications of the implicit solver to high-speed flow simulations also need to be equipped with a well-performing shock-capturing strategy. It is critical for a robust and efficient compressible solver that dissipates oscillations at shocks and has negligible dissipation in smooth regions. We modify the bulk-stress based artificial viscosity by adding extra density-related terms. We further design an extra shear-stress based artificial viscosity to help stabilize simulations where strong shear layers exist. The new form of artificial viscosity performs well in problems involving

shocks in so far as the simulations can achieve more accurate shock profiles and catch more small-scale and small-magnitude structures in resolved regions.

1.6 Outline

The governing equations, the spatial and temporal discretization, and the Jacobian-free Newton-Krylov method are introduced in Section 2. Section 3 analyzes the spectral properties of several widely-used preconditioners and discusses the practical implementation of an iterative preconditioner in Nektar++ in consideration of its storage requirements and computational cost. We propose a systematic set of strategies aimed to automatically adjust the parameters within the implicit solver in Section 4. Section 5 presents the application of the implicit solver to high-speed flow simulations. A new shock-capturing strategy that absorbs the best features of shear-stress based, bulk-stress based, and Laplacian artificial viscosities is proposed.

Chapter 2

Implicit spectral/hp element solver

This chapter presents the governing equations, numerical methods adopted for the implicit compressible flow solver, as well as the process of how we implement and verify the codes of the solver. Section 2.1 introduces the mathematical models governing inviscid and viscous compressible flows, namely the compressible Euler and Navier-Stokes equations. Then, Section 2.2 and Section 2.3 introduce the spatial and temporal discretization methods of the governing equation respectively. The discretized nonlinear equation system is then resolved by a Newton-type nonlinear solver and a Krylov linear solver, which is introduced in Section 2.4 and Section 2.6. After the introduction of the above numerical methods applied to the compressible flow solver, Section 2.7 gives a brief demonstration of the construction of the solver. Lastly, Section 2.8 utilizes several benchmark tests to verify the codes of the solver.

2.1 Governing equations

The system of governing equations for compressible flow is given by the continuity, momentum and energy equations, which are written in conservative form

$$\frac{\partial \mathbf{U}}{\partial t} = L(\mathbf{U}) = -\left(\nabla \cdot \mathbf{F}(\mathbf{U}) + \nabla \cdot \mathbf{G}(\mathbf{U}, \nabla \mathbf{U})\right), \tag{2.1}$$

where L is the nonlinear spatial operator, $\mathbf{U} = [\rho, \rho u_1, \ldots, \rho u_d, E]^T$ is the vector of conservative variables, and the subscript d denotes the number of dimensions of the problem.

The inviscid flux, $\mathbf{F} = \mathbf{F}(\mathbf{U})$, and the viscous flux, $\mathbf{G} = \mathbf{G}(\mathbf{U}, \nabla \mathbf{U})$ in the i^{th} direction are given by

$$\mathbf{F}_i = \begin{bmatrix} \rho u_i \\ \rho u_1 u_i + \delta_{1,i} p \\ \vdots \\ \rho u_d u_i + \delta_{d,i} p \\ u_i (E + p) \end{bmatrix}, \quad \mathbf{G}_i = \begin{bmatrix} 0 \\ -\tau_{i1} \\ \vdots \\ -\tau_{id} \\ q_i - \sum_{j=1}^{d} u_j \tau_{ij} \end{bmatrix}, \tag{2.2}$$

where p is the pressure, ρ is the density, $E = \frac{p}{\gamma - 1} + \rho \frac{u_k u_k}{2}$ is the total energy, and γ is the ratio of specific heats. The viscous tensor is

$$\tau_{ij}(\mu, \theta, \mathbf{U}) = \mu\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) + \left(\theta - \frac{2}{3}\mu\right)\frac{\partial u_k}{\partial x_k}\delta_{ij}, \tag{2.3}$$

and the heat flux is

$$q_i = -\kappa \frac{\partial T}{\partial x_i}. \tag{2.4}$$

The bulk viscosity θ is typically set to be zero, and the dynamic viscosity μ is a function of the temperature T through the Sutherland's law

$$\mu = \mu_0 \left(\frac{T}{T_0}\right)^{\frac{3}{2}} \frac{T_0 + 110}{T + 110}. \tag{2.5}$$

The thermal conductivity κ satisfies the following equation of state

$$\kappa = (\mu + \theta)\frac{\gamma R}{Pr(\gamma - 1)}, \tag{2.6}$$

where Pr is the Prandtl number representing the ratio of momentum diffusivity to thermal diffusivity and R is the gas constant satisfying $R = C_p - C_v$. C_p is the specific heat constant for constant pressure while C_v is the specific heat constant for constant volume.

The system of governing equation Eq. (2.1) is supplemented by the constitutive equation for an ideal gas

$$p = \rho RT. \tag{2.7}$$

In the following we will assume a fluid with the properties of air at sea level, namely, $\mu_0 = 1.7894 \times 10^{-5} kg.m.s^{-1}$, $T_0 = 288.15K$, $C_p = 1.01 kJ.kg^{-1}$, $C_v = 0.718 kJ.kg^{-1}$, $Pr = 0.72$ and $\gamma = 1.4$.

2.2 Discontinuous Galerkin formulations

In the discontinuous Galerkin methods, the domain Ω is partitioned into N_e non-overlapping elements Ω_e, such that $\Omega = \bigcup_{e=1}^{N_e} \Omega_e$. In 2D, these elements could be a mixture of quadrilaterals and triangles, and in 3D, of hexahedra, triangular prisms, square-based pyramids and tetrahedra. For convenience, the local element coordinates defined as $\mathbf{x} \in \Omega_e$ are mapped onto standard elements $\boldsymbol{\xi} \in \Omega_{st}$, so that $\mathbf{x} = \boldsymbol{\chi}_e(\boldsymbol{\xi})$, where the $\boldsymbol{\xi}$ are d-dimensional coordinates representing positions in the standard elements. For example, a quadrilateral is defined as $\Omega_{st} = \{(\xi_1, \xi_2)| -1 \leq \xi_1, \xi_2 \leq 1\}$. Other supported region shapes can be found in (Cantwell et al., 2015). The mapping $\boldsymbol{\chi}_e$ need not necessarily exhibit a constant Jacobian. For deformed elements, Nektar++ represents the curvatures of these elements by taking a sub-parametric or iso-parametric mapping for $\boldsymbol{\chi}_e$ (Karniadakis and Sherwin, 2013). With the mapping $\boldsymbol{\chi}_e$, the discrete approximation $\mathbf{U}_\delta$ to the solution $\mathbf{U}$ on a physical

element can be represented as

$$\mathbf{U} \simeq \mathbf{U}_\delta = \sum_{q=1}^{N_{dof}(e)} \phi_q \left([\boldsymbol{\chi}_e]^{-1}(\mathbf{x}) \right) \mathbf{u}_q(t), \tag{2.8}$$

where $N_{dof}(e)$ is the total degrees of freedom (Dofs) in Ω_e, and $\mathbf{u}_q$ is the coefficient of the trial function ϕ_q. The $\mathbf{U}_\delta$ on the quadrature points can be calculated by

$$\mathbf{U}_{\delta,i} = \sum_{q=1}^{N_{dof}(e)} \phi_q(\boldsymbol{\xi}_i)\mathbf{u}_q(t) = \sum_{q=1}^{N_{dof}(e)} \mathbf{B}_{i,q}\mathbf{u}_q(t), \tag{2.9}$$

where $\boldsymbol{\xi}_i$ are the coordinates of the i^{th} quadrature point and $\mathbf{B}$ is the backward transform matrix. Eq. (2.9) is simplified as

$$\mathbf{U}_\delta = \mathbf{B}\mathbf{u}. \tag{2.10}$$

Assuming $\mathbb{P}_P(\Omega_e)$ denotes the space of polynomials spanned by $N_{dof}(e)$ basis functions, with the maximum polynomial order P in Ω_e, the space required in discontinuous Galerkin methods is defined as

$$D_P = \{v \in L^2(\Omega) : v|\Omega_e \in \mathbb{P}_P(\Omega_e), \forall \Omega_e\}, \tag{2.11}$$

which allows the functions to be discontinuous across elemental boundaries.

Following the standard Galerkin approach, the test functions are the same as trial functions. Assuming ϕ_p is a test function lying in $\mathbb{P}_P(\Omega_e)$, the weak form of Eq. (2.1) is thus given by

$$\int_{\Omega_e} \frac{\partial \mathbf{U}}{\partial t} \phi_p d\Omega = \int_{\Omega_e} L(\mathbf{U}) \phi_p d\Omega. \tag{2.12}$$

After integration by parts, reads

$$\begin{aligned}\int_{\Omega_e} \frac{\partial \mathbf{U}}{\partial t}\phi_p d\Omega =& \left(\int_{\Omega_e} \mathbf{F}\cdot\nabla\phi_p d\Omega - \int_{\Gamma_e} \mathbf{F}_n\phi_p d\Gamma\right) \\ &+ \left(\int_{\Omega_e} \mathbf{G}\cdot\nabla\phi_p d\Omega - \int_{\Gamma_e} \mathbf{G}_n\phi_p d\Gamma\right) \\ =&\mathbf{R}_I(\mathbf{F}) + \mathbf{R}_V(\mathbf{G}),\end{aligned} \tag{2.13}$$

where the set of elemental faces for Ω_e is denoted as Γ_e, $\mathbf{F}_n = \mathbf{F}\cdot\mathbf{n}$, $\mathbf{G}_n = \mathbf{G}\cdot\mathbf{n}$ are inviscid and viscous fluxes normal to the trace, $\mathbf{n}$ is the elemental outward normal, and $\mathbf{R}_I(\mathbf{F})$, $\mathbf{R}_V(\mathbf{G})$ denote the RHS of the inviscid and viscous fluxes.

Advection term The weak form without viscous terms in Eq. (2.13) gives

$$\int_{\Omega_e} \frac{\partial \mathbf{U}}{\partial t}\phi_p d\Omega = \int_{\Omega_e} \mathbf{F}\cdot\nabla\phi_p d\Omega - \int_{\Gamma_e} \mathbf{F}_n\phi_p d\Gamma = \mathbf{R}_I(\mathbf{F}). \tag{2.14}$$

Substituting Eq. (2.9) into Eq. (2.14), the fluxes are evaluated at the same quadrature points. A quadrature rule with N_Q quadrature points $\boldsymbol{\xi}$ is adopted for the volume integration while a quadrature rule with N_Q^Γ quadrature points $\boldsymbol{\xi}^\Gamma$ is adopted for the trace integration. This transfers the weak form into a semi-discrete form

$$\sum_{i=1}^{N_Q}\sum_{q=1}^{N_{dof}(e)} \phi_p(\boldsymbol{\xi}_i) w_i J_i \phi_q(\boldsymbol{\xi}_i)\frac{d\mathbf{u}_q}{dt} = \sum_{i=1}^{N_Q} w_i J_i \nabla\phi_p(\boldsymbol{\xi}_i)\cdot\mathbf{F}(\mathbf{U}_{\delta,i}) - \sum_{i=1}^{N_Q^\Gamma} w_i^\Gamma J_i^\Gamma \phi_p(\boldsymbol{\xi}_i^\Gamma)\mathbf{F}_n, \tag{2.15}$$

where w_i and J_i are the quadrature weight and grid metric Jacobian at the i^{th} volume quadrature point, w_i^Γ and J_i^Γ are the quadrature weight and grid metric Jacobian at the i^{th} trace quadrature point.

The system is complete as long as the trace flux $\mathbf{F}_n$ is defined. Using a Riemann solver to deal with the discontinuous advection flux, the numerical trace flux can be expressed as

$$\mathbf{F}_n(\mathbf{U}) = \tilde{\mathbf{F}}_n(\mathbf{U}^+, \mathbf{U}^-), \tag{2.16}$$

where $\mathbf{U}^+$ and $\mathbf{U}^-$ denote the variable values exterior and interior to the local element trace. $\tilde{\mathbf{F}}_n$ is a numerical advection flux that is calculated through the solution of a local Riemann problem in the normal direction to the trace. A review of exact and approximate Riemann solvers can be found in the textbook by Toro (2009). Most widely-used Riemann solvers, including the exact Riemann solver and approximated Riemann solvers such as Roe solver, HLLC solver, etc, are available in Nektar++.

Diffusion term The discretization of the diffusion term becomes more complex because the evaluation of numerical trace flux $\mathbf{G}_n = \tilde{\mathbf{G}}_n(\mathbf{U}^+, \mathbf{U}^-, \nabla\mathbf{U}^+, \nabla\mathbf{U}^-)$, also depends on the gradients, $\nabla\mathbf{U}^+$ and $\nabla\mathbf{U}^-$, at both sides of interface. It is a simple and natural choice to impose both values of the double-valued solutions and gradients on interior faces are identical, which was proposed in (Bassi and Rebay, 1997a), namely Bassi-Rebay (BR) scheme. However, the application of the BR scheme to the Poisson problem was found to be problematic (Brezzi et al., 1999), because the existence of the approximate solution can not be guaranteed. The modified version of BR scheme adding additional stabilization terms was proposed in (Bassi et al., 1997), which is referred to as BR2 scheme. Similar to the manner of BR2, the family of local discontinuous Galerkin (LDG) methods adds different versions of stability terms to the fluxes (Cockburn, 1998; Cockburn and Shu, 1998a; Cockburn and Dawson, 1999). Its 'local' property is because $\mathbf{U}$-flux does not depend on $\nabla\mathbf{U}$-flux. Therefore the second-order equation of calculating the diffusion term can be separated into two first-order equations. Thus the solving of the gradient and the solution, $\nabla\mathbf{U}$ and $\mathbf{U}$, can be decoupled and resolved sequentially. Another family of interior penalty (IP) methods arises popularity nearly at the same time as BR, LDG methods, but is independently developed. IP methods firstly calculate $\nabla\mathbf{U}$ through the differentiation with respect to $\mathbf{U}$ within local elements. The Dirichlet boundary condition embodied in the DG methods and the requirement of continuity for approximate solutions are with the aid of straightforward adding boundary penalty and interior penalty terms. Compared with BR and LDG methods, the family of

IP methods generally has a smaller stencil, which leads to lower memory consumption and fewer parallel communication. Especially for the implicit solver that we preferred to use in high Reynolds number compressible flow simulation, the benefits of reduction in the computational cost and storage consumption during preconditioning the linear solver within the implicit solver are more obvious. Therefore, IP methods are adopted in the implicit compressible flow solver of Nektar++.

Since the inviscid terms have been treated in Eq. (2.14), we will focus on the discretization of the viscous term. Firstly, the equivalent form of IP diffusion flux is defined as

$$\mathbf{G}_s(\mathbf{U}, \nabla\mathbf{U}) = \mathbf{K}_s \cdot \nabla\mathbf{U}, \tag{2.17}$$

where $\mathbf{G}_s$ is the flux in the s^{th} direction. The expressions of the matrices $\mathbf{K}$ can be found in (Dolejší and Feistauer, 2015). Using the above relationship Eq. (2.17), the following expression summarizes the IP method in primal form

$$\begin{aligned}\mathbf{R}_V(\mathbf{G}) = &\int_{\Omega_e} \mathbf{G} \cdot \nabla\phi_p d\Omega - \int_{\Gamma_e} \mathbf{n}_s \cdot \{\mathbf{K}_s \nabla\mathbf{U}\} [\phi_p] d\Gamma \\ &-\theta \int_{\Gamma_e} \mathbf{n}_s \cdot \left\{\mathbf{K}_s^T \nabla\phi_p\right\} [\mathbf{U}] d\Gamma - \int_{\Gamma_e} \delta[\mathbf{U}][\phi_p] d\Gamma,\end{aligned} \tag{2.18}$$

where the jump across the trace is $[w] = w_i n_i + w_j n_j$ and the average of the values from the two sides of the element interface is $\{w\} = \frac{w_i + w_j}{2}$ (w is auxiliary variable, i and j are the indexes of neighbouring elements of a trace). The third term at the right-hand side is the symmetric term. When $\theta = 1$ and symmetric term is included, this scheme is called symmetric interior penalty Galerkin (SIPG) method, which is adopted in the solver. It is advantageous to have some type of symmetry to satisfy adjoint consistency condition (Arnold et al., 2002). This property is critical to guarantee the optimal order of convergence using different orders of DG approximations, which is not shared by some other schemes such as nonsymmetric interior penalty Galerkin (NIPG, $\theta = -1$) method (Rivière et al., 1999, 2001) and incomplete interior penalty Galerkin (IIPG, $\theta = 0$) method (Sun, 2003). The fourth term is the penalty term to help stabilize, where the parameter δ controls the extent of penal-

ization. A choice $\delta = \frac{(P+1)^2}{h}$ is selected for quadrilateral and hexahedral meshes. h is the characteristic length of elements, which is an average of the element lengths on both sides of the trace. The element lengths are obtained from grid derivatives in the trace normal direction. Other choices of penalty terms can be found in (Rivire, 2008; Hillewaert, 2013). The IP formulation incorporates two parameters, θ and δ, in the last two terms of the RHS of Eq. (2.18) that aims at controlling the amount of dissipation at the trace generated by the viscous and inviscid contributions to the jumps at the trace.

After the discretization of the advection and the diffusion terms and substituting the variables expressed by polynomial functions in Eq. (2.8), the weak form Eq. (2.13) can be written as a semi-discrete form of the form

$$\frac{\partial \mathbf{u}}{\partial t} = \mathcal{L}(\mathbf{u}, t) = \mathbf{M}^{-1}(\mathbf{R}_I + \mathbf{R}_V), \tag{2.19}$$

where $\mathbf{M}$ is the mass matrix of element Ω_e.

The corresponding semi-discrete form for $\mathbf{U}_\delta$ can be achieved through Eq. (2.10) and

$$L_\delta(\mathbf{U}(\mathbf{u})) = \mathbf{B}\mathcal{L}(\mathbf{u}), \tag{2.20}$$

reads

$$\frac{\partial \mathbf{U}_\delta}{\partial t} = L_\delta(\mathbf{U}_\delta). \tag{2.21}$$

The spatial truncation error measured in L_2 norm verifies SIPG without super-penalization can guarantee the solution achieve an optimal order of accuracy, which is mentioned in (Arnold et al., 2002) and shown in the results in Section 2.8. For a simulation using a fixed mesh of size h and P^{th} order DG polynomial, the spatial truncation error can be defined as the difference between theoretically exact solution $\mathbf{U}$ and the spatial discretized solution $\mathbf{U}_\delta$. Because the error is directly linked with flux residual L, the truncation error arising from the spatial discretization can be

also defined as the difference between Eq. (2.1) and Eq. (2.21)

$$\mathbf{E}_s = L_\delta(\mathbf{U}_\delta) - L(\mathbf{U}) = C_s D_s(\mathbf{U}) h^{P+1} + O(h^{P+2}), \tag{2.22}$$

where $C_s D_s(\mathbf{U}) h^{P+1}$ is the leading term of truncation error, $D_s(\mathbf{U})$ is the spatial derivative only depending on $\mathbf{U}$, and C_s is a coefficient independent of $\mathbf{U}$ but only related to spatial discretization scheme. Currently, we restrict our study to deal with smooth flows, thus avoiding potential inexact estimates of errors in shock-capturing problems.

2.3 Implicit time integration methods

For unsteady problems, the time step of explicit time integration schemes is restricted by the Courant-Friedrichs-Lewy (CFL) condition. This restriction is more serious in highly-stretched mesh, high-Reynolds number simulations. Generally, the use of implicit schemes can relax this restriction and is expected to achieve better efficiency. In this section, two popular series of implicit time integration schemes are adopted to discretize the governing equation Eq. (2.1).

Backward differentiation formulas A family of implicit multi-step methods (Bijl et al., 2002), named backward differentiation formulas (BDF) is expressed as

$$\sum_{i=1}^{N+1} a_i \mathbf{U}^{n+2-i} = \Delta t L(\mathbf{U}^{n+1}), \tag{2.23}$$

where N is the order of time integration schemes, n is the index of time step iteration, and $\mathbf{a}$ are the consistent coefficients. Taking the second-order BDF as an example, $N = 2$, $a_1 = 3/2$, $a_2 = -2$, and $a_3 = 1/2$.

Within implicit time integration schemes, BDF methods are widely used because only one nonlinear system needs to be resolved per time step. The first-order and the second-order BDF are widely used in industry because engineering problems usually

do not require a high-level accuracy. However, the schemes higher than second-order have limited stability properties, such as BDF3 lacks A-stability [1] that may be problematic in the presence of convection (Bijl et al., 2002) and BDF4 seldom remains stability in large-scale problems (Melson et al., 1993).

Diagonally implicit Runge-Kutta In contrast to multi-step schemes, no bound on the order for A-stable schemes has been proved for implicit Runge-Kutta (RK) methods (Birken, 2013). Therefore, higher-order RK schemes can be constructed, thus achieving higher temporal accuracy. For s-stage Runge-Kutta schemes, the coefficients used during processing such as $\mathbf{b} = [b_1 \dots b_s]^T$, $\mathbf{c} = [c_1 \dots c_s]^T$, $\mathbf{A} = [a_{i,j}]_{i,j} \in R^{s,s}$ can be written in a compact form in Butcher array. The diagonally implicit Runge-Kutta (DIRK) means the coefficients in the Butcher array (Table. 2.1) are zero in the upper triangle. The advantage of DIRK schemes is the stage solutions can be evaluated sequentially rather than be solved as one coupled large implicit system. Singly DIRK (SDIRK) schemes are adopted because the diagonals $a_{i,i}$ are identical. This is advantageous because it permits the reuse of the similar linearization in preconditioning over all s sub-stages. Someone may doubt the efficiency of DIRK schemes because at each stage there is one nonlinear system that needs to be resolved. In practice, DIRK schemes can reach a lower error level more efficiently compared with their BDF counterparts (Bijl et al., 2002). This is because the diagonal coefficients $a_{i,i}$ are typically smaller than one. The smaller real time step $a_{i,i}\Delta t$ at each stage reduces the stiffness.

$$\begin{array}{c|c} \mathbf{c} & \mathbf{A} \\ \hline & \mathbf{b}^T \end{array} = \begin{array}{c|cccc} c_1 & a_{1,1} & 0 & 0 & 0 \\ c_2 & a_{2,1} & a_{2,2} & 0 & 0 \\ \vdots & \vdots & \ddots & a_{i,i} & 0 \\ c_s & a_{s,1} & \dots & a_{s,s-1} & a_{s,s} \\ \hline & b_1 & \dots & \dots & b_s \end{array}$$

Table 2.1: Butcher array for the DIRK schemes.

[1] To be A-stable, the stability function must have no poles in the left half-plane. Also, the magnitude of the stability function must be bounded by 1. A-stability is a very important property for a robust time integration scheme. More information can refer to (Butcher, 2016).

The time integration by the DIRK schemes consists of the following steps:

1. The initial guess is the previous time step's solution

$$\mathbf{U}^{(0)} = \mathbf{U}^n. \tag{2.24}$$

2. At each stage, the solutions are updated by

$$\mathbf{S}^{(i)} = \mathbf{U}^n + \Delta t \sum_{j=1}^{i-1} a_{i,j} L(\mathbf{U}^{(j)}), \tag{2.25}$$

$$\mathbf{U}^{(i)} = \mathbf{S}^{(i)} + \Delta t a_{i,i} L(\mathbf{U}^{(i)}), \tag{2.26}$$

   where the superscript (i) represents the i^{th} stage, $i = 1, 2, \ldots, s$.

3. The solution at the new time step t^{n+1} is obtained by

$$\mathbf{U}^{n+1} = \mathbf{U}^n + \Delta t \sum_{i=1}^{s} b_i L(\mathbf{U}^{(i)}). \tag{2.27}$$

4. If $t^{n+1} < T$, go to Step 1; else stop.

Another advantage of using high-order Rung-Kutta schemes is that it is easier to do time step adaptivity. This special group of time integration schemes (Bu et al., 2016; Jörgensen et al., 2018) calculates an extra embedded stage of solution to provide a different order of approximation $\hat{\mathbf{U}}$. The differences between the approximations of $\mathbf{U}$ and $\hat{\mathbf{U}}$ can be an estimate of temporal error at the current step, which can be used for time step adaptivity.

The Butcher array of the embedded version of the DIRK schemes is given by Table. 2.2

$$
\begin{array}{c|c}
\mathbf{c} & \mathbf{A} \\
\hline
 & \mathbf{b}^T \\
 & \hat{\mathbf{b}}^T
\end{array}
=
\begin{array}{c|ccccc}
c_1 & a_{1,1} & & & & \\
c_2 & a_{2,1} & a_{2,2} & & & \\
\vdots & \vdots & \ddots & a_{i,i} & & \\
c_s & a_{s,1} & \dots & a_{s,s-1} & a_{s,s} & 0 \\
c_{s+1} & a_{s+1,1} & \dots & \dots & a_{s+1,s} & a_{s+1,s+1} \\
\hline
 & b_1 & \dots & \dots & b_s & 0 \\
\hline
 & \hat{b}_1 & \dots & \dots & \dots & \hat{b}_{s+1}
\end{array}
$$

Table 2.2: Butcher array for the embedded DIRK schemes.
(The values of the coefficients in Butcher array are listed in Appendix 6.2.)

The different order's approximation $\hat{\mathbf{U}}$ using $(s+1)$ stages is

$$\hat{\mathbf{U}}^{n+1} = \mathbf{U}^n + \Delta t \sum_{i=1}^{s+1} \hat{b}_i L(\mathbf{U}^{(i)}). \tag{2.28}$$

The family of embedded DIRK schemes proposed in (Kvrn, 2004) is adopted for the adaptive time-stepping strategy in Section 4.1. It is an embedded version of singly diagonally implicit Runge-Kutta methods with an explicit first stage (ES-DIRK) scheme (Jörgensen et al., 2018). The explicit first stage, namely $a_{1,1} = 0$ in Table. 2.2, is mentioned in (Jörgensen et al., 2018) that can avoid order reduction in stiff problems and guarantee at least a second-order accuracy. The embedded solution $\hat{\mathbf{U}}^{n+1}$ is one order higher than $\mathbf{U}^{n+1}$, the leading term of the temporal error of $\mathbf{U}^{n+1}$ can be accurately estimated as

$$
\begin{aligned}
\mathbf{E}_t &= \mathbf{U}^{n+1} - \hat{\mathbf{U}}^{n+1} \\
&= \Delta t \sum_{i=1}^{s+1} (b_i - \hat{b}_i) L(\mathbf{U}^{(i)}) \\
&\approx \Delta t (C_t D_t(U) \Delta t^N + O(\Delta t^{N+1})) \\
&= C_t D_t(\mathbf{U}) \Delta t^{N+1} + O(\Delta t^{N+2}),
\end{aligned}
\tag{2.29}
$$

where N is the order of accuracy of DIRK scheme, $C_t D_t(\mathbf{U})\Delta t^{N+1}$ is the leading term of truncation temporal error, $D_t(\mathbf{U})$ is a temporal derivative term only depending on $\mathbf{U}$ and C_t is a coefficient independent of $\mathbf{U}$ but only related to time integration

discretization scheme. To notice, here the derivation of temporal error does not consider the spatial discretization. The analytical operator L in Eq. (2.1) is used rather than the discretized spatial operator L_δ in Eq. (2.21), thus the spatial error is not included. The estimated $\mathbf{E}_t$ also excludes the accumulated temporal errors from previous time steps, which is considered as local temporal error.

2.4 Newton-type nonlinear solver

At each stage of DIRK methods, given Eq. (2.26), the nonlinear equation system can be written as

$$\mathbf{N}(\mathbf{U}^{(i)}) = \mathbf{U}^{(i)} - \mathbf{S}^{(i)} - \Delta t a_{i,i} L(\mathbf{U}^{(i)}) = 0. \tag{2.30}$$

At each step of BDF, the nonlinear system Eq. (2.23) can be written in a similar form but here we will not focus on it.

Considering the spatial discretization, the discrete formulation of the nonlinear system replaces $\mathbf{U}$ and L by $\mathbf{U}_\delta$ and L_δ

$$\mathbf{N}(\mathbf{U}_\delta^{(i)}) = \mathbf{U}_\delta^{(i)} - \mathbf{S}^{(i)} - \Delta t a_{i,i} L_\delta(\mathbf{U}_\delta^{(i)}) = 0. \tag{2.31}$$

Because the length of the coefficient vector $\mathbf{u}$ is usually smaller than $\mathbf{U}_\delta$, to reduce the size of the nonlinear system, we choose to solve the system in the coefficient space

$$\mathbf{N}(\mathbf{u}^{(i)}) = \mathbf{u}^{(i)} - \mathbf{s}^{(i)} - \Delta t a_{i,i} \mathcal{L}(\mathbf{u}^{(i)}) = 0, \tag{2.32}$$

and update $\mathbf{U}_\delta$ through

$$\mathbf{U}_\delta^{(i)} = \mathbf{B}\mathbf{u}^{(i)}. \tag{2.33}$$

The Jacobian-free Newton-Krylov method (JFNK) is selected to solve the discretized equation system. JFNK is a combination of a Newton-type nonlinear solver and a consistent Krylov linear solver. This method saves the storage of the Jaco-

bian matrix by storing the vectors after Jacobian-vector multiplication operation in Krylov solver resolution. It is widely used in implicit solvers. More information can be found in the review (Knoll and Keyes, 2004).

The Newton solver is chosen as the nonlinear solver because its quadratic convergence rate can be guaranteed as long as its initial guess is near the exact solution (Kelley, 1995; Dennis Jr and Schnabel, 1996). This convergence rate is much faster than other types of nonlinear solvers such as dichotomy, chord method, etc.

The process of Newton iteration is seeking for the roots of the nonlinear system Eq. (2.32). Using the variable $\mathbf{v}$ to replace $\mathbf{u}^{(i)}$ during Newton iteration, the nonlinear system is rewritten as

$$\mathbf{N}(\mathbf{v}) = \mathbf{v} - \mathbf{s} - \Delta t a_{i,i} \mathcal{L}(\mathbf{v}) = 0. \tag{2.34}$$

The Newton iteration consists of the following steps:

1. Choose the last stage's solution $\mathbf{u}^{(i)}$ as the initial approximation

$$\mathbf{v}^0 = \mathbf{u}^{(i)}. \tag{2.35}$$

   Since the initial guess is very important to retain the quadratic convergence property of Newton's method, the introduction of other advanced methods such as pre-iterations or extrapolating a high-order initial solution using previous solutions can be found in (Kelley, 2003).

2. Newton's method applied to the nonlinear system Eq. (2.34) at k^{th} step, reads

$$\mathbf{v}^{k+1} = \mathbf{v}^k + \Delta \mathbf{v}, \tag{2.36}$$

   where the Newton step $\Delta \mathbf{v}$ is updated through the following linearized equation

$$\left(\frac{\partial \mathbf{N}(\mathbf{v}^k)}{\partial \mathbf{v}^k} \right) \Delta \mathbf{v} = -\mathbf{N}(\mathbf{v}^k). \tag{2.37}$$

3. If the Newton residual of the updated solution $\mathbf{v}^{k+1}$ satisfies

$$\|\mathbf{N}(\mathbf{v}^{k+1})\| \leq \tau_{\mathrm{N}}, \tag{2.38}$$

where τ_{N} is the Newton tolerance. A proper choice of τ_{N} is important because it determines the magnitude of the iterative error introduced by the iterative solver, which may degrade the overall error and the convergence order of accuracy (Noventa et al., 2020). Here, we introduce two common choices of Newton tolerance.

Absolute Newton tolerance One widely-used tolerance is the absolute Newton tolerance (Bijl et al., 2002), which limits the Newton residual up to an expected accuracy level

$$\|\mathbf{N}(\mathbf{v}^{k+1})\| \leq \theta_1 \|\mathbf{v}^0\|, \tag{2.39}$$

where θ_1 is a constant safety factor and $\|\mathbf{v}^0\|$ is the norm of initial fluid input. One can directly give a specific value as the tolerance, but the $\|\mathbf{v}^0\|$ is multiplied here so that the whole tolerance scales together with the scale of the problem and a lower bound of θ_1 can be derived based on the rounding error of the simulation platform.

Absolute Newton tolerance $\tau_{\mathrm{N}} = \theta_1 \|\mathbf{v}^0\|$ requires the residual up to a user-defined accuracy level, which is relatively straight forward. However, this method is very sensitive and case-dependent. For example, if the time step of a simulation is decreased, the tolerance should be manually adjusted to maintain accuracy.

Relative Newton tolerance Another commonly used approach is relative Newton tolerance (Persson and Peraire, 2006a; Persson, 2009), which stops iterations when the Newton residual reduces to a relative level compared with

its initial Newton residual ($\|\mathbf{N}(\mathbf{v}^0)\|$)

$$\frac{\|\mathbf{N}(\mathbf{v}^{k+1})\|}{\|\mathbf{N}(\mathbf{v}^0)\|} \leq \theta_2, \tag{2.40}$$

where θ_2 is a safety factor.

This strategy linearly links the tolerance to the time step because the Newton residual $\mathbf{N}$ changes with the size of Δt. Therefore, the choice of constant θ_2 can be less sensitive than the absolute tolerance. However, for simulations near steady state, the iteration will have difficulty converging because the $(\theta_2\|\mathbf{N}(\mathbf{v}^0)\|)$ term maybe close to or even smaller than the rounding error.

In Nektar++, we default use relative Newton tolerance $\tau_{\mathrm{N}} = \theta_2\|\mathbf{N}(\mathbf{v}^0)\|$ and $\theta_2 = 10^{-3}$, which is based on numerical tests and other papers' choices (Persson, 2009; Noventa et al., 2016). We also develop an adaptive Newton tolerance in Section 4.2, which is much less case-dependent.

4. The approximated solution $\mathbf{v}$ is then assigned into the time integration process

$$\mathbf{u}_{it}^{(i)} = \mathbf{v}^{k+1}, \tag{2.41}$$

where $\mathbf{u}_{it}^{(i)}$ is regarded as the approximate solution of the nonlinear system Eq. (2.32). It is used to distinguish with the discretized solution $\mathbf{u}^{(i)}$.

The remaining Newton residual after convergence is denoted as $\mathbf{R}(\mathbf{u}_{it}^{(i)})$. Recalling $\mathbf{u}$ is the exact solution of nonlinear equation system Eq. (2.32) and $\mathbf{u}_{it}^{(i)}$ is the iterative approximation, we can get $\mathbf{N}(\mathbf{u}^{(i)}) = 0$ and $\mathbf{N}(\mathbf{u}_{it}^{(i)}) = \mathbf{R}^{(i)}$. Thus we can derive the following relationship

$$\mathbf{N}(\mathbf{u}_{it}^{(i)}) - \mathbf{N}(\mathbf{u}^{(i)}) = (\mathbf{u}_{it}^{(i)} - \mathbf{u}^{(i)}) - \Delta t a_{i,i}\left(\mathcal{L}(\mathbf{u}_{it}^{(i)}) - \mathcal{L}(\mathbf{u}^{(i)})\right) = \mathbf{R}^{(i)}. \tag{2.42}$$

Using the above relationship, we will further discuss the contributions from the iterative error to the total discretization error in the following Section 2.5.

2.5 Analysis of error estimates

After the discretization, it is meaningful to exploit the components of errors, which could be a good reminder of accuracy convergence. Based on the estimates of these errors, we develop an adaptive time-stepping strategy and adaptive Newton tolerance for the nonlinear solver, which will be introduced in Section 4.1 and Section 4.2 respectively.

The local discretization errors introduced at the current time step mainly consist of spatial error, temporal error, and iterative error. The accumulation of these errors is referred to as global errors, which are challenging and time-consuming to estimate, such as considering the influence from initial condition and boundary conditions, the propagation of errors from old time steps and neighboring elements. Studies on the global error can be found in references such as (Shampine, 2005; Meisrimel and Birken, 2020). In the contrast, the estimate of the local error introduced at the current time step could help improve the solver's efficiency such as doing time-step adaptivity (Söderlind, 2002; Loppi et al., 2019), mesh adaptivity (Houston, 1999) without much costs.

We will give a brief derivation of the relationship among the local errors. We firstly recall the notations $\mathbf{U}$ is the mathematically exact solution, $\mathbf{U}_\delta$ is the spatial discretized solution, $\mathbf{U}^n$ is the temporal discretized solution at n^{th} time step, $\mathbf{U}^{(i)}$ is the i^{th} stage's temporal discretized solution, and $\mathbf{U}_{it} = \mathbf{B}\mathbf{u}_{it}$ is the iterative approximation (numerical solution). Because we are only evaluating the local time step errors, at time t^n, we assume all the values at $t = t^n$ are exact

$$\mathbf{U}(t^n) = \mathbf{U}^n = \mathbf{U}^{(0)} = \hat{\mathbf{U}}^n = \hat{\mathbf{U}}^{(0)}. \tag{2.43}$$

We recall that the numerical solution at the time step $t = t^{n+1}$ is calculated by

$$\mathbf{U}_{it}^{n+1} = \mathbf{U}^n + \Delta t \sum_{i=1}^{s} b_i L_\delta(\mathbf{U}_{it}^{(i)}), \tag{2.44}$$

and the exact solution $\mathbf{U}^{n+1}$ based on $\mathbf{U}^n$ can be expressed as

$$\begin{aligned}\mathbf{U}(t^{n+1}) &= \hat{\mathbf{U}}^{n+1} + O(\Delta t^{N+2}) \\ &= \mathbf{U}^n + \Delta t \sum_{i=1}^{s+1} \hat{b}_i L(\mathbf{U}^{(i)}) + O(\Delta t^{N+2}),\end{aligned} \tag{2.45}$$

where $O(\Delta t^{N+2})$ represents all the high-order truncation terms and is not detailed here. Subtracting Eq. (2.44) by Eq. (2.45) and ignoring the high-order truncation terms $O(\Delta t^{N+2})$, the total local error can be seen as the following difference

$$\mathbf{U}_{it}^{n+1} - \mathbf{U}(t^{n+1}) \approx \Delta t \sum_{i=1}^{s} b_i L_\delta(\mathbf{U}_{it}^{(i)}) - \Delta t \sum_{i=1}^{s+1} \hat{b}_i L(\mathbf{U}^{(i)}). \tag{2.46}$$

If we add then subtract the summation terms of $\Delta t \sum_{i=1}^{s+1} b_i L(\mathbf{U}^{(i)})$, $\Delta t \sum_{i=1}^{s} b_i L_\delta(\mathbf{U}_\delta^{(i)})$ to the right-hand side of Eq. (2.46), we can construct the following relationship

$$\begin{aligned}\mathbf{U}_{it}^{n+1} - \mathbf{U}(t^{n+1}) \approx\ & \Delta t \sum_{i=1}^{s+1} (b_i - \hat{b}_i) L(\mathbf{U}^{(i)}) \\ & + \Delta t \sum_{i=1}^{s} b_i \left(L_\delta(\mathbf{U}_\delta^{(i)}) - L(\mathbf{U}^{(i)}) \right) \\ & + \Delta t \sum_{i=1}^{s} b_i \left(L_\delta(\mathbf{U}_{it}^{(i)}) - L_\delta(\mathbf{U}_\delta^{(i)}) \right),\end{aligned} \tag{2.47}$$

where b_{s+1} is always zero in the embedded DIRK schemes we adopted (Kvrn, 2004). In the last term of Eq. (2.47), $\left(L_\delta(\mathbf{U}_{it}^{(i)}) - L_\delta(\mathbf{U}_\delta^{(i)}) \right)$ is related to the residual defined in Eq. (2.42) and can be considered as the iterative error $\mathbf{E}_{it}^{(i)}$. Recalling the spatial discretized error defined in Eq. (2.22) and the temporal discretized error defined in (2.29), and ignoring high-order terms $O(\Delta t^{N+2})$, $O(h^{P+2})$, Eq. (2.47) is rewritten as

$$\begin{aligned}\mathbf{U}_{it}^{n+1} - \mathbf{U}(t^{n+1}) \approx & C_t D_t(\mathbf{U}) \Delta t^{N+1} + \Delta t C_s \bar{D}_s(\mathbf{U}) h^{P+1} + \Delta t \bar{\mathbf{E}}_{it} \\ \approx & \mathbf{E}_t^{n+1} + \Delta t \bar{\mathbf{E}}_s + \Delta t \bar{\mathbf{E}}_{it} \\ = & \mathbf{B} \left(\mathbf{e}_t^{n+1} + \Delta t \bar{\mathbf{e}}_s + \Delta t \bar{\mathbf{e}}_{it} \right).\end{aligned} \tag{2.48}$$

The over-bar on variables such as $\bar{\mathbf{e}}_s$, $\bar{\mathbf{e}}_{it}$ denotes implementing a weighed average operation of the form of $\bar{\psi} = \sum_{i=1}^{s} b_i \psi^{(i)}$, where $\sum_{i=1}^{s} b_i = 1$.

Eq. (2.48) shows that the total local error per step is the sum of local temporal error, averaged spatial error and averaged iteration error. This indicates that to decrease the total error, these three errors all play an important role and should be decreased simultaneously. The solver will not improve much if only one of them is optimized to the best and the other two still contribute much to the total error.

2.6 Krylov linear solver

To calculate the Newton step $\Delta\mathbf{v}$ in Eq. (2.36), we require the solution of the linear equation system Eq. (2.37), that is simplified as

$$\mathbf{A}\mathbf{x} = \mathbf{b}, \tag{2.49}$$

where $\mathbf{A} = \frac{\partial \mathbf{N}(\mathbf{v}^k)}{\partial \mathbf{v}^k}$, $\mathbf{x} = \Delta\mathbf{v}$, and $\mathbf{b} = -\mathbf{N}(\mathbf{v}^k)$.

The matrix $\mathbf{A}$ arising from the discretization of the Navier-Stokes equations is large, sparse, and non-symmetric. The solving of Eq. (2.49) thus requires the use of Krylov solvers such as GMRES methods adopted here.

GMRES aims to minimize the following functional

$$J(\mathbf{y}) = \|\mathbf{b} - \mathbf{A}\mathbf{x}\| = \|\mathbf{b} - \mathbf{A}(\mathbf{x}_0 + \mathbf{W}_J \mathbf{y}_J)\| \tag{2.50}$$

in a J-dimensional Krylov subspace $(\mathbf{x}_0 + Kr_J)$, where the subspace is defined as

$$Kr_J = span\{\mathbf{r}_0, \mathbf{A}\mathbf{r}_0, \mathbf{A}^2\mathbf{r}_0, \ldots \mathbf{A}^{J-1}\mathbf{r}_0\}, \tag{2.51}$$

and $\mathbf{x}_0$ is the initial guess, $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$ is the initial residual.

Assuming GMRES stops if its residual is smaller than a pre-set convergence tolerance τ_{GMRES} after J iterations, the updated solution is expressed by J orthogonal

bases $\mathbf{W}_J = [\mathbf{w}_1 \ldots, \mathbf{w}_J]$ and the corresponding coefficient vector $\mathbf{y}$

$$\mathbf{x} = \mathbf{x}_0 + \mathbf{W}\mathbf{y}, \tag{2.52}$$

where the process of seeking orthogonal bases is called Arnoldi process (Walker, 1988; Saad, 2003). The algorithm of GMRES includes the Arnoldi process and the backward function to calculate the coefficients $\mathbf{y}$, the details of which are introduced in Appendix 6.2.

The process of classical GMRES is non-recurrent, the whole set of bases $\mathbf{W}$ has to be stored. Therefore the storage and cost per step increase linearly with the iteration number. The restarted GMRES(M) is adopted, which only stores M orthogonal bases (Saad and Schultz, 1986). After updating $\mathbf{x}_{\text{new}} = \mathbf{x}_0 + \mathbf{W}_M\mathbf{y}_M$ in the first round , namely, in the first Krylov space ($\mathbf{x}_0 + Kr_M$), the calculation is restarted in another Krylov subspace ($\mathbf{x}_{\text{new}} + Kr_M$). When M is small, the performance can be very poor. Through numerical tests and following other researchers' experience (Birken, 2013), we default set $M = 30$.

Given the large size of the matrix $\mathbf{A}$, GMRES is usually implemented using a matrix-free method. Instead of storing the details of the whole Jacobian matrix, matrix-free method calculates matrix-vector products in the following form

$$\mathbf{N}'(\mathbf{v}^k)\mathbf{w} \approx \frac{\mathbf{N}(\mathbf{v}^k + \epsilon\mathbf{w}) - \mathbf{N}(\mathbf{v}^k)}{\epsilon}, \tag{2.53}$$

where $\mathbf{w}$ refers to any search bases in $\mathbf{W}$ and ϵ is a perturbation parameter. The accuracy of the approximation Eq. (2.53) increases with the decreasing value of ϵ, while for small values, the accuracy is affected by the cancelling error and the rounding error. The choice for ϵ suggested in (Turner and Walker, 1992) is taken to be $\epsilon = 10^{-8}\|\mathbf{v}^k\|$ when using the first-order finite difference approximation. By storing $\mathbf{N}(\mathbf{v}^k)$, one extra evaluation of Newton residual $\mathbf{N}(\mathbf{v}^k + \epsilon\mathbf{w})$ is required per GMRES iteration.

However, because this type of matrix-free method approximates the multipli-

cations in finite difference form, the orthogonality of vectors in Krylov space is influenced since the Jacobian matrix keeps changing even though the disturbance is not obvious. How the accuracy of approximations influences the convergence will be discussed in Section 4.3.

In summary, the above process of combined use of Newton-type nonlinear solver, Krylov linear solver, and matrix-free technique are called JFNK. The whole process of JNFK is also summarized as Fig. 2.1. More references can be found in the review (Knoll and Keyes, 2004).

In practice, preconditioning techniques to improve the performance and reliability of a Krylov subspace solver play an important role in challenging problems. A preconditioner is expected to transform the original linear equation system into a less stiff system. Generally speaking, the preconditioner attempts to improve the spectral properties of the Jacobian matrix (Benzi, 2002). For normal Jacobian matrices, hopefully, the eigenvalues are clustered or the condition number is reduced after the transform, which indicates the speed-up of convergence rate. For non-normal systems, the relationship between spectral properties and the convergence rate of an iterative solver such as GMRES may not be directly related. However, if the system is not far from normal, spectral analysis is still an effective tool to assess the performance of a preconditioner. The details of the spectral analysis and the efficient implementation of a preconditioner are introduced in Section 3.

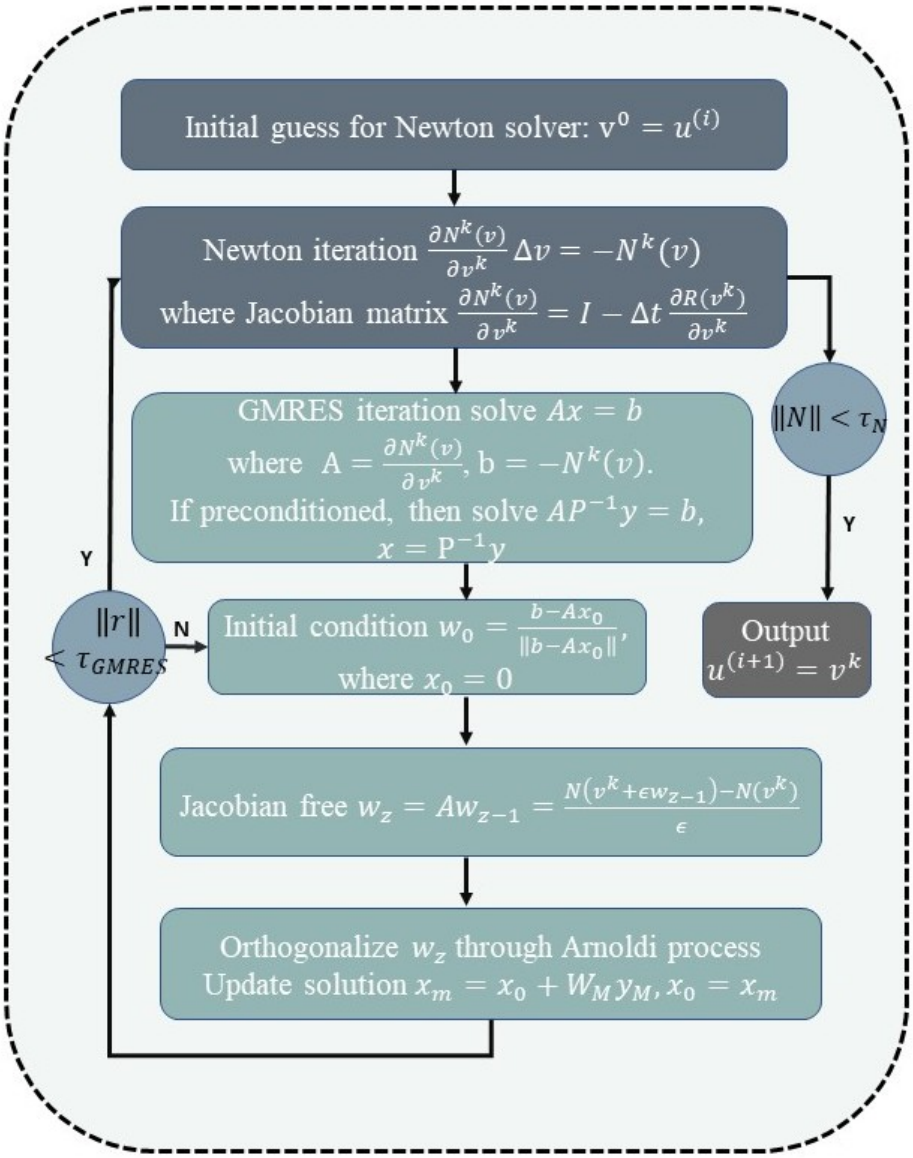


Figure 2.1: A flow chart of the JFNK process.

2.7 Implementation in Nektar++ framework

Nektar++ is open-source software that supports the development of high-performance solvers for partial differential equations (PDEs) based on the spectral/hp element framework, such as the diffusion solver, the incompressible flow solver, the compressible flow solver, etc. As illustrated in Fig. 2.2, a solver usually begins from a driver, which follows the traditional order of solving a PDE equation including initializing conditions, imposing boundary conditions, solving equations, and post-processing.

For scientific studies, Nektar++ provides a set of pre-developed solvers and separates the commonly-used operators into the library for further development. The library includes the projection operators, spatial discretization operators, temporal discretization operators, etc. Fig. 2.2 shows part of the functionalities of Nektar++. For implementing an efficient implicit compressible flow solver, the author contributes to adding the singly diagonally implicit Runge-Kutta to the time integration library, adding the interior penalty approach to the original spatial discretization library, creating a new library of nonlinear methods, complementing the GMRES linear solver within the original linear solver library, creating a new library of preconditioners including BRJ preconditioner and extra exploitation of shock-capturing methods. The more details about the routines of the implementation and the results simulated by the implicit solver are described in paper (Yan et al., 2020).

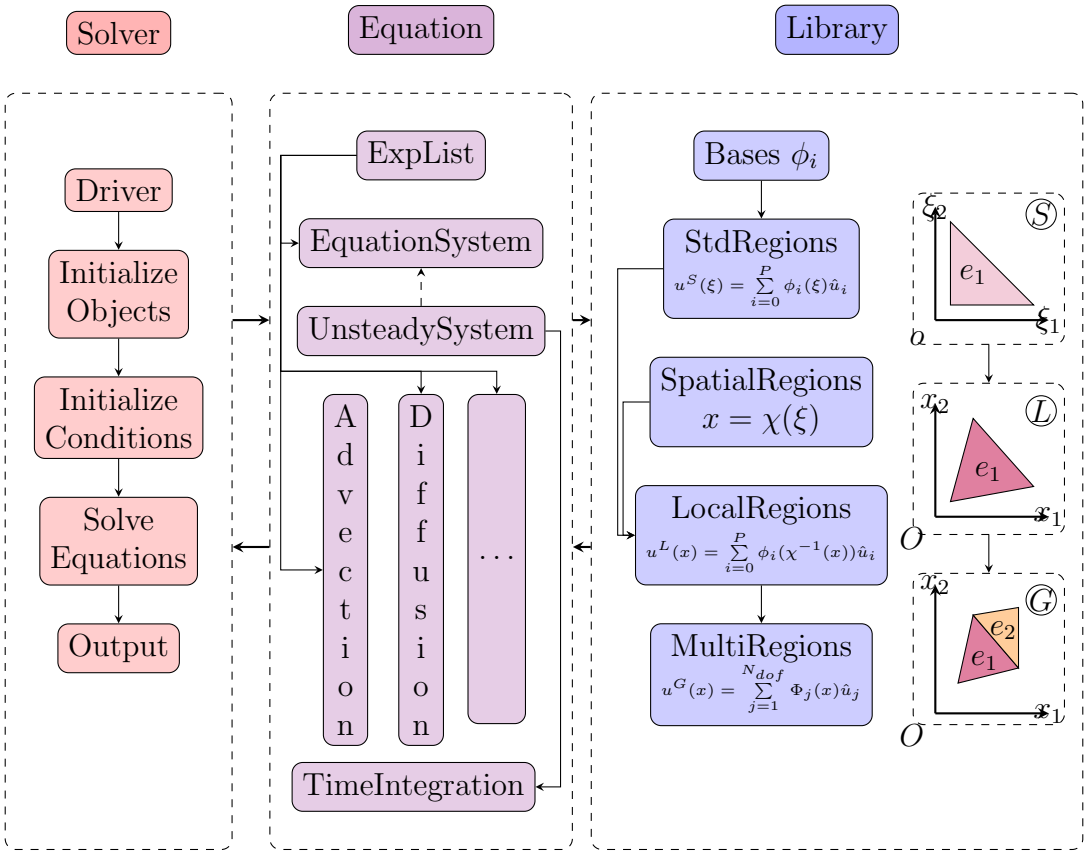


Figure 2.2: The components of a flow solver within Nektar++.

2.8 Verification of the implementations

The testing of the convergence order of accuracy of a solver is an effective tool to verify the codes (Roy, 2005; Oberkampf and Roy, 2010). In this section, we verify that simulations by the implicit compressible flow solver can achieve the desired order of accuracy in space and time through several benchmark tests. The 2D unsteady isentropic vortex convection is used to verify the discretization of advection terms while the 2D Couette flow is used to verify the diffusion terms. Furthermore, the unsteady flow past a cylinder case is utilized to verify the implementation of the time integration part.

2.8.1 Advection: 2D isentropic vortex convection

The 2D unsteady isentropic vortex convection is one standard test case for the verification of codes regarding with Euler equation. The test is widely-adopted because its analytical solution is available. An inviscid isentropic vortex is convected downstream in the computational region [0,10]×[-5,5], the analytical solutions of which at time t are

$$u = u_\infty - \varphi\bar{y}\exp(1-\bar{r}^2), \tag{2.54}$$

$$v = v_\infty + \varphi\bar{x}\exp(1-\bar{r}^2), \tag{2.55}$$

$$T = 1 - \frac{(\gamma-1)\varphi^2}{8\gamma\pi^2}\exp(2-2\bar{r}^2), \tag{2.56}$$

$$\rho = T^{\frac{1}{\gamma-1}}, \tag{2.57}$$

where $\varphi = 0.5$ is a parameter that controls the strength of the vortex, (u_∞, v_∞)=$(1.0, 0.0)$ are the far-field x, y direction velocities. The initial coordinate of the centre of perturbation is $x_0 = 5$ and $y_0 = 0$, the other input variables are

$$\bar{x} = x - x_0 - u_\infty t, \quad \bar{y} = x - x_0 - v_\infty t, \quad \bar{r} = \sqrt{\bar{x}^2 + \bar{y}^2}. \tag{2.58}$$

The boundary conditions are periodic and the initial condition is depicted in

Fig. 2.3.

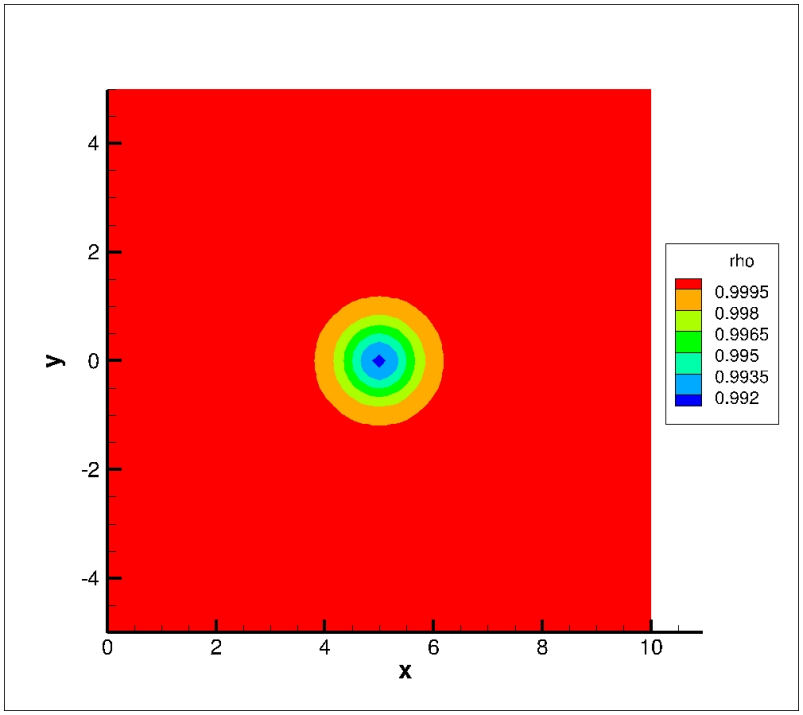


Figure 2.3: 2D convected isentropic vortex: Initial condition.

The simulations utilize a small time step CFL = 0.1 and a fourth-order DIRK scheme to ensure the spatial error is dominant. To notice, the commonly-adopted absolute/relative Newton tolerances (Eq. 2.39/Eq. 2.40) are always problem dependent. To ensure the iteration errors do not influence the estimate of discretization order, the Newton tolerance is set very strict $\theta_2 = 10\times^{-6}$. The cartesian meshes are gradually refined from $[10\times10]$ to $[40\times40]$ elements. The numerical results extracted at $T = 1s$ using different DG polynomials from the first order to the fourth-order are compared with the analytical solution respectively. Their differences are referred to as spatial discretization errors. The error distributions are drawn with respect to the mesh size h in Fig. 2.4. The observed order of accuracy (OoA) of all the simulations, tested by the slopes of the error distribution lines, approximately reaches the designed order of accuracy, which verifies the implementation of the advection discretization codes.

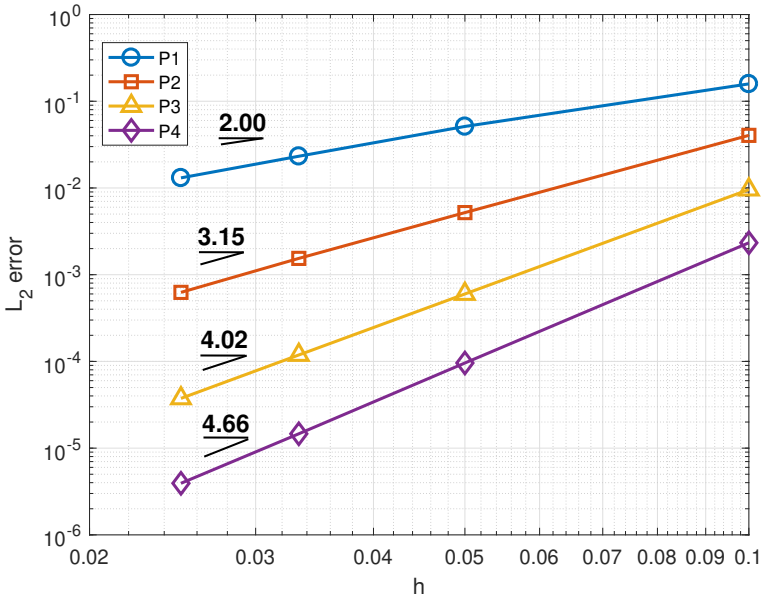


Figure 2.4: 2D convected isentropic vortex:
Observed orders of spatial accuracy.

2.8.2 Diffusion: 2D Couette flow

This is one of the very few compressible flow cases available with an analytical solution. The 2D Couette flow is a laminar viscous flow between two parallel plates. The exact solution (Liepmann and Roshko, 2001) needs complicated integration and the dynamic viscosity needs to obey specific power law $\frac{\mu}{\mu_\infty} = (\frac{T}{T_\infty})^{0.76}$. If the Mach number is low, the exact solution is close to the analytical solution of the incompressible Couette flow

$$u = \frac{y}{H} U_\infty, \tag{2.59}$$

$$v = 0, \tag{2.60}$$

$$p = p_\infty, \tag{2.61}$$

$$T = T_0 + \frac{y}{H}(T_1 - T_0) + \frac{y}{H}\left(1 - \frac{y}{H}\right)\frac{P_r U_\infty^2}{2C_p}, \tag{2.62}$$

where in the tests, the flow conditions are set as M = 0.1 with $U_\infty = 34m/s$. The temperature at the bottom wall and the upper wall are $T_0 = 240K$, $T_1 = 255K$

respectively.

The order of accuracy (OoA) study is then applied to verify the codes for solving the compressible Navier-Stokes equations using interior penalty (IP) discretization. The meshes are only gradually refined in the y-direction with mesh densities N_e =2, 3, 4, and 5. To keep spatial discretization errors dominant, the simulations are run using a small time step CFL = 0.1 and a fourth-order DIRK scheme. The results are extracted when the norm of the residual $L_\delta(\mathbf{U}_\delta)$ defined in Eq. (2.21) is reduced by 10^{-10} compared with the initial residual norm $\|L_\delta(\mathbf{U}_0)\|$ of the simulation. The results of simulations using DG polynomial order from the first-order to the fourth-order are compared with the analytical solution. The simulation using $P = 4$ does not run in the densest mesh because the simulation is so easy to reach a low-level error, which is near the rounding error level. The error distribution lines drawn in Fig. 2.5 show the simulations with polynomial orders P from 1 to 4 all approximately reach their designed order of accuracy, which verifies the implementation of the spatial discretization codes in the compressible flow solver.

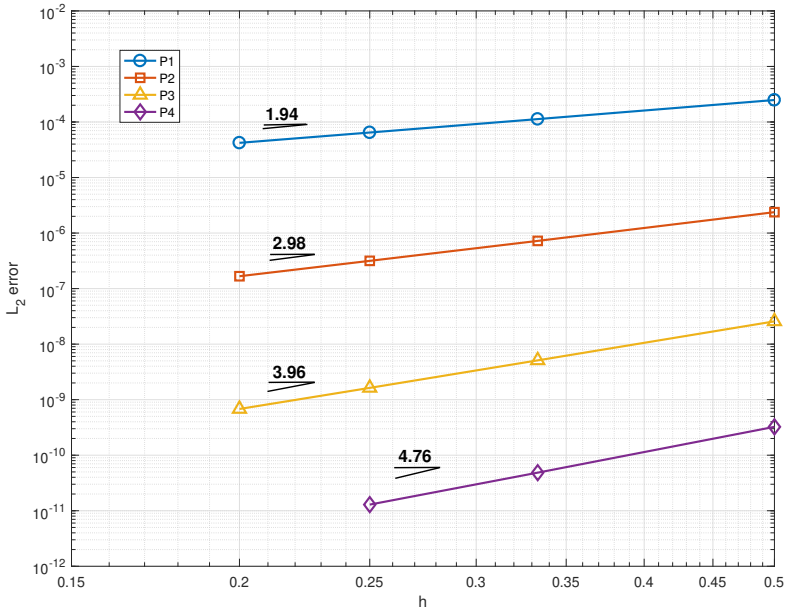


Figure 2.5: 2D Couette flow:
Observed orders of spatial accuracy.

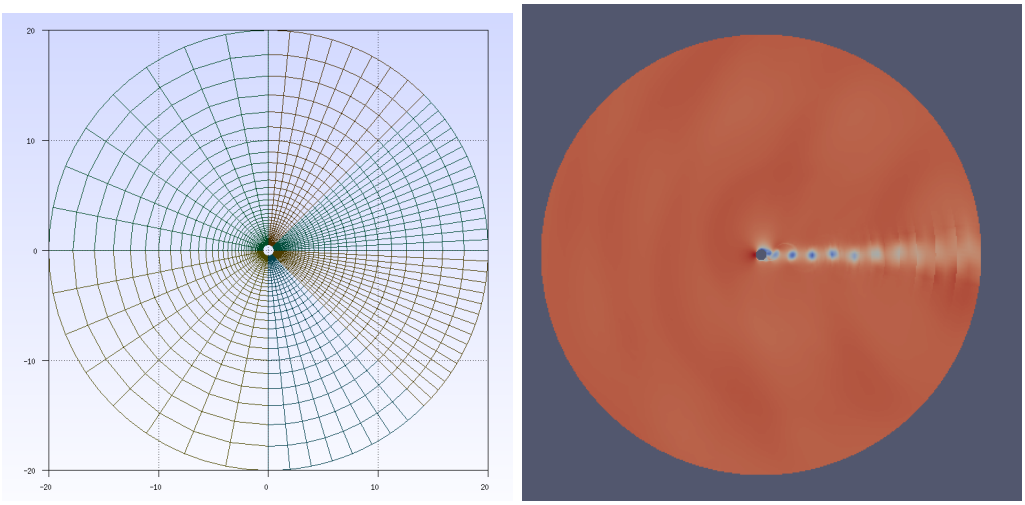
2.8.3 Time integration: 2D flow past a circular cylinder

An unsteady vortex flow past a circular cylinder is used to verify the temporal accuracy of the implicit time integration schemes. The simulation is run under free-stream Mach number M = 0.3 and Reynolds number Re = $\frac{\rho u D}{\mu}$ = 1200 based on the free-stream velocity u and cylinder diameter D. The far-field boundary condition is applied at radial distance $r = 20D$ from the center of the cylinder and non-slip wall boundary condition is set at cylinder surface $r = D/2$. Before testing the order of accuracy (OoA) of time integration schemes, the mesh convergence study is used to ensure the spatial error is not dominant. We test in a set of refined mesh with N_r = 20, 40, 60 and 80 (N_r is the mesh density in radial direction while the tangential mesh density $N_t = 64$ remains constant). The flow is periodic and the computed Strouhal number St shown in Table. 2.3 is compared with the results by (Bijl et al., 2002). The following case running under mesh density $N_r = 60$ has converged. The later analysis of temporal accuracy is based on the mesh with $N_r = 60$.

Mesh	Strouhal number
20	0.2397
30	0.2407
60	0.2438
80	0.2438
Bijl et al. (2002)	0.2467

Table 2.3: Mesh convergence study.

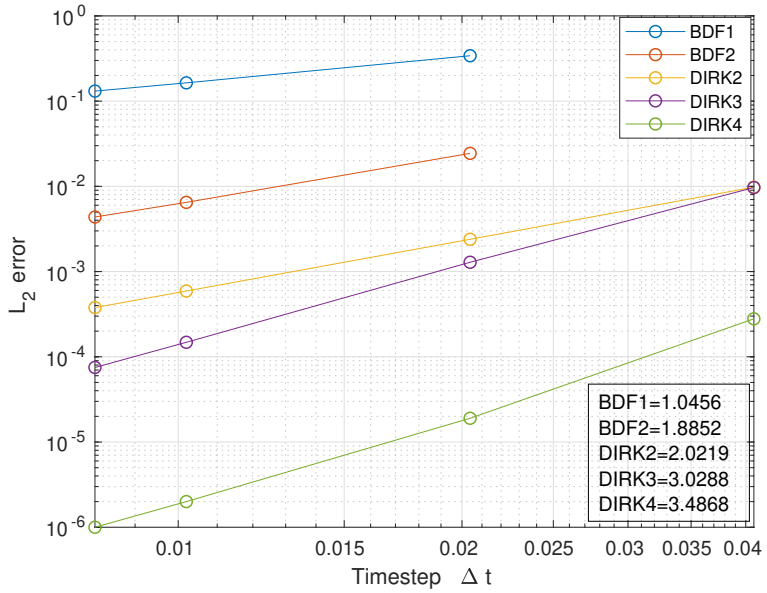
The simulation is run for more than 50 periods to ensure the vortex sheds periodically. The converged mesh and the initial field is shown as Fig. 2.6.



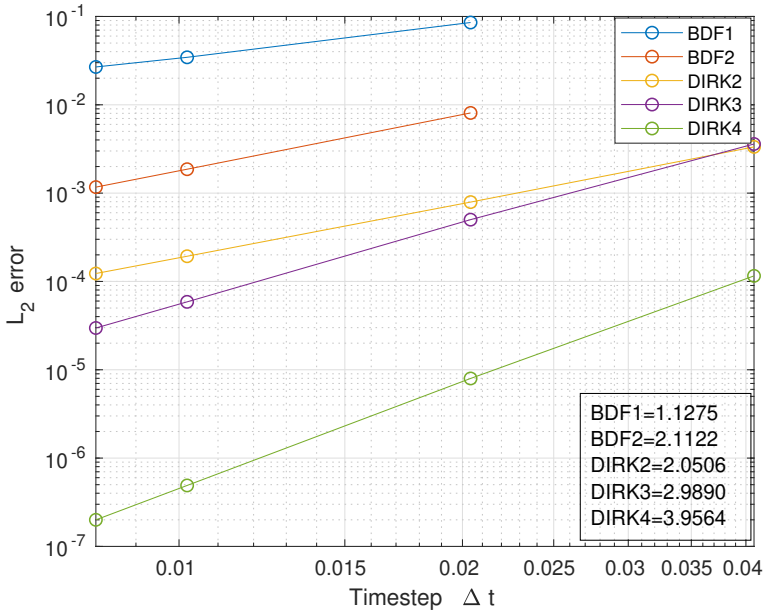
(a) Mesh (b) Periodic vortex shedding

Figure 2.6: 2D flow past a circular cylinder: Mesh and initial field.

A time interval of 1.7 periods is enough to accumulate temporal errors (Bijl et al., 2002). Therefore, after 1.7 periods, lift and drag coefficients are chosen to test the accuracy of time integration schemes. The time step samples for BDF series are (a) 8×10^{-5} (b) 10^{-4}, and (c) 2×10^{-4} while for DIRK series are (a) 8×10^{-5} (b) 10^{-4} (c) 2×10^{-4}, and (d) 4×10^{-4}. The result obtained by the simulation with $\Delta t = 4 \times 10^{-5}$ using the fourth-order DIRK is treated as the 'numerically exact' reference. The error distributions and consistently observed order of accuracy (OoA) of time integration schemes are obtained through the comparisons between the sample results with the reference solution. Fig. 2.7 shows all the schemes achieve their designed order of accuracy. The OoA of BDF1 and BDF2 are tested approximately 1.0 and 2.0 while OoA of DIRK2, DIRK3 are tested approximately 2.0, 3.0 shown in Fig. 2.7a and Fig. 2.7b. The OoA test using the lift coefficient of DIRK4 is only 3.48 because the lift coefficient of this case is small. Since the accuracy level achieved using DIRK4 is very close to that of the 'numerically exact' solution, even rounding errors could result in an inexact test of OoA. This phenomenon does not appear when the OoA is tested using drag coefficient, in which the OoA of DIRK4 is approximately 4 (3.98). In conclusion, the achievement of expected temporal accuracy in these OoA tests verifies the codes of the implicit solver.



(a) Lift OoA comparison



(b) Drag OoA comparison

Figure 2.7: 2D flow past a circular cylinder: Observed orders of temporal accuracy.

Chapter 3

Preconditioners for linear solvers

In this chapter, we consider the preconditioning of the linear problem arising from the linearization of the nonlinear system during Newton iterations. Although one Krylov solver can be efficient compared with other types of linear solvers, it still suffers from the slow speed of convergence in large-scale stiff problems. The idea of preconditioning is to transform the system into a less stiff problem, which was first proposed in (Evans, 1968), and is the key ingredient to accelerate the convergence.

As discussed in Section 2.6, the calculation of the Newton step $\Delta\mathbf{v}$ is a linear system resolution

$$\mathbf{A}\mathbf{x} = \mathbf{b}, \tag{3.1}$$

where $\mathbf{A} = \frac{\partial \mathbf{N}(\mathbf{v})}{\partial \mathbf{v}}$, $\mathbf{x} = \Delta\mathbf{v}$, and $\mathbf{b} = -\mathbf{N}(\mathbf{v})$.

In terms of solving a linear system, it is obvious that when $\mathbf{A}$ is close to the identity matrix $\mathbf{I}$, the system is easier to solve. A nonsingular preconditioner matrix $\mathbf{P}$, which approximates $\mathbf{A}$, is therefore designed. Through left or right multiplication of $\mathbf{P}^{-1}$, the linear system is transformed to Eq. (3.2) or Eq. (3.3)

$$\mathbf{P}^{-1}\mathbf{A}\mathbf{x} = \mathbf{P}^{-1}\mathbf{b}, \tag{3.2}$$

$$\mathbf{A}\mathbf{P}^{-1}\hat{\mathbf{x}} = \mathbf{b}, \tag{3.3}$$

where the preconditioned matrices $\mathbf{P}^{-1}\mathbf{A}$ and $\mathbf{A}\mathbf{P}^{-1}$ are expected to approximate

the identity matrix. The corresponding preconditioning methods are called left and right preconditioning. The right preconditioned system Eq. (3.3) needs the following extra step to achieve the final result

$$\mathbf{x} = \mathbf{P}^{-1}\hat{\mathbf{x}}. \tag{3.4}$$

The default strategy of preconditioning in Nektar++ is right preconditioning because this form keeps the residual unchanged after preconditioning and makes it easier to observe the residual convergence.

Recalling the formulation of using Jacobian-free Eq. (2.53), the approximation of the matrix-vector products for right preconditioning is

$$\left(\frac{\partial \mathbf{N}(\mathbf{v})}{\partial \mathbf{v}}\mathbf{P}^{-1}\right)\mathbf{w} = \frac{\mathbf{N}(\mathbf{v} + \epsilon\mathbf{P}^{-1}\mathbf{w}) - \mathbf{N}(\mathbf{v})}{\epsilon}, \tag{3.5}$$

thus the equivalent system of preconditioning can be denoted as

$$\mathbf{P}\mathbf{y} = \mathbf{w}. \tag{3.6}$$

In terms of the studies on the convergence theory for an efficient preconditioner, as mentioned in (Greenbaum, 1997), the exact sense in which the preconditioned matrix should approximate the identity depends on the iterative method being used. For simple iterative methods, the spectral radius $\|\mathbf{I}-\mathbf{A}\mathbf{P}^{-1}\| \ll 1$ is a good indicator of achieving fast asymptotic convergence rate. For symmetric systems using the conjugate gradient (CG) method, it can be proved, for example see (Benzi, 2002; Hogben, 2006), that the convergence rate is linked with the eigenvalue distributions. In other words, a system with preconditioned matrices of only a few tightly clustered eigenvalues are typically easier to solve.

When using GMRES to solve the nonsymmetric linear system, unfortunately, there is not a general answer to assess the performance of a preconditioner (Nachtigal et al., 1992). If the preconditioned matrix is not far from normal, the cluster of

eigenvalues could still be a good indicator of an efficient preconditioning strategy. Other spectral properties like condition number can also indicate the stiffness of preconditioned linear equation system. Therefore, the eigenvalue analysis of the system after preconditioning is still an acceptable tool to assess a preconditioner's performance.

In terms of the classification of preconditioners, the most widely-used preconditioners come from the modification of linear solvers. Splitting-type linear solvers include the Jacobi method, Gauss-Seidel method, etc. These methods, even though inferior to Krylov solvers as linear solvers, have numerically shown in (Diosady and Darmofal, 2007; Ghai et al., 2019) to be efficient preconditioners. The idea of incomplete LU (ILU) factorization preconditioner arises from the use of the inexact linear solver. Alternatively, p-multigrid solver can be used as an efficient linear solver. It utilizes multigrid theory to smooth high-frequency errors in low-order resolution. Its counterpart as a preconditioner can also be used to accelerate a Krylov solver efficiently in some problems (Luo et al., 2006). The details of these preconditioners will be introduced in Section 3.1. The convergence properties after preconditioning will also be investigated through eigenvalue analysis in the test case of Lid-driven cavity flow. Subsequently in Section 3.2, considering the balance of storage and efficiency, a practical implementation of the preconditioner in Nektar++ will be introduced and the efficiency of the implicit solver using the developed preconditioner will be tested.

3.1 Eigenspectral analysis of preconditioners

Due to the use of the Jacobian-free method, the Jacobian matrix is not explicitly stored. To perform eigenspectral analysis, we explicitly generate the Jacobian matrix tested using a first-order finite difference method as follows

$$\mathbf{A}(\mathbf{v})\vec{\mathbf{e}}^l = \frac{\mathbf{N}(\mathbf{v} + \epsilon\vec{\mathbf{e}}^l) - \mathbf{N}(\mathbf{v})}{\epsilon}, \tag{3.7}$$

where $\mathbf{e}^l$ is a unit vector of l^{th} entry equals 1 and the perturbation parameter is set $\epsilon = 10^{-12}$. The right hand side of Eq.(3.7) is an approximation of l^{th} column's derivatives in Jacobian matrix $\mathbf{A}$.

After the numerical generation of $\mathbf{A}$, the matrix is then aligned as a global matrix including $(N_e \times N_e)$ block matrices, where the derivatives of the i^{th} element's residual $\mathbf{N}(\mathbf{v}_{e_i})$ with respect to the j^{th} element's solutions $\mathbf{v}_{\mathrm{j}}$ are clustered in the $(i,j)^{th}$ block matrix. A sketch of the alignment of Jacobian matrix is shown as Fig. 3.1. To notice, the sketch does not consider the number of conservative variables.

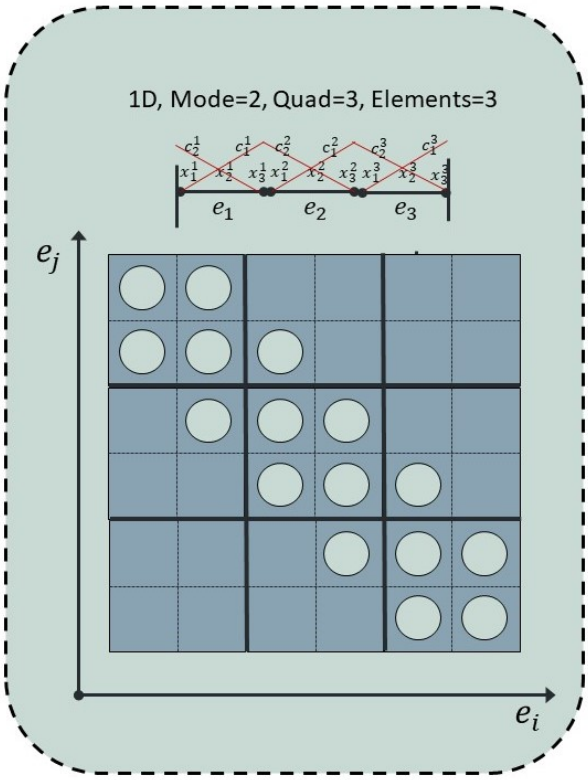


Figure 3.1: Sketch of the Jacobian matrix:
3 elements using $P = 1$ DG polynomial (nDof=2 per element).

The operations of preconditioning are block matrix based, where each block matrix operation is treated similarly to an entry operation.

The preconditioners studied in this section are block Jacobi (BJac), block Gauss-Seidel (BGS), block ILU (BILU), and block p-multigrid preconditioners. The generation and usage of these preconditioners are also based on the numerically generated

Jacobian matrix.

The process of numerically generating a Jacobian matrix takes up large storage and spends costly computation. This is not practical for large-scale problems and is not implemented in Nektar++. This spectral analysis is only tested in a small-scale problem Lid-driven flow to help us understand the performance of preconditioners before developing a practical preconditioner.

3.1.1 Block spliting preconditioner

A standard linear equation system is denoted as

$$\mathbf{A}\mathbf{y} = \mathbf{c}. \tag{3.8}$$

The 'splitting' type linear solvers are named because the target matrix $\mathbf{A}$ is split into several parts

$$\mathbf{A} = \mathbf{L} + \mathbf{D} + \mathbf{U}, \tag{3.9}$$

where $\mathbf{L}$, $\mathbf{D}$ and $\mathbf{U}$ are the lower triangular, diagonal, and upper triangular parts.

Considering $\mathbf{D}$ is nonsingular, Eq. (3.8) is equal to the following fixed point equation

$$\mathbf{y} = \mathbf{D}^{-1}(\mathbf{c} - (\mathbf{L} + \mathbf{U})\mathbf{y}). \tag{3.10}$$

The corresponding splitting method is called Jacobi iterations. The process of Z Jacobi iterations is

$$\begin{aligned} \mathbf{y}_0 &= 0 \\ \mathbf{y}^z &= \mathbf{D}^{-1}(\mathbf{c} - (\mathbf{L} + \mathbf{U})\mathbf{y}^{z-1}) \\ z &= 1, 2 \ldots, Z \end{aligned} \tag{3.11}$$

During the above process, $\mathbf{A}^{-1}$ is approximated by

$$\mathbf{D}^{-1}\{\mathbf{I} - (\mathbf{L} + \mathbf{U})\mathbf{D}^{-1} \ldots - \left((\mathbf{L} + \mathbf{U})\mathbf{D}^{-1}\right)^{Z-1}\}, \tag{3.12}$$

where $\mathbf{I}$ is the identity matrix. Thus the process can implicitly work as preconditioning, namely Jacobi preconditioning. The preconditioning process Eq. (3.6) is implicitly solved as long as $\mathbf{c}$ is replaced by $\mathbf{w}$.

The consistent block Jacobi (BJac) preconditioner replaces the entry operation during Jacobi iterations by block operation.

When $Z = 1$, we have a block diagonal (BDia) preconditioner. Only the diagonal blocks in $\mathbf{D}$ are evaluated, inverted and stored for preconditioning. The implementation of the BDia preconditioner is simple and saves storage. However, it performs poorly in stiff problems because it is not enough to approximate $\mathbf{A}^{-1}$ by $\mathbf{D}^{-1}$. As the smoothing iteration Z increases, the information from neighboring elements is utilized, which typically leads to better preconditioning performance. However, the factors of extra storage, computational cost, and parallel communication also need consideration. It is a key gradient to balance the storage and computational cost to construct an efficient iterative preconditioner.

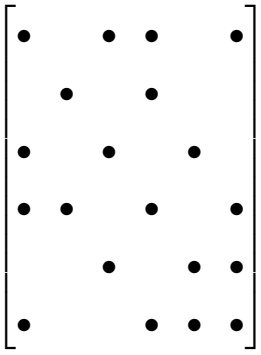
Gauss-Seidel method is another splitting method. During z^{th} iteration, after calculating the i^{th} block's variables in vector $\mathbf{y}^z$, the updated solution is immediately used for the solving of the next block's variables. The process of Z Guass-Seidel iterations is

$$\begin{aligned} \mathbf{y}_0 &= 0 \\ \mathbf{y}^z &= \mathbf{D}^{-1}(\mathbf{c} - \mathbf{L}\mathbf{y}^z - \mathbf{U}\mathbf{y}^{z-1}) \\ z &= 1, 2 \ldots, Z \end{aligned} \tag{3.13}$$

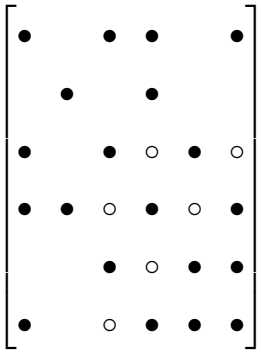
The corresponding block preconditioner is block Gauss-Seidel (BGS) preconditioner. Compared with the BJac preconditioner, the information is propagated further and updated more immediately after each Gauss-Seidel iteration. It is advantageous in situations like encountering shocks. However, it also has its drawback such as more frequent parallel communication due to the sequential process. For methods to improve parallel efficiency such as multi-coloring reordering, etc, readers can read (Sato et al., 2013; Li and Saad, 2013).

3.1.2 Block ILU preconditioner

The standard LU factorization is factoring $\mathbf{A} = \tilde{\mathbf{L}}\tilde{\mathbf{U}}$ through Gauss elimination. The structure of the remaining matrix to store $\tilde{\mathbf{L}}$ and $\tilde{\mathbf{U}}$, in the form of $\mathbf{C} = \tilde{\mathbf{L}} + \tilde{\mathbf{U}} - \mathbf{I}$ (3.15), is typically denser than the original matrix (3.14), where $\mathbf{I}$ is the identity matrix, $\tilde{\mathbf{L}}$ is the lower triangular matrix, and $\tilde{\mathbf{U}}$ is the upper triangular matrix. The empty circles in (3.15) are examples to mark the new generated entries after the matrix factorization.



(3.14)



(3.15)

The new generated entries usually appear in the positions surrounded by the original entries, which depends on the process of LU factorization. These new entries are called fill-ins.

For a large sparse matrix $\mathbf{A}$, a large number of fill-ins make the storage enlarged significantly. Incomplete LU factorization denoted as ILU(0) is designed so that the remaining matrix $\mathbf{C}$ does not have any fill-ins after factorization. The ILU factorization algorithms that keep more fill-in are also good black-box preconditioners, but obviously require more storage and cost. Assuming a set S includes the positions of the entries in original matrix $\mathbf{A}$ such as the black circles demonstrated in Eq. (3.14), the algorithm of ILU(0) is as follows

for $i = 2, \ldots, N_e$ **do**

  for $k = 1, \ldots, i-1$ **And** $(i,k) \in S(\mathbf{A})$ **do**

    Calculate the block matrix $\mathbf{A}_{i,k} = \mathbf{A}_{i,k}/\mathbf{A}_{k,k}$

for $j = k+1, \ldots, N_e$ **And** $(i,j) \in S(\mathbf{A})$ **do**

Calculate the block matrix $\mathbf{A}_{i,j} = \mathbf{A}_{i,j} - \mathbf{A}_{i,k} \times \mathbf{A}_{k,j}$

end for

end for

end for

By discarding the calculation when generating fill-in entries, it leads to significantly less computational cost for a sparse matrix. Although ILU factorization can not be used as an exact linear solver, it can be treated as an efficient preconditioner for a Krylov iterative solver. It makes use of the lines of maximum coupling information of the flow, which promotes convergence (Diosady and Darmofal, 2007). The efficiency of the ILU preconditioner has been numerically verified in various works (Magolu monga Made et al., 2000; Diosady and Darmofal, 2007; Persson and Peraire, 2008), especially for steady problems (Ur Rehman et al., 2008).

The applications of ILU preconditioners also encounter some challenging problems. The storage requirement in large-scale problems is a headache. Most implementations of ILU both store the linearization $\mathbf{A}$ and incomplete factorization $\tilde{\mathbf{L}}\tilde{\mathbf{U}}$. Even though implementing ILU using some optimized algorithms, such as the in-place storage mentioned in (Diosady and Darmofal, 2007), the process of ILU(0) preconditioning still takes up more than the size of a Jacobian matrix. Furthermore, parallel implementation of an ILU preconditioner has still been recognized as a challenging problem. This drawback is inherent from Gaussian elimination, on which ILU is based. The forward and backward triangular solving during ILU preconditioning are also highly sequential, which are ill-suited for parallelization. Domain decomposition techniques such as Schur complement methods, alternating Schwarz methods (Smith et al., 1998) can improve the parallel efficiency, but most CFD solvers lack the required functions of mesh decomposition or mesh overlapping.

During the process of block ILU (BILU) preconditioning, the operations of block matrices are treated as an entry operation in ILU processing. In the later tests, the BILU(0) preconditioner is referred to as BILU.

3.1.3 p-multigrid preconditioner

For linear iterative solvers such as Jacobi or Gauss-Seidel solvers, the number of operation counts required for convergence still scales poorly with the problem size (Brandt, 1982; Braess, 1982; Kovac and Strang, 2005). This is because the iteration process involves the coupled low-frequency and high-frequency errors. The high-frequency errors can be smoothed within a few iterations while low-frequency errors are hard to reduce. To remove the low-frequency errors, a multigrid solver attempts to transform the algebraic equation to be solved in a hierarchical coarse mesh. The original 'low-frequency' errors behave like high-frequency errors in the coarse mesh. If optimized well, the required iteration number could be mesh-independent. The multigrid algorithms using several sets of mesh are called geometric multigrid or h-multigrid. The natural extensions of h-multigrid to high-order finite element formulation such as spectral/hp element methods, where the equations using a different order of polynomial approximations solved recursively, are called p-multigrid. The application of p-multigrid algorithms on preconditioning is called p-multigrid preconditioners.

The process of p-multigrid can be summarized as 'smooth high-frequency errors in high-order system and solve the equation in the lower-order system'. The first step is to pre-smooth the solutions at the current level using smoothers like BJac or BGS mentioned in Section 3.1.1.

$$\mathbf{y}_1^{Z_1} = \mathbf{D}^{-1}(\mathbf{c} - (\mathbf{L} + \mathbf{U})\mathbf{y}_1^{Z_1 - 1}). \tag{3.16}$$

After obtaining the solution after Z_1 iterations, the equation residual vector $\mathbf{r} = \mathbf{c} - \mathbf{A}\mathbf{y}_1^{Z_1}$ at high-level system is projected to the lower-level resolution

$$\mathbf{r}^l = \tilde{\mathbf{R}}(\mathbf{c} - \mathbf{A}\mathbf{y}_1^{Z_1}), \tag{3.17}$$

where the restriction matrix $\tilde{\mathbf{R}}$ is a projection matrix to map the residual to the lower level. The superscript l denotes the lower level.

The defect equation at the lower level is resolved to obtain the corrected error $\mathbf{e}^l$.

$$\mathbf{A}^l\mathbf{e}^l = \mathbf{r}^l, \tag{3.18}$$

where $\mathbf{A}^l$ is the operation matrix for the defect equation. It is related to the projection matrices $\tilde{\mathbf{R}}$ and $\tilde{\mathbf{P}}$ (mentioned below).

The error is then interpolated to correct high-level approximations $\mathbf{x}_1^{Z_1}$

$$\mathbf{y}_2 = \mathbf{y}_1^{Z_1} + \tilde{\mathbf{P}}\mathbf{e}^l, \tag{3.19}$$

where the prolongation matrix $\tilde{\mathbf{P}}$ is a projection matrix to map the error back to the high-level system.

At the working level (highest level), the corrected solution needs post-smoothing to obtain the final solution $\mathbf{y}_2^{Z_2}$

$$\mathbf{y}_2^{Z_2} = \mathbf{D}^{-1}(\mathbf{c} - (\mathbf{L} + \mathbf{U})\mathbf{y}_2^{Z_2-1}). \tag{3.20}$$

The transform matrices $\tilde{\mathbf{P}}$ and $\tilde{\mathbf{R}}$ are determined by the methods of projection. Recalling the DG methods introduced in Section 2.2, for P^{th} order DG discretization, the space of test functions is defined as

$$D_P = \{v \in L^2(\Omega) : v|\Omega_e \in \mathbb{P}_P(\Omega_e), \forall\Omega_e\}, \tag{3.21}$$

where $\mathbb{P}_P(\Omega_e)$ represents the space of all polynomials of degree ($p \leq P$) in Ω_e.

Additionally, the inner product in D_P space is denoted as $(.,.)_P$ and the lower-order space D_{P-1} is defined as

$$D_{P-1} = \{v \in L^2(\Omega) : v|\Omega_e \in \mathbb{P}_{P-1}(\Omega_e), \forall\Omega_e\}. \tag{3.22}$$

The prolongation operator $\tilde{\mathbf{P}}$ is to interpolate the approximation from D_{P-1} to

D_P

$$\tilde{\mathbf{P}}v = v, \forall v \in D_{P-1} \subset D_P. \tag{3.23}$$

The variational formulation of Eq. (3.23) is

$$(\tilde{\mathbf{P}}v, w)_P = (v, w)_P, \forall v \in D_{P-1}, w \in D_P. \tag{3.24}$$

Therefore, the prolongation operator $\tilde{\mathbf{P}}$ can be calculated by

$$\tilde{\mathbf{P}} = (\mathbf{M}_P^P)^{-1}\mathbf{M}_{P-1}^P, \tag{3.25}$$

where the rectangular matrix $\mathbf{M}_{P-1}^P(i,j) = (\phi_i^P, \phi_j^{P-1})_P$, and $\mathbf{M}_P^P$ is the mass matrix. The superscript of ϕ^P denotes the order of approximation.

The derivation of restriction operator $\tilde{\mathbf{R}}$ is similar. Because $\tilde{\mathbf{R}}$ must the the Hilbert adjoint of $\tilde{\mathbf{P}}$, the following relations need to satisfy when restricting D_P to D_{P-1}

$$(\tilde{\mathbf{R}}w, v)_{P-1} = (w, \tilde{\mathbf{R}}v)_P = (w, v)_P, \forall v \in V_{P-1}, w \in V_P. \tag{3.26}$$

Therefore, the restriction operator $\tilde{\mathbf{R}}$ is calculated by

$$\tilde{\mathbf{R}} = (\mathbf{M}_{P-1}^{P-1})^{-1}\mathbf{M}_P^{P-1}, \tag{3.27}$$

where $\mathbf{M}_P^{P-1} = (\mathbf{M}_{P-1}^P)^T$ because $\mathbf{M}_P^{P-1}(i,j) = (\phi_i^{P-1}, \phi_j^P)_P$, $\mathbf{M}_{P-1}^{P-1}$ is the mass matrix of the lower-order system.

It is also easy to further derive the lower system operation matrix $\mathbf{A}^l$ in Eq. (3.18) as

$$\mathbf{A}^l = \tilde{\mathbf{R}}\mathbf{A}\tilde{\mathbf{P}}. \tag{3.28}$$

The above process only demonstrates the process of two-level multigrid. The process can involve more levels and solve the system recursively. If the resolution reaches the lowest level, the governing equation Eq. (3.18) can also utilize BILU as its smoother. Because the resolution at the lowest level requires much less compu-

tational cost.

3.1.4 Spectral analysis of preconditioned matrices for a Lid-driven cavity flow

The lid-driven cavity problem is an isothermal, incompressible flow in a 2D square domain, where all the boundaries are walls. The top wall moves in u=1m/s in x-direction while the other three walls are stationary. This test is adopted for spectral analysis of the preconditioned matrices in consideration of its small size. Especially in the case using high polynomial order with p-multigrid preconditioner, the storage of the numerically-generated Jacobian matrix and the calculation of spectral information, such as eigenvalues and condition numbers, are costly if testing in a large-scale problem. The simulations are run under at a low Reynolds number of Re = 5. The spectral analysis is based on the converged flow field, whose residual $L_\delta(\mathbf{U}_\delta)$ is reduced by a factor of 10^{-9}.

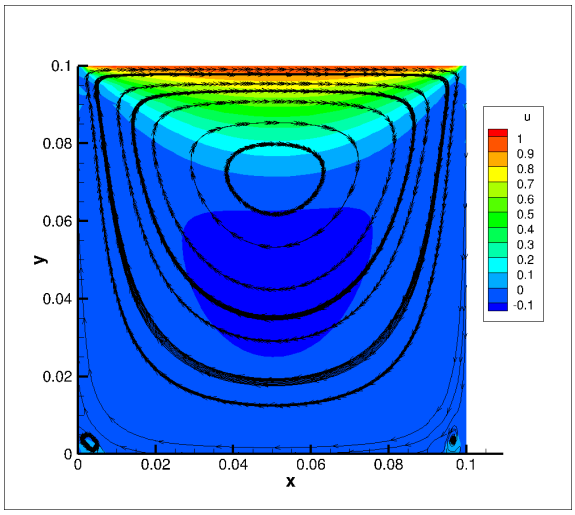


Figure 3.2: Lid-driven cavity flow: Steady-state field (Re=5).

Based on the above initial field, the solver is run using a backward Euler time integration scheme and $P = 4$ order DG scheme. The tolerance convergence of GMRES and Newton solvers were set to reduce the residual by a factor of 10^{-6}

respectively. The preconditioned matrices are extracted for analysis after only one time step $\Delta t = 0.001$.

To do spectral analysis, we compare two properties, the condition number , and the eigenvalue distributions. The condition number is defined as the ratio of the largest singular value to the smallest singular value (Kincaid et al., 2009). The condition number of unpreconditioned Jacobian matrix is 9.71×10^{13} and the eigenvalue distributions are shown as Fig. 3.3. The eigenvalues are widely distributed and the maximum magnitude of eigenvalue reaches over 3000. It is always the case that widely spread eigenvalues indicate a slow convergence (Wathen, 2015). Without preconditioning, it is hard to converge.

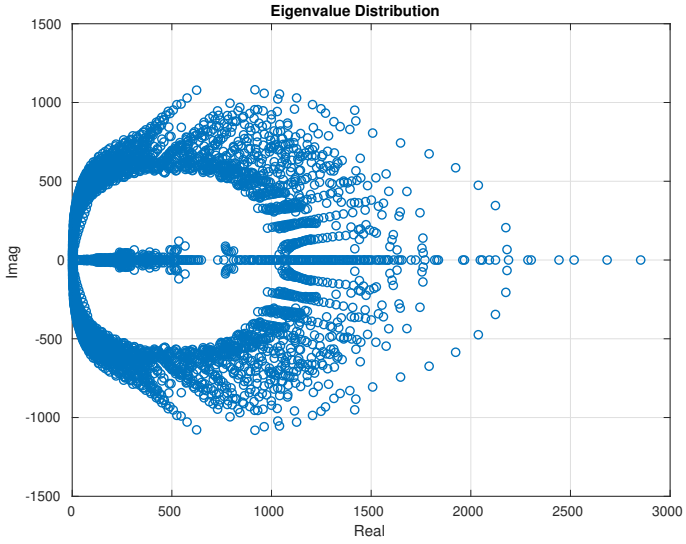


Figure 3.3: Eigenvalue distributions of the unpreconditioned Jacobian matrix.

In the first test, we compare the performance of different BJac preconditioners. The number in the bracket represents the number of smoothing iterations. The simulation using BJac(1) does not converge, the result of which is not shown here. Fig. 3.4 demonstrates that many eigenvalues of BJac(1) are distributed around the circle with its center at (1,0) and radius=1. The eigenvalues of BJac(3) and BJac(5) tend to cluster in a smaller circle and are distributed near the center. The condition number after BJac(1), BJac(3), and BJac(5) preconditioning are 3.17×10^{10}, 9.61×10^{9}, and 5.80×10^{9} respectively, which are much smaller than the case before

preconditioning. Lastly, the simulations with BJac(3) and BJac(5) take 187 and 181 GMRES iterations to converge. The changes of eigenvalue distributions and condition number are consistent with the improvement of convergence after precondition, which indicates in this case, the spectral analysis is a good tool to compare preconditioners' performance.

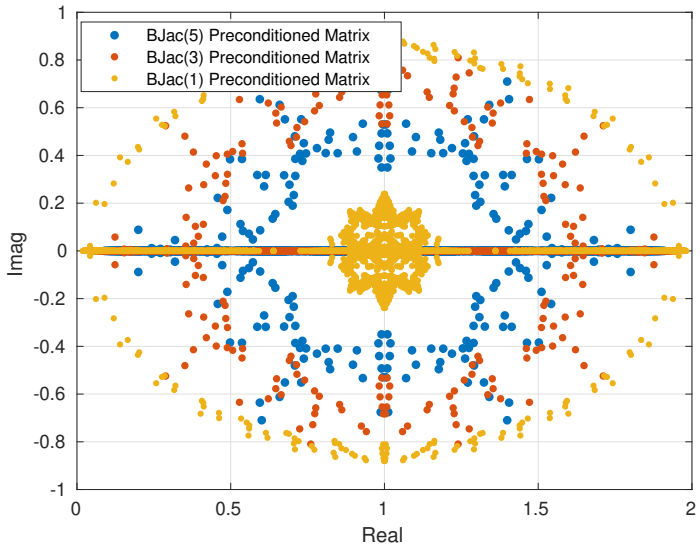


Figure 3.4: Eigenvalue distributions of the block Jacobi preconditioned matrix.

The second test compares the performance of different BGS preconditioners. The simulation using BGS(1) also does not converge. Different from the eigenvalue distributions of BJac preconditioners Fig. 3.5 shows the eigenvalues of BGS preconditioned matrices are non-symmetrically distributed in this test case. The condition number after BGS(1), BGS(3), and BGS(5) preconditioning are 3.07×10^{10}, 9.54×10^{9}, and 5.64×10^{9} respectively, which are smaller than their counterparts of BJac preconditioners. Lastly, the simulations with BGS(3) and BGS(5) take 174 and 127 GMRES iterations to converge, which are also consistent with the changes of spectral properties.

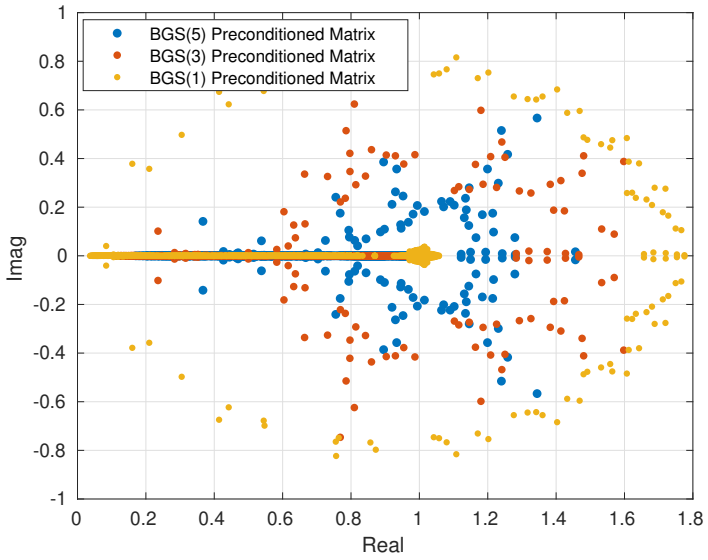


Figure 3.5: Eigenvalue distributions of the Block Gauss-Seidel preconditioned matrix.

The third test compares the performance of p-multigrid preconditioners of different levels. The smoothers for these preconditioners are BGS solvers. The simulations using p-multigrid of Level2(3), Level3(5), and Level4(7) take 174, 129, and 101 GMRES iterations to converge. The number in the bracket represents the total iteration number and 'Level' represents the levels for a multi-level p-multigrid preconditioner. For example, Level3(5) is a 3 levels p-multigrid preconditioner, where the highest level resolution at P=3, the second level is at P=2 and the lowest level is at P=1. The number 5 in Level3(5) means totally 5 smoothing iterations, consisting of 1 pre-smooth in the highest level, 1 pre-smooth in the second level, 1 smoothing in the lowest level, 1 post-smooth in the second level, 1 post-smooth in the highest level. Compared with the performance BGS(5) preconditioner using 127 GMRES iterations, Level3(5)'s performance is a little inferior to it with 129 iterations. The smoother performance at a low level still needs to be optimized to achieve the same performance as high-level smoother. But the low-level resolution saves computational cost. Lastly, the condition number after Level2, Level3 and Level4 p-multigrid preconditioning are tested 9.54×10^9, 5.77×10^9, and 4.16×10^9.

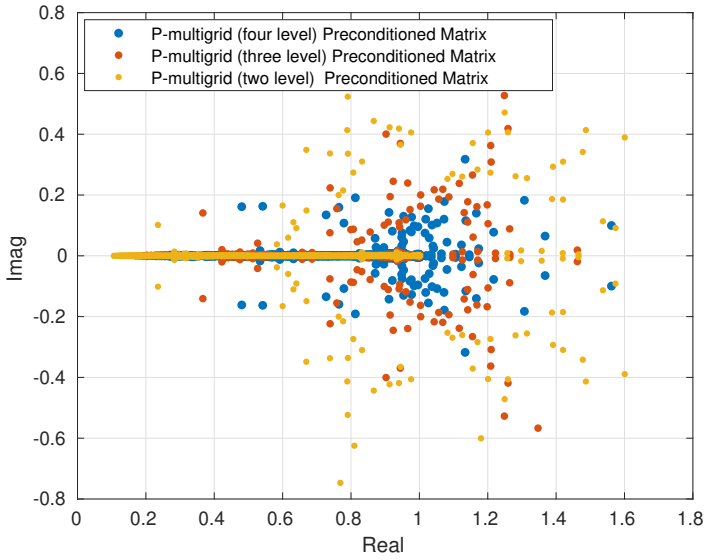


Figure 3.6: Eigenvalue distributions of the p-multigrid preconditioned matrix.

In the fourth test, the simulation using BILU takes 139 GMRES iterations to converge. The condition number after BILU preconditioning is 5.11×10^9. The eigenvalue distributions of the Jacobian matrix after BILU preconditioning are not compared separately. Fig. 3.7 compares the distributions of BILU preconditioner with other types of preconditioners to demonstrate some interesting details.

To summarize, for the same type of iterative preconditioners, when we apply more iterations, such as BJac(5) compared with BJac(3), the system converges faster. The clustering of eigenvalues is consistent with better convergence property as expected. However, for different types of preconditioners, they cluster in different directions and there is no obvious rule between different types of preconditioners. When the smoothing number is the same, BGS generally performs better than BJac. Because after each smoothing iteration, BGS can influence more elements and smooth errors in a wider computational domain. The BILU preconditioner does not perform well compared with BGS(3) in this case. Comparing Fig. 3.5 and Fig. 3.6, the eigenvalue distributions of p-multigrid are quite similar to that of directly using its smoother as the preconditioner. To be clear, as shown in Fig. 3.7, the p-multigrid (BGS as its smoother) preconditioner and the BGS preconditioner use

the same number of iterations, their eigenvalue distributions are quite similar. The condition numbers after preconditioning are also close. For example, the condition numbers after BGS(5) and Level3(5) p-multigrid preconditioning are both approximately 5×10^9. The spectral properties are very similar between the p-multigrid preconditioner and its smoother. It, therefore, hints an efficient smoother play an important role for p-multigrid preconditioner. Most importantly, this property indicates that even though p-multigrid preconditioner does not obviously improve the convergence property compared with its original smoothing method (BGS) in this test when the total smoothing number is the same, the computational cost can be potentially reduced since part of smoothing iterations of p-multigrid are at lower levels.

Extra spectral analysis based on the converged field of Lid-driven cavity flow at $\mathrm{Re} = 100$ is presented in Appendix 6.2. Similar conclusions can be verified, such as the spectral properties of the p-multigrid preconditioner are directly related to that of its smoother.

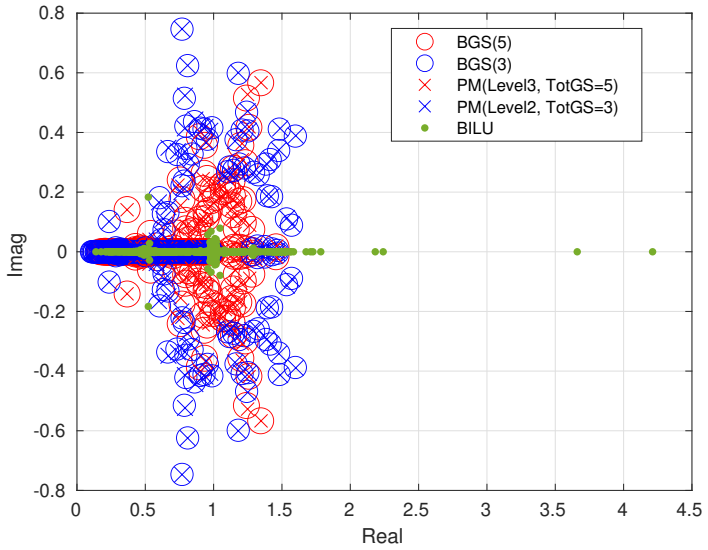


Figure 3.7: Comparisons between eigenvalue distributions of BGS, BILU and p-multigrid preconditioned matrices.

3.2 Efficient implementation of block relaxed Jacobi preconditioner

The computational cost and memory consumption to calculate and store the whole Jacobian matrix $\frac{\partial \mathbf{N}}{\partial \mathbf{v}}$ explicitly in large-scale problem is expensive. The calculation of a Jacobian matrix requires much more cost compared with calculating the residual $\mathbf{N}$. The frequent update of the preconditioner in an unsteady problem also does not allow the direct calculation of the whole Jacobian. In terms of storage consumption, the straightforward way to store a Jacobian matrix in 3D simulations takes up approximately $8N_nN_e(N_{var}(P+1)^3)^2$ in double precision (Yan et al., 2020). N_n is the number of non-zero blocks per row, N_e is the number of partitioned elements, N_{var} is the number of conservative variables, and P is the DG polynomial order. One Gbyte of memory can only support a 3D simulation using $P = 4$ DG polynomial and 45 hexahedral elements. Moreover, frequently loading this large matrix into the CPU cache will seriously influence the computational speed since most modern high-performance computers (HPCs) are memory bandwidth limited (Knoll and Keyes, 2004; Witherden et al., 2014).

3.2.1 Block relaxed Jacobi preconditioner

Through the spectral analysis in Section 3.1, we understand some spectral properties of different types of preconditioners. When we practically implement an efficient preconditioner, we optimize the block Jacobi (BJac) preconditioner through the consideration of balancing the storage requirement and computational cost.

Different from the BJac, the block relaxed Jacobi (BRJ) adds a relaxation factor ω to the iteration. This factor can adjust the convergence rate and stability (Datta,

2010). The process of Z step block relaxed Jacobi preconditioner BRJ(Z) is

$$\begin{gathered}\mathbf{y}_0 = 0 \\ \mathbf{y}^z = \omega \mathbf{D}^{-1}(\mathbf{c} - (\mathbf{L}+\mathbf{U})\mathbf{y}^{z-1}) + (1-\omega)\mathbf{y}^{z-1} \\ z = 1, 2 \ldots, Z\end{gathered} \tag{3.29}$$

which is equivalent to BJac if $\omega = 1$.

To minimize the memory consumption, only the matrix $\mathbf{D}^{-1}$ is stored while the product $(\mathbf{L}+\mathbf{U})\mathbf{y}^{z-1}$ is calculated on the fly. Since we do not store the off-diagonal blocks, it takes up much less storage than the straightforward approach of storing the whole Jacobian matrix.

The calculation of $\mathbf{D}^{-1}$ is very critical to an efficient preconditioner. The algorithm to construct $\mathbf{D}$ is very difficult because it includes all the components of the solver such as the discretized advection term, the discretized diffusion term, boundary conditions and Riemann numerical fluxes. Considering the various choices of Riemann fluxes and boundary conditions, it would be tedious to code the Jacobian matrix for each of these options. Since the derivation of diffusion term's Jacobian is much more complexed, the following section takes the derivation of Euler equation (without diffusion term) as an example

$$\frac{\partial \mathbf{N}(\mathbf{v}_{e_i})}{\partial \mathbf{v}_{e_j}} = \mathbf{I}\delta_{i,j} - \Delta t a_{k,k} \frac{\partial \mathcal{L}(\mathbf{v}_{e_i})}{\partial \mathbf{v}_{e_j}}, \tag{3.30}$$

where $a_{k,k}$ is the diagonal coefficient of DIRK scheme, $\frac{\partial \mathbf{N}(\mathbf{v}_{e_i})}{\partial \mathbf{v}_{e_j}}$ is the Jacobian of the i^{th} element' fluxes with respect to j^{th} elements' field variables, $\mathcal{L}(\mathbf{v}_{e_i})$ is the discretized operator of the advection term

$$\mathcal{L}(\mathbf{v}_{e_i}) = \mathbf{M}_{e_i}^{-1} \left(\int_{\Omega_{e_i}} \mathbf{F} \cdot \nabla \phi_p^{e_i} d\Omega - \int_{\Gamma_{e_i}} \tilde{\mathbf{F}}^n(\mathbf{V}^+, \mathbf{V}^-) \phi_p^{e_i} d\Gamma \right), \tag{3.31}$$

the variables' meanings of which can be found in Section 2.2, except here $\mathbf{V}$, $\mathbf{v}$ are the solution vector and the coefficient vector during Newton iterations.

The volume Jacobian $\frac{\partial \mathbf{F}(\mathbf{V}_{e_i})}{\partial \mathbf{v}_{e_j}}$ and the trace Jacobian $\frac{\partial \tilde{\mathbf{F}}^n_{e_i}(\mathbf{V}^+,\mathbf{V}^-)}{\partial \mathbf{v}_{e_j}}$ are the only variables depending on $\mathbf{V}$. The other operators such as integration, the mass matrix $\mathbf{M}_{e_i}$ are stored at the beginning and unchanged throughout the simulation. The volume Jacobian $\frac{\partial \mathbf{F}(\mathbf{V}_{e_i})}{\partial \mathbf{v}_{e_j}}$ are calculated analytically. The analytical expression of volume Jacobian matrix can be found in references such as (Masatsuka, 2013). The trace Jacobian matrix needs neighboring elements' information reads

$$\frac{\partial \tilde{\mathbf{F}}^n_{e_i}(\mathbf{V}^+,\mathbf{V}^-)}{\partial \mathbf{v}_{e_j}} = \left(\frac{\partial \tilde{\mathbf{F}}^n_{e_i}(\mathbf{V}^+,\mathbf{V}^-)}{\partial \mathbf{V}^+} \frac{\partial \mathbf{V}^+}{\partial \mathbf{V}_{e_j}} + \frac{\partial \tilde{\mathbf{F}}^n_{e_i}(\mathbf{V}^+,\mathbf{V}^-)}{\partial \mathbf{V}^-} \frac{\partial \mathbf{V}^-}{\partial \mathbf{V}_{e_j}} \right) \mathbf{B}, \tag{3.32}$$

where the backward transform matrix $\mathbf{B}$, the interpolation matrices $\frac{\partial \mathbf{V}^+}{\partial \mathbf{V}_{e_2}}$ and $\frac{\partial \mathbf{V}^-}{\partial \mathbf{V}_{e_2}}$ are unchanged maps.

It is non-trivial to calculate the analytical solution of the trace Jacobian in consideration of different types of Riemann solvers that have their consistent trace Jacobian. Some analytical expressions of trace Jacobian of Riemann fluxes such as Roe flux, HLLC flux can be found in (Rinaldi et al., 2014). Most of other types of trace Jacobian are complicated to derive and explicitly expressed. It is also mentioned that in most cases, the analytical and numerical Jacobian are identical in practical use (Vanden and Orkwis, 1996). Therefore to provide a general formulation for all these options, a finite difference approximation is adopted to calculate the trace Jacobian matrices numerically

$$\frac{\partial \tilde{\mathbf{F}}^n_{e_i}(\mathbf{V}^+,\mathbf{V}^-)}{\partial \mathbf{V}^\pm} = \frac{\tilde{\mathbf{F}}^n_{e_i}\left(\mathbf{V}^\pm + \chi \vec{\mathbf{e}}^l, \mathbf{V}^\mp\right) - \tilde{\mathbf{F}}^n_{e_i}\left(\mathbf{V}^\pm, \mathbf{V}^\mp\right)}{\chi}, \tag{3.33}$$

where $\vec{\mathbf{e}}^l$ is a unit vector of l^{th} entry equaling to 1. The parameter χ is evaluated in a fashion similar to ϵ used in Eq. 2.53. However, the scale is replaced by the L_2 norm of each variable, which makes χ depends on each variable. Physical boundary conditions need to be imposed on $\mathbf{V}^+$ specially in the $\tilde{\mathbf{F}}^n_{e_i}\left(\mathbf{V}^\pm + \chi \vec{\mathbf{e}}^l, \mathbf{V}^\mp\right)$ evaluation so that the Jacobian matrices of the boundary conditions are already included in Eq. (3.33). The entries of $\frac{\partial \tilde{\mathbf{F}}^n_{e_i}(\mathbf{V}^+,\mathbf{V}^-)}{\partial \mathbf{V}^\pm}$ are stored since they are also needed for the evaluation of the product $(\mathbf{L}+\mathbf{U})\,\hat{\mathbf{y}}^{z-1}$ in Eq. (3.29). The computational cost and

storage consumption of the trace Jacobian matrices $\frac{\partial \tilde{\mathbf{F}}^n_{e_i}(\mathbf{V}^+,\mathbf{V}^-)}{\partial \mathbf{V}^{\pm}}$ are however small compared with the other parts of the implicit solver.

Considering the costly consumption of generating the preconditioner, we do not update the preconditioner every time step, especially for the simulation of smooth evolution. The frequency to update the preconditioner in Nektar++ is every 10 time steps. In other words, the 'freezing number' is set as 10. In Section 4.4, two freezing strategies of the preconditioner will also be introduced and compared.

3.2.2 Efficiency comparison between explicit and implicit solvers: 2D flow over a circular cylinder

After the completion of the BRJ preconditioner, the major part of the developed implicit solver has been introduced. The efficiency of the implicit solver is compared to that of an explicit solver through the case of 2D flow over a circular cylinder. This case has been used as a verified test case in a wide range of Re (Bijl et al., 2002; Gautier et al., 2013). The efficiency is compared in two states Re=1200 and Re=40. The simulation run under Re=1200 is an unsteady problem while under Re=40, the flow is near a steady state.

Re=1200 In the first test, the flow is simulated under the Mach number M=0.3 and the Reynolds number Re=1200. The initial setup has been introduced in Section 2.8.3. After reaching a periodic unsteady state, the St is tested approximately 0.243.

The efficiency (total CPU) is compared within one period. In terms of the implicit solver, it utilizes a third-order DIRK (DIRK3) scheme and a second-order DG polynomial, with a time step $\Delta t = 5 \times 10^{-2}$ (CFL=475). BRJ(7) preconditioner with a freezing number of 10 is set. For the explicit solver, it utilizes a third-order Runge-Kutta (RK3) scheme and a second-order DG polynomial, with time step $\Delta t = 5 \times 10^{-5}$ (CFL=0.0475). The time step used for the explicit solver is the maximum allowed time step to guarantee stability. Fig. 3.8 shows the variation of density at the history point $(2r, 0)$ within one period.

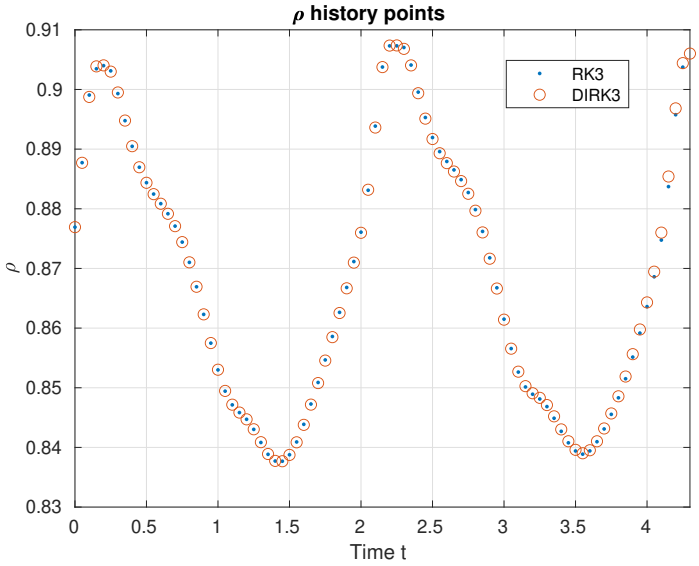


Figure 3.8: 2D flow over a circular cylinder (Re=1200): ρ variation within one period.

Table .3.1 compares the efficiency of the implicit solver and the explicit solver using (a) one core, (b) 8 cores, (c) 16 cores, (d) 24 cores. The speed-up of the developed implicit solver can reach over 30 in this test.

Cores	RK3 CPU Time (s)	DIRK3 CPU Time (s)	Speed-up
Serial	377815	11160	34
8	37315	919	41
16	22241	544	41
24	12703	377	44

Table 3.1: 2D flow over a circular cylinder (Re=1200): Efficiency comparison.

Re=40 In the second test, the flow is simulated under the Mach number M=0.3 and the Reynolds number Re=40. The solvers use the same schemes as the first test (a) implicit solver uses DIRK3 and $P = 2$ DG schemes, (b) explicit solver uses RK3 and $P = 2$ DG schemes. To avoid the limitations of using a very small time step when starting, the initial field is run for a while from a uniform field using low-order schemes, BDF1 scheme and $P = 1$ DG scheme. The initial ρ distribution is shown as Fig. 3.9. The implicit and explicit solvers are run using their respective

largest time step $\Delta t = 10^{-2}$ and $\Delta t = 10^{-7}$, equally CFL = 100 and CFL = 0.001. The history of residual is shown in Fig. 3.10. Compared with the explicit solver, it shows the implicit solver speeds up by approximately two orders of magnitude in this test. After the normalized residual $L_\delta(\mathbf{U}_\delta)$ of the implicit simulation reaches 10^{-7}, we extract the result to compare with the results in (Gautier et al., 2013). Fig. 3.11 shows the contour of ρ distribution. The separation angle θ_s and the location of recirculation centre (x_r, y_r) of the numerical simulation are consistent with the reference, which are shown in Table. 3.2.

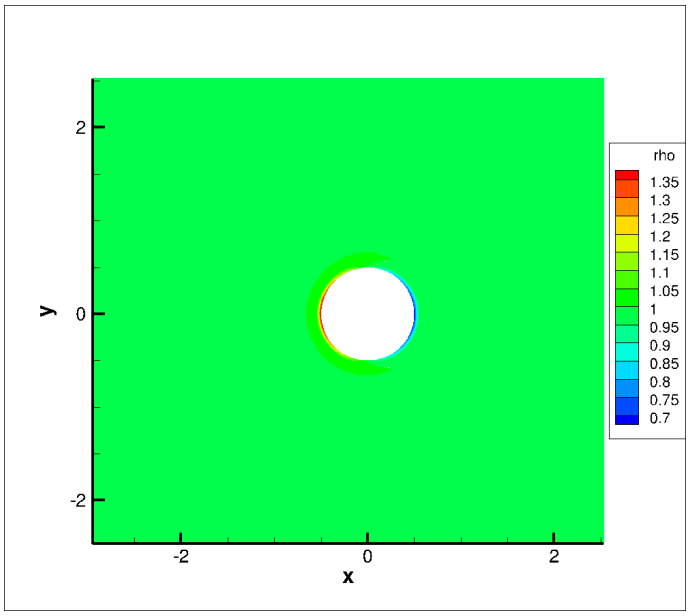


Figure 3.9: 2D flow past a circular cylinder (Re=40): Initial field.

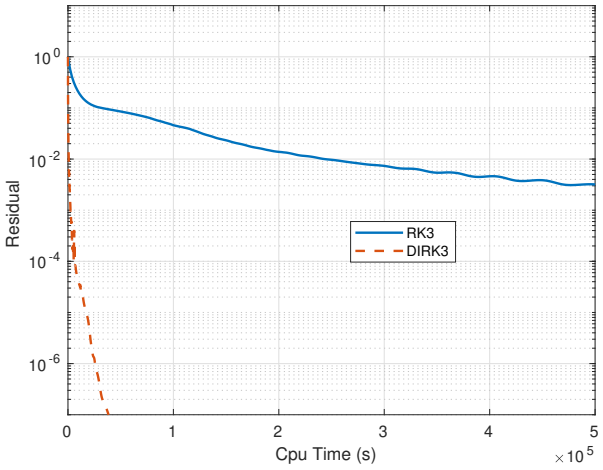


Figure 3.10: 2D flow past a circular cylinder (Re=40): Residual history of implicit and explicit simulations.

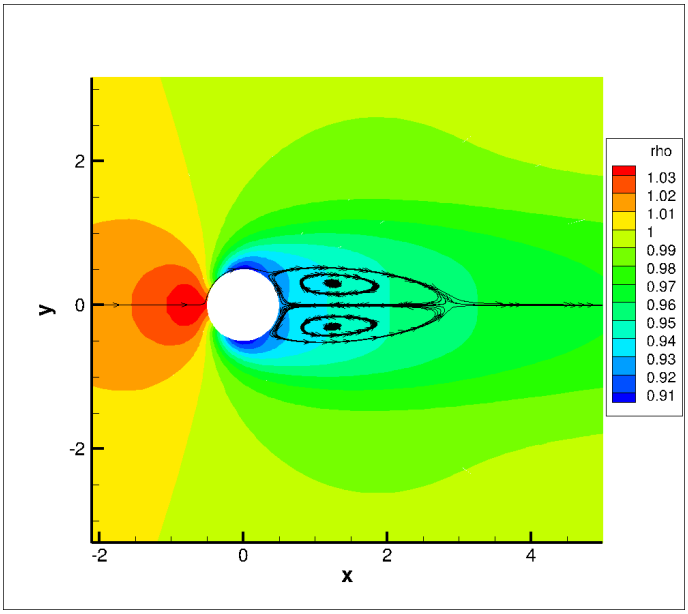


Figure 3.11: 2D flow past a circular cylinder (Re=40): ρ distributions.

	θ_s	x_r	y_r
Reference	126.4	0.71	0.59
Numerical result	127	0.7	0.6

Table 3.2: 2D flow past a circular cylinder (Re=40): Comparison of flow features.

Chapter 4

Choices of parameters for a reliable implicit solver

Unlike explicit solvers which are typically limited by the time step restriction, implicit solvers can utilize a much larger time step. Therefore, implicit solvers have been widely used in a great variety of steady and unsteady simulations as a more efficient choice (Bassi et al., 2016; Vandenhoeck and Lani, 2019; Noventa et al., 2020). However, implicit solvers are generally more complex and introduce more free parameters. Taking the solver we developed for example, the combined use of diagonally implicit Runge-Kutta (DIRK) methods and the Jacobian-free Newton-Krylov (JFNK) method introduces parameters such as the size of the time step, the convergence tolerance for Newton nonlinear solver, the convergence tolerance for Krylov linear solver, the order of finite difference approximation of matrix-vector products, etc (Yan et al., 2020). If the system is preconditioned, other parameters such as the number of smoothing iterations and the update frequency of the preconditioner are involved. These parameters are nonlinearly coupled with each other and dramatically influence the accuracy and efficiency of an implicit solver. It would not be surprising that a naive set of parameter choices makes the simulations several times slower than the optimal one.

This chapter aims to provide a set of strategies to avoid improper choices of

parameters based on the consideration of accuracy requirement and efficiency improvement. Firstly, in Section 4.1, an adaptive time-stepping strategy is developed based on the idea that the local temporal error should be smaller than the spatial error in fixed mesh simulations. This strategy can also help relax the difficulty to choose the order of time integration schemes. Secondly, based on a similar error analysis, an adaptive Newton tolerance is proposed in Section 4.2, considering the numerical error generated during Newton iteration should be smaller than the temporal error to ensure the temporal accuracy. The systematic introductions to the adaptive time-stepping strategy and the adaptive Newton tolerance have also been presented in (Pan et al., 2021). Section 4.3 studies the accuracy of approximated Jacobian's influence on the efficiency. In Section 4.4, an adaptive strategy to update the preconditioner based on the observation of GMRES convergence state is discussed. Combined with the strategy proposed in (Eisenstat and Walker, 1996) to link the convergence tolerance for Newton solver with the tolerance for Krylov linear solver, the implemented implicit solver can almost remove the headache to choose user-defined parameters.

4.1 Error-based adaptive time step

Recalling the error relationship Eq. (2.48) derived in Section 2.5, the total error generated within one time step is the sum of local temporal error $\mathbf{E}_t$, averaged spatial error $\bar{\mathbf{E}}_s$ and averaged iteration error $\bar{\mathbf{E}}_{it}$

$$\begin{aligned} \mathbf{E}_{\text{tot}} =& \mathbf{E}_t^{n+1} + \Delta t \bar{\mathbf{E}}_s + \Delta t \bar{\mathbf{E}}_{it} \\ =& \mathbf{B}\left(\mathbf{e}_t^{n+1} + \Delta t \bar{\mathbf{e}}_s + \Delta t \bar{\mathbf{e}}_{it}\right), \end{aligned} \tag{4.1}$$

where $\mathbf{e}_t$, $\bar{\mathbf{e}}_s$ and $\bar{\mathbf{e}}_{it}$ are corresponding coefficient vectors. The over-bar on $\bar{\mathbf{e}}_s$ and $\bar{\mathbf{e}}_{it}$ denotes implementing a weighted averaged operation over substages in the form of $\bar{\psi} = \sum_{i=1}^{s} b_i \psi^{(i)}$, where $\sum_{i=1}^{s} b_i = 1$.

This indicates that to decrease the total error, these three errors should be

reduced simultaneously. It will not help much if one of them is optimized to a small level but much smaller than the largest one.

Based on the error estimate, an adaptive time-stepping strategy is proposed that the time step should be as large as possible without obviously influencing the total discretization error. If the spatial error fixed in a specific simulation, this implies that

$$\|\mathbf{E}_t^{n+1}\| = \beta \Delta t \|\bar{\mathbf{E}}_s\|, \tag{4.2}$$

with $\beta < 1.0$ and a proper defined norm. When $\|\mathbf{E}_t^{n+1}\|$ is orders of magnitude smaller than $\Delta t\|\bar{\mathbf{E}}_s\|$, the total error $\mathbf{E}_{\text{tot}}$ will be dominated by the spatial error and further decreasing the temporal error through adjusting the time step will not improve the solution accuracy.

To ensure Eq. (4.2) is satisfied, the spatial error and temporal error need to be estimated. The embedded ESDIRK schemes offer a suitable way to estimate temporal error $\mathbf{e}_t^{n+1}$ using Eq. (2.29). In terms of the spatial truncation error $\bar{\mathbf{E}}_s$, it is estimated through a one order higher approximation at the end of each time step, reads

$$\begin{aligned} \bar{\mathbf{E}}_s &\approx L_\delta^{P+1}(\mathbf{U}_{it}^{n+1}) - L_\delta^{P}(\mathbf{U}_{it}^{n+1}) \\ &= C_s D_s(\mathbf{U}) h^{P+1} + O(h^{P+2}), \end{aligned} \tag{4.3}$$

the leading term of which is the same as that in Eq. (2.22) and P is the order of DG polynomial. Therefore, Eq.(4.2) implies that the desired time step ($\Delta\hat{t}$) should satisfy

$$\|\mathbf{E}_t^{n+1}(\Delta\hat{t})\| \approx \|C_t D_t(\mathbf{U}) \Delta\hat{t}^{N+1}\| = \beta \Delta\hat{t} \|\bar{\mathbf{E}}_s\|. \tag{4.4}$$

Dividing Eq. (4.4) by Eq. (2.29), the expected time step $\Delta\hat{t}$ using the N^{th} order time integration scheme can be calculated by

$$\Delta\hat{t} = \Delta t^n \left(\frac{\beta \Delta t \|\bar{\mathbf{E}}_s\|}{\|\mathbf{E}_t\|} \right)^{\frac{1}{N}}, \tag{4.5}$$

which can be used as the time step for the next time step. The simplest elementary

controller (Sóderlind and Wang, 2006) is adopted to determine the time step and other choices are also possible (Sóderlind and Wang, 2006).

In the designs of adaptive time-stepping in (Blom et al., 2016; Noventa et al., 2020), a L_2 norm of the $\mathbf{E}_t^{n+1}$ and $\bar{\mathbf{E}}_s$ for all the variables throughout the flow field is adopted as the norm in their adaptive strategy. However, the norm of the error with this choice may be dominated by a specific variable or the large-scale unsteady structures. As a result, the adapted time step may be determined by one of the variables and/or the dominating unsteady structure in the flow field, which could be too large for the small unsteady structures of interest.

To avoid these problems, the norm is defined in each element. The L_2 norm of $\mathbf{E}_t^{n+1}$ and $\bar{\mathbf{E}}_s$ are calculated in element e for variable m and the elemental time step is calculated using

$$\Delta t_{e,m}^{n+1} = \Delta t^n \left(\frac{\beta \Delta t \|\bar{\mathbf{E}}_s\|_{e,m} + 1.5^N \epsilon_m}{\|\mathbf{E}_t\|_{e,m} + \epsilon_m} \right)^{1/N}. \tag{4.6}$$

Here, the scaling factor β controls the relation between the temporal error and the spatial error. The perturbation parameter ϵ_m is added to the denominator to avoid division by zero. But it also serves as a threshold of the error magnitude of interest. If the local temporal error magnitudes are much smaller than ϵ_m, the local unsteady structures are either too weak to be of interest or are already accurately captured, which indicates the current time step choice is already excessively accurate in that element. Thus, a term $1.5^N \epsilon_m$ is also added to the numerator to ensure the elemental time step will at least grow by 1.5 times. The choice of 1.5, which is not optimized, ensures that the time step will increase but not too fast.

The value of ϵ_m adopted here is,

$$\epsilon_m = 10^{-12} \mathbf{U}_{it,m}^{\text{rms}}, \tag{4.7}$$

where $\mathbf{U}_{it,m}^{\text{rms}}$ denotes the root mean square value over the whole flow field of the mass, momentum magnitude and energy variables. Such a threshold value is based

on the rounding error of the m^{th} variable. One purpose of introducing this term is to avoid noisy elemental time steps caused by rounding errors in uniform flow areas.

With elemental time steps $\Delta t_{e,m}^{n+1}$, the time step for variable m can be calculated using different strategies. For example, a minimum of $\Delta t_{e,m}^{n+1}$ would maintain that Eq. (4.2) is satisfied in every element. Here we choose to calculate it using a weighted average of $\Delta t_{e,m}^{n+1}$ as

$$\Delta t_m^{n+1} = \frac{\sum_{e=1}^{N_e} \|\mathbf{E}_t\|_{e,m}^r \Delta t_{e,m}^{n+1}}{\sum_{e=1}^{N_e} \|\mathbf{E}_t\|_{e,m}^r}. \tag{4.8}$$

The temporal error $\|\mathbf{E}_t\|_{e,m}^r$ is used as the weights such that elements with larger temporal errors have larger contributions to the global time step. The value of $r \in [0, \infty)$ can be chosen based on the smallest temporal structure of interest in the simulations, a larger value of which would lead to a larger contribution from elements with large temporal errors. In this work, $r = 1$ is adopted. This weighted averaging helps maintain a relatively smooth change of the time step. Finally, the time step Δt^{n+1} is calculated by

$$\Delta t^{n+1} = \min(\Delta t_m^{n+1}). \tag{4.9}$$

4.1.1 Discussion of the new adaptive time-stepping strategy

The basic idea behind the time step adaptation strategy proposed in Section 4.1 is the relation between temporal and spatial errors that is sufficient for maintaining temporal accuracy in unsteady simulations. Here, we give a brief discussion to illustrate that it is a rational choice because it shares a very similar logic behind the convergence tests in CFD, which is also a key point in guaranteeing the high efficiency in implicit time integration methods.

The implicit time integration methods are broadly accepted to be much more efficient than explicit ones in simulations of stiff problems such as problems with low Mach numbers, high Reynolds number, and highly-stretched grids (Bassi et al., 2016; Vandenhoeck and Lani, 2019). The assumption behind this conclusion is that

we can safely increase the time steps and temporal errors when using implicit time integration methods without obviously degrading the simulation results. In another word, the comparison of implicit and explicit time integration in efficiency is not based on the computation time of the same temporal error level as is done in most studies of time integrations methods (Bijl et al., 2002; Holst et al., 2020) but is based on the computation time as long as the results are sufficiently accurate [1]. This implicitly defined sufficiently accurate condition is the maximum error level without obviously degrading the target quantity of interest, which is the one we are seeking when performing convergence tests in CFD simulations. However, this condition usually depends on the unsteady properties of the problem (the magnitude and frequency of the unsteady waves), the scale of interest, the acceptable error level, etc, and is not known before an accurate simulation result is obtained.

Similarly, the error condition behind the time adaptation strategy in this study, is to achieve the maximum value of Δt without obviously degrading the results of the discrete PDE (partial differential equation) system. Although, convergence tests are still needed to ensure physical accuracy, the strategy avoids improper error relations, which may seriously lower the efficiency without obviously improving the results. In a case presented in (Noventa et al., 2016), the improper error relations can increase the CPU time by about 20 times without obviously improving the results.

In some adaptive time-stepping strategies, the adapted time step is limited to be no larger than a user-defined time step Δt_{limit}, which is based on their understanding of the physical time scale to be resolved as in (Noventa et al., 2020). However, Eqs. (4.1) and (4.2) indicate that limiting the time step will not improve the results but possibly will increase the computational costs. Instead, refining the spatial grid will be much more efficient in improving the results. Therefore, it is advised to refine the mesh instead of limiting the time step in our time adapting strategy when the adaptive time step is obvious too large to capture the unsteady flow. As a

[1] These studies are still of great value because as long as the time step can be freely chosen for the two methods in comparison, the results can always get the same (desired) error level. However, there is a large difference between these two conditions when comparing explicit and implicit methods, since the time step of explicit methods is restricted by the stability condition.

by-product of this observation, if the temporal scale of interest is well defined and the properties of the ODE solver (ESDIRK in this paper) is well understood, the adapted time step can also serve as a good indicator of mesh refinement.

In the adaptation strategy, extra computational costs are required to estimate spatial and temporal errors. The embedded schemes (Kvrn, 2004) require an extra implicit RK stage, which will approximately increase the computational costs by $1/(s-1)$, where s is the number of stages of the main ESDIRK scheme, which has values of 3, 4, and 6 for the second, third and fourth order ESDIRK schemes adopted, respectively. Similarly, the spatial error estimation in Eq. (4.3) requires the calculation of a higher-order spatial discretization operator $L_\delta^{P+1}(\mathbf{U}_{it}^{n+1})$, which is similar in costs as $L_\delta^P(\mathbf{U}_{it}^{n+1})$ since the same number of quadrature points $3(P+1)/2$ is adopted and most of the computations in the DG methods are related to the number of quadrature points. Moreover, these extra costs could be further reduced by exploring the hierarchical structure of the base functions. The extra costs from spatial error estimation are lower than those of temporal error estimation since up to ten or more L_δ operations are needed in one implicit RK stage. In summary, we can roughly estimate the extra costs to be $100/(s-1)\%$. For simulations with little or slow time scale changes, a freezing strategy of the time step can also be adopted to lower the additional costs.

Compared with adaptive time-stepping strategies reported in (Birken et al., 2013; Noventa et al., 2016; Blom et al., 2016; Noventa et al., 2020), the current strategy is improved in the following ways:

- The adaptive time step is determined by the relation between the temporal and spatial error instead of a user-defined temporal error level. Therefore, highly problem-dependent user-defined tolerances are avoided.

- An embedded ESDIRK scheme with an order of $(N+1)$ is employed instead of a $(N-1)^{th}$ order embedded scheme, which predicts the temporal error more accurately. This also helps avoid problem-dependent calibration parameters of the estimated errors.

- The adaptation is determined element by element first instead of using the L_2 norm of the temporal error of the whole flow field. Therefore, it has the potential to avoid time steps being determined primarily by the large-scale unsteady structures and avoid small-scale unsteady structures being under-resolved.

4.1.2 2D isentropic vortex convection

This test case is adopted because its analytical solution is available, which makes it easier for spatial and temporal error estimations. In the computational domain $[0, 10] \times [-5, 5]$ with periodic boundary conditions in both directions, an inviscid isentropic vortex is convected down stream. The analytical solutions have been introduced in Section 2.8.1. The cartesian meshes and the density distribution have also been illustrated in previous Fig. 2.3.

To estimate different errors, different solutions are adopted as the reference in the error estimation. The total discretization error of a solution is obtained by subtracting it from the analytical solution of Eq. (2.57). As mentioned in (Bijl et al., 2002), the difference between the specific solution and a solution with a very small time step is regarded as the sum of temporal and iterative errors as the two solutions share the same spatial discretization. The difference between the solution with a very small time step and the analytical solution is regarded as the spatial error, since the error is dominated by the spatial error with a very small time step.

This Section aims at studying spatial, temporal, and total discretization errors characterized in terms of parameters such as the CFL number and the mesh size, h. The adaptive time-stepping strategy is evaluated at a polynomial order $P = 6$. The total discretization errors and temporal errors are calculated with respect to the analytical solution and a reference solution computed at a CFL number of 0.01.

The total errors of the third-order ESDIRK (ESDIRK3) with the adaptive time-stepping, CFL=10.0, CFL=1.0, and CFL=0.2 using different mesh sizes, $h = 0.2$, $h = 0.1$, $h = 0.067$ and $h = 0.05$, are presented in Fig. 4.1. The corresponding

temporal errors are shown in Fig. 4.2. The spatial errors are also presented for comparison. The temporal errors with the adaptive time steps are always smaller than the spatial errors, as per design, and the total error converges at the same rate as the spatial error. However, for simulations with CFL=10.0, the total errors are dominated by the temporal errors and converge at a lower order. For simulations with CFL=0.2, the total errors are dominated by spatial errors. However, the temporal errors are much smaller than the spatial errors, which may lead to extra costs without obviously improving the results. For simulations with CFL=1.0, temporal error is smaller than the spatial error for $h = 0.2$ but larger than that for $h = 0.05$, which emphasizes the need for calibration for each time step when using a time step based on a CFL number. Referring to the adaptive time-stepping methods in (Noventa et al., 2020; Blom et al., 2016), the temporal error curve should be horizontal lines because the temporal error tolerance in these methods is user defined and independent of h.

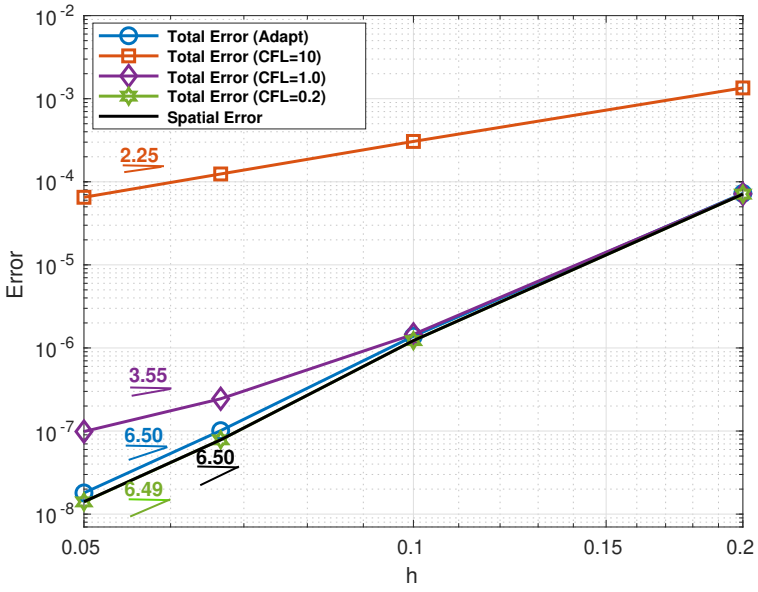


Figure 4.1: Isentropic vortex problem:
Effect of time-stepping method on total errors.

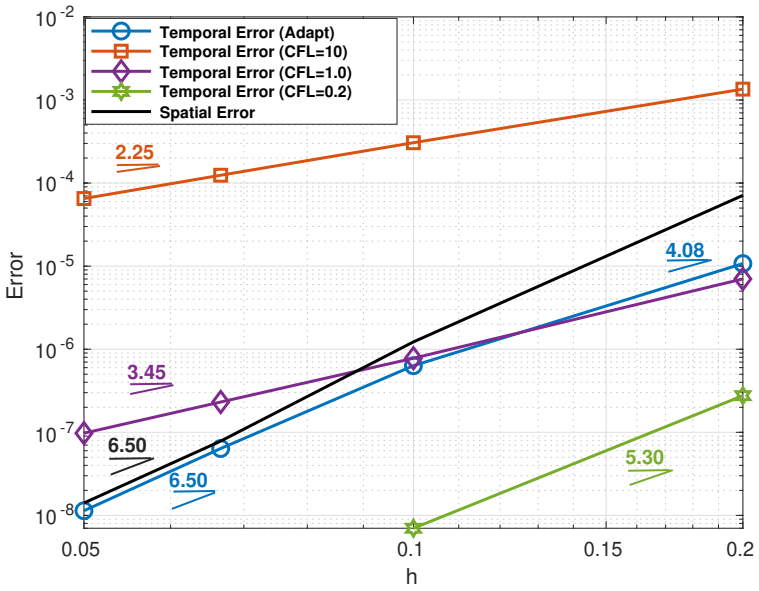


Figure 4.2: Isentropic vortex problem:
Effect of time-stepping method on temporal errors and spatial errors.

The CPU time of the above simulations is compared in Fig. 4.3. The simulation with the adaptive strategy is close to the most efficient ones at all the error levels. It is slightly more expensive than the most efficient one at some error levels partly due to the extra costs in estimating the errors. Fig. 4.4 and Fig. 4.5 also show the corresponding curves of residual evaluation number and GMRES iteration number. These numbers are the summation of all the time steps during the time interval. The residual evaluation number and GMRES iteration number of the adaptive method are also close to the smallest among different time-stepping methods. For the parameters adopted in this test, the efficiency of the adaptive stepping can be up to ten times more efficient at the same error level than the simulations with a naive choice of CFL, which highlights the importance of balancing the spatial and temporal error from the point view of efficiency.

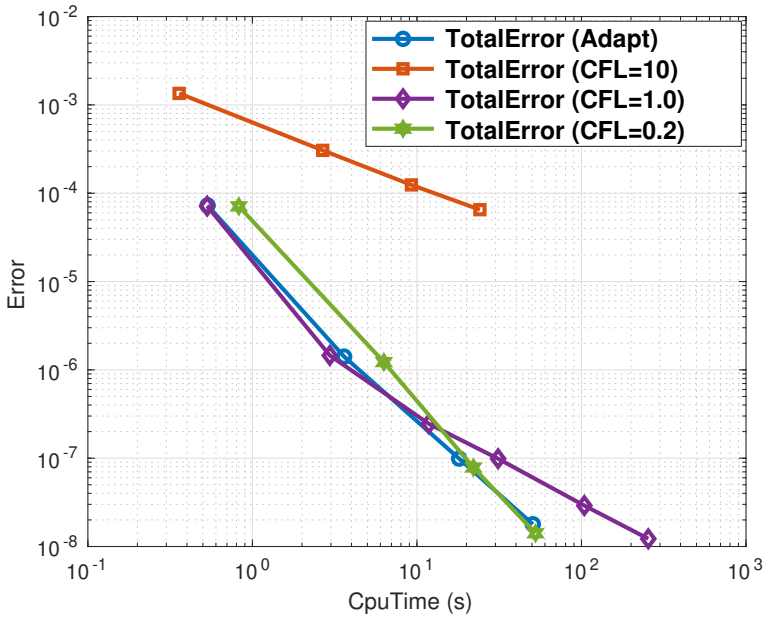


Figure 4.3: Isentropic vortex problem: CPU time.

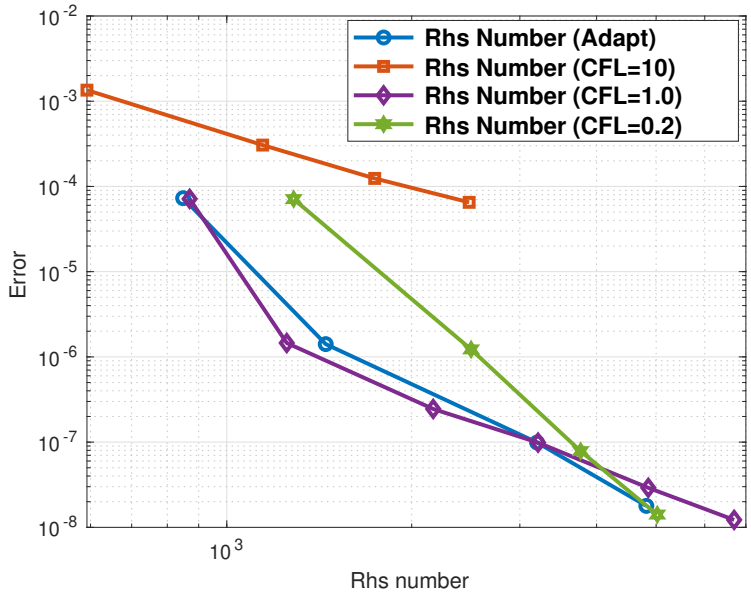


Figure 4.4: Isentropic vortex problem: Residual evaluation numbers.

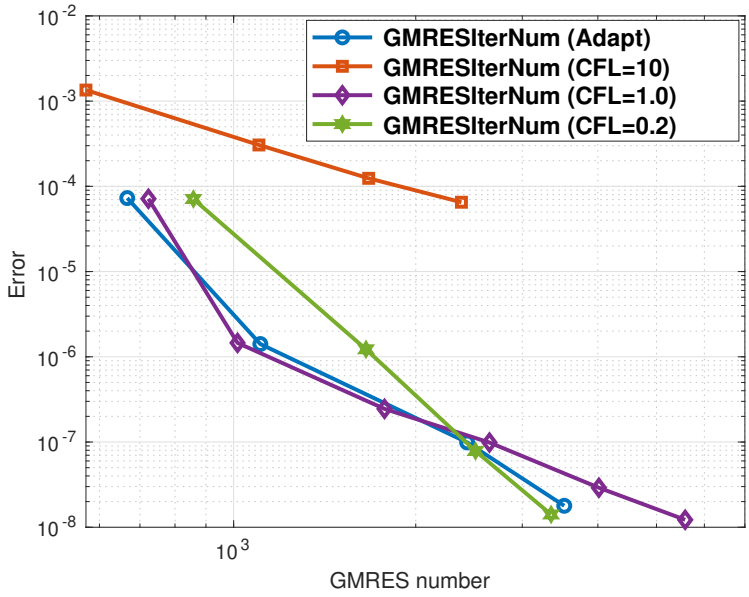


Figure 4.5: Isentropic vortex problem: GMRES iteration numbers.

The errors of the second-order, third-order, and fourth-order ESDIRK schemes are compared in Fig. 4.6. All temporal errors are smaller than the spatial errors and the total errors almost coincide with each other for different values of N. Therefore, the simulation results are almost independent of the ESDIRK scheme adopted in the adaptive time-stepping. This property can also be achieved by the adaptive methods proposed in (Blom et al., 2016; Noventa et al., 2020), but it requires an additional calibrating process which is problem dependent. The temporal errors are just a little smaller than the spatial errors in the fine mesh cases ($h = [0.05 \sim 0.1]$) while the spatial errors are larger than the temporal errors in the coarse mesh cases ($h = 0.2$). The phenomenon shown in coarse mesh cases indicates the adaptive strategy cannot help improve accuracy further mesh refinement is in need in such situations. This phenomenon will also be explained in later 3D cylinder flow simulation (4.1.5) that the post-processing of the adaptive strategy can be an indicator of regions in need of mesh refinement.

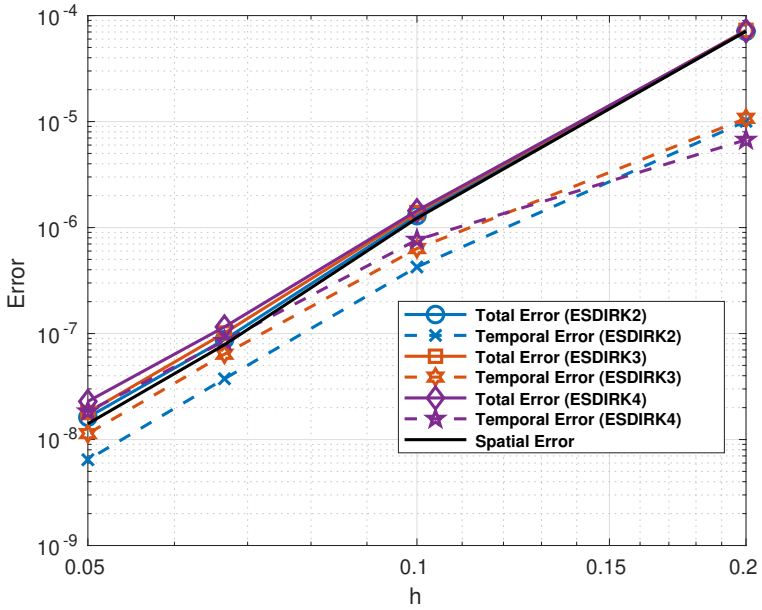


Figure 4.6: Isentropic vortex problem:
Performance of adaptive time-stepping for different ESDIRK schemes.

4.1.3 2D steady-state flat plate boundary layer flow

The steady-state flat plate boundary layer flow is a validation case involving flow past a flat plate. The Mach number of the flow is M = 0.1 and the Reynolds number is Re = 1.6×10^6. The low Mach number assumes the flow is incompressible and the high Reynolds number motivates the flow to gradually grow into a thin laminar boundary layer along with the plate.

The objective of this example is to illustrate how the adaptive time-stepping strategy could be used to accelerate convergence towards steady-state in a practical case. The geometry and boundary conditions are depicted in Fig. 4.7. The details of boundary conditions can be found in (Mengaldo, 2015; Yan et al., 2020). The simulations are performed using a second-order DG polynomial and a third-order ESDIRK (ESDIRK3) scheme with the highly-stretched meshes shown in Fig. 4.8(a). The computed profile of the horizontal velocity is shown and compared with the analytical Blasius profile in Fig. 4.8(b). In all the simulations below, the time step is set to be no larger than that corresponding to a CFL number of 5×10^4 to maintain

the stability of the simulations.

For the steady-state problem, we adopt the same adaptive time-stepping strategy and parameters used for the unsteady simulations to show that the proposed adaptive strategy can adjust itself based on the properties of the problem. For comparison, a time-stepping strategy based on a growing CFL is also presented, which starts from CFL=0.1 and grows by a rate of 1.5 and 2.0 up to CFL=5×10^4. A relative Newton tolerance of $\theta_2 = 10^{-6}$ in Eq. (2.40) is adopted since the default choice of $\theta_2 = 10^{-3}$ cannot guarantee the stable simulation at such a large CFL number. [2] The growing CFL number method is a commonly-adopted strategy for steady-state simulations because a small time step is needed to maintain stability in the initial transient phase, while a large time step is needed to accelerate the convergence when the flow field is close to the steady-state solution (Bucker et al., 2009; Vanderstraeten, 2001).

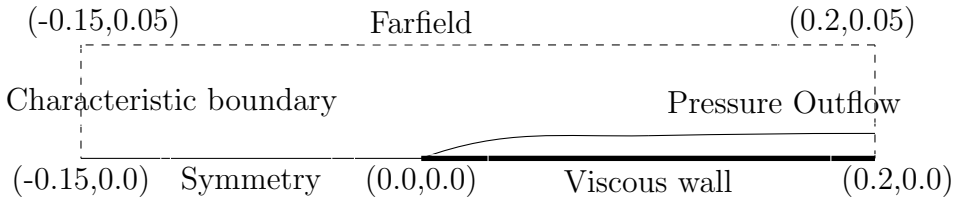
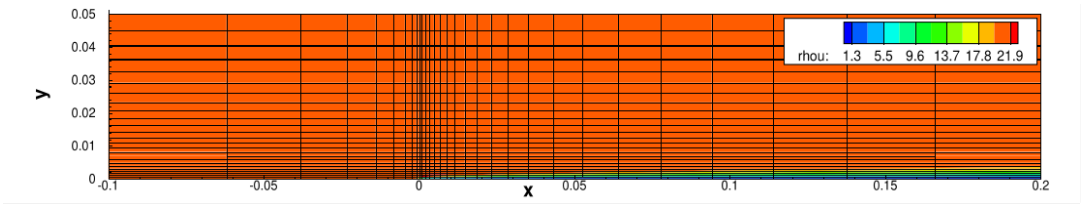
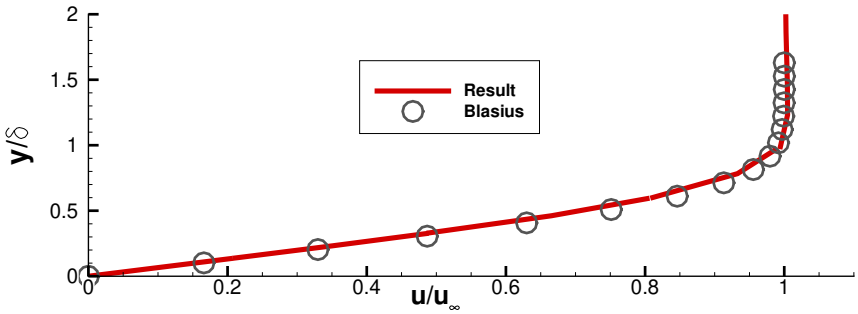


Figure 4.7: Boundary-layer flow past a flat plate: Geometry and boundary conditions.

[2] Instability problem occurred using a CFL number larger than 5×10^4 in this case. This problem may be caused by the nonlinear instability of ESDRIK (huu Cong, 1993; Kvrn, 2004) or by the approximations in constructing the preconditioner. The headache to choose proper parameters such as maximum CFL and Newton tolerance in this case also highlights the advantage of the adaptive strategy.



(a) Mesh and x-momentum.



(b) Velocity profile, where δ is the boundary layer thickness,

Figure 4.8: Boundary-layer flow past a flat plate: mesh and velocity profile.

The convergence history of the normalized residual norm $\|L_\delta(\mathbf{U}_\delta)\|$ is presented in Fig. 4.9. The simulation with the adaptive time-stepping proposed in Section 4.1 converges almost as fast as the simulations with a CFL growth rate of 2.0. The residual, Δt and CPU time at different steps are illustrated in Figs. 4.10, 4.11 and 4.12 respectively. It shows that the Δt of the adaptive method grows much slower at the first 100 time steps. However, the extra CPU cost because of the extra steps is low since it is cheap to converge at a small Δt. Fig. 4.11 also demonstrates that the adaptive Δt is small at the beginning of the simulation. This is because the transient flow is highly unsteady leading to a large temporal derivative term $D_t(\mathbf{U})$ in Eq. (2.29) and consequently to a smaller Δt. As the flow field becomes closer to the steady state, the Δt increases because the magnitude of the unsteady waves in the flow field becomes smaller, which leads to a smaller term $D_t(\mathbf{U})$ and larger Δt based on the relation of Eq. (4.2). The adaptive time-stepping adjusts the Δt based on the unsteady properties of the flow field and achieves fast convergence whilst maintaining stability.

Although the adaptive time-stepping may not be as efficient as methods specially

designed for steady-state simulations, such as local time-stepping and multi-grid, it is general and, with a fixed set of parameters, is capable of efficiently simulating a wide range of steady and unsteady flows.

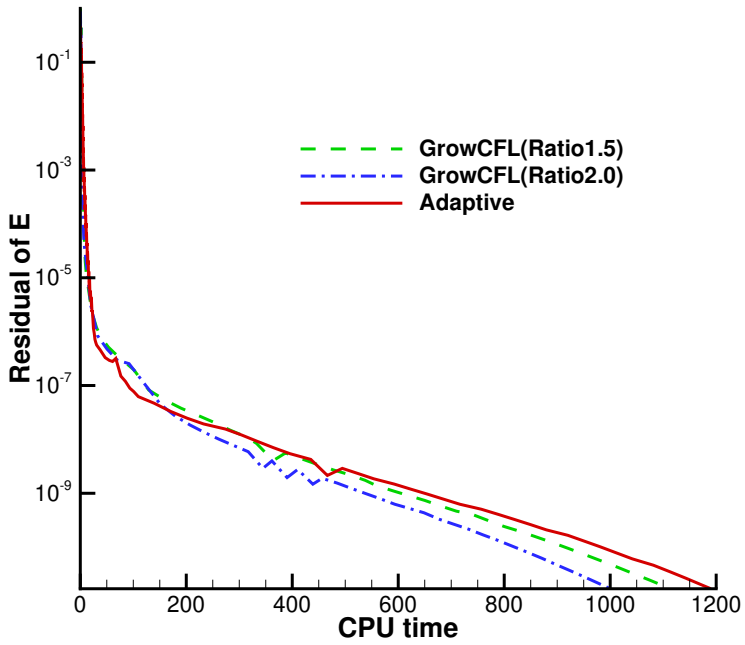


Figure 4.9: Boundary-layer flow past a flat plate: Residual convergence history. The number in the parenthesis indicates the growth rate of the CFL number.

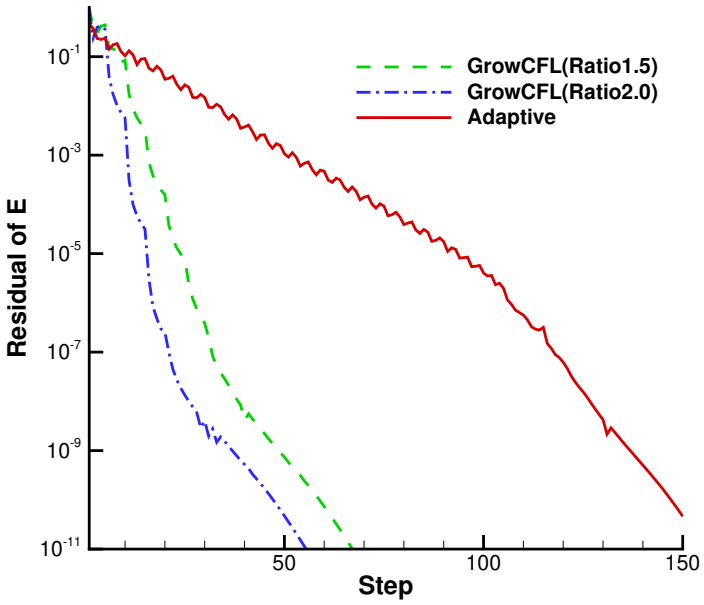


Figure 4.10: Boundary-layer flow past a flat plate: Residual with respect to the number of steps.

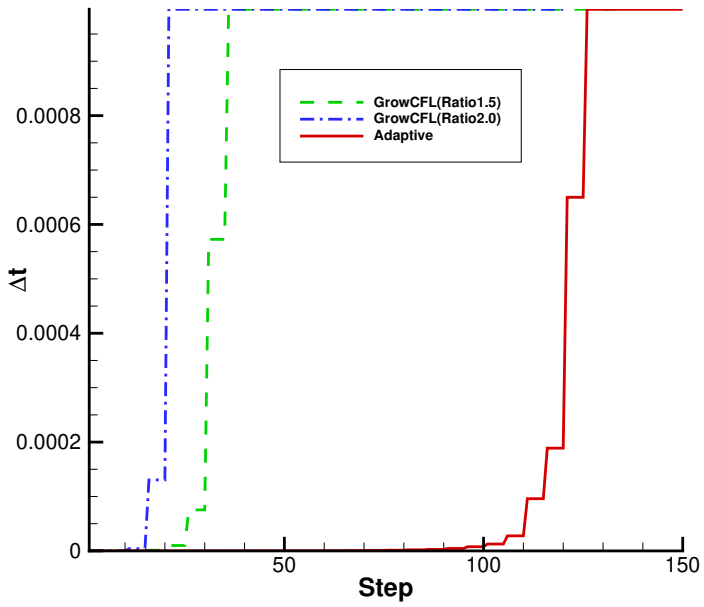


Figure 4.11: Boundary-layer flow past a flat plate: Time step size with respect to the number of steps.

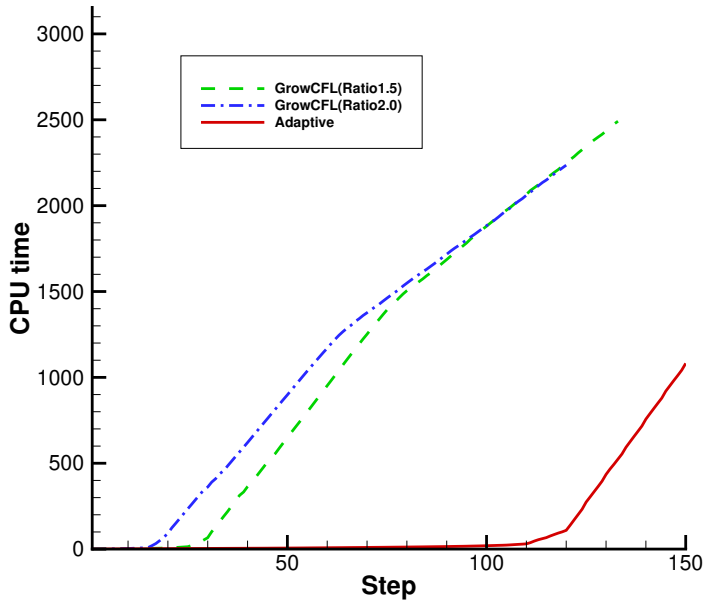


Figure 4.12: Boundary-layer flow past a flat plate: CPU time with respect to the number of steps.

4.1.4 Taylor-Green vortex

We next apply the adaptive time-stepping strategy to the simulation of the Taylor-Green vortex, which represents an unsteady flow of decaying vortices. The initial large vortices break down into smaller-scale vortices and eventually transition into turbulent flow. The simulations are run under the Mach number M = 0.1, the Reynolds number Re = 1600, and with periodic boundary conditions on a 32^3 uniformly distributed mesh using a fourth-order DG polynomial and a fourth-order ESDIRK (ESDIRK4). Note that, to simulate nearly incompressible conditions, the flow is taken to be compressible but at a low Mach number of M = 0.1.

Fig. 4.13 presents the evolution of $2\mu\varepsilon/\rho$, where ε denotes the enstrophy. According to (De Wiart et al., 2014), this variable is a good estimate of the kinetic energy dissipation rate in the incompressible limit. Fig. 4.14 presents the errors of the dissipation rate as defined by Eq. (18) of (De Wiart et al., 2014). The simulations are run with the set of values $\beta = 0.1$ and $\beta = 0.01$ for the adaptive time-stepping

strategy and they are also run with $\Delta t = 0.1$ and $\Delta t = 0.2$ for comparison. The decay curves of $2\mu\varepsilon/\rho$ are in relatively good agreement with each other for all the simulations. Their differences can be analyzed from the distributions of the errors in Fig. 4.14. The error with adaptive time-stepping and $\beta = 0.1$ is sightly larger than other errors. The error with $\beta = 0.01$ almost coincides with the error of $\Delta t = 0.1$. Both simulations with $\beta = 0.1$ and $\beta = 0.01$ are accurate in temporal accuracy especially compared with the spatial error, which can be roughly estimated by the deviation from the DNS result (De Wiart et al., 2014).

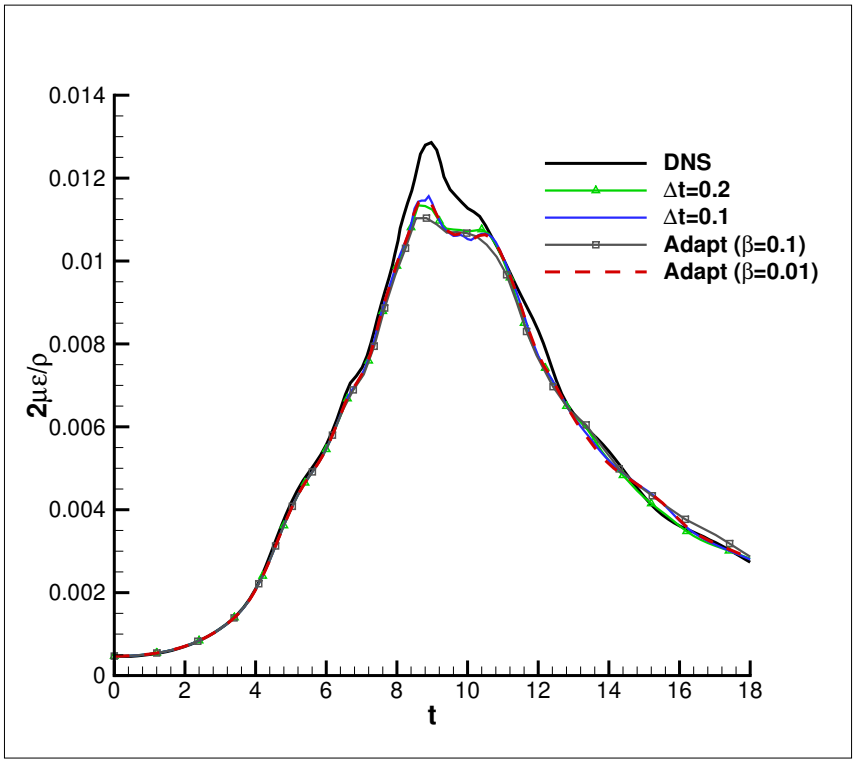


Figure 4.13: Taylor-Green vortex: Evolution of the enstrophy.

We enforce the local temporal error to be smaller than the spatial error by design, but there is no reason to introduce a bias towards the spatial error if the main consideration is simply to minimize the error magnitude. This consideration may be of relevance for under-resolved turbulent simulations where the spatial truncation error terms sometimes serve as implicit modeling of the under-resolved scales, which is referred to as the implicit large eddy simulation model (Grinstein et al., 2007).

To avoid the temporal error from influencing the modeling of the under-resolved scales, a smaller temporal error compared with the spatial error, $\beta = 0.01$, is adopted in implicit LES simulations of turbulent flows in our study. This approach gives reasonable results as shown in Fig. 4.14 and in Section 4.1.5. The computational CPU time of the simulations with $\beta = 0.01$ is slightly larger than that of simulation with $\Delta t = 0.1$ (1.27 times) partly because of the additional costs required for the error estimation.

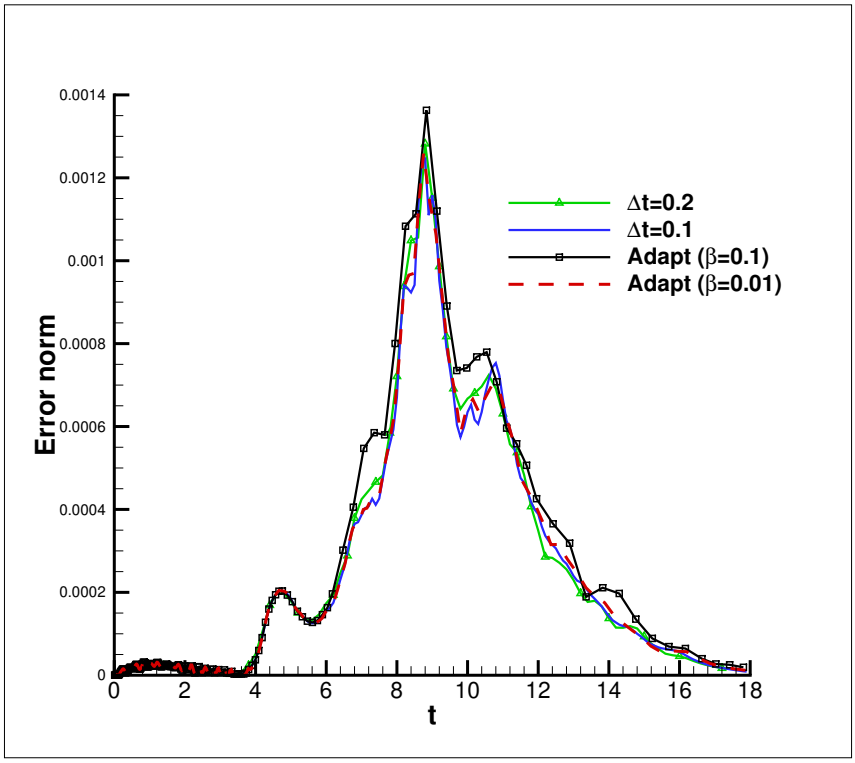


Figure 4.14: Taylor-Green vortex:
Error norms of the kinetic energy dissipation rate.

4.1.5 Turbulent flow over a circular cylinder at $\mathrm{Re} = 3\ 900$

The widely studied test case of the turbulent flow over a circular cylinder at $\mathrm{Re} = 3\ 900$ is adopted here to further illustrate the performance of the adaptive strategy in wall-bounded turbulence simulations. Details of the settings of the test case can be found in (Yan et al., 2020), which employ 123 360 curved unstructured hexahedral elements and polynomial order of $P = 2$. The unstructured mesh distributions

and the vortex structures behind the cylinder are illustrated in Fig. 4.15. Yan et al. (2020) adopted a second-order DIRK scheme (DIRK2) with a time step of 0.01, which corresponds to a CFL number of about 40. The time step was determined based on some numerical tests on its efficiency and accuracy in (Yan et al., 2020). Here we simulate the same case using the ESDIRK2, ESDIRK3, and ESDIRK4, together with the adaptive time-stepping strategy, to demonstrate their performance.

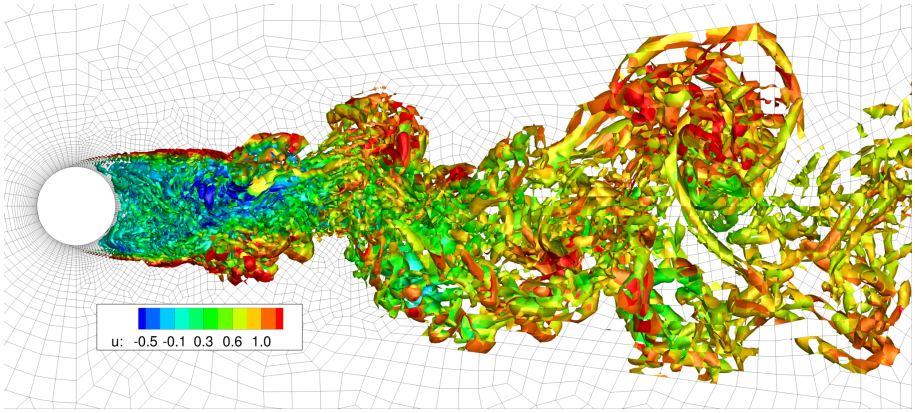


Figure 4.15: Turbulent flow over a circular cylinder:
Q criteria iso-surface ($Q = 5$) and mesh from (Yan et al., 2020).

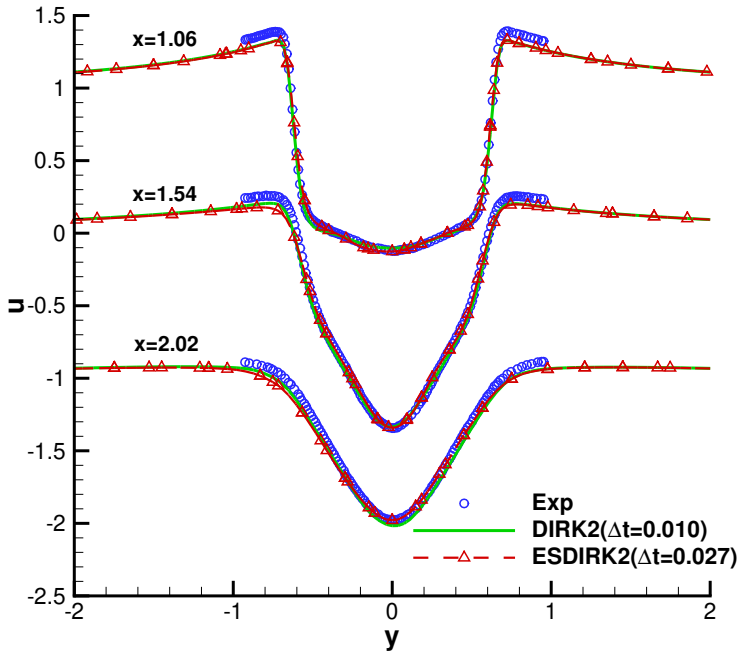


Figure 4.16: Turbulent flow over a circular cylinder:
Comparison of time-averaged velocity distribution. The velocity profiles at $x = 1.54$ and $x = 2.02$ are shifted down by increments of $u = -1$ and $u = -2$, respectively.

The adaptive time steps for a duration corresponding to approximately 19 vortex shedding periods are shown in Fig. 4.17. Table. 4.1 presents the corresponding averages, $\Delta\bar{t}$. The averaged time step of ESDIRK2 is approximately 0.027, which is 2.7 times larger than the $\Delta t = 0.01$ of the DIRK2 scheme. The adapted time steps are larger for the ESDIRK3 and ESDIRK4 with averaged values of 0.043 and 0.127, respectively, because of the improved resolution of the high-order ESDIRK schemes.

Table. 4.1 summarizes the averaged time steps and CPU time of the simulations. The results show that the simulations with the adaptive time-stepping strategy achieve good efficiency with only a small overhead in CPU time partly due to the extra costs of error estimations. The time-averaged velocity distributions are presented in Fig. 4.16. The result of ESDIRK2 with $\Delta t = 0.027$ almost coincides with the result of DIRK2 with $\Delta t = 0.01$, both of which are in good agreement with experimental results (Parnaudeau et al., 2008). The simulations with adaptive time

steps give accurate predictions of the time-averaged quantities.

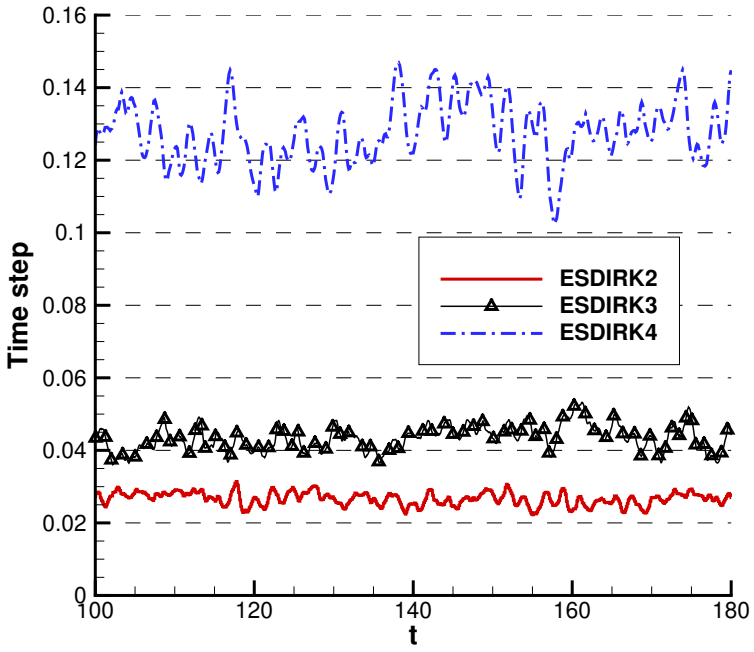


Figure 4.17: Turbulent flow over a circular cylinder:
Adaptive time steps (Δt) of ESDIRK2, ESDIRK3 and ESDIRK4.

Methods	$\Delta\bar{t}$	CPU time	CPU time ratio
DIRK2	0.010	7.2×10^2	1.00
ESDIRK2	0.027	7.9×10^2	1.09
ESDIRK3	0.043	9.8×10^2	1.36
ESDIRK4	0.127	1.1×10^3	1.53

Table 4.1: Turbulent flow over a circular cylinder:
Comparisons of averaged time steps and CPU times for different DIRK schemes with the CPU time ratio defined as the ratio of CPU time between the current method and the DIRK2 method.

Fig. 4.18 depicts the distribution of elemental Δt for the ESDIRK2 scheme corresponding to the variable ρu on a slice of the flow field. To highlight the elements with a smaller elemental time step than the global time step, the contour coloring is cut below the global time step and these elements are presented in white. One

important observation is that the relation of Eq. (4.2) is satisfied in the majority of elements. Only fewer than 1% of the elements (white region), the elemental time step is smaller than the global time step. The influence of ϵ_m in Eq. (4.6) is negligible in the global time step in this test case, which is only effective in elements with very small error magnitudes.

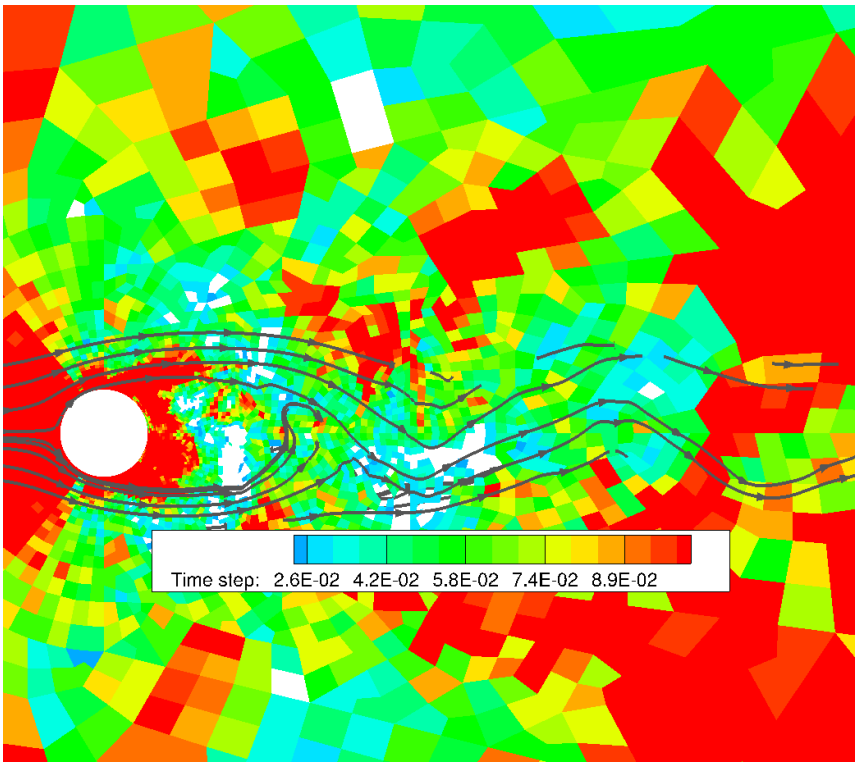


Figure 4.18: Turbulent flow over a circular cylinder:
Values of the elemental time steps.

4.1.6 Summary of parameters in the simulations

The tunable parameters in the adaptive strategy are listed in Table. 4.2. The parameter β controls the relation between the temporal error and spatial error, which is set to 0.1 for general simulations, but to 0.01 for implicit large-eddy simulations as discussed in Section 4.1.4. The value r in Eq. (4.8) determines the contribution of elemental time steps to the global time step. For $r = 0$, a simple algebraic average is used. A larger r leads to a larger contribution from elements with larger

temporal errors. The term ϵ_m in Eq. (4.6) avoids division by zero and also serves as a threshold values for the error level of interest. The current choice of Eq. (4.7) is motivated by the truncation error of the time integration process.

Parameter	Position	Range	Current choice
β	Eq. (4.2)	$0 < \beta < 1$	$\beta = 0.1$ or $\beta = 0.01$
r	Eq. (4.8)	$r \in [0, \infty)$	$r = 1$
ϵ_m	Eq. (4.6)	Small relative to $\mathbf{U}$	Eq. (4.7)

Table 4.2: Summary of tunable parameters in the adaptive strategy.

The current adaptive time-stepping strategy is used together with the adaptive Newton tolerance introduced in later Section 4.2, because they are both based on error estimates and there is no extra cost to utilize the adaptive Newton tolerance. Without using the adaptive strategy, the choices of Δt and Newton tolerance have to be changed case by case to achieve a good performance as summarized in Table. 4.3. These case by case choices are obtained at costs of repeated numerical experiments, which could be expensive especially when starting the simulations of a new test case. The comparison of computational efficiency in Table. 4.3 shows that the adaptive strategy is close to the most efficient choices in all the cases without case by case optimized parameters, which demonstrates the remarkable generality of the adaptive strategy.

Test case	CFL	Newton Tolerance	CPU time ratio
Isentropic Vortex ($h = 0.2$)	1.0	$\theta_2 = 10^{-3}$	0.96
Isentropic Vortex ($h = 0.05$)	0.2	$\theta_2 = 10^{-7}$	1.01
Boundary layer	$0.1 - 5 \times 10^4$	$\theta_2 = 10^{-6}$	1.21
Taylor-Green vortex	≈ 29	$\theta_2 = 10^{-3}$	1.27
Circular cylinder	≈ 40	$\theta_2 = 10^{-3}$	1.09

Table 4.3: Summary of parameters in different test cases and comparison of efficiency. The CPU time ratio is defined as the ratio of CPU time using the adaptive strategy and that using the parameters listed above.

4.2 Error-based adaptive Newton tolerance

In (Noventa et al., 2016; Blom et al., 2016), the authors proposed choices of Newton tolerance that ensure the iteration error should not interfere with the temporal accuracy. Based on the similar idea and previous error analysis, an adaptive Newton tolerance τ_{N} is proposed by directly relating the Newton tolerance to the temporal error, that has the following form

$$\tau_{\mathrm{N}} = \eta \|\mathbf{e}_t\|, \tag{4.10}$$

with $\eta = 0.1$. However, the relation between the Newton residual vector, $\mathbf{R}$, and the iteration error, $\Delta t\, \bar{\mathbf{e}}_{it}$, introduced into Eq. (4.1) is not trivial. In the following this relation is estimated to illustrate the assumptions made in choosing the Newton tolerance of Eq. (4.10).

Considering that

$$\Delta t \|\bar{\mathbf{e}}_{it}\| = \Delta t \|\sum_{i=1}^{s} b_i \mathbf{e}_{it}^{(i)}\| \leq \Delta t \sum_{i=1}^{s} b_i \|\mathbf{e}_{it}^{(i)}\|, \tag{4.11}$$

and $\sum_{i=1}^{s} b_i = 1$, a sufficient condition for

$$\Delta t \|\bar{\mathbf{e}}_{it}\| \leq \eta \|\mathbf{e}_t\|, \tag{4.12}$$

is that

$$\Delta t \|\mathbf{e}_{it}^{(i)}\| \leq \eta \|\mathbf{e}_t\|, \tag{4.13}$$

which maintains that the iterative error is smaller than the temporal error in Eq. (4.1) with $\eta < 1.0$.

The relation between $\Delta t \mathbf{e}_{it}^{(i)}$ and $\mathbf{R}^{(i)}$ is estimated in the following. The $\mathbf{R}^{(i)}$ can expressed as

$$\mathbf{R}^{(i)} = \mathbf{N}(\mathbf{u}_{it}^{(i)}) - \mathbf{N}(\mathbf{u}^{(i)}) = \mathbf{N}'(\mathbf{u}_{it}^{(i)}) \left(\mathbf{u}_{it}^{(i)} - \mathbf{u}^{(i)} \right) + O\left(\Delta \mathbf{u}^2\right), \tag{4.14}$$

where $\Delta\mathbf{u} = \mathbf{u}_{it}^{(i)} - \mathbf{u}^{(i)}$. Therefore, the iterative error of the solution $\mathbf{u}_{it}^{(i)}$ can be approximated as

$$\begin{aligned}\mathbf{u}_{it}^{(i)} - \mathbf{u}^{(i)} &= (\mathbf{N}')^{-1}\left(\mathbf{R}^{(i)} - O\left(\Delta\mathbf{u}^2\right)\right) \\ &= \left(\mathbf{I} - \Delta t a_{ii}\mathcal{L}'_\delta\left(\mathbf{u}_{it}^{(i)}\right)\right)^{-1}\left(\mathbf{R}^{(i)} - O\left(\Delta\mathbf{u}^2\right)\right).\end{aligned} \tag{4.15}$$

Subtracting Eq. (4.15) by Eq. (2.42) and ignoring high-order terms of $O\left(\Delta\mathbf{u}^2\right)$, we can further deduce the relation between $\Delta t\mathbf{e}_{it}$ and $\mathbf{R}$ as

$$\begin{aligned}\Delta t\mathbf{e}_{it}^{(i)} =& \Delta t\left(\mathcal{L}_\delta\left(\mathbf{u}_{it}^{(i)}\right) - \mathcal{L}_\delta\left(\mathbf{u}^{(i)}\right)\right) \\ =& \Delta t\mathcal{L}'_\delta\left(\mathbf{u}_{it}^{(i)}\right)(\mathbf{N}')^{-1}\mathbf{R}^{(i)},\end{aligned} \tag{4.16}$$

substituting Eq. (4.16) into Eq. (4.13), we get

$$\|\mathcal{L}'_\delta\left(\mathbf{I}/\Delta t - a_{ii}\mathcal{L}'_\delta\right)^{-1}\mathbf{R}^{(i)}\| \leq \eta\|\mathbf{e}_t\|. \tag{4.17}$$

The presence of $\mathcal{L}'_\delta\left(\mathbf{I}/\Delta t - a_{ii}\mathcal{L}'_\delta\right)^{-1}$, which is difficult or expensive to estimate accurately, makes the relation between $\Delta t\mathbf{e}_{it}$ and $\mathbf{R}$ problem dependent and difficult to evaluate. As Δt becomes larger $\mathcal{L}'_\delta\left(\mathbf{I}/\Delta t - a_{ii}\mathcal{L}'_\delta\right)^{-1}$ becomes closer and closer to a diagonal matrix with diagonal values of $1/a_{ii}$ and at the limit of $\Delta t \to \infty$, Eq. (4.17) becomes

$$\|\mathbf{R}^{(i)}\| \leq \eta a_{ii}\|\mathbf{e}_t\|, \tag{4.18}$$

with $0.25 < a_{ii} < 0.45$ in the ESDIRK schemes adopted. When Δt becomes smaller and smaller, the entities of the matrix $\mathcal{L}'_\delta\left(\mathbf{I}/\Delta t - a_{ii}\mathcal{L}'_\delta\right)^{-1}$ will tend to zeros. Therefore, a even larger Newton tolerance will maintain that Eq. (4.17) is satisfied. Overall, Eq. (4.18) appears to provide a good estimate of the convergence upper bound at least at the limits of $\Delta t \to \infty$ and $\Delta t \to 0$. Based on numerical experiments, we observe that the Newton tolerances of Eq. (4.10) seems good enough to maintain temporal convergence accuracy, as will be shown in Section 4.2.1. However, the assumptions made in the derivation of the Newton tolerance may be invalid

in some simulations, and if an accurate estimate of Newton tolerance is needed, $\mathcal{L}'_\delta(\mathbf{I}/\Delta t - a_{ii}\mathcal{L}'_\delta)^{-1}$ can be estimated using the Jacobian matrix or simply by a calibration for a specific problem.

Although based on some assumptions, Eq. (4.10) is still much less problem dependent compared with the commonly-used absolute Newton tolerance and relative Newton tolerance. It will also vary in a more consistent manner with the temporal error when Δt changes. Here, for convenience, we rewrite the forms of the absolute Newton tolerance Eq. (4.19) and the relative Newton tolerance Eq. (4.20)

$$\tau_{\mathrm{N}} = \theta_1 \|\mathbf{v}^0\|, \tag{4.19}$$

$$\tau_{\mathrm{N}} = \theta_2 \|\mathbf{N}(\mathbf{v}^0)\|. \tag{4.20}$$

4.2.1 2D isentropic vortex convection

The setup of the isentropic vortex flow has been introduced in Section 4.1.2. This test is also used to compares the performance of the proposed Newton tolerance, given by Eq. (4.10) and $\eta = 0.1$, with commonly-adopted tolerances given by Eq. (4.19) using values $\theta_1 = (10^{-6}, 10^{-10})$, and Eq. 4.20 using $\theta_2 = (10^{-3}, 10^{-7})$. The main purpose is to show that Eq. (4.10) can give a reasonable tolerance and still maintain temporal accuracy. In this case, we employ a second-order DG polynomial and a third-order ESDRIK (ESDIRK3) scheme with mesh size $h = 1/3$ and three different time steps: $\Delta t = 0.05$, 0.1, and 0.2.

Fig. 4.19 shows that large values of the parameters θ_1 and θ_2 in the Newton tolerances from Eq. (4.19) and Eq. (4.20) lead to a degradation of the order of temporal accuracy. Small values of these parameters achieve desired temporal order of accuracy but are not necessarily the best choice or the most efficient choice. Furthermore, they need to be calibrated for each test case and for each time step to obtain the best performance because in general the term $\Delta t\bar{\mathbf{e}}_{it}$ based on the tolerances of Eq. (4.19)/Eq. (4.20) does not scale simultaneously with the temporal

error. On the other hand as shown in Fig. 4.19, the adaptive Newton tolerance of Eq. (4.10) achieves the optimal order of accuracy.

To verify that the Newton tolerance is not unnecessarily small, we numerically seek the maximum Newton tolerance τ_{max} which maintains a difference between $\|\mathbf{e}_t^{n+1} + \Delta t \bar{\mathbf{e}}_{it}\|$ and $\|\mathbf{e}_t^{n+1}\|$ is within 1%. Compared with τ_{max}, the designed adaptive Newton tolerance $\eta\|\mathbf{e}_t\|$ should be no larger than required tp maintain temporal accuracy and should not be too small to maintain efficiency. Table. 4.4 shows that the ratios between the adaptive Newton tolerance and the τ_{max} is of the same order of 0.1 for different time steps. This indicates that the iterative error scales simultaneously with temporal error and $\eta = 0.1$ is close to the maximum choice that can maintain temporal accuracy in this test.

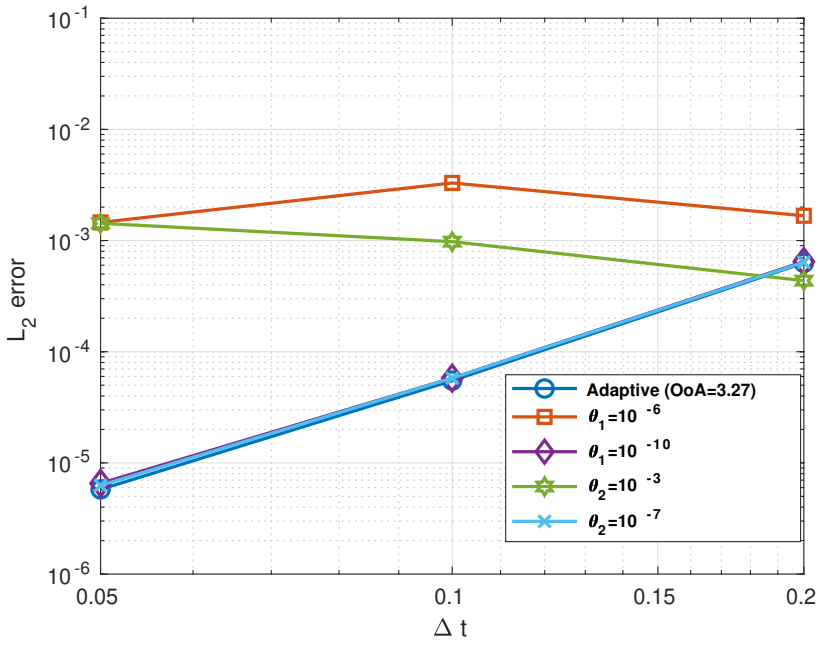


Figure 4.19: Isentropic vortex problem:
Effect of different Newton tolerances on temporal accuracy.

Δt	τ_{max}	$\eta\|\mathbf{e}_t\|$	$\eta\|\mathbf{e}_t\|/\tau_{max}$
0.05	1.0×10^{-5}	5.7×10^{-6}	0.57
0.1	4.0×10^{-5}	5.5×10^{-5}	0.14
0.2	2.0×10^{-4}	6.3×10^{-4}	0.31

Table 4.4: Isentropic vortex problem:
Comparison of the adaptive Newton tolerance and the maximum Newton tolerance that maintains temporal accuracy.

4.2.2 2D flow past a circular cylinder

To verify the adaptive Newton tolerance has a more general application, we do the same test to compare the adaptive tolerance with the 'numerically tested maximum Newton tolerance' τ_{max} in 2D flow past a circular cylinder case. The setup of this test case has been introduced in Section 2.8.3. A 2D flow past a cylinder is simulated under free-stream Mach number M = 0.3 and Reynolds number Re = 1200. Based on the flow field running after 50 periods when the vortex sheds periodically, we do the following tests. The simulations are run using a second-order DG polynomial and a fourth-order ESDIRK (ESDIRK4) scheme. We adopt three time step samples Δt = 0.02, 0.05 and 0.1 and treat the simulation of Δt = 0.005 as the reference. After running $t = 0.4s$, the errors are tested by comparing the values of momentum (ρu) with that of the reference. τ_{max} is numerically found in the same manner as Section 4.2.1. Shown in Table. 4.5, the adaptive tolerance is consistent with the optimal tolerance at all time step samples. Therefore, the adaptive Newton tolerance strategy has a general applications.

Δt	τ_{max}	$\eta\|\mathbf{e}_t\|$	$\eta\|\mathbf{e}_t\|/\tau_{max}$
0.02	2.41×10^{-4}	2.84×10^{-5}	0.12
0.05	2.40×10^{-3}	5.87×10^{-4}	0.24
0.1	2.42×10^{-2}	4.50×10^{-3}	0.31

Table 4.5: 2D flow past a circular cylinder:
Comparison of the adaptive Newton tolerance and the maximum Newton tolerance that maintains temporal accuracy.

4.3 Accuracy of Jacobian matrix approximation

For the JFNK method introduced in Section 2.4 and Section 2.6, the products of Jacobian matrix $\mathbf{N}'(\mathbf{v})$ and vectors such as the orthogonal bases $\mathbf{w}$ during the Arnoldi process are approximated by a specific finite difference method to save storage. In most cases, the matrix-vector products $\mathbf{N}'(\mathbf{v})\mathbf{w}$ use a forward difference (FD) approximation like

$$\frac{\partial \mathbf{N}(\mathbf{v})}{\partial \mathbf{v}}\mathbf{w} \approx \frac{\mathbf{N}(\mathbf{v}+\epsilon\mathbf{w}) - \mathbf{N}(\mathbf{v})}{\epsilon}. \tag{4.21}$$

Clearly, other finite difference schemes are also possible, for instance, a centered difference (CD) approximation is given by

$$\frac{\partial \mathbf{N}(\mathbf{v})}{\partial \mathbf{v}}\mathbf{w} \approx \frac{\mathbf{N}(\mathbf{v}+\epsilon\mathbf{w}) - \mathbf{N}(\mathbf{v}-\epsilon\mathbf{w})}{2\epsilon}. \tag{4.22}$$

It was proved by (An et al., 2011) that the matrix-free method using CD has a second-order accuracy while FD is only first-order accurate. The higher-order approximation can achieve higher-order accuracy but spends more time calculating extra Newton residual $\mathbf{N}$. Therefore, it is important to choose the right approximation to improve efficiency in different problems.

According to (Turner and Walker, 1992), the optimized perturbation parameter ϵ for different order finite difference schemes are

$$\epsilon \approx \bar{u}^{\frac{1}{\bar{p}+1}} \|\mathbf{v}\|, \tag{4.23}$$

where $\bar{u}$ is the machine epsilon of computer systems $\bar{u} \approx 10^{-16}$ and $\bar{p}$ is the order of the finite difference scheme. Therefore, ϵ used in later tests for FD and CD approximations are set 10^{-8} and 4.6416×10^{-6}, respectively.

4.3.1 2D steady-state flat plate boundary layer flow

The setup of this laminar flat plate boundary layer flow has been introduced in Section 4.1.3. The simulation is run under Mach number M = 0.1 and Reynolds number Re = 1.6×10^6. This case is selected because the boundary layer flow using high Reynolds number and highly-stretched meshes is a relatively stiff problem. Therefore, we can observe the residual history of GMRES and compare the convergence rate more clearly when using FD approximation and CD approximation. Secondly, we will extract the exact (analytical) Jacobian matrix $\mathbf{N}'$ for comparison with approximated Jacobian since the computational cost and storage of this case are acceptable.

The geometry, mesh and boundary conditions are also the same as shown in Fig. 4.7 and Fig. 4.8a. Unlike the test running to the converged result as shown in Fig. 4.8b, the initial field for this test is near the converged state shown as Fig. 4.20.

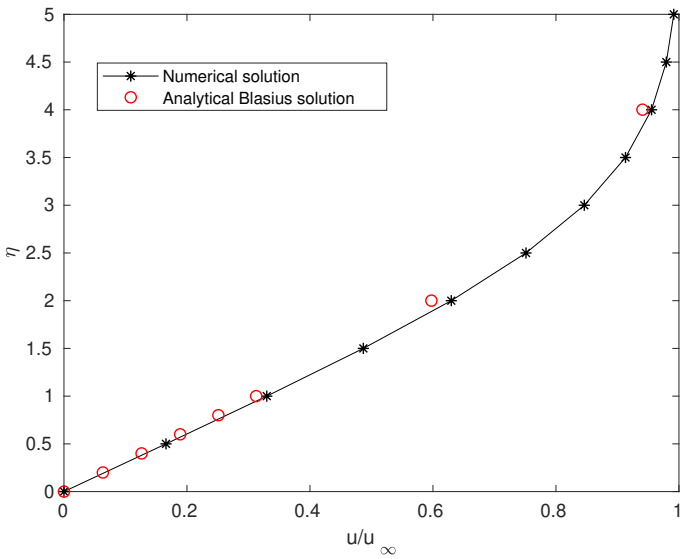


Figure 4.20: 2D steady-state flat plate boundary layer flow: Test based on a near converged flow field.

The compressible flow solver uses a second-order DG scheme and a second-order DIRK scheme. At each stage, the JFNK solver is applied to the nonlinear equation system. We focus on the resolution of the linear equation system in the first Newton

step on the non-converged initial field.

In the first test, two types of restarted GMRES(60) are utilized, where the matrix-vector products adopt FD approximation and CD approximation respectively. The tolerance of GMRES is set very low thereby reducing the initial residual norm $\mathbf{r}_0$ by a factor of 10^{-12} so that the whole residual history can be observed. Fig. 4.21 compares the residual between FD and CD in the cases: (a) without preconditioning, (b) preconditioned by BRJ(1), (c) preconditioned by BRJ(3). Fig. 4.22 compares the residual history of cases with time step Δt: (a) 4×10^{-5}, (b) 2×10^{-5}, (c) 10^{-5}. The cases using fewer preconditioning iterations and larger time steps are considered stiffer. It can be seen that in the case without preconditioning and the case using a larger time step, the stiffest problem, CD is advantageous in reducing to a lower level residual.

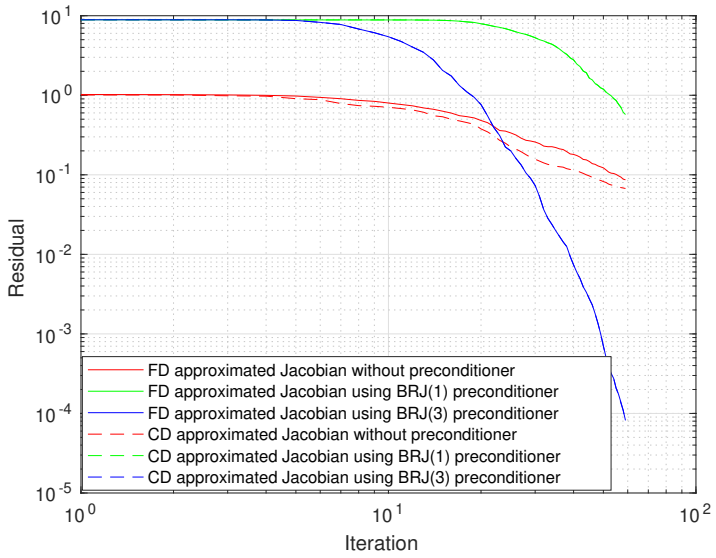


Figure 4.21: 2D steady-state flat plate boundary layer flow: Residual history of using different preconditioners.

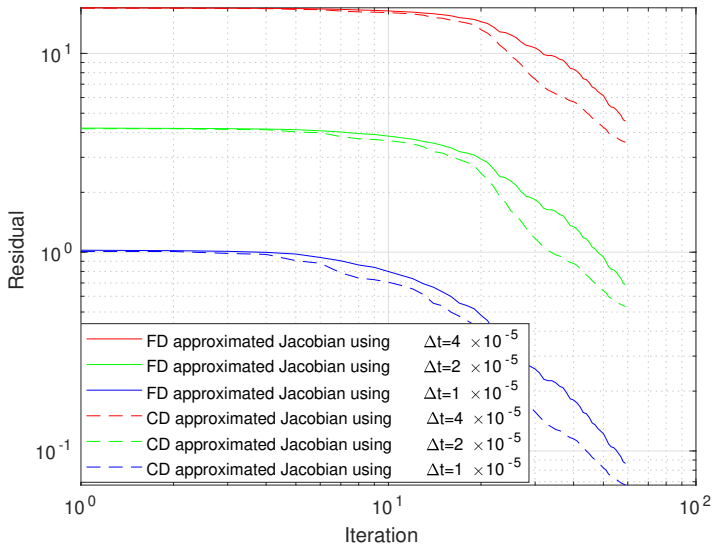


Figure 4.22: 2D steady-state flat plate boundary layer flow: Residual history of using different time steps.

In the second test, we compare the situations after restarting. We still use GMRES(60) linear solver and further observe the residual history after restarting. For comparison, we run the reference case using the exact Jacobian matrix $\mathbf{N}'$ rather than the approximated Jacobian (Jacobian-free). In other words, when we calculate the matrix-vector products during GMRES process, we compare the following three approaches: (a) $\mathbf{N}'(\mathbf{v})\mathbf{w}$, (b) $\frac{\mathbf{N}(\mathbf{v}+\epsilon\mathbf{w})-\mathbf{N}(\mathbf{v})}{\epsilon}$, and (c) $\frac{\mathbf{N}(\mathbf{v}+\epsilon\mathbf{w})-\mathbf{N}(\mathbf{v}-\epsilon\mathbf{w})}{2\epsilon}$. Shown in Fig. 4.23, the residual nearly overlap in all the cases before restarting. After restarting, there is a big jump of residual using approximated Jacobian methods because the searching bases $\mathbf{w}$ using the approximated Jacobian (Jacobian-free) method will gradually lose its orthogonality and influence the convergence after restarting. This situation is more serious for the case using FD approximation because the approximation is less accurate. This is why in Nektar++, we default set GMRES(M) not restart and directly jump into next Newton step after M GMRES iterations.

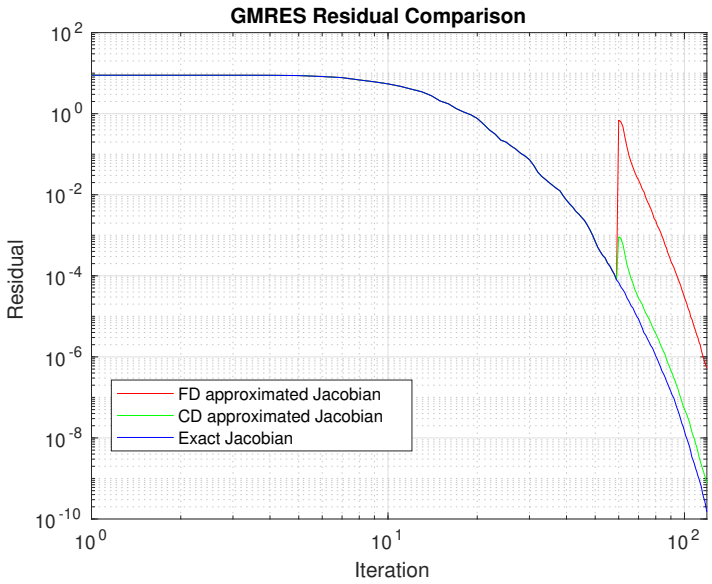


Figure 4.23: 2D steady-state flat plate boundary layer flow:
GMRES residual using approximated Jacobian and exact Jacobian.

After studying the influence from the accuracy of the approximated Jacobian matrix, it is obvious that a higher-order approximation can help reach a low-level residual after the same number of GMRES iterations. However, as mentioned at the beginning, the higher-order approximation spends more time calculating extra Newton residual $\mathbf{N}$. The choice of approximation method depends on the stiffness of the problem. A higher-order approximation is more advantageous in stiff problems.

4.3.2 Turbulent flow over a circular cylinder at $\mathrm{Re} = 3\ 900$

The 3D turbulent flow over a circular cylinder flow has been introduced in the previous Section 4.1.5. From previous observation, we found a more accurate approximation of matrix-vector products can help GMRES reach a lower level residual. This is advantageous to accelerate convergence in stiff problems. As this case is more difficult and stiffer to solve with 3.3 million DoFs, the reduction in iteration number of GMRES is more obvious when using CD approximation. Table 4.6 lists the total GMRES iterations and CPU time within one time step $\Delta t = 8 \times 10^{-3}$. There is a 19% improvement in CPU time when using CD matrix-free scheme.

	GMRES number	CPU time (s)
FD	48	1.9892×10^2
CD	25	1.6618×10^2

Table 4.6: Turbulent flow over a circular cylinder:
GMRES iterations and CPU time within one time step.

4.4 Control of freezing number of preconditioner

The frequency of updating the preconditioner is another important user-defined parameter that could influence the efficiency of the solver. The construction of a preconditioner takes up a large part of the computational cost during simulation. During intervals of smooth evolution, variables and their derivatives do not change much and the preconditioner can still remain a good approximation of the Jacobian matrix. In such situations, there is no need to update the preconditioner frequently. A commonly-used practice is to freeze the update of the preconditioner every constant number of time steps. However, when the field variables vary greatly such as flowing across a shock, delayed updates of the preconditioner based on time step will slow down, or even ruin the simulation. There are also other studies on the freezing strategies such as (Birken et al., 2008; Birken, 2013) based on ILU decomposition. But such detection of variations of the Jacobian matrix to trigger the update of the preconditioner is costly. We study the following strategy to update the preconditioner for every const number of GMRES iterations. Combined with the proposed time-stepping strategy in Section 4.1, it is expected to automatically adjust the update frequency of the preconditioner according to the stiffness of problems.

4.4.1 Taylor Green vortex

The Taylor-Green vortex test case has been described in Section 4.1.4. This case is selected because the stiffness of the problem varies during the evolution. We compare the above two freezing strategies, one is based on time step and the other is based on GMRES iterations, using three cases: (a) freezing the preconditioner

every 600 GMRES iterations, (b) freezing the preconditioner every 5 time steps, (c) freezing the preconditioner every 10 time steps.

The simulations are run using a fourth-order DG scheme and a fourth-order ESDRIK (ESDIRK4). The results are compared by the evolution of enstrophy. Using the same order of spatial and temporal discretization, Fig. 4.24 shows the results are similar.

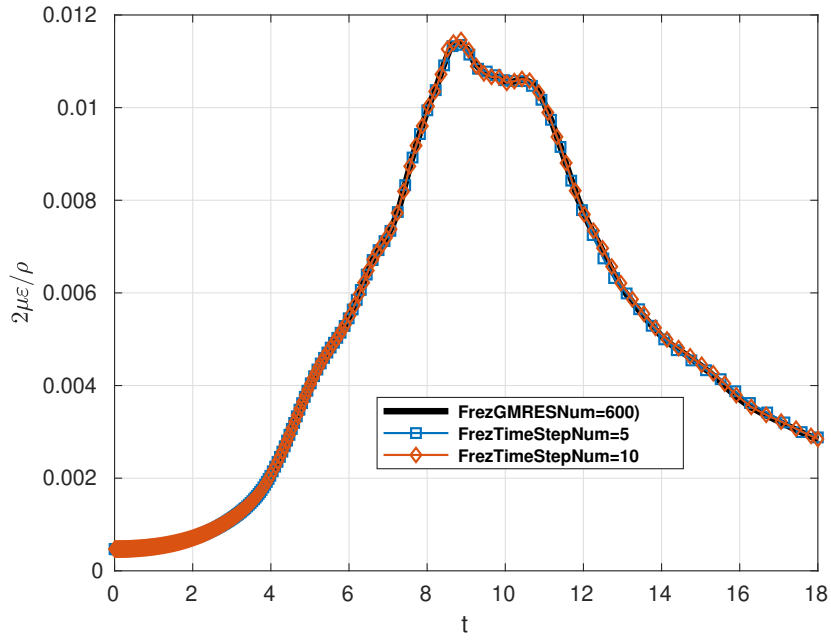


Figure 4.24: Taylor Green vortex: Evolution of the enstrophy.

Shown in Fig. 4.25, at the beginning of smooth evolution, the time step grows slowly from a small time step. When the vortex develops, the time step becomes larger and varies. Finally, the time step is large when the flow is close to a converged state. During the development of vortex, the time step varies largely and the preconditioner is expected to be updated more properly according to the stiffness of resolution.

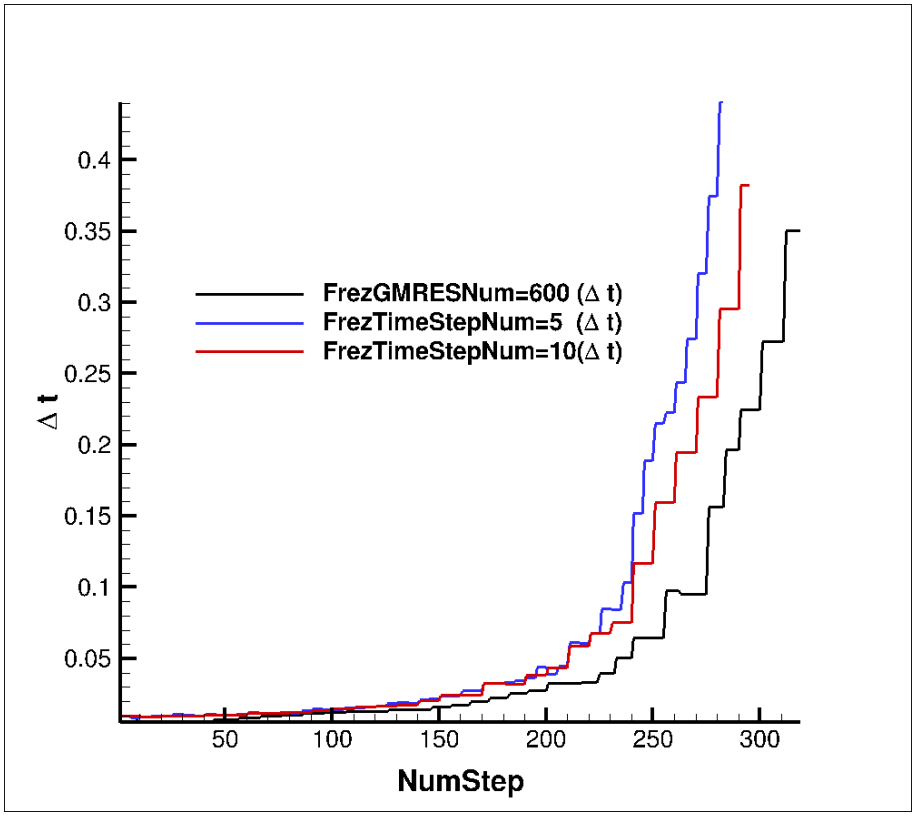


Figure 4.25: Taylor Green vortex: Time step evolution.

Fig. 4.26 demonstrates the evolution of the CPU cost per time step. The jumps on the lines are due to the extra cost to construct the preconditioner. As Fig. 4.26 shows, during the first 100 time steps, Case (b) takes more time due to more frequent updates of the preconditioner (more jumps shown in the blue line). When the time step becomes larger, some slowly converged situations happened when the update of the preconditioner is not immediate. Case (c) wastes much time around the 250^{th} time step. This is the drawback of using the freezing strategy based on time step. For smooth simulations, frequent update of the preconditioner wastes much time while for stiff problems, delayed update leads to slow convergence. The strategy of freezing the preconditioner based on GMRES iterations improves the situation as the results of case (a) show. The line of CPU time per time step behaves very average and smooth. As the stiffness becomes larger, the preconditioner is updated frequently.

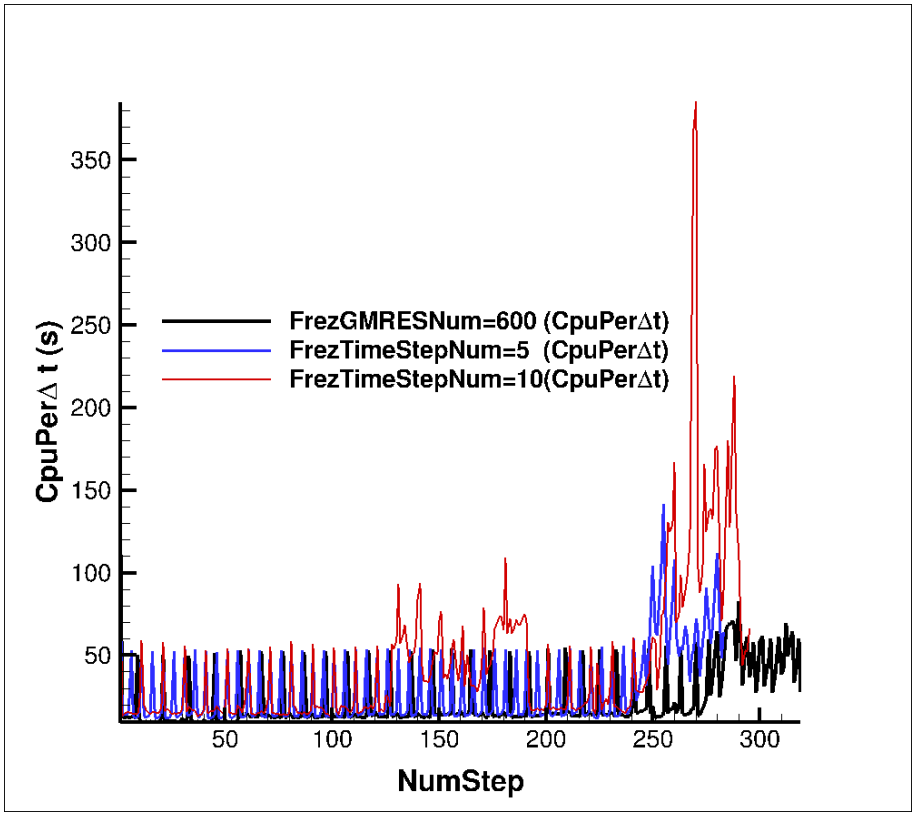


Figure 4.26: Taylor Green vortex: CPU per Δ t.

Table. 4.7 compares the total CPU time, in this case, the freezing strategy based on GMRES iterations is more efficient compared with the other two cases using freezing strategies based on time step.

One might doubt the number of GMRES iterations to freeze the preconditioner is also a user-defined parameter. However, it is much less case-dependent than the freezing strategy based on the time step, because it adjusts the update frequency automatically based on the stiffness of problems.

Case	Total CPU time (s)
Freeze GMRES (a)	7.1450×10^3
Freeze time step (b)	8.2150×10^3
Freeze time step (c)	1.3539×10^4

Table 4.7: Taylor Green vortex: Total CPU time.

4.5 Discussion and conclusions

An implicit solver always performs more efficiently than an explicit solver in situations such as steady flows, low-Mach high-Reynolds-number flows, or simulations using highly-stretched meshes. However, an implicit solver is more complicated and involves many user-defined parameters. In this section, we have introduced the strategies to relax the difficulty of choosing proper parameters in the implicit solver, which makes the implemented solver more user-friendly and effective.

The adaptive time-stepping strategy introduced in Section 4.1 is based on the idea of balancing the errors generated within one time step. This strategy maintains temporal accuracy in the sense that the total error is dominated by the spatial error and further decreasing the temporal error will not obviously improve the results of the discrete PDE system, which is verified by numerical experiments in the isentropic vortex, Taylor-Green vortex and turbulent flow over a circular cylinder problem. Moreover, this adaptive strategy is a relatively efficient choice because it is around the maximum value with temporal accuracy, the efficiency of which is tested in a variety of steady state and unsteady problems, including the isentropic vortex, Taylor–Green vortex, flat plate boundary layer, and turbulent flow over a circular cylinder. In all these tests, the adaptive methods are close to the most efficient one possibly with a small overhead due to extra costs in error estimate.

This adaptive time-stepping strategy performs well in a variety of problems without the need of tuning parameters or requiring a priori knowledge of the flow properties, thus reducing the necessity for user intervention. This feature will facilitate the development of automatic CFD simulation pipelines in a variety of application areas such as, for instance, shape optimization.

The main idea of the adaptive time-stepping is to construct a proper relation between different errors, which is not limited to the spatial discretization methods, temporal discretization methods and error estimators adopted in this paper. Provided proper defined spatial and temporal error estimators, the idea should also be applicable to other unsteady PDE solvers. Currently, the adaptive method is

based on the local errors generated within one time step, which in theory can not guarantee the global error is still spatial error dominated. This could be improved by adopting some global error estimators. The adaptive strategy provides an upper bound of the time step based on the requirement of temporal accuracy. However, this upper bound does not necessarily maintain simulation stability in challenging problems. Methods for improving robustness could be considered in the future.

In Section 4.2, based on the similar idea that the iteration error should not be larger than the temporal error to guarantee temporal accuracy, an automatic Newton tolerance is developed. Combined with the adaptive time-stepping strategy, there is no extra computational cost to do the error estimate. This strategy has been theoretically proved much less problem-dependent than other common choices of Newton tolerance such as absolute/relative Newton tolerance (Eqs. (2.39) and (2.40)). This adaptive Newton tolerance has also been numerically-tested in the benchmark tests of isentropic vortex flow and 2D flow past a circular cylinder that the Newton tolerance based on the error analysis can be close to the most efficient choice of Newton tolerance ($\tau_{\max}$ in Section 4.2). Therefore, this strategy can be used in a general application.

The studies on the approximation of Jacobian matrices' influence in Section 4.3 tell us the accuracy of the approximation method influences the orthogonality of searching bases in GMRES, thus influencing the convergence. The restart of GMRES will worsen this situation because the initial residual $\mathbf{r}_0$ after restarting is not accurate. That is why we do not restart in practice. Even though the computational cost per GMRES iteration using a higher-order approximation such as the centered difference approximation is larger, the reduction in GMRES iteration number can balance the extra cost. In the test of 3D turbulent flow past a circular cylinder, there is 19 percent improvement in efficiency.

In Section 4.4, we compare the common freezing strategy to update the preconditioner based on the number of time step with the proposed freezing strategy based on the number of GMRES iterations. The proposed strategy can automatically

adjust the update frequency of the preconditioner according to the stiffness of the problem. When the simulation is smoothly evolved, the preconditioner is updated not so frequently while meeting stiff situations such as shocks, the preconditioner can be updated immediately as a reaction. By default, we set the period of updating the preconditioner every 600 GMRES iterations based on the experience. Even though it is also a user-defined parameter, it is much less problem-dependent than the freezing strategy based on the time step.

Chapter 5

An improved shock-capturing strategy for high-order DG compressible flow simulations

We present a novel shock-capturing strategy for high-order compressible Navier-Stokes solvers, that improves the robustness and efficiency in high-order discontinuous Galerkin (DG) simulations using a combination of the best-leading terms of existing artificial viscosity approaches, which include bulk stress, shear stress, and Laplacian operators. This strategy includes two components: a modified bulk stress and an additional shear stress. The majority of the artificial viscosity is based on the bulk stress modification, which has good performance in capturing sharp shock profiles in steady simulations (Ramshaw and Mousseau, 1990; Masatsuka, 2013). Inspired by the paper (Guermond et al., 2011), it mentioned extra density-related artificial viscous flux is needed in the mass conservation equation for compressible Navier-Stokes solver to increase stability. And also impressed by Laplacian artificial viscosity's good performance in later 1D benchmark tests, we realized the importance of the density term. We further add extra density-related terms to the momentum equations and energy equation. This modified version helps dissipate oscillations at shocks and has negligible dissipation in smooth regions, which is very critical for a

robust and efficient compressible solver. We also analyze the use of auxiliary shear-stress based artificial viscosity to help improve the robustness in simulating strong shear layers. The performance of the new shock-capturing extension is verified on a number of benchmark problems.

The shock sensor adopted is briefly introduced in Section 5.1.1. Section 5.1.2 lists the typical forms of artificial viscosity models for comparison, like shear-stress based artificial viscosity, bulk-stress based artificial viscosity, and Laplacian artificial viscosity. Section 5.2 and Section 5.3 introduce the main contribution of this chapter which is the proposed new shock-capturing strategy, including the major part of modified bulk-stress based artificial viscosity dealing with shocks and the auxiliary extra shear-stress based artificial viscosity to help improve robustness in shear stress dominant regions. Results of benchmark tests are presented in Section 5.4, which also presents the design process of the new shock-capturing strategy through theoretical analysis and numerical tests. Finally, conclusions are drawn in Section 5.5.

5.1 High-order DG methods with artificial viscosity shock-capturing

This section introduces the various forms of artificial viscosity employed in shock-capturing methods for the simulations of the compressible Navier-Stokes equations.

Firstly, recall the governing equations Eq.(2.1) are

$$\frac{\partial \mathbf{U}}{\partial t} + \nabla \cdot \mathbf{F}(\mathbf{U}) + \nabla \cdot \mathbf{G}(\mathbf{U}, \nabla \mathbf{U}) = 0, \tag{5.1}$$

where $\mathbf{F}$ and $\mathbf{G}$ are the inviscid and viscous fluxes. Because the components of the viscous term such as viscous tensor τ (Eq. (2.3)) and heat flux q (Eq. (2.4)) are both functions of the dynamic viscosity μ, bulk viscosity θ and $\mathbf{U}$, we denote the viscous

term as

$$\mathbf{G} = \mathbf{G}(\mu, \theta, \mathbf{U}). \tag{5.2}$$

The shock-capturing ability of high-order DG methods is enhanced through the explicit addition of artificial diffusion flux to the original governing equation. AV methods mainly include two components, a sensor and a consistent PDE-based dissipation term. The former is designed to detect if a discontinuity exists in certain regions. The AV flux determines the form and amount of AV to each equation of the system. The employed discontinuity sensor and various forms of AV fluxes are presented in Sections 5.1.1 and 5.1.2, respectively.

5.1.1 Discontinuity sensor

Various kinds of discontinuity sensors have been developed based on the physical (Ducros et al., 1999) and mathematical (Tonicello et al., 2020) properties of discontinuities. Here, we adopt a sensor based on the decay rate of coefficients in polynomial modal space. As discussed in (Klöckner et al., 2011), these coefficients should decay sufficiently quickly for a smooth function. A highest modal decay (MDH) model (Persson and Peraire, 2006b) was proposed based on this mathematical property of smooth flows. The sensor is defined as the energy fraction of the highest modes of a variable f, which is

$$S_k = \frac{\|f - \hat{f}\|^2}{\|f\|^2}, \tag{5.3}$$

where $\hat{f}$ is the polynomial expansion containing the first $(P-1)$ terms in an orthonormal polynomial coefficient space and P denotes the polynomial order of the numerical approximation. The choice of f can be solution components like density, entropy or characteristic variables (Krivodonova et al., 2004). Here, density is selected as the variable for the sensor as suggested in (Persson and Peraire, 2006b). Based on the sensor, an artificial viscosity coefficient is defined to determine the

amount of artificial viscosity

$$\bar{\mu}^{\text{av}} = \mu_{\max} \begin{cases} 0, & s_k < c_0 - c_k, \\ \frac{1}{2}\left(1 + \sin(\frac{\pi(s_k - s_0)}{2c_k})\right), & c_0 - c_k \leq s_k < c_0 + c_k, \\ 1, & s_k \geq c_0 + c_k, \end{cases} \tag{5.4}$$

$$\hat{\mu}^{\text{av}} = \frac{h w_{max}}{P} \bar{\mu}^{\text{av}}, \tag{5.5}$$

where the variables $s_k = \log S_k$, $c_0 = -c_A - 4\log P$, and $\mu_{\max}$, c_A and c_k are tunable parameters. Suggested by the optimal parameters achieved using neural networks in (Discacciati et al., 2020), we set $\mu_{\max} = 0.5$, $c_A = 2.5$ and $c_k = 0.2$. h is the element characteristic size, $w_{max} = \text{Max}(c + |\mathbf{u}|)$ is the maximum wave speed. Persson and Peraire (2006b) proposed this scaling of viscosity coefficient with $\frac{h}{P} w_{max}$ to add appropriate amount of diffusion and achieve sub-cell shock resolutions.

A simple global smoothing mentioned in (Yu and Hesthaven, 2017) is applied to the artificial coefficient to reduce the numerical oscillations and enhance the viscosity sub-cell resolution (Discacciati et al., 2020).

The discontinuity sensor determines when and where to add artificial viscosity, and therefore has a significant impact on the performance in shock-capturing. The performance of various widely-used sensors is analyzed in more detail in studies such as (Fernandez et al., 2018; Discacciati et al., 2020).

5.1.2 Artificial viscous flux

An artificial viscosity (dissipation) flux $\mathbf{H}$ based on the discontinuity sensor is added to the original governing equations (5.1) to smear out oscillations near discontinuities, to read

$$\frac{\partial \mathbf{U}}{\partial t} + \nabla \cdot \mathbf{F} + \nabla \cdot \mathbf{G} = \nabla \cdot \mathbf{H}, \tag{5.6}$$

where the artificial viscous flux $\mathbf{H}$ is the same formulation as $\mathbf{G}$, namely $\mathbf{H} = \mathbf{H}(\mu^{\text{av}}, \theta^{\text{av}}, \mathbf{U})$.

Different forms of artificial viscous fluxes have been proposed, including the Laplacian (Persson and Peraire, 2006b), shear-stress based (or physical) (Klöckner et al., 2011) and bulk-stress based viscous fluxes (Fernandez et al., 2018).

The Laplacian artificial viscous flux is expressed as

$$\mathbf{H}_i^{\mathrm{L}} = \hat{\mu}^L \frac{\partial \mathbf{U}}{\partial x_i}, \tag{5.7}$$

which is a straightforward extension of the artificial viscous flux of scalar equations.

The shear-stress based artificial viscous flux is defined as

$$\mathbf{H}_i^{\mathrm{S}} = \mathbf{H}_i \left(\mu^S, 0, \mathbf{U}\right), \tag{5.8}$$

which is the same as the physical diffusion term but with an artificial viscosity coefficient. μ^S includes the dimension of density.

The bulk-stress based artificial viscous flux is defined as

$$\mathbf{H}_i^{\mathrm{B}} = \mathbf{H}_i \left(0, \mu^B, \mathbf{U}\right), \tag{5.9}$$

which has zero viscosity based on the shear stress but a non-zero viscosity based on the bulk term. μ^B includes the dimension of density.

A comparison of the Laplacian and shear-stress based artificial viscosities is performed in (Persson and Peraire, 2006b), where the authors conclude that shear-stress based (or physical) flux is superior to the Laplacian one since it is consistent with the physical diffusion and keeps the enthalpy constant across shocks. However, it is pointed out in (Klöckner et al., 2011) that the shear-stress based (or physical) diffusion flux cannot suppress the oscillations in density, which is also observed in our numerical tests. The bulk viscosity is shown to be suitable for large-eddy simulations in a DG framework in (Fernandez et al., 2018). However, it shares a similar problem with the shear-stress based artificial viscosity as still exhibits unwanted oscillations, which will be shown in later tests.

5.2 Development of a modified bulk-stress based artificial viscosity

An appraisal of the artificial viscous fluxes is conducted in Section 5.2.1. Based on this analysis, a modified bulk viscosity is proposed in Section 5.2.2 that combines the good shock-capturing ability and low dissipation in vortex dominated flows of different artificial viscosity forms.

5.2.1 Appraisal of different artificial viscosity forms

Numerical tests are conducted based on Sod's shock tube problem to appraise the discontinuity capturing ability of different forms of artificial viscous fluxes. The initial condition is

$$(\rho, u, p) = \begin{cases} (1.0, 0, 1.0), & x < 0 \\ (0.125, 0, 0.1), & x < 0 \end{cases} \tag{5.10}$$

This shock tube problem is a traditional test with shocks, contact discontinuities and rarefaction waves. A contact discontinuity separates the gases with different pressure and density in uniform flow. The computational region $\Omega = [0, 1]$ is partitioned using 100 elements. The simulations are run by a solver using a forward Euler time integration scheme and a fourth-order DG scheme. The results are compared at $t = 0.4$s. The results with the three different forms of artificial viscosity in Section 5.1.2 and the same sensor in Section 5.1.1 are presented in Fig. 5.1. It shows that there are obvious overshoots near the contact discontinuity and the rarefaction wave when using shear-stress based artificial viscosity and bulk-stress based artificial viscosity. In contract, the Laplacian artificial viscosity shows better performance in suppressing oscillations, which is consistent with the conclusion in (Klöckner et al., 2011).

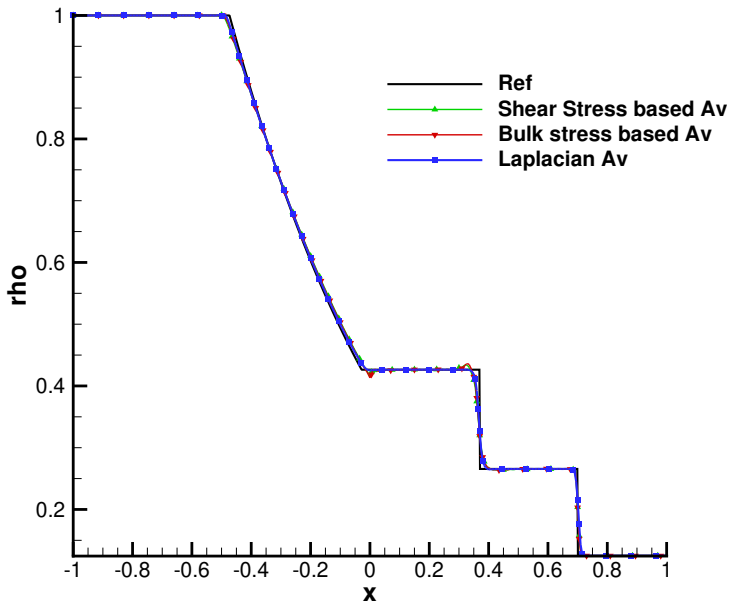


Figure 5.1: Sod shock tube (ρ distribution).

5.2.2 Modified bulk-stress based artificial viscosity

Inspired by the paper (Guermond et al., 2011), it mentioned extra density-related artificial viscous flux is needed in the mass conservation equation for compressible Navier-Stokes solver to increase stability. We further modify the momentum equations and energy equation through adding extra density-related terms on the original form of bulk-stress based artificial viscosity.

$$\mathbf{H}_i^{\mathrm{M}} = \begin{pmatrix} \hat{\mu}^M \partial\rho/\partial x_i \\ \tau_{1i}^{\mathrm{M}} \\ \cdots \\ \tau_{di}^{\mathrm{M}} \\ u_k \tau_{ki}^{\mathrm{M}} - q_i^{\mathrm{M}} \end{pmatrix}, \tag{5.11}$$

The consistent heat flux and stresses are defined as

$$q_i^{\mathrm{M}} = -\frac{\hat{\mu}^{\mathrm{M}}}{\gamma - 1}\frac{\partial p}{\partial x_i}, \tag{5.12}$$

and

$$\tau_{ij}^{\mathrm{M}} = \hat{\mu}^{\mathrm{M}}\frac{\partial \rho u_k}{\partial x_k}\delta_{ij}. \tag{5.13}$$

where the parameters $\hat{\mu}^{\mathrm{M}}$ and $\hat{\kappa}^{\mathrm{M}}$ do not include the dimension of density.

The modified artificial viscosity model is developed through comparing the differences between the original bulk-stress based artificial viscosity model and Laplacian artificial viscosity model in 1D case. We research different terms' dissipation performance. In the later section, the modification process will be introduced through researching the Sod shock tube problem. Laplacian artificial viscosity performs well in this case. We modify the bulk-stress based artificial viscosity by comparing the differences in these two models' forms and finally improve its performance.

5.3 Extra shear-stress based artificial viscosity

Many shock-involving simulations also encounter boundary layer resolutions, where shear stress is dominant. The adding of bulk viscous flux alone sometimes cannot maintain stability. Specific artificial viscosity aimed at diffusing oscillations in strong shear layers is needed to help stabilize the resolution.

Because the shear stress is linked with curl. The auxiliary shear layer sensor is designed to detect vorticity in the flow and it is defined as

$$s_{\mathrm{extra}} = \frac{h}{P}\frac{\|\nabla \times u\|}{v_{\max}}, \tag{5.14}$$

where $\|\nabla \times u\|$ is elemental L_2 norm of all the curl coefficients within local element and v_{max} is the maximum isentropic velocity, defined as the velocity the flow if all total energy was converted into kinetic energy through an isentropic expansion

(Fernandez et al., 2018)

$$v_{max} = \sqrt{\frac{2}{\gamma - 1}c^2 + \mathbf{u}^2}. \tag{5.15}$$

The multiplication of $\frac{h}{P}\frac{1}{v_{\max}}$ ensures the amount of artificial viscosity approximated by P^{th} order polynomial to resolve a shock profile is only $O(h/P)$ (Persson and Peraire, 2006b). Further division of v_{max} to guarantee dimensional consistence.

Next step is to bound the viscosity coefficient s_{extra} within a reasonable range. The coefficient should be beyond zero and under by an a priori positive value throughout the simulation to avoid accuracy and stability issues.

The limiting function L proposed by (Fernandez et al., 2018) is utilized to scale the coefficient

$$L(s_{\text{extra}}, s_0, s_{\max}) = l_{\min}(l_{\max}(a \times (s_{\text{extra}} - s_0)) - s_{\max}) + s_{\max}, \tag{5.16}$$

where

$$l_{\max}(s_{\text{extra}}) = \frac{s_{\text{extra}}}{\pi}\text{actan}(b \times s_{\text{extra}}) + \frac{s_{\text{extra}}}{2} - \frac{1}{\pi}\text{actan}(b) + \frac{1}{2}, \tag{5.17}$$

$$l_{\min}(s_{\text{extra}}) = s_{\text{extra}} - l_{\max}(s_{\text{extra}}). \tag{5.18}$$

The shape of the limiting function is shown as Fig. 5.2. This limiting function links curl $\nabla \times u$ linearly in the working region, which enables directly add extra dissipation to the positions of strong shear stress. $l_{\max}(s)$ and $l_{\min}(s)$ are smooth approximations of $\text{Max}(s, 0)$ and $\text{Min}(s, 0)$, which guarantee the calculation of partial derivatives at turning points. a is the slope of the function. b adjusts the curvature of the arc around the turning points. s_0 is the position of the first turning point, which is the cutoff of the viscosity coefficient. $s_{\max}$ sets the upper bound for scaled viscosity coefficient.

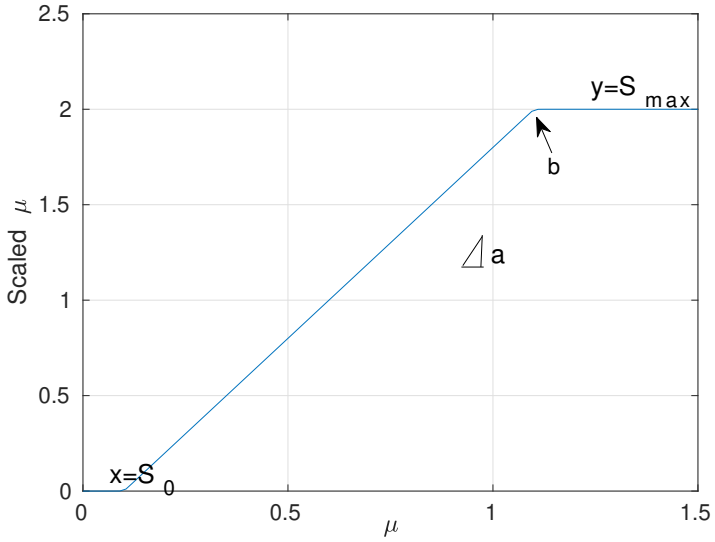


Figure 5.2: Sketch of limiting function L.

The viscosity coefficient should include the dimension of density

$$\mu^{\mathrm{E}} = \rho L(s_{\mathrm{extra}}). \tag{5.19}$$

Using the same form of Eq.(5.8), finally, the extra shear-stress based artificial viscous flux is defined as

$$\mathbf{H}_i^{\mathrm{E}} = \mathbf{H}_i\left(\mu^E, 0, \mathbf{U}\right). \tag{5.20}$$

5.4 Numerical tests

5.4.1 Sod shock tube problem

Since in 1D, the forms of Laplacian artificial viscous flux and bulk-stress based artificial viscous flux are very similar. Laplacian artificial viscosity performs well in this case. We research what differences between Laplacian artificial viscosity and bulk-stress based artificial viscosity change the performance through numerical tests as follows.

In 1D, the form of Laplacian artificial viscous flux is

$$\mathbf{H}_i^{\mathrm{L}} = \hat{\mu}^{\mathrm{L}} \nabla \mathbf{U} = \hat{\mu}^{\mathrm{L}} \begin{pmatrix} \frac{\partial \rho}{\partial x} \\ \frac{\partial \rho u}{\partial x} \\ \frac{1}{\gamma - 1} \frac{\partial p}{\partial x} + \frac{1}{2} \frac{\partial \rho u^2}{\partial x} \end{pmatrix}, \tag{5.21}$$

while the form of bulk-stress based artificial viscous flux is

$$\mathbf{H}_i^{\mathrm{B}} = \mathbf{H}_i \left(0, \mu^B, \mathbf{U}\right) = \begin{pmatrix} 0 \\ \hat{\mu}^{\mathrm{B}} \rho \frac{\partial u}{\partial x} \\ u \hat{\mu}^{\mathrm{B}} \rho \frac{\partial u}{\partial x} - q_x^{\mathrm{B}} \end{pmatrix}, \tag{5.22}$$

We do the following modifications on the original form of bulk-stress based artificial viscosity: (a) adding the term $\frac{\partial \rho}{\partial x}$ in continuity equation, (b) replacing the velocity derivative $\rho \frac{\partial u}{\partial x}$ by $\frac{\partial \rho u}{\partial x}$ in momentum and energy equations, (c) replacing the thermal term $-q_x^{\mathrm{B}}$ by $\frac{\hat{\mu}^{\mathrm{B}}}{\gamma - 1} \frac{\partial p}{\partial x}$ in energy equation. Other modifications of combined adding terms are also tested. Modification (a) overcomes the oscillations compared with original bulk-stress based artificial viscosity. However, there seems too much dissipation added near the shock. The result of modification (b) is not shown in plots because the simulation broke down. The result of modification (c) shows little artificial viscosity is added leading to large oscillations if replacing the thermal term alone. Finally, we found the combined modification of adding $\frac{\partial \rho}{\partial x}$ and replacing the thermal term obviously improves the performance shown in Fig. 5.3 and Fig. 5.4. The extra oscillations are smoothed at the positions of rarefaction fans and shock. The final form of modified bulk-stress based artificial viscosity defined in Eq. (5.11) also replace $\rho \frac{\partial u}{\partial x}$ by $\frac{\partial \rho u}{\partial x}$. Even though in 1D, the form of modified bulk-stress based artificial viscosity is now similar to that of Laplacian artificial viscosity, it is different in 2D and 3D.

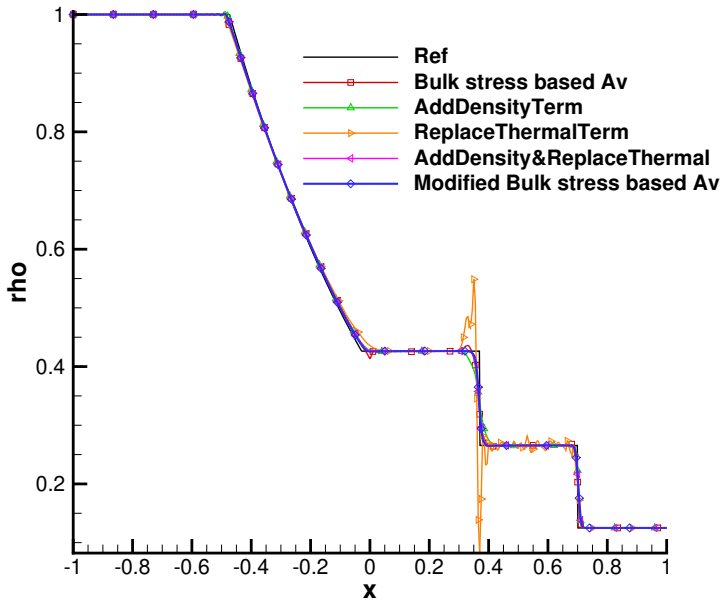


Figure 5.3: Sod shock tube (ρ distribution).

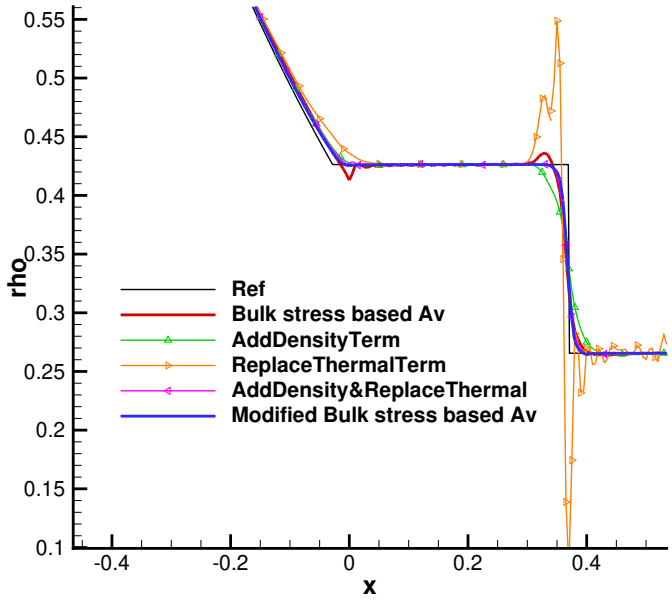


Figure 5.4: Sod shock tube (details in ρ distribution).

5.4.2 Shu-Osher problem

The Shu-Osher problem is a 1D simulation where a front shock moves within an inviscid flow, generating frequent density fluctuations. Large gradients in smooth regions lead the problem challenging for a high-order solver to describe the exact solution. Despite to further comparing the shock-capturing performance, this benchmark is expected to demonstrate if the new shock-capturing model can deal with oscillations accurately using high-order DG methods.

Consider the computational domain $\Omega = [-5, 5]$ partitioned using $N_e = 200$ elements. Even though the Shu-Osher problem is a 1D case, it is run as a 2D simulation, where the y-direction boundaries are set as periodic. The simulation is run to $t = 1.8$ s using a solver of a fourth-order Runge-Kutta time integration scheme and a fourth-order DG scheme. The initial condition is set as

$$(\rho, u, p) = \begin{cases} (3.857143, 2.629369, 10.333333), & -5 \leq x \leq -4 \\ (1 + 0.2\sin(5x), 0, 1), & -4 < x \leq 5 \end{cases} \tag{5.23}$$

The values of the primitive variables are also set as the initial values at the left-side BC of $x = -5$. A pressure outflow is imposed at $x = 5$ where the pressure is fixed and a Neumann BC is used for density.

All the artificial viscosity models can capture shocks rightly at positions near x=-3, x=-2, x=-1. Between these shocks x=-2.5, x=-1.5, from Fig. 5.6, small oscillations exist when using shear-stress based artificial viscosity and bulk-stress based artificial viscosity. At these smooth positions, there should not be extra artificial viscosity added. Therefore, the cause of these oscillations can also be improper amount of artificial viscosity added at shocks. During the positions of frequent physical oscillations $x \in [0.5, 2]$, modified bulk-stress based artificial viscosity diffuses less and describes the solution distribution more exactly compared with original bulk-stress based artificial viscosity. It is even better than Laplacian artificial viscosity. Because the simulations are actually run in a 2D computational region. The benefit that the model of modified bulk-stress based artificial viscosity can describe shock

shape through adding less diffusion in later 2D cases.

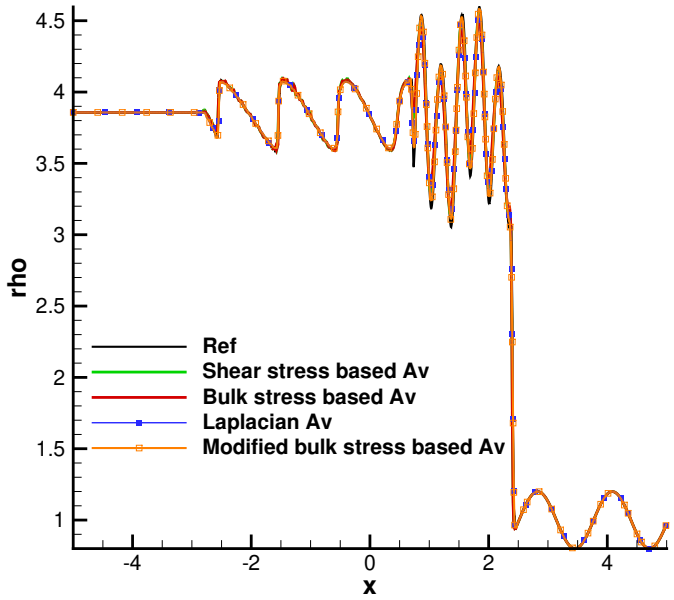


Figure 5.5: Shu-Osher problem (ρ distribution).

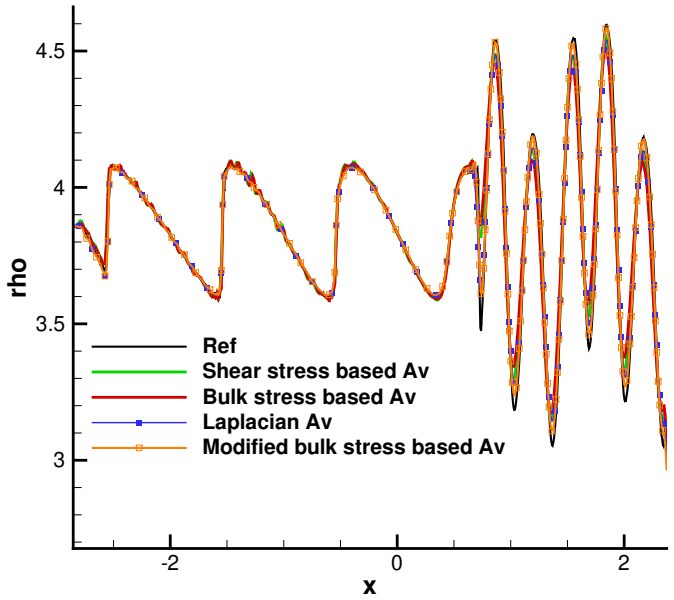


Figure 5.6: Shu-Osher problem (details in ρ distribution).

5.4.3 2D Riemann problem

The 2D Riemann problem was firstly proposed in (Schulz-Rinne et al., 1993). It is a square domain with an initial condition that comprises of four regions separated by two shocks perpendicular to each other and parallel to the axes. During the further evolution of shock interactions, nearly all the physical phenomena such as shock reflections, vortex-shock interactions, the spiral formation can be observed. Therefore, many researchers considered it as a verification test case, such as (Chang et al., 1995; Zhang and Zheng, 1990) analyzed the configurations from theoretical point of view and (Han et al., 2011; Rathan and Naga Raju, 2018) simulated it numerically. We consider the 2D Riemann problem mentioned in (Chiavassa et al., 2001), which has a modified geometry and is run longer time until $t = 0.8s$ to study physical behavior at later time. This case is also used to demonstrate why the extra shear-stress based artificial viscosity introduced in Section. 5.3 is needed and how we tune the parameters.

The computational region $\Omega = [0, 1] \times [0, 1]$ is partitioned into $[200 \times 200]$ elements. The initial filed is set as

$$(\rho, u, v, p) \begin{cases} (1.5, 0, 0, 1.5), & 0.8 < x < 1.0; \quad 0.8 < y < 1.0 \\ (0.5323, 1.206, 0.0, 0.3), & 0.0 < x < 0.8; \quad 0.8 < y < 1.0 \\ (0.138, 1.206, 1.206, 0.029), & 0.0 < x < 0.8; \quad 0.0 < y < 0.8 \\ (0.5323, 0.0, 1.206, 0.3), & 0.8 < x < 1.0; \quad 0.0 < y < 0.8 \end{cases} \tag{5.24}$$

Since the exact solution is unknown, the referred results shown in Fig.5.7 is from (Ha et al., 2013), using the fifth-order WENO-JS scheme and dense mesh $[800 \times 800]$ elements. A well-developed scheme should simulate the roll-up vortexes near the jet structure in the center due to the presence of Kelvin-Helmholtz instabilities such as Fig.5.7d.

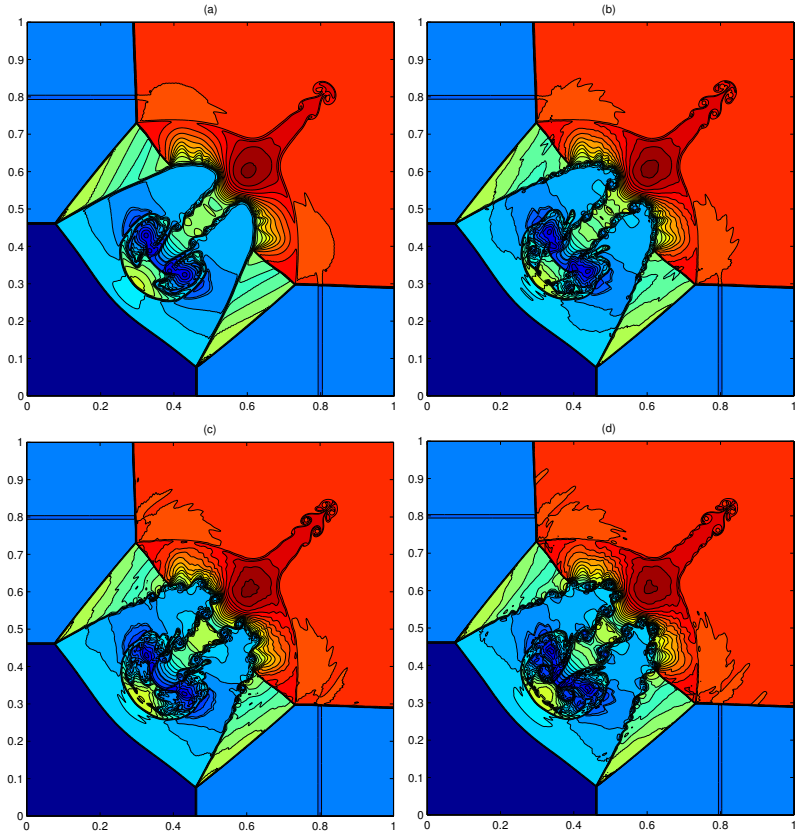
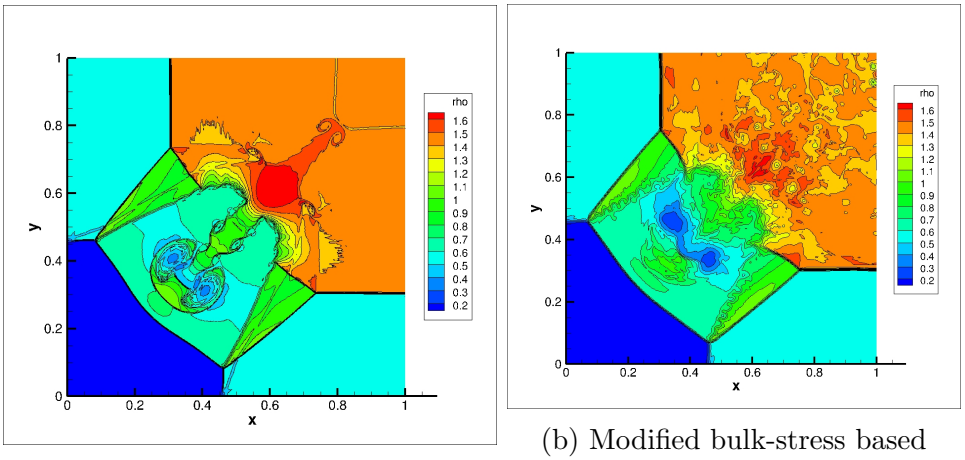


Figure 5.7: 2D Riemann problem: Reference density profiles at $t = 0.8$s (a)WENO-JS, (b)WENO-M, (c)WENO-Z, (d)WENO-NS (Ha et al., 2013).

The simulations are run by a compressible Euler solver using a second-order Runge-Kutta time integration scheme and a fourth-order DG scheme. Fig.5.8a shows the result using Laplacian artificial viscosity. Compared with reference contour, even though Laplacian artificial viscosity captures shock at the right positions and smooth the oscillations around shocks, it also smooths most of the small-scale vortexes.

As shown in Fig. 5.8b, the numerical diffusion of the modified artificial viscosity model cannot suppress the influence from shear layer interaction. The numerical vortex explodes from the left bottom part, where shear stress interaction exists. This is because the simulations are run by an inviscid solver. The shear interaction region is only diffused by numerical dissipation. The modified bulk-stress based artificial viscosity does not add any shear stress dissipation. It is challenging for the model to diffuse the oscillations in this region.



(a) Laplacian artificial viscosity

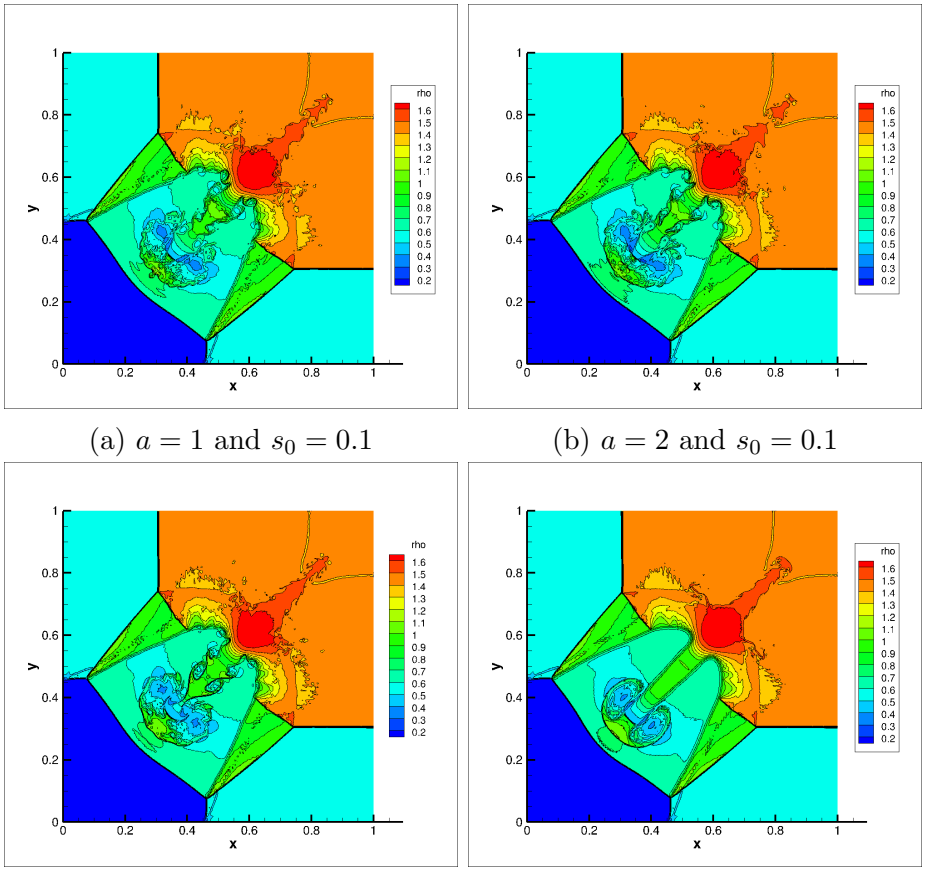
(b) Modified bulk-stress based artificial viscosity

Figure 5.8: 2D Riemann problem: ρ distribution.

Extra shear-stress based artificial viscosity is designed for a stability issue in such situations when strong shear stresses exist. The parameters of the limiting function Eq. 5.16 for extra artificial viscosity are tuned through this test. The parameters of cutoff point s_0 and the slope a play important roles. The former sets the criteria when to add artificial viscosity and the latter determines the sensitivity to add artificial viscosity.

First group's results shown in Fig. 5.9a, Fig. 5.9b and Fig. 5.9c compare simulations using limiting function: (a) a=1, (b) a=2, and (c) a=5. More and more scales are diffused and when we run the case using $a = 100$, which is not shown here, the simulation breaks out due to too much artificial viscosity being added. After a set of tests, $a = 2$ is enough for most cases.

Comparing the results between Fig. 5.9b and Fig. 5.9d, the smaller criteria s_0 will lead to larger region of artificial viscosity added. $s_0 = 0.1$ is acceptable for most cases because smaller criteria will lead to smooth many small-scale physical features in interesting places.

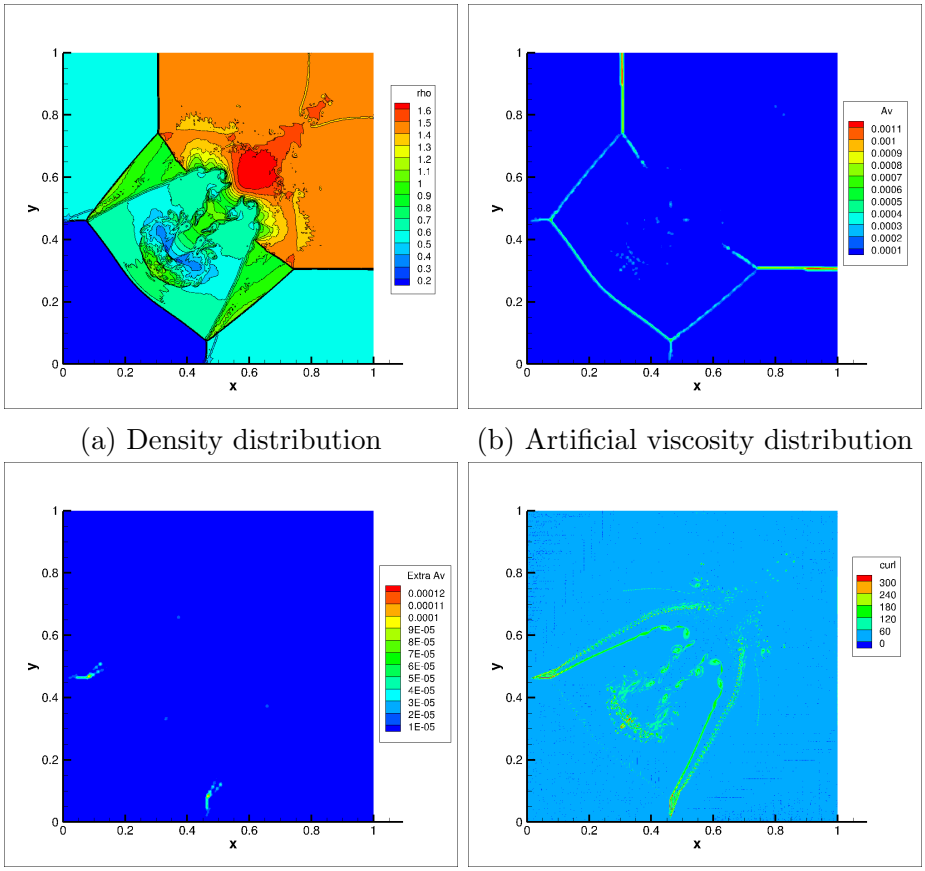


(a) $a = 1$ and $s_0 = 0.1$ (b) $a = 2$ and $s_0 = 0.1$

(c) $a = 5$ and $s_0 = 0.1$ (d) $a = 2$ and $s_0 = 0.05$

Figure 5.9: 2D Riemann problem ρ distribution:
Tunable parameters' influence.

Fig. 5.10b and Fig. 5.10c demonstrate the modified bulk artificial viscosity distribution and extra shear-stress based artificial viscosity distribution. The modified bulk artificial viscosity works at the right place around shocks and the extra artificial viscosity just works around the vorticity center. Comparing the curl distribution shown in Fig. 5.10d, the addition of extra artificial viscosity does little influence on the development of vortexes. The amount of extra artificial viscosity is also ignorable compared with that of modified bulk artificial viscosity, around 10 percent. However, in conclusion, there is not a standard set of parameters for all tests. The parameters a and s_0 are easy to adjust in different problems.



(a) Density distribution (b) Artificial viscosity distribution

(c) Extra artificial viscosity (d) Curl distribution

Figure 5.10: 2D Riemann problem: Field distributions.

5.4.4 2D shock vortex interaction

The test is introduced by (Discacciati et al., 2020). It simulates a 2D inviscid interaction between a strong vortex and a strong shock with complicated physical phenomena occurring after the interaction. For example, the intensity of vorticity is strengthened after flowing across the shock. The vortex is also split due to the compression effect of the shock passage. This case is used to verify that a well-designed shock-capturing scheme should diffuse rightly at the shock and has little influence on interesting structures such as the vortical structures.

The computational domain $\Omega = [0, 2] \times [0, 1]$ is partitioned into $[80 \times 40]$ elements. Slip, adiabatic wall boundary conditions are applied on the top and bottom surfaces, while the characteristics-based, non-reflecting boundary conditions are applied at

inflow and outflow. A counter-clockwise vortex is initially located upstream a normal shock at $x_s = 0.5$, and gradually advects through the shock. The initial flow field is given by

$$v(r) = v_\alpha + u_\infty \vec{\mathbf{e}}_x, \tag{5.25}$$

$$v_\alpha = u_m \vec{\mathbf{e}}_\alpha \cdot \begin{cases} \dfrac{r}{r_a}, & r \le r_a \\ \dfrac{r_a}{r_a^2 - r_b^2}\left(r - \dfrac{r_b^2}{r}\right), & r_a \le r \le r_b \\ 0, & r_b \le r \end{cases} \tag{5.26}$$

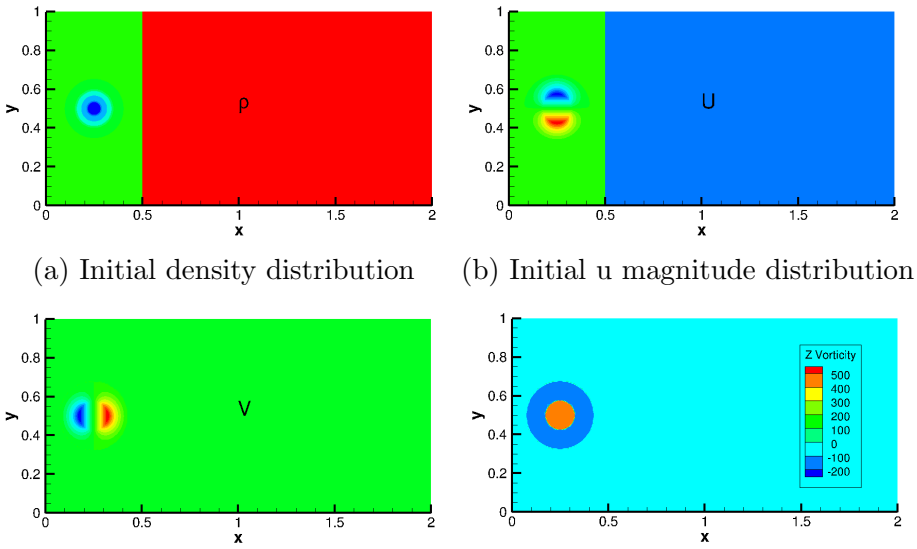
$$T(r) = \begin{cases} T_\infty - \displaystyle\int_r^{r_b} \frac{1}{r_a} \frac{|v_\alpha(r')|^2}{r'} dr', & r < r_b \\ 0, & r \ge r_b \end{cases}, \tag{5.27}$$

$$\rho(r) = \rho_\infty \left(\frac{T(r)}{T_\infty}\right)^{\frac{1}{\gamma-1}}, \tag{5.28}$$

$$p(r) = p_\infty \left(\frac{T(r)}{T_\infty}\right)^{\frac{\gamma}{\gamma-1}}, \tag{5.29}$$

where $r_a = 0.075$, $r_b = 0.175$, $\gamma = 1.4$, r denotes the distance from vortex center, u_∞, ρ_∞, p_∞ are inflow velocity magnitude, inflow density and inflow pressure. The inflow Mach number is $M_\infty = 1.5$. $u_m = 0.6u_\infty$ and x, α represent horizontal and tangential direction.

The initial field is presented as Fig. 5.11



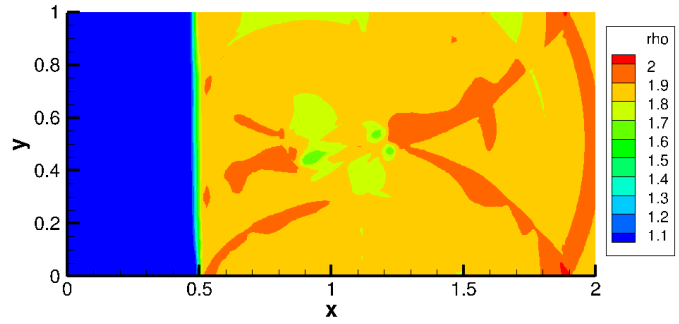
(a) Initial density distribution (b) Initial u magnitude distribution

(c) Initial v magnitude distribution (d) Initial z vorticity distribution

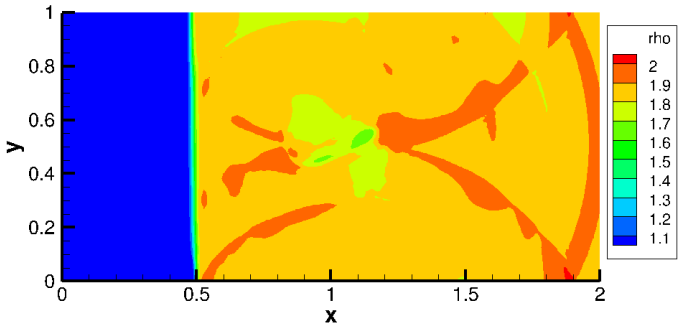
Figure 5.11: 2D shock vortex interaction: Initial field.

The simulations are run using a second-order Runge-Kutta time integration scheme and a third-order DG scheme from $t = 0$s to $t = 0.0413$s after the vortex flowing across the shock. This simulation is compared with a numerical reference using a denser mesh [400×200] and a sixth-order DG scheme.

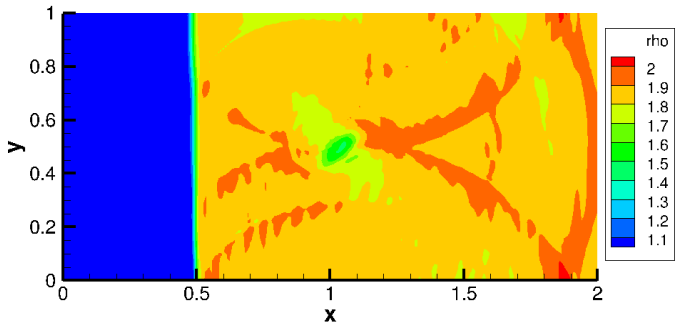
Compared with the original bulk-stress based artificial viscosity, the simulation using the modified edition has fewer oscillations even though the big-scale physical structures are similar. The simulations using shear-stress based artificial viscosity and bulk-stress based artificial viscosity experience the similar level oscillations. The results of simulations using Laplacian artificial viscosity and modified bulk-stress based artificial viscosity are more smooth. Therefore, here not all the results are shown.



(a) Modified bulk-stress based artificial viscosity.



(b) Modified bulk-stress based artificial viscosity with added shear-stress based artificial viscosity.

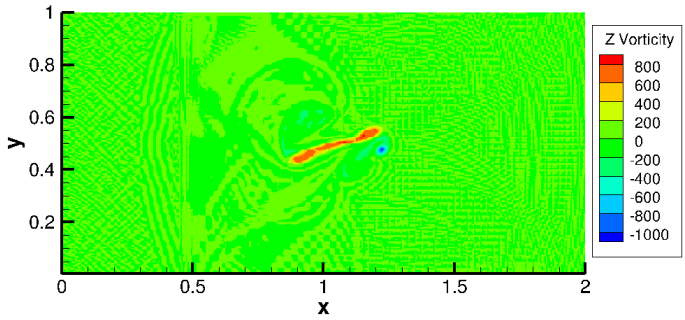


(c) Shear stress-based artificial viscosity.

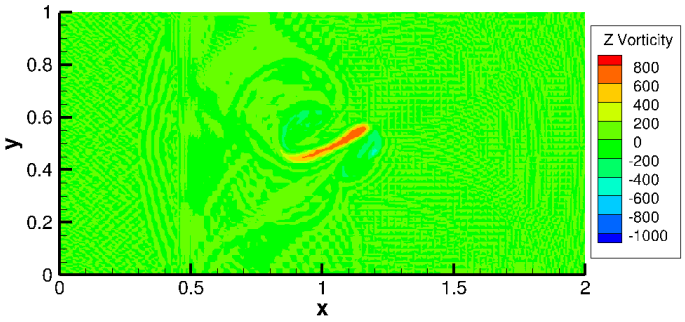
Figure 5.12: 2D shock vortex interaction: Density distribution.

To see the differences between the performance of Laplacian artificial viscosity and modified bulk-stress based artificial viscosity, we plot the vorticity magnitude distribution. Compared with the initial vorticity Fig. 5.11d, it can be seen in the following figures that the intensity of vorticity gets strengthened after flowing cross-

ing the shock. The case using modified bulk-stress based artificial viscosity adds less diffusion and catches more small scales near the vorticity when comparing with the result of the reference simulation in Fig. 5.13d. However, if adding extra shear-stress based artificial viscosity, some small scales could be smoothed (shown in Fig. 5.13b). However, using the tuned set of parameters $a = 1$ and $s_0 = 0.1$, the result still demonstrates more details than the result using Laplacian artificial viscosity.

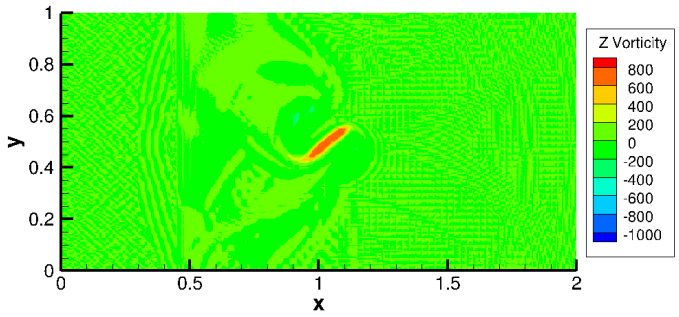


(a) Modified bulk-stress based artificial viscosity.

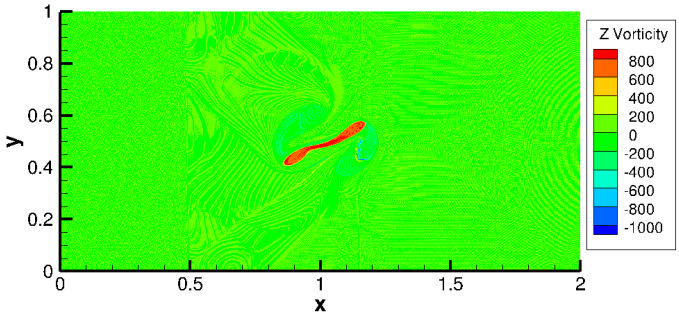


(b) Modified bulk-stress based artificial viscosity with added shear-stress based artificial viscosity.

Figure 5.13: 2D shock vortex interaction: Z vorticity.



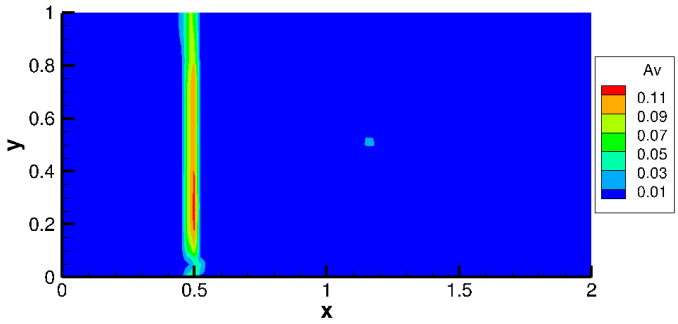
(c) Laplacian artificial viscosity.



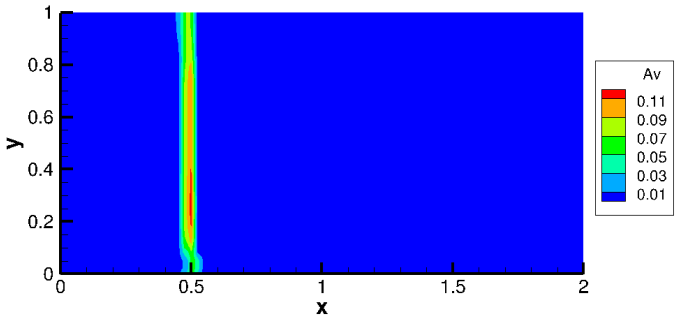
(d) Reference using dense mesh.

Figure 5.13: 2D shock vortex interaction: Z vorticity.

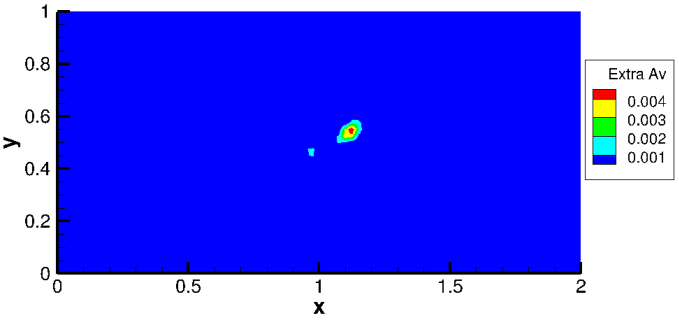
The artificial viscosity distributions are as follows: (a) Modified bulk case' artificial viscosity (Fig. 5.14a), (b) Modified bulk with added artificial viscosity case' artificial viscosity (Fig. 5.14b), (c) Modified bulk with added artificial viscosity case' extra artificial viscosity (Fig. 5.14c), and (d) Laplacian case's artificial viscosity (Fig. 5.14d). Comparing Fig. 5.14a and Fig. 5.14b, the distribution of modified bulk artificial viscosity are the same. It is important to conclude that the adding of extra artificial viscosity is independent of the adding of modified bulk artificial viscosity. Therefore, it is easy to adjust a reasonable amount of extra artificial viscosity if simulations involve strong shear stress interactions.



(a) Modified bulk-stress based artificial viscosity.

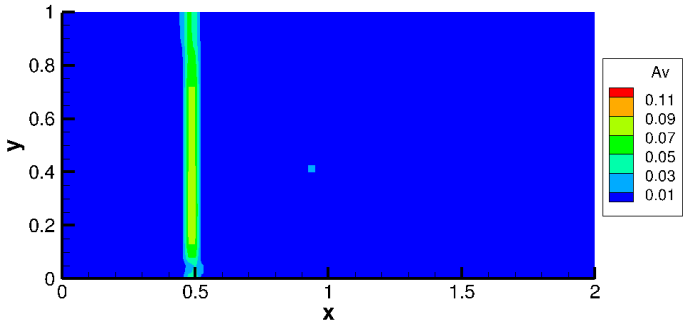


(b) Modified bulk-stress based artificial viscosity with added shear-stress based artificial viscosity.



(c) Modified bulk-stress based artificial viscosity with added shear-stress based artificial viscosity.

Figure 5.14: 2D shock vortex interaction: Artificial viscosity distribution.



(d) Laplacian artificial viscosity.

Figure 5.14: 2D shock vortex interaction: Artificial viscosity distribution.

5.5 Discussion and conclusions

We propose a new shock-capturing strategy by combining some of the best features of bulk-stress based, shear-stress based, and Laplacian artificial viscosities. The modified bulk-stress based artificial viscosity plays the main role in shock-capturing with an additional shear-stress based artificial viscosity to help stabilize situations involving strong shear layers. This strategy provides accurate and sharp shock profiles in 1D test cases like the Sod shock tube and Shu-Osher problems. The numerical results in more complex multi-dimensional cases like 2D Riemann and shock-vortex problems show an adequate amount of artificial viscosity is added near shocks and relatively negligible dissipation is enforced on resolved vortical structures. As a result, sharp shock profiles and more small-scale features can be simulated compared with the results using other types of artificial viscosity.

The modified bulk artificial viscosity is the key component in our shock-capturing strategy. It enables capture sharp shock profiles but has a negligible impact on vortical structures. The extra shear-stress based artificial viscosity is not necessary for most situations and it works independently of modified bulk artificial viscosity. The amount of extra artificial viscosity only accounts for a small part (less than 10%) compared with modified bulk-stress based artificial viscosity.

Chapter 6

Conclusions and future work

Section 6.1 summarizes the contributions of this thesis and Section 6.2 discusses some potential future work.

6.1 Conclusions

This thesis focuses on developing a solver for simulating unsteady compressible flows efficiently and accurately within the framework of Nektar++. The developed solver chooses to employ spectral/hp element methods, specially DG methods, for spatial discretization and implicit schemes, such as BDF/DIRK methods, for temporal discretization. High-order DG methods can satisfy the accuracy requirement in high-fidelity compressible simulations, and implicit schemes can potentially achieve better efficiency since the time step is not constrained by the CFL stability condition as explicit schemes. The nonlinear system after discretization is solved by the Jacobian-free Newton-Krylov (JFNK) method. This method combines the use of a Newton-type nonlinear solver and a Krylov linear solver. The Jacobian matrix-vector products are approximated in a finite difference way to save storage. To improve the efficiency, a block relaxed Jacobi (BRJ) preconditioner is designed. This preconditioner is partially matrix-free implementation with a hybrid calculation of analytical and numerical Jacobian in consideration of the storage requirement and

computational cost. After the implementation, the problem of too many user-defined parameters within the implicit solver is studied. Lastly, to extend the application of the developed solver to high-speed flow simulations, a novel shock-capturing strategy is proposed.

The main contributions of this thesis are summarized in the following aspects:

1. An efficient implicit compressible flow solver is developed within Nektar++. The robustness and efficiency have been tested in various steady/unsteady compressible flow simulations. In the efficiency comparison test with an explicit solver in 2D flow past a circular cylinder, the speedup of the implicit solver is approximately 64 at an unsteady state of $\mathrm{Re} = 1200$ while it is up to 100 even 1000 at a steady state of $\mathrm{Re} = 40$.

2. A systematic framework of adaptive strategies is proposed to overcome the additional problem that too many user-defined parameters within the implicit solver. A novel adaptive time-stepping strategy is proposed based on the observation that in a fixed mesh simulation, when the total error is dominated by the spatial error, the further decreasing of temporal error through decreasing the time step cannot help increase the accuracy but only slow down the solver. An adaptive Newton tolerance is designed based on the assumption that the iterative error should not overlap the temporal error to guarantee temporal accuracy. The freezing strategy of updating preconditioner based on GMRES convergence state is tested more efficient and less problem-dependent. The efficiency of the adaptive solver has been verified in a number of steady/unsteady, resolved/unresolved cases, such as the 2D isentropic vortex, 2D steady-state flat plate boundary layer flow, 2D flow past a circular cylinder, Taylor-Green vortex, and turbulent flow over a circular cylinder. In all the tests, the efficiency of the adaptive solver is close to the most efficient one with a little overhead due to the extra cost to do error estimates. The development of this adaptive solver avoids repeated tests of tunning parameters and provides an efficient solver for users with an accurate result.

3. A novel shock-capturing strategy is designed to help dissipate oscillations at shocks but have less dissipation in smooth regions. The majority of this strategy considers the density's influence and modifies the original bulk-stress based artificial viscosity through adding extra density-related terms. The new shock-capturing model describes more accurate and sharp shock profiles, shown as the results of Sod shock tube, Shu-Osher problems, but has less dissipation in smooth regions shown as the results in shock-vortex interaction problem.

6.2 Future work

There are still many interesting aspects that can improve the performance of the current solver. We will focus on the following topics in future work

1. A more efficient *hp*-multigrid solver: From the spectral analysis of *p*-multigrid preconditioner, we found the spectral properties of the *p*-multigrid preconditioner are quite similar to that of its smoother. For example, if direct using a BJac preconditioner or using BJac as the smoother for a *p*-multigrid preconditioner, the condition number and eigenvalue distributions of the preconditioned matrices are quite similar. That means the efficiency of an efficient *p*-multigrid preconditioner is promising to be optimized better than directly using its smoother for preconditioning, because some smoothing iterations are at the lower level. However, the efficiency is expected to be improved more obviously if combined with the use of *h*-multigrid. The smoothing of low-frequency errors using *p*-multigrid is only restricted to local elements while *h*-multigrid will help smooth the errors in the whole computational region.

2. A partially-updated preconditioner: Now the update of the preconditioner based on the GMRES convergence state avoids the frequent reconstruction of the preconditioner during a smoothly evolved flow field. However, the whole preconditioner is needed to be updated, which is still time-consuming. If the block preconditioner is only updated at those blocks where the flow field

evolves dramatically, the efficiency can be further improved.

3. A more robust shock-capturing strategy: The choice of characteristic length to scale the artificial viscosity is worth studying. Since a complex simulation involves different physical scales, different characteristic lengths for different scales' physical phenomena can help add a more suitable amount of artificial viscosity.

4. A combined application of p-type adaptivity, the proposed time-stepping adaptivity, and the novel shock-capturing strategy: The distributions of local adaptive time steps can work as an indicator of under-resolved regions. The accuracy can be selectively increased in the regions of highly vorticity gradients through p-adaptivity while the regions of sharp discontinuities can be dissipated through the proposed shock-capturing strategy.

Bibliography

Alexander, R. (1977). Diagonally implicit Runge-Kutta methods for stiff ODEs. SIAM Journal on Numerical Analysis, 14(6):1006–1021.

Aliabadi, S., Tu, S., and Watts, M. (2004). An alternative to limiter in discontinous Galerrkin finite element method for simulation of compressible flows. In 42nd AIAA Aerospace Sciences Meeting and Exhibit, page 76.

Alzaeili, J. S. and Mazaheri, K. (2006). Bulk viscosity damping for accelerating convergence of compressible viscous flow solvers. In ECCOMAS CFD 2006: Proceedings of the European Conference on Computational Fluid Dynamics, Egmond aan Zee, The Netherlands, September 5-8, 2006. Delft University of Technology; European Community on Computational Methods .

An, H.-B., Wen, J., and Feng, T. (2011). On finite difference approximation of a matrix-vector product in the Jacobian-free Newton-Krylov method. Journal of Computational and Applied Mathematics, 236(6):1399–1409.

Arévalo, C., Söderlind, G., Hadjimichael, Y., and Fekete, I. (2021). Local error estimation and step size control in adaptive linear multistep methods. Numerical Algorithms, pages 537–563.

Arnold, D. N., Brezzi, F., Cockburn, B., and Marini, L. D. (2002). Unified analysis of discontinuous Galerkin methods for elliptic problems. SIAM Journal on Numerical Analysis, 39(5):1749–1779.

Ashby, S., Brown, P., Dorr, M., and Hindmarsh, A. (1995). A linear algebraic analysis of diffusion synthetic acceleration for the Boltzmann transport equation. SIAM Journal on Numerical Analysis, 32(1):128–178.

Barter, G. and Darmofal, D. (2007). Shock capturing with higher-order, PDE-based artificial viscosity. In 18th AIAA Computational Fluid Dynamics Conference, page 3823.

Barter, G. E. and Darmofal, D. L. (2010). Shock capturing with PDE-based artificial viscosity for DGFEM: Part I. formulation. Journal of Computational Physics, 229(5):1810–1827.

Bassi, F., Botti, L., Colombo, A., Crivellini, A., Ghidoni, A., and Massa, F. (2016). On the development of an implicit high-order discontinuous Galerkin method for DNS and implicit LES of turbulent flows. European Journal of Mechanics-B/Fluids, 55:367–379.

Bassi, F., Crivellini, A., Rebay, S., and Savini, M. (2005). Discontinuous Galerkin solution of the Reynolds-averaged Navier–Stokes and k–ω turbulence model equations. Computers & Fluids, 34(4-5):507–540.

Bassi, F. and Rebay, S. (1997a). A high-order accurate discontinuous finite element method for the numerical solution of the compressible Navier–Stokes equations. Journal of Computational Physics, 131(2):267–279.

Bassi, F. and Rebay, S. (1997b). High-order accurate discontinuous finite element solution of the 2D Euler equations. Journal of Computational Physics, 138(2):251–285.

Bassi, F. and Rebay, S. (2000). A high order discontinuous Galerkin method for compressible turbulent flows. In Discontinuous Galerkin Methods, pages 77–88. Springer.

Bassi, F., Rebay, S., Mariotti, G., Pedinotti, S., and Savini, M. (1997). A high-order accurate discontinuous finite element method for inviscid and viscous turbomachinery flows. In Proceedings of the 2nd European Conference on Turbomachinery Fluid Dynamics and Thermodynamics, pages 99–109. Antwerpen, Belgium.

Bastian, P., Müller, E. H., Müthing, S., and Piatkowski, M. (2019). Matrix-free multigrid block-preconditioners for higher order discontinuous Galerkin discretisations. Journal of Computational Physics, 394:417–439.

Benzi, M. (2002). Preconditioning techniques for large linear systems: A survey. Journal of Computational Physics, 182(2):418–477.

Berezin, I. and Zhidkov, N. (1962). Computational methods. Metody vychislenii, 2.

Bijl, H., Carpenter, M. H., Vatsa, V. N., and Kennedy, C. A. (2002). Implicit time integration schemes for the unsteady compressible Navier-Stokes equations: Laminar flow. Journal of Computational Physics, 179(1):313–329.

Birken, P. (2013). Numerical Methods for the Unsteady Compressible Navier-Stokes Equations. PhD thesis, Kassel, Universität, FB 10, Mathematik und Naturwissenschaften, Institut für Mathematik.

Birken, P., Gassner, G., Haas, M., and Munz, C.-D. (2013). Preconditioning for modal discontinuous Galerkin methods for unsteady 3D Navie-Stokes equations. Journal of Computational Physics, 240:20–35.

Birken, P., Tebbens, J. D., and Meister (2008). Preconditioner updates applied to CFD model problems. Applied Numerical Mathematics, 58(11):1628–1641.

Blom, D. S., Birken, P., Bijl, H., Kessels, F., Meister, A., and van Zuijlen, A. H. (2016). A comparison of Rosenbrock and ESDIRK methods combined with iterative solvers for unsteady compressible flows. Advances in Computational Mathematics, 42(6):1401–1426.

Bolemann, T., Beck, A., Flad, D., Frank, H., Mayer, V., and Munz, C.-D. (2015). High-order discontinuous Galerkin schemes for large-eddy simulations of moderate Reynolds number flows. In IDIHOM: Industrialization of high-order methods-a top-down approach, pages 435–456. Springer.

Bonfiglioli, A. and Paciorri, R. (2014). Convergence analysis of shock-capturing and shock-fitting solutions on unstructured grids. AIAA Journal, 52(7):1404–1416.

Braess, D. (1982). The convergence rate of a multigrid method with Gauss-Seidel relaxation for the Poisson equation. In Multigrid methods, pages 368–386. Springer.

Brandt, A. (1977). Multi-level adaptive solutions to boundary-value problems. Mathematics of Computation, 31(138):333–390.

Brandt, A. (1982). Guide to multigrid development. In Multigrid methods, pages 220–312. Springer.

Brezzi, F., Manzini, G., Marini, D., Pietra, P., and Russo, A. (1999). Discontinuous finite elements for diffusion problems. Atti Convegno in onore di F. Brioschi (Milano 1997), Istituto Lombardo, Accademia di Scienze e Lettere, 1999:197–217.

Brown, P. N. and Saad, Y. (1990). Hybrid Krylov methods for nonlinear systems of equations. SIAM Journal on Scientific and Statistical Computing, 11(3):450–481.

Bu, S., Jung, W., and Kim, P. (2016). An error embedded Runge-Kutta method for initial value problems. Kyungpook Mathematical Journal, 56(2):311–327.

Bücker, H. M., Pollul, B., and Rasch, A. (2009). On cfl evolution strategies for implicit upwind methods in linearized euler equations. International Journal for Numerical Methods in Fluids, 59(1):1–18.

Bucker, H. M., Pollul, B., and Rasch, A. (2009). On CFL evolution strategies for implicit upwind methods in linearized Euler equations. International Journal for Numerical Methods in Fluids, 59(1):1–18.

Burrage, K. and Erhel, J. (1998). On the performance of various adaptive preconditioned GMRES strategies. Numer. Linear Algebra Appl., 5(2):101–121.

Burrage, K. and Petzold, L. (1990). On order reduction for Runge–Kutta methods applied to differential/algebraic systems and to stiff systems of ODEs. SIAM journal on numerical analysis, 27(2):447–456.

Butcher, J. C. (1964). Implicit Runge-Kutta processes. Mathematics of Computation, 18(85):50–64.

Butcher, J. C. (2016). Numerical methods for ordinary differential equations. Wiley, third edition.

Butcher, J. C. and Diamantakis, M. (1998). DESIRE: diagonally extended singly implicit Runge–Kutta effective order methods. Numerical Algorithms, 17(1):121–145.

Campbell, S. L., Ipsen, I. C. F., Kelley, C. T., and Meyer, C. D. (1996). GMRES and the minimal polynomial. BIT Numerical Mathematics, 36(4):664–675.

Cantwell, C., Moxey, D., Comerford, A., Bolis, A., Rocco, G., Mengaldo, G., De Grazia, D., Yakovlev, S., Lombard, J.-E., Ekelschot, D., Jordi, B., Xu, H., Mohamied, Y., Eskilsson, C., Nelson, B., Vos, P., Biotto, C., Kirby, R., and Sherwin, S. (2015). Nektar++: An open-source spectral/hp element framework. Computer Physics Communications, 192:205–219.

Canuto, C., editor (2006). Spectral methods: fundamentals in single domains. Springer-Verlag.

Cassinelli, A., Montomoli, F., Adami, P., and Sherwin, S. J. (2018). High fidelity spectral/hp element methods for turbomachinery. In Turbo Expo: Power for Land, Sea, and Air, volume 51012, page V02CT42A020. American Society of Mechanical Engineers.

Chan, T. F. and Jackson, K. R. (1984). Nonlinearly preconditioned Krylov subspace methods for discrete Newton algorithms. SIAM Journal on Scientific and Statistical Computing, 5(3):533–542.

Chang, T., Chen, G.-Q., and Yang, S. (1995). On the 2D Riemann problem for the compressible Euler equations I. Interaction of shocks and rarefaction waves. Discrete & Continuous Dynamical Systems, 1(4):555.

Chapman, A., Saad, Y., and Wigton, L. (2000). High-order ILU preconditioners for CFD problems. International Journal for Numerical Methods in Fluids, 33(6):767–788.

Cherednichenko, S., Frey, C., and Ashcroft, G. (2012). On the application of the Discontinuous Galerkin method to turbomachinery flows. In 6th European Congress on Computational Methods in Applied Sciences and Engineering, pages 2359–2375.

Chiavassa, G., Donat, R., and Marquina, A. (2001). Fine-mesh numerical simulations for 2D Riemann problems with a multilevel scheme. In Hyperbolic problems: theory, numerics, applications, pages 247–256. Springer.

Chudanov, V., Aksenova, A., Goreinov, S., Makarevich, A., and Pervichko, V. (2014). Validation of a new method for solving of CFD problems in nuclear engineering using petascale HPC. In International Conference on Nuclear Engineering, volume 45943, page V004T10A009. American Society of Mechanical Engineers.

Cockburn, B. (1998). An introduction to the discontinuous Galerkin method for convection-dominated problems. In Advanced numerical approximation of nonlinear hyperbolic equations, pages 150–268. Springer.

Cockburn, B. and Dawson, C. (1999). Some extensions of the local discontinuous Galerkin method for convection-diffusion equations in multidimensions. The Mathematics of Finite Elements and Applications.

Cockburn, B., Hou, S., and Shu, C.-W. (1990). The Runge-Kutta local projection discontinuous Galerkin finite element method for conservation laws. IV. The multidimensional case. Mathematics of Computation, 54(190):545–581.

Cockburn, B., Lin, S.-Y., and Shu, C.-W. (1989). TVB Runge-Kutta local projection discontinuous Galerkin finite element method for conservation laws III: One-dimensional systems. Journal of Computational Physics, 84(1):90–113.

Cockburn, B. and Shu, C.-W. (1989). TVB Runge-Kutta local projection discontinuous Galerkin finite element method for conservation laws. II. General framework. Mathematics of Computation, 52(186):411–435.

Cockburn, B. and Shu, C.-W. (1991). The Runge-Kutta local projection-discontinuous-Galerkin finite element method for scalar conservation laws. ESAIM: Mathematical Modelling and Numerical Analysis, 25(3):337–361.

Cockburn, B. and Shu, C.-W. (1998a). The local discontinuous Galerkin method for time-dependent convection-diffusion systems. SIAM Journal on Numerical Analysis, 35(6):2440–2463.

Cockburn, B. and Shu, C.-W. (1998b). The Runge-Kutta discontinuous Galerkin method for conservation laws V: multidimensional systems. Journal of Computational Physics, 141(2):199–224.

Cockburn, B. and Shu, C.-W. (2001). Runge–Kutta discontinuous Galerkin methods for convection-dominated problems. Journal of Scientific Computing, 16(3):173–261.

Cook, A. W. (2013). Effects of heat conduction on artificial viscosity methods for shock capturing. Journal of Computational Physics, 255:48–52.

Cook, A. W. and Cabot, W. H. (2005). Hyperviscosity for shock-turbulence interactions. Journal of Computational Physics, 203(2):379–385.

Datta, B. N. (2010). Numerical linear algebra and applications. SIAM, 2nd edition.

De Laborderie, J., Duchaine, F., Gicquel, L., Vermorel, O., Wang, G., and Moreau, S. (2018). Numerical analysis of a high-order unstructured overset grid method for compressible LES of turbomachinery. Journal of Computational Physics, 363:371–398.

De Wiart, C. C., Hillewaert, K., Duponcheel, M., and Winckelmans, G. (2014). Assessment of a discontinuous Galerkin method for the simulation of vortical flows at high Reynolds number. International Journal for Numerical Methods in Fluids, 74(7):469–493.

DeCougny, H. L., Devine, K. D., Flaherty, J. E., Loy, R., Özturan, C., and Shephard, M. S. (1994). Load balancing for the parallel adaptive solution of partial differential equations. Applied Numerical Mathematics, 16(1-2):157–182.

Dennis Jr, J. E. and Schnabel, R. B. (1996). Numerical methods for unconstrained optimization and nonlinear equations. SIAM.

Diosady, L. and Darmofal, D. (2007). Discontinuous Galerkin solutions of the Navier-Stokes equations using linear multigrid preconditioning. In 18th AIAA Computational Fluid Dynamics Conference, page 3942.

Diosady, L. T. and Darmofal, D. L. (2009). Preconditioning methods for discontinuous Galerkin solutions of the Navier–Stokes equations. Journal of Computational Physics, 228(11):3917–3935.

Discacciati, N., Hesthaven, J. S., and Ray, D. (2020). Controlling oscillations in high-order discontinuous Galerkin schemes using artificial viscosity tuned by neural networks. Journal of Computational Physics, page 109304.

Dolejší, V. and Feistauer, M. (2015). Discontinuous Galerkin Method, volume 48 of Springer. Springer International Publishing.

Ducros, F., Ferrand, V., Nicoud, F., Weber, C., Darracq, D., Gacherieu, C., and

Poinsot, T. (1999). Large-eddy simulation of the shock/turbulence interaction. Journal of Computational Physics, 152(2):517–549.

Edalatpour, V., Hezari, D., and Khojasteh Salkuyeh, D. (2015). Accelerated generalized SOR method for a class of complex systems of linear equations. Mathematical Communications, 20(1):37–52.

Eisenstat, S. and Walker, H. (1996). Choosing the forcing terms in an inexact Newton method. SIAM Journal on Scientific Computing, 17(1):16–32.

Evans, D. J. (1968). The use of preconditioning in iterative methods for solving linear equations with symmetric positive definite matrices. IMA Journal of Applied Mathematics, 4(3):295–314.

Feistauer, M., Feistauer, M., Felcman, J., and Straškraba, I. (2003). Mathematical and computational methods for compressible flow. Oxford University Press.

Fernandez, P., Nguyen, C., and Peraire, J. (2018). A physics-based shock capturing method for unsteady laminar and turbulent flows. In 2018 AIAA Aerospace Sciences Meeting, Kissimmee, Florida. American Institute of Aeronautics and Astronautics.

Fernandez, P., Nguyen, N., and Peraire, J. (1994). Physics capturing of discontinuous Galerkin methods for under-resolved turbulence simulations. J. Comput. Phys., In preparation for submission.

Ferracina, L. and Spijker, M. (2008). Strong stability of singly-diagonally-implicit Runge–Kutta methods. Applied Numerical Mathematics, 58(11):1675–1686.

Ferrand, R., Galtier, S., Sahraoui, F., and Federrath, C. (2020). Compressible turbulence in the interstellar medium: new insights from a high-resolution supersonic turbulence simulation. The Astrophysical Journal, 904(2):160.

Gautier, R., Biau, D., and Lamballais, E. (2013). A reference solution of the flow over a circular cylinder at Re=40. Computers & Fluids, 75:103–111.

Ghai, A., Lu, C., and Jiao, X. (2019). A comparison of preconditioned Krylov subspace methods for largescale nonsymmetric linear systems. Numerical Linear Algebra with Applications, 26(1):e2215.

Gholami, A., Malhotra, D., Sundar, H., and Biros, G. (2016). FFT, FMM, or multigrid? A comparative study of state-of-the-art Poisson solvers for uniform and nonuniform grids in the unit cube. SIAM Journal on Scientific Computing, 38(3):C280–C306.

Glowinski, R., Keller, H., and Reinhart, L. (1985). Continuation-conjugate gradient methods for the least squares solution of nonlinear boundary value problems. SIAM Journal on Scientific and Statistical Computing, 6(4):793–832.

Greenbaum, A. (1997). Iterative methods for solving linear systems. Frontiers in applied mathematics. SIAM.

Grinstein, F. F., Margolin, L. G., and Rider, W. J. (2007). Implicit Large Eddy Simulation: Computing Turbulent Fluid Dynamics. Cambridge University Press.

Guermond, J.-L., Pasquetti, R., and Popov, B. (2011). Entropy viscosity method for nonlinear conservation laws. Journal of Computational Physics, 230(11):4248–4267.

Gustafsson, K. (1991). Control theoretic techniques for stepsize selection in explicit Runge-Kutta methods. ACM Transactions on Mathematical Software (TOMS), 17(4):533–554.

Gustafsson, K. (1992). Control of error and convergence in ODE solvers. PhD thesis, Lund University.

Gustafsson, K. (1994). Control-theoretic techniques for stepsize selection in implicit Runge-Kutta methods. ACM Transactions on Mathematical Software (TOMS), 20(4):496–517.

Ha, Y., Ho Kim, C., Ju Lee, Y., and Yoon, J. (2013). An improved weighted essentially non-oscillatory scheme with a new smoothness indicator. Journal of Computational Physics, 232(1):68–86.

Han, E., Li, J., and Tang, H. (2011). Accuracy of the adaptive GRP scheme and the simulation of 2D Riemann problems for compressible Euler equations. Communications in Computational Physics, 10(3):577–609.

Hartmann, R. (2013). Higher-order and adaptive discontinuous Galerkin methods with shock-capturing applied to transonic turbulent delta wing flow. International Journal for Numerical Methods in Fluids, 72(8):883–894.

Hartmann, R. and Houston, P. (2002). Adaptive discontinuous Galerkin finite element methods for the compressible Euler equations. Journal of Computational Physics, 183(2):508–532.

Hartmann, R. and Houston, P. (2006a). Symmetric interior penalty DG methods for the compressible Navier–Stokes equations I: Method formulation. International Journal of Numerical Analysis & Modeling, 1(2):1–20.

Hartmann, R. and Houston, P. (2006b). Symmetric interior penalty DG methods for the compressible Navier–Stokes equations II: Goal–oriented a posteriori error estimation. International Journal of Numerical Analysis & Modeling, 3(2):141–162.

Higueras, I. and Roldán, T. (2018). Order barrier for low-storage DIRK methods with positive weights. Journal of Scientific Computing, 75(1):395–404.

Hillewaert, K. (2013). Development of the discontinuous Galerkin method for high-resolution, large scale CFD and acoustics in industrial geometries. PhD Thesis, Univ. Louvain.

Hirsch, C., Hillewaert, K., Hartmann, R., Couaillier, V., Boussuge, J.-F., Chalot,

F., Bosniakov, S., and Haase, W. (2021). TILDA: Towards industrial LES/DNS in aeronautics. Applied Science, 148(21):10202.

Hogben, L. (2006). Handbook of linear algebra. CRC press.

Holst, K. R., Glasby, R. S., and Bond, R. B. (2020). On the effect of temporal error in high-order simulations of unsteady flows. Journal of Computational Physics, 402:108989.

Houston, P. (1999). Discontinuous Galerkin FEMs for CFD: A posteriori error estimation and adaptivity. University of Nottingham, page 202.

huu Cong, N. (1993). A-stable diagonally implicit Runge-Kutta-Nyström methods for parallel computers. Numerical Algorithms, 4(2):263–281.

Il'in, V. (2021). Iterative preconditioned methods in Krylov spaces: Trends of the 21st century. Computational Mathematics and Mathematical Physics, 61(11):1750–1775.

Ipsen, I. C. and Meyer, C. D. (1998). The idea behind Krylov methods. The American Mathematical Monthly, 105(10):889–899.

Jameson, A. (2017). Origins and further development of the jameson–schmidt–turkel scheme. AIAA Journal, 55(5):1487–1510.

Jameson, A. and Mavriplis, D. (1986). Finite volume solution of the two-dimensional Euler equations on a regular triangular mesh. AIAA Journal, 24(4):611–618.

Johnsen, E., Larsson, J., Bhagatwala, A. V., Cabot, W. H., Moin, P., Olson, B. J., Rawat, P. S., Shankar, S. K., Sjögreen, B., Yee, H. C., et al. (2010). Assessment of high-resolution methods for numerical simulations of compressible turbulence with shock waves. Journal of Computational Physics, 229(4):1213–1237.

Jörgensen, J. B., Kristensen, M. R., and Thomsen, P. G. (2018). A family of ESDIRK integration methods. arXiv:1803.01613.

Kalkote, N., Assam, A., and Eswaran, V. (2019). Acceleration of later convergence in a density-based solver using adaptive time stepping. AIAA Journal, 57(1):352–364.

Karniadakis, G. and Sherwin, S. (2013). Spectral/hp element methods for computational fluid dynamics. Oxford University Press.

Ke, C., Ablowitz, M., Olver, P., Davis, S., Iserles, A., Ockendon, J., and Hinch, E. (2005). Matrix preconditioning techniques and applications. Mathematical Physics and Mathematics.

Kelley, C. T. (1995). Iterative methods for linear and nonlinear equations. SIAM.

Kelley, C. T. (2003). Solving nonlinear equations with Newton's method. SIAM.

Kennedy, C. and Carpenter, M. (2016). Diagonally implicit Runge-Kutta methods for ordinary differential equations. A Review. NASA Report. Langley research center. Hampton VA, 23681:162.

Kennedy, C. A. and Carpenter, M. H. (2019). Diagonally implicit Runge-Kutta methods for stiff ODEs. Applied Numerical Mathematics, 146:221–244.

Kennedy, C. A., Carpenter, M. H., and Lewis, R. M. (2000). Low-storage, explicit Runge–Kutta schemes for the compressible Navier–Stokes equations. Applied Numerical Mathematics, 35(3):177–219.

Kincaid, D., Kincaid, D. R., and Cheney, E. W. (2009). Numerical analysis: mathematics of scientific computing, volume 2. American Mathematical Soc.

Klöckner, A., Warburton, T., and Hesthaven, J. S. (2011). Viscous shock capturing in a time-explicit discontinuous Galerkin method. Mathematical Modelling of Natural Phenomena, 6(3):57–83.

Knoll, D. A. and Keyes, D. E. (2004). Jacobian-free Newton-Krylov methods: A survey of approaches and applications. Journal of Computational Physics, 193(2):357–397.

Kovac, J. and Strang, G. (2005). The fundamentals and advantages of multi-grid techniques. SIAM Journal on Numerical Analysis, 101(1):12–126.

Krivodonova, L. (2007). Limiters for high-order discontinuous Galerkin methods. Journal of Computational Physics, 226(1):879–896.

Krivodonova, L., Xin, J., Remacle, J.-F., Chevaugeon, N., and Flaherty, J. E. (2004). Shock detection and limiting with discontinuous Galerkin methods for hyperbolic conservation laws. Applied Numerical Mathematics, 48(3-4):323–338.

Kroll, N., Bieler, H., Deconinck, H., Couaillier, V., Van der Ven, H., and Sorensen, K. (2010). ADIGMA–A European Initiative on the Development of Adaptive Higher-Order Variational Methods for Aerospace Applications: Results of a Collaborative Research Project Funded by the European Union, 2006-2009, volume 113. Springer.

Kvrn, A. (2004). Singly diagonally implicit Runge–Kutta methods with an explicit first stage. BIT Numerical Mathematics, 44(3):489–502.

Landmann, B., Kessler, M., Wagner, S., and Krämer, E. (2008). A parallel, high-order discontinuous Galerkin code for laminar and turbulent flows. Computers & Fluids, 37(4):427–438.

Larsen, E. W. (1982). Unconditionally stable diffusion-synthetic acceleration methods for the slab geometry discrete ordinates equations. Part I: Theory. Nuclear Science and Engineering, 82(1):47–63.

Larsson, J., Lele, S., and Moin, P. (2007). Effect of numerical dissipation on the predicted spectra for compressible turbulence. Annual Research Briefs, pages 47–57.

Lesaint, P. and Raviart, P.-A. (1974). On a finite element method for solving the neutron transport equation. Publications mathématiques et informatique de Rennes, 32(S4):1–40.

LeVeque, R. J. (1992). Numerical methods for conservation laws, volume 132. Springer.

Li, R. and Saad, Y. (2013). GPU-accelerated preconditioned iterative linear solvers. The Journal of Supercomputing, 63(2):443–466.

Lian, C., Xia, G., and Merkle, C. L. (2009). Solution-limited time stepping to enhance reliability in CFD applications. Journal of Computational Physics, 228:4836–4857.

Liepmann, H. W. and Roshko, A. (2001). Elements of gasdynamics. Courier Corporation.

Lo, S.-C., Blaisdell, G., and Lyrintzis, A. (2010). High-order shock capturing schemes for turbulence calculations. International Journal for Numerical Methods in Fluids, 62(5):473–498.

Lomtev, I., Quillen, C., and Karniadakis, G. (1998). Spectral/hp methods for viscous compressible flows on unstructured 2D meshes. Journal of Computational Physics, 144(2):325–357.

Loppi, N., Witherden, F., Jameson, A., and Vincent, P. (2019). Locally adaptive pseudo-time stepping for high-order flux reconstruction. Journal of Computational Physics, 399:108913.

Luo, H., Baum, J. D., and Löhner, R. (2006). A p-multigrid discontinuous Galerkin method for the Euler equations on unstructured grids. Journal of Computational Physics, 211(2):767–783.

Magolu monga Made, M., Beauwens, R., and Warzée, G. (2000). Preconditioning of discrete Helmholtz operators perturbed by a diagonal complex matrix. Communications in Numerical Methods in Engineering, 16(11):801–817.

Mani, A., Larsson, J., and Moin, P. (2009). Suitability of artificial bulk viscosity for large-eddy simulation of turbulent flows with shocks. Journal of Computational Physics, 228(19):7368–7374.

Marty, J., Lantos, N., Michel, B., and Bonneau, V. (2015). LES and hybrid RANS/LES simulations of turbomachinery flows using high order methods. In Turbo Expo: Power for Land, Sea, and Air, volume 56659, page V02CT44A003. American Society of Mechanical Engineers.

Masatsuka, K. (2013). I do Like CFD, vol. 1, volume 1. Lulu.com.

Mavriplis, D. J. (1998). Multigrid strategies for viscous flow solvers on anisotropic unstructured meshes. Journal of Computational Physics, 145(1):141–165.

Mazaheri, K. and Roe, P. L. (2003). Bulk viscosity damping for accelerating convergence of low Mach number Euler solvers. International journal for numerical methods in fluids, 41(6):633–652.

Meisrimel, P. and Birken, P. (2020). Goal oriented time adaptivity using local error estimates. Algorithms, 13(5):113.

Melson, N. D., Atkins, H. L., and Sanetrik, M. D. (1993). Time-accurate Navier-Stokes calculations with multigrid acceleration. In The Sixth Copper Mountain Conference on Multigrid Methods, Part 2.

Mengaldo, G. (2015). Discontinuous spectral/hp element methods: development, analysis and applications to compressible flows. PhD thesis, Imperial college London.

Moro, D., Nguyen, N. C., and Peraire, J. (2016). Dilation-based shock capturing for high-order methods. International Journal for Numerical Methods in Fluids, 82(7):398–416.

Nachtigal, N. M., Reddy, S. C., and Trefethen, L. N. (1992). How fast are nonsymmetric matrix iterations? SIAM Journal on Matrix Analysis and Applications, 13(3):778–795.

Nguyen, C. and Peraire, J. (2011). An adaptive shock-capturing HDG method for compressible flows. In 20th AIAA Computational Fluid Dynamics Conference, page 3060.

Nguyen, N. C., Peraire, J., and Cockburn, B. (2009). An implicit high-order hybridizable discontinuous Galerkin method for nonlinear convection–diffusion equations. Journal of Computational Physics, 228(23):8841–8855.

Nigro, A., Ghidoni, A., Rebay, S., and Bassi, F. (2014). Modified extended BDF scheme for the discontinuous Galerkin solution of unsteady compressible flows. International Journal for Numerical Methods in Fluids, 76(9):549–574.

Noventa, G., Massa, F., Bassi, F., Colombo, A., Franchina, N., and Ghidoni, A. (2016). A high-order discontinuous Galerkin solver for unsteady incompressible turbulent flows. Computers & Fluids, 139:248–260.

Noventa, G., Massa, F., Rebay, S., Bassi, F., and Ghidoni, A. (2020). Robustness and efficiency of an implicit time-adaptive discontinuous Galerkin solver for unsteady flows. Computers & Fluids, 204:104529.

Oberkampf, W. L. and Roy, C. J. (2010). Verification and validation in scientific computing. Cambridge University Press.

Olson, B. J. and Lele, S. K. (2013). Directional artificial fluid properties for compressible large-eddy simulation. Journal of Computational Physics, 246:207–220.

Owren, B. and Zennaro, M. (1992). Derivation of efficient, continuous, explicit Runge–Kutta methods. SIAM Journal on Scientific and Statistical Computing, 13(6):1488–1501.

Pan, Y., Yan, Z.-G., Peiró, J., and Sherwin, S. J. (2021). Development of a balanced adaptive time-stepping strategy based on an implicit JFNK-DG compressible flow solver. Communications on Applied Mathematics and Computation, pages 1–30.

Parnaudeau, P., Carlier, J., Heitz, D., and Lamballais, E. (2008). Experimental and numerical studies of the flow over a circular cylinder at Reynolds number 3900. Physics of Fluids, 20(8):085101.

Patera, A. T. (1984). A spectral element method for fluid dynamics: laminar flow in a channel expansion. Journal of Computational Physics, 54(3):468–488.

Peraire, J. and Persson, P.-O. (2008). The compact discontinuous Galerkin (CDG) method for elliptic problems. SIAM Journal on Scientific Computing, 30(4):1806–1824.

Persson, P.-O. (2009). Scalable parallel Newton-Krylov solvers for discontinuous Galerkin discretizations. In 47th AIAA Aerospace Sciences Meeting including The New Horizons Forum and Aerospace Exposition, page 606.

Persson, P.-O. and Peraire, J. (2006a). An efficient low memory implicit DG algorithm for time dependent problems. In 44th AIAA Aerospace Sciences Meeting and Exhibit, page 113.

Persson, P.-O. and Peraire, J. (2006b). Sub-cell shock capturing for discontinuous Galerkin methods. In 44th AIAA Aerospace Sciences Meeting and Exhibit.

Persson, P.-O. and Peraire, J. (2008). Newton-GMRES preconditioning for discontinuous Galerkin discretizations of the Navier-Stokes equations. SIAM Journal on Scientific Computing, 30(6):2709–2733.

Peterson, J. W., Lindsay, A. D., and Kong, F. (2018). Overview of the incompressible Navier–Stokes simulation capabilities in the MOOSE framework. Advances in Engineering Software, 119:68–92.

Prothero, A. and Robinson, A. (1974). On the stability and accuracy of one-step methods for solving stiff systems of ordinary differential equations. Mathematics of Computation, 28(125):145–162.

Qiu, W. and Shi, K. (2016). An HDG method for convection diffusion equation. Journal of Scientific Computing, 66(1):346–357.

Ramshaw, J. and Mousseau, V. (1990). Accelerated artificial compressibility method for steady-state incompressible flow calculations. Computers & Fluids, 18(4):361–367.

Rathan, S. and Naga Raju, G. (2018). A modified fifth-order WENO scheme for hyperbolic conservation laws. Computers & Mathematics with Applications, 75(5):1531–1549.

Rawat, P. and Zhong, X. (2011). Direct numerical simulations of turbulent flow interactions with strong shocks using shock-fitting method. In 49th AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition, page 649.

Reed, W. H. and Hill, T. R. (1973). Triangular mesh methods for the neutron transport equation. Technical report, Los Alamos Scientific Lab., N. Mex.(USA).

Rinaldi, E., Pecnik, R., and Colonna, P. (2014). Exact Jacobians for implicit Navier–Stokes simulations of equilibrium real gas flows. Journal of Computational Physics, 270:459–477.

Rivière, B., Wheeler, M. F., and Girault, V. (1999). Improved energy estimates for interior penalty, constrained and discontinuous Galerkin methods for elliptic problems. part i. Computational Geosciences, 3(3):337–360.

Rivière, B., Wheeler, M. F., and Girault, V. (2001). A priori error estimates for finite element methods based on discontinuous approximation spaces for elliptic problems. SIAM Journal on Numerical Analysis, 39(3):902–931.

Rivire, B. (2008). Discontinuous Galerkin methods for solving elliptic and parabolic Equations: Theory and implementation. SIAM.

Roy, C. J. (2005). Review of code and solution verification procedures for computational simulation. Journal of Computational Physics, 205(1):131–156.

Saad, Y. (2003). Iterative methods for sparse linear systems. SIAM.

Saad, Y. and Schultz, M. H. (1986). Gmres: A generalized minimal residual algorithm for solving nonsymmetric linear systems. SIAM Journal on Scientific and Statistical Computing, 7(3):856–869.

Sato, Y., Hino, T., and Ohashi, K. (2013). Parallelization of an unstructured Navier–Stokes solver using a multi-color ordering method for OpenMP. Computers & Fluids, 88:496–509.

Schulz-Rinne, C. W., Collins, J. P., and Glaz, H. M. (1993). Numerical solution of the Riemann problem for two dimensional gas dynamics. SIAM Journal on Scientific Computing, 14(6):1394–1414.

Sevilla, R. and Huerta, A. (2018). HDG-NEFEM with degree adaptivity for Stokes flows. Journal of Scientific Computing, 77(3):1953–1980.

Shampine, L. F. (2005). Error estimation and control for ODEs. Journal of Scientific Computing, 25(1).

Slotnick, J., Khodadoust, A., Alonso, J., Darmofal, D., Gropp, W., Lurie, E., and Mavriplis, D. (2014). CFD vision 2030 study: a path to revolutionary computational aerosciences. NASA/CR, 1(218):178.

Smith, B. F., Bjorstad, P. E., Gropp, W. D., and Pasciak, J. E. (1998). Domain decomposition: Parallel multilevel methods for elliptic partial differential equations. SIAM Review, 40(1):169–170.

Söderlind, G. (2002). Automatic control and adaptive time-stepping. Numerical Algorithms, 31(1):281–310.

Sóderlind, G. (2002). Automatic control and adaptive time-stepping. Numerical Algorithms, 31(1):281–310.

Sóderlind, G. and Wang, L. (2006). Adaptive time-stepping and computational stability. Journal of Computational and Applied Mathematics, 185(2):225–243.

Sun, S. (2003). Discontinuous Galerkin methods for reactive transport in porous media. The University of Texas at Austin.

Titley-Peloquin, D., Pestana, J., and Wathen, A. J. (2014). GMRES convergence bounds that depend on the right-hand-side vector. IMA Journal of Numerical Analysis, 34(2):462–479.

Tonicello, N., Lodato, G., and Vervisch, L. (2020). Entropy preserving low dissipative shock capturing with wave-characteristic based sensor for high-order methods. Computers & Fluids, 197:104357.

Toro, E. F. (2009). Riemann solvers and numerical methods for fluid dynamics: A practical introduction. Springer, 3rd edition.

Toselli, A. and Widlund, O. B. (2005). Domain decomposition methods–algorithms and theory. Springer.

Trefethen, L. N. and Bau III, D. (1997). Numerical linear algebra, volume 50. SIAM.

Trefethen, L. N. and Embree, M. (2005). Spectra and pseudospectra: the behavior of nonnormal matrices and operators. Princeton University Press.

Trottenberg, U., Oosterlee, C. W., and Schuller, A. (2000). Multigrid. Elsevier.

Turner, K. and Walker, H. (1992). Efficient high accuracy solutions with GMRES(m). SIAM Journal on Scientific and Statistical Computing, 13(3):815–825.

Tyacke, J., Vadlamani, N., Trojak, W., Watson, R., Ma, Y., and Tucker, P. (2019). Turbomachinery simulation challenges and the future. Progress in Aerospace Sciences, 110:100554.

Ur Rehman, M., Vuik, C., and Segal, G. (2008). Preconditioners for the steady incompressible Navier-Stokes problem. International Journal of Applied Mathematics, 38(4):1–10.

van Buuren René (1999). Time integration methods for compressible flow. Computer Science.

Van Leer, B., Lo, M., and van Raalte, M. (2007). A discontinuous Galerkin method for diffusion based on recovery. In 18th AIAA computational fluid dynamics conference, page 4083.

Vanden, K. J. and Orkwis, P. D. (1996). Comparison of numerical and analytical Jacobians. AIAA Journal, 34(6):1125–1129.

Vandenhoeck, R. and Lani, A. (2019). Implicit high-order flux reconstruction solver for high-speed compressible flows. Computer Physics Communications, 242:1–24.

Vanderstraeten, D. (2001). An expert system to control the CFL number of implicit upwind methods. In Toro, E. F., editor, Godunov Methods: Theory and Applications, pages 977–984. Springer US.

VonNeumann, J. and Richtmyer, R. D. (1950). A method for the numerical calculation of hydrodynamic shocks. Journal of Applied Physics, 21(3):232–237.

Walker, H. F. (1988). Implementation of the GMRES method using Householder transformations. SIAM Journal on Scientific and Statistical Computing, 9(1):152–163.

Wang, Z. and Rahmani, S. (2021). Implicit large eddy simulation of the NASA CRM high-lift configuration near stall. Computers & Fluids, 220:104887.

Wang, Z. J., Fidkowski, K., Abgrall, R., Bassi, F., Caraeni, D., Cary, A., Deconinck, H., Hartmann, R., Hillewaert, K., Huynh, H. T., et al. (2013). High-order CFD methods: current status and perspective. International Journal for Numerical Methods in Fluids, 72(8):811–845.

Wathen, A. J. (2015). Preconditioning. Acta Numerica, 24:329–376.

Wesseling, P. (1995). Introduction to multigrid methods. Technical report, CASE HAMPTON VA.

Witherden, F. D., Farrington, A. M., and Vincent, P. E. (2014). PyFR: An open source framework for solving advection–diffusion type problems on streaming architectures using the flux reconstruction approach. Computer Physics Communications, 185(11):3028–3040.

Yan, Z.-G., Pan, Y., Castiglioni, G., Hillewaert, K., Peiró, J., Moxey, D., and Sherwin, S. J. (2020). Nektar++: design and implementation of an implicit, spectral/hp element, compressible flow solver using a Jacobian-free Newton Krylov approach. Computers & Mathematics with Applications, page S0898122120301073.

Yang, U. M. (2006). Parallel algebraic multigrid methods high performance preconditioners. In Bruaset, A. M. and Tveito, A., editors, Numerical Solution of Partial Differential Equations on Parallel Computers, Lecture Notes in Computational Science and Engineering, pages 209–236, Berlin, Heidelberg. Springer.

Yildirim, A., Kenway, G. K. W., Mader, C. A., and Martins, J. R. R. A. (2019). A Jacobian-free approximate Newton-Krylov startup strategy for RANS simulations. Journal of Computational Physics, 397:108741.

Yu, J. and Hesthaven, J. S. (2017). A comparative study of shock capturing models for the discontinuous Galerkin method. Technical report, Elsevier.

Yu, J., Yan, C., and Jiang, Z. (2018). Revisit of dilation-based shock capturing for

discontinuous Galerkin methods. Applied Mathematics and Mechanics, 39(3):379–394.

Zaporozhets, O., Tokarev, V., and Attenborough, K. (2019). Aircraft Noise: Assessment, prediction and control. CRC Press.

Zhang, T. and Zheng, Y. X. (1990). Conjecture on the structure of solutions of the Riemann problem for two-dimensional gas dynamics systems. SIAM Journal on Mathematical Analysis, 21(3):593–630.

Appendix A: GMRES algorithm

The algorithm of GMRES(M) is as follows

$\mathbf{r} = \mathbf{b} - \mathbf{A}\mathbf{x}_0$ for a given initial guess $\mathbf{x}_0$

Step 1: Arnoldi process to calculate orthogonal vectors

for $i = 1, 2, \ldots$ **do**

$\mathbf{w}_0 = \mathbf{r}/\|\mathbf{r}\|$

for $j = 1, 2, \ldots, M$ **do**

$\mathbf{v} = \mathbf{A}\mathbf{w}_i$

for $k = 1, 2, \ldots, j$ **do**

$h_{k,j} = \mathbf{w}_k^T \mathbf{v}$, $\mathbf{v} = \mathbf{v} - h_{k,j}\mathbf{w}_k$

end for

$h_{j+1,j} = \|\mathbf{v}\|$, $\mathbf{w}_{j+1} = \mathbf{v}/h_{j+1,j}$

Step 2: QR factorization to seek for y to minimize residual

$r_{1,j} = h_{1,j}$

for $k = 2, 3, \ldots, j$ **do**

$\gamma = c_{k-1} r_{k-1,j} + s_{k-1} h_{k,j}$

$r_{k,j} = -s_{k-1} r_{k-1,j} + c_{k-1} h_{k,j}$

$r_{k-1,j} = \gamma$

end for

$\delta = \sqrt{r_{j,j}^2 + h_{j+1,j}^2}$, $c_j = r_{j,j}/\delta$, $s_j = h_{j+1,j}/\delta$

$r_{j,j} = c_j r_{j,j} + s_j h_{j+1,j}$

$\hat{b}_{j+1} = -s_j \hat{b}_j$, $\hat{b}_j = c_j \hat{b}_j$

$\eta = \|\hat{b}_{j+1}\|$

if $\eta \leq \tau_{\text{GMRES}}$ **then**

$N = j$, converged

else

$N = M$

end if

end for

Step 3: backward method to calculate y

$y_N = \hat{b}_N / r_{N,N}$

for $k = N-1, \ldots, 1$ **do**

$y_k = (\hat{b}_k - \sum_{j=k+1}^{N} r_{k,j} y_j) / r_{k,k}$

end for

$\mathbf{x}_{\text{new}} = \mathbf{x}_0 + \sum_{j=1}^{N} y_j \mathbf{w}_j$, $\mathbf{r} = \mathbf{b} - \mathbf{A}\mathbf{x}_{\text{new}}$

If $\eta \leq \tau_{\text{GMRES}}$, **Quit**

end for

Appendix B: Butcher array of ESDIRK

The coefficients of ESDIRK schemes adopted in this paper can be expressed in the form of the general Butcher tableau shown in Table. 6.1 Kvrn (2004), where S and $R = S + 1$ are the number of stages for the main scheme and the embedded scheme. The coefficients of the matrix A are sufficient to define the whole Butcher tableau since $c_i = \sum_{j=1}^{S+1} a_{ij}$, $b_i = a_{Si}$, and $\hat{b}_i = a_{Ri}$. The last two relations indicate that the main and the embedded schemes are both stiffly accurate. The diagonal coefficients are given by $a_{11} = 0$ and $a_{ii} = \alpha; i = 2, \ldots, S + 1$. The coefficients of the matrix A for the ESDIRK2, ESDIRK3 and ESDIRK4 schemes are presented in Tab. 6.4, Tab. 6.2 and Tab. 6.3, respectively.

$$
\begin{array}{c|c}
c & A \\
\hline
 & b^T \\
 & \hat{b}^T
\end{array}
=
\begin{array}{c|cccccc}
0 & 0 & 0 & \ldots & \ldots & \ldots & 0 \\
c_2 & a_{21} & \alpha & 0 & \ldots & \ldots & 0 \\
c_3 & a_{31} & a_{32} & \alpha & 0 & \ldots & 0 \\
\vdots & \vdots & \vdots & & \ddots & \ddots & \vdots \\
c_S & a_{S1} & a_{S2} & \ldots & \ldots & \alpha & 0 \\
c_R & a_{R1} & a_{R2} & \ldots & \ldots & \ldots & \alpha \\
\hline
 & b_1 & \ldots & \ldots & \ldots & b_S & 0 \\
 & \hat{b}_1 & \ldots & \ldots & \ldots & \ldots & \hat{b}_R
\end{array}
$$

Table 6.1: General form of the Butcher tableau of the embedded ESDIRK. Here R is shorthand for $R = S + 1$.

A =	0	0	0	0
	0.292893218800000	0.2928932188000000	0	0
	0.353553390567523	0.353553390632477	0.292893218800000	0
	0.215482203122508	0.686886723913539	-0.195262145836047	0.292893218800000

Table 6.2: Butcher array of embedded ESDIRK3 with $S + 1 = 5$.

	0	0	0	0	0
	0.43586652150846	0.43586652150846	0	0	0
A =	0.14073777472471	-0.10836555138132	0.43586652150846	0	0
	0.10239940061991	-0.37687845225556	0.83861253012719	0.43586652150846	0
	0.15702489786032	0.11733044137044	0.61667803039212	-0.32689989113134	0.43586652150846

Table 6.3: Butcher array of embedded ESDIRK4 with $S + 1 = 7$.

	0	0	0	0	0	0	0
	0.27000000000000000	0.27000000000000000	0	0	0	0	0
	0.13500000000000000	0.87265371804359686	0.27000000000000000	0	0	0	0
A=	0.24814211234447322	0.13282088522859322	-0.03886686658917771	0.27000000000000000	0	0	0
	0.25494479822150471	0.13106196422347200	-0.04522093930235708	0.03389121682051642	0.27000000000000000	0	0
	0.17549975523182941	0	-0.01641725931492383	3.59357175290010625	-3.02265424881701182	0.27000000000000000	0
	0.15847612643670410	0	-0.07384703732094983	5.26056776397634893	-4.83946947758407500	0.22427262449197180	0.270

Table 6.4: Butcher array of embedded ESDIRK2 with $S + 1 = 4$.

Appendix C: Lid-driven flow ($\mathbf{Re} = 100$)

The converged density and velocity distributions of Re = 100 are shown as Fig. 6.1 and Fig. 6.2, which are extracted when the residual $\mathcal{L}_\delta(\mathbf{U})$ is reduced by a factor of 10^{-9}.

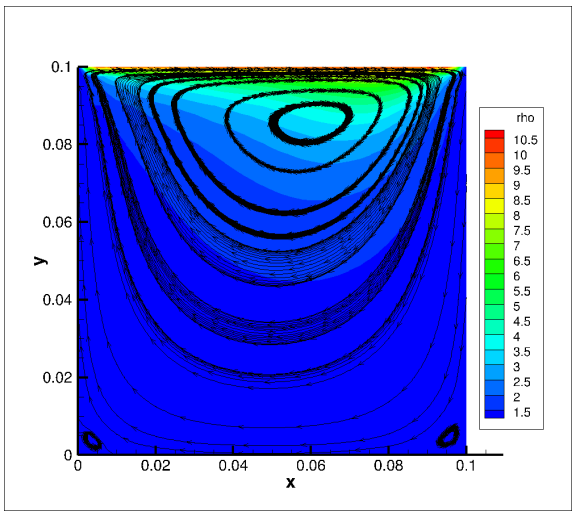


Figure 6.1: Lid-driven cavity flow (Re = 100): Density distribution.

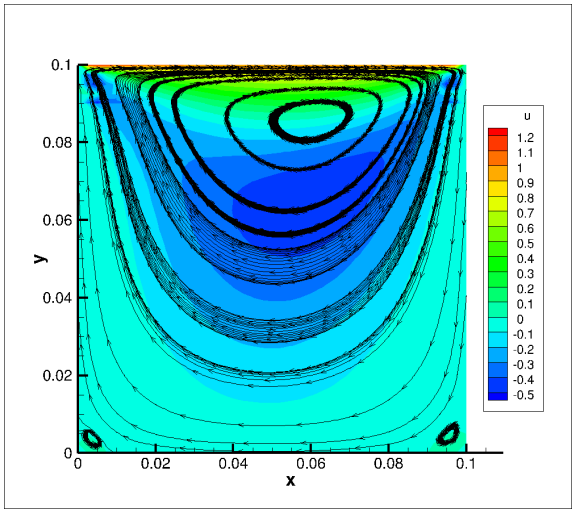


Figure 6.2: Lid-driven cavity flow (Re = 100): Velocity distribution.

This the eigenvalue analysis is bases on the converged field and then run extra one time step $\Delta t = 0.0002$ using the backward Euler time-integration scheme and $P = 4$ order DG scheme. The eigenvalue distributions of the Jacobian matrix before preconditioning and after different methods of preconditioning are shown as Fig. 6.3 and Fig. 6.4.

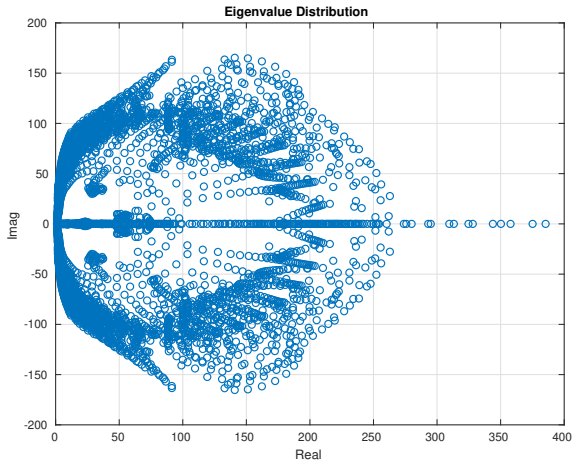


Figure 6.3: Lid-driven cavity flow (Re = 100):
Eigenvalue distributions of unpreconditioned Jacobian matrix.

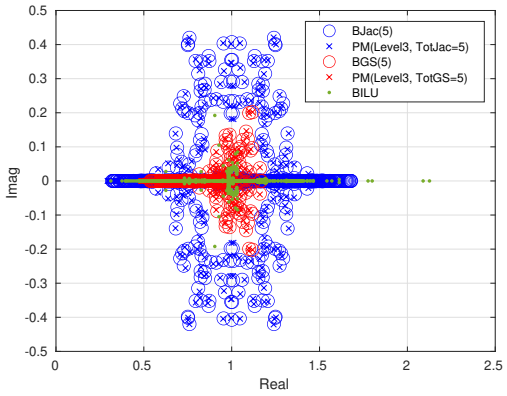


Figure 6.4: Lid-driven cavity flow (Re = 100):
Eigenvalue distributions comparisons of different preconditioned Jacobian matrix.

Table 6.5 lists the total GMRES within one time step and the condition number after different types of preconditioning.

Preconditioner type	GMRES iteration number	Condition number
Unpreconditioned	×	2.83×10^{12}
BJac(3)	265	2.57×10^{9}
BJac(5)	187	1.78×10^{9}
PM-level2 (TotBJac=3)	296	2.93×10^{9}
PM-level3 (TotBJac=5)	206	1.81×10^{9}
BGS(3)	145	2.18×10^{9}
BGS(5)	109	8.71×10^{8}
PM-level2 (TotBGS=3)	167	2.22×10^{9}
PM-level3 (TotBGS=5)	119	8.88×10^{8}
BILU	147	2.49×10^{8}

Table 6.5: Lid-driven flow(Re=100):
Comparison of performance of preconditioned Jacobian matrices
(PM denotes p-multigrid).