

Imperial College London
Department of Aeronautics

Efficient implicit spectral/*hp* element DG techniques for compressible flows

Zhen-Guo Yan

Submitted in part fulfilment of the requirements for the
degree of Doctor of Philosophy at Imperial College London,
October 2021

Abstract

In the simulation of stiff problems, such as fluid flows at high Reynolds numbers, the efficiency of explicit time integration is significantly limited by the need to use very small time steps. To alleviate this limitation and to accelerate compressible flow simulations based on high-order spectral/*hp* element methods, an implicit time integration method is developed using singly diagonally implicit Runge-Kutta temporal discretization schemes combined with a Jacobian-free Newton Krylov (JFNK) method. This thesis studies several topics influencing the efficiency, accuracy and robustness of the solver.

Firstly, an efficient partially matrix-free block relaxed Jacobi (BRJ) preconditioner is proposed, in which the Jacobian matrix and preconditioning matrices are properly approximated based on studies of their influences on convergence. The preconditioner only forms and stores the diagonal part of the Jacobian matrix while the off-diagonal operators are calculated on the fly. Used together with techniques such as using single precision data, the BRJ can largely reduce the memory consumption when compared with matrix-based ones like incomplete LU factorization preconditioners (ILU). To further accelerate the solver, influences of different parts of the flux Jacobian on the preconditioning effects are studied and terms with minor influences are neglected. This reduces the computational cost of the BRJ preconditioner by about 3 times while maintaining similar preconditioning effects.

Secondly, adaptive strategies for a suitable choice of some free parameters are designed to maintain temporal accuracy and relatively high efficiency. The several free parameters in the implicit solver have significant influences on the accuracy, efficiency and stability. Therefore, designing proper strategies in choosing them is essential for developing a robust general purpose solver. Based on the idea of constructing proper relations between the temporal, spatial and iterative errors, adaptive strategies are designed for determining the time step and Newton tolerance. These parameters maintain temporal accuracy of the solver in the sense that further

decreasing temporal and iterative errors will not obviously improve the efficiency. Meanwhile, they maintain relatively efficient by avoiding excessively small time step and Newton tolerance. The strategies are tested in different types of cases to illustrate their performance and generality.

Finally, the implicit solver is studied in high-fidelity simulations of turbulent flows based on a hierarchical implementation in the open-source spectral/*hp* element framework Nektar++. The solver is applied to large-eddy simulations of Taylor-Green vortex flow, turbulent channel flow and flow over a circular cylinder cases. The efficiency of the solver and the prediction accuracy of these problems are studied. The results show that the solver yields good predictions in turbulence simulations whilst keeping good stability and high efficiency.

Acknowledgements

I want to thank my supervisors, Profs. Spencer Sherwin and Joaquim Peiró, who gave me this great chance studying CFD and high-order numerical methods. They have not only taught me knowledge in this area but also scientific ways in analyzing and studying new problems. I have been deeply impressed by their profound knowledge and strict attitude to scientific researches. They have been supportive both for my PhD research project and my life in London.

Special thanks goes to my family for their support. My wife, Huajun, has shouldered a lot while I am away from home. Also thanks my parents for raising me up, educating me and supporting me. The support and company from my parents, Huajun and our two kids have been an unforgettable and irreplaceable part of my life.

I thank the financial support and care from my employer in China. I am also grateful to the mentors, colleagues and collaborators in my previous education and research experiences, who have taught me basic knowledge and research abilities which are foundations of my current research project. Special thanks to Prof. Xiaogang Deng, Prof. Meiliang Mao, Prof. Jianqiang Chen, Prof. Laiping Zhang, Prof. Xianxu Yuan, Prof. Qin Li, Dr. Huayong Liu and everyone who have supported and helped.

My gratitude goes also to 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 software design and technical details. Special thanks goes to Yu Pan, whose collaborations have helped greatly. This work would not have been possible without their kind help.

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

Dedication

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.

Zhen-Guo Yan (2021)

The copyright of this thesis rests with the author. Unless otherwise indicated, its contents are licensed under a Creative Commons Attribution-Non Commercial 4.0 International Licence (CC BY-NC).

Under this licence, you may copy and redistribute the material in any medium or format. You may also create and distribute modified versions of the work. This is on the condition that: you credit the author and do not use it, or any derivative works, for a commercial purpose.

When reusing or sharing this work, ensure you make the licence terms clear to others by naming the licence and linking to the licence text. Where a work has been adapted, you should indicate that the work has been changed and describe those changes.

Please seek permission from the copyright holder for uses of this work that are not included in this licence or permitted under UK Copyright Law.

Contents

Abstract	i
Acknowledgements	iii
1 Introduction	1
1.1 The need for implicit schemes	1
1.2 Literature review	3
1.2.1 High-order DG methods	3
1.2.2 Implicit time integration methods	7
1.3 Objectives	20
1.4 Outline	21
2 Governing equations and numerical methods	22
2.1 Compressible Navier-Stokes equations	22
2.2 Spatial discretization	24
2.2.1 Discontinuous Galerkin formulation	24

2.2.2	Symmetric interior penalty Galerkin method	27
2.2.3	Shock-capturing method	29
2.3	Implicit time discretization schemes	30
2.3.1	Diagonally implicit Runge-Kutta schemes	31
2.3.2	Discrete formulation	33
2.3.3	Newton-Krylov method	34
2.4	Discretization error estimate	36
3	Approximating Jacobian and preconditioning matrices	38
3.1	Matrix-free Jacobian matrix approximations	38
3.1.1	Jacobian-free method	39
3.1.2	Jacobian approximation errors	40
3.2	Constructing efficient preconditioners	44
3.2.1	Block relaxed Jacobi iterative (BRJ) preconditioner	45
3.2.2	Approximations of matrices in preconditioning	46
4	Implementation and verification	54
4.1	Software design in Nektar++	54
4.1.1	Introduction to Nektar++	55
4.1.2	Implicit solver architecture and implementation	58
4.2	Performance analysis	61
4.2.1	Performance based on different linear algebra libraries	62

4.2.2	Performance for different orders of accuracy	65
4.2.3	Scalability performance	70
4.3	Memory consumption analysis	73
4.4	Verification of the implicit solver	74
4.4.1	Isentropic vortex convection	74
4.4.2	Manufactured compressible Poiseuille flow	77
4.4.3	Laminar flow over a circular cylinder	80
4.4.4	Shock boundary layer interaction	81
5	Robust solver parameter choice	85
5.1	Balanced adaptive time-stepping	86
5.2	Adaptive convergence tolerances	90
5.3	Summary of the adaptive strategy and discussion	92
5.4	Numerical tests of the parameter choices	96
5.4.1	Isentropic vortex	96
5.4.2	Flat plate boundary layer	103
5.4.3	Taylor-Green vortex	105
5.4.4	Turbulent flow over a circular cylinder at $Re = 3\,900$	110
5.4.5	Summary of parameters	113

6	Application to high-fidelity simulations	116
6.1	Stabilization techniques	117
6.2	Taylor-Green vortex	120
6.3	Flow over a circular cylinder at $Re = 3900$	121
6.4	Turbulent channel flow	124
6.5	Summary	128
7	Conclusions and future work	132
7.1	Concluding remarks	132
7.2	Future work	135
7.3	List of publications and conference presentations	136
A	Butcher Tableau of ESDIRK schemes	139
B	Forcing terms of manufactured compressible Poiseuille flow	142
	Bibliography	142

List of Tables

3.1	The number of iteration for the GMRES method to converge by 5 orders of magnitudes with different time steps and different preconditioners.	49
4.1	Comparison of computational time with different BLAS libraries. JF represents the Jacobian-free operation in the GMRES. PREC-DIAG represents the $\hat{\mathbf{D}}^{-1}$ vector inner product operation. PREC-OFFDIAG represents the $\hat{\mathbf{L}} + \hat{\mathbf{U}}$ vector inner product operator. VOL, BND and INV present the element volume, element boundary and matrix inversion operations in calculating $\hat{\mathbf{D}}^{-1}$	64
4.2	Iteration information of 3D and 2D cylinder cases. The Newton iteration is the averaged number per implicit stage. The GMRES iteration is the averaged number per Newton iteration. The computational time is the average time per time step.	65
4.3	Memory consumption of BRJ in bytes for each hexahedral mesh. $\mathbf{BRJ}_D$, $\mathbf{BRJ}_O$ and $\mathbf{BRJ}_T$ represent the storage for $\hat{\mathbf{D}}^{-1}$, $\partial \hat{\mathbf{H}}^n / \partial \hat{\mathbf{Q}}^\pm$ and their sum. $\mathbf{Jac}_S$ and $\mathbf{Jac}_D$ represent the storage for the Jacobian matrix considering the sparsity patterns and not. The superscript D represent the storage is for double precision data while all others without the superscript represent the storage for single precision data.	74

4.4	Efficiency comparison between explicit and implicit time integration schemes in the laminar cylinder case. All simulations are conducted using one HPC node as discussed in Section 4.2.	82
4.5	Efficiency comparison between explicit and implicit time integration schemes of shock wave boundary-layer interaction.	82
5.1	Parameters of the implicit solver. A , E and S represent that the parameter has influence on the accuracy, efficiency and stability of the implicit solver, respectively.	86
5.2	Isentropic vortex problem: Comparison of the adaptive Newton tolerance and the maximum Newton tolerance that maintains temporal accuracy.	98
5.3	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.	111
5.4	Summary of tunable parameters in the adaptive strategy.	114
5.5	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.	114
6.1	Efficiency comparison between explicit and implicit time integration schemes of TGV. All simulations are conducted using four HPC nodes, the information of which is discussed in Section 4.2.	121

6.2	Efficiency comparison between explicit and implicit time integration schemes of turbulent circular cylinder at $Re = 3900$. All simulations are conducted using four HPC nodes, the information of which is discussed in Section 4.2.	124
6.3	Mesheres of turbulent channel flow simulations.	128
6.4	Summary of computational efficiency. All tests are ran using 48 cores. The CPU time is the averaged time for one time step. The speed-up in the last column is relative to the explicit solver with second-order Runge-Kutta scheme.	131
A.1	General form of the Butcher tableau of the embedded ESDIRK. Here R is shorthand for $R = S + 1$	140
A.2	Butcher array of embedded ESDIRK2 with $S + 1 = 4$	141
A.3	Butcher array of embedded ESDIRK3 with $S + 1 = 5$	141
A.4	Butcher array of embedded ESDIRK4 with $S + 1 = 7$	141

List of Figures

3.1	Influence of finite difference Jacobian approximation errors with different condition numbers. The system is generated using random numbers. The <i>Averaged</i> , <i>Maximum</i> , <i>Minimum</i> indicate the algebraic, maximum and minimum errors of 200 different random linear systems to reduce the specificity of the system.	43
3.2	Influence of finite difference Jacobian approximation errors with different time steps. The <i>Averaged</i> , <i>Maximum</i> , <i>Minimum</i> indicate the algebraic, maximum and minimum errors of 200 different random linear systems to reduce the specificity of the system.	43
3.3	Eigenspectra distributions based on different approximations of the preconditioning matrices. <i>NoPrec</i> represents the one without preconditioning, <i>BRJ(3)</i> represents the one preconditioned by the BRJ with $J = 3$, the suffixes <i>Diff</i> and <i>Adv</i> represents the matrix BRJ constructed using only the advection and diffusion flux Jacobian. . . .	48
3.4	Performance comparison of the BRJ preconditioner with different iteration numbers and implementations.	53

4.1	Class structure of the explicit and implicit solvers. The equation system classes (<code>EulerCFE</code> or <code>NavierStokesCFE</code>) contain access to the main functionalities of the solver, such as time integration, and the solution fields. They make use of numerical methods from the libraries, such as <code>AdvectionWeakDG</code> , and equations system related functions, such as the advection flux (<code>F</code>) and diffusion flux (<code>G</code>), to form the spatial discretization operator <code>A</code> . The spatial operator <code>A</code> and the time integration method form the explicit solver using the method of lines. For the implicit solver, additional classes like the Newton solver (<code>NewtonSolver</code>), GMRES solver (<code>GMRES</code>) and linear algebra solver of preconditioners (e.g. <code>PrecondBRJ</code>) are instantiated. Together with operators related to the nonlinear system (<code>C</code>), the Jacobian matrix (<code>D</code>) and preconditioning matrix (<code>E</code>), an implicit system solver (<code>B</code>) is constructed, which is linked to the implicit time integration scheme to form the implicit solver.	57
4.2	Nassi–Shneiderman diagram of the implicit solver with the corresponding class names. Brown: library class ; cyan: solver class.	61
4.3	Performance analysis of the implicit solver in a 3D cylinder case with polynomial orders of $P = 2, 3, 4$	66
4.4	Performance analysis of the implicit solver in a 2D cylinder case with polynomial orders of $P = 2, 3, 4$. The meanings of different operations can be found in Fig. 4.3.	69
4.5	Strong parallel scalability performance of the implicit solver for polynomial orders $P = 1, 2, 3$	72
4.6	Isentropic vortex problem: Mesh and density distribution.	76

4.7	Convergence order of accuracy (OoA) tests of the advection scheme with polynomial orders $P = 1, 2, \dots, 5$. The L2 error norms at $t=0.01$ are calculated using the analytical solution as the reference. The OoA between the two finest mesh levels is given in the legend.	76
4.8	Non-dimensional total energy (E) distribution of the manufactured compressible Poiseuille flow.	78
4.9	Convergence order of accuracy (OoA) of the IP methods in compressible Poiseuille flow where the OoA between the two finest mesh levels is given in the legend.	78
4.10	Residual distribution of implicit and explicit simulations in compressible Poiseuille flow.	79
4.11	Convergence order of accuracy (OoA) of BDF1, BDF2, DIRK2, DIRK3 and DIRK4 implicit time integration schemes in the laminar cylinder case.	81
4.12	Mach number and skin friction coefficient distributions in shock wave boundary-layer interaction.	83
5.1	Isentropic vortex problem: Effect of Newton tolerances in Eqs. (5.18) and (5.9) on temporal accuracy.	98
5.2	Isentropic vortex problem: Effect of time-stepping method on total errors.	100
5.3	Isentropic vortex problem: Effect of time-stepping method on temporal errors and spatial errors.	100
5.4	Isentropic vortex problem: Effect of time-stepping method on efficiency.	101
5.5	Isentropic vortex problem: Performance of adaptive time-stepping for different ESDIRK schemes.	102

5.6	Boundary-layer flow past a flat plate: Geometry and boundary conditions.	104
5.7	Boundary-layer flow past a flat plate: mesh and velocity profile. . . .	104
5.8	Boundary-layer flow past a flat plate: Residual convergence history. The number in the parenthesis indicates the growth rate of the CFL number.	106
5.9	Boundary-layer flow past a flat plate: residual, CPU time and time step evolution curves as a function of steps. The number in the parenthesis indicates the growth rate of the CFL number.	107
5.10	Taylor-Green vortex: Evolution of the enstrophy.	108
5.11	Taylor-Green vortex: Error norms of the kinetic energy dissipation rate.	109
5.12	Turbulent flow over a circular cylinder: Q-criteria iso-surface ($Q = 5$) and mesh from reference [159].	111
5.13	Turbulent flow over a circular cylinder: Adaptive time steps (Δt) of ESDIRK2, ESDIRK3 and ESDIRK4.	111
5.14	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. .	112
5.15	Turbulent flow over a circular cylinder: Spatial distributions of the values of the elemental time steps.	113
6.1	Evolution of the dissipation rate in Taylor-Green vortex.	122
6.2	Taylor-Green vortex case: Comparison of contours of surface normal vorticity on the plane $y = \pi$ at $t = 8$ calculated using ERK2 (red dashed lines) and SDIRK2 with $\Delta t = 0.1$ (blue solid lines).	123

6.3	Non-dimensional mean u velocity of turbulent flow over a circular cylinder at $Re = 3900$	125
6.4	Mean streamwise and crossflow velocity distributions in the wake of the circular cylinder. Kravchenko-LES in [83], Ma-DNS in [94], and Parnaudeau-LES, Parnaudeau-Exp and Lourenco-Exp in [114]. . . .	126
6.5	Mean velocity fluctuations in the wake of the circular cylinder. . . .	127
6.6	Flow field, mean velocity and turbulent fluctuation distributions of turbulent channel flow.	129
6.7	Comparison of turbulent channel flow simulation results at $Re_\tau = 550$ for different polynomial orders and mesh resolutions.	130

Chapter 1

Introduction

Section 1.1 gives a brief discussion of the motivation for developing the implicit time integration solver. Section 1.2 presents the state-of-the-art literature review of major topics of the implicit solver including the development of DG methods themselves, the development of DG (discontinuous Galerkin) methods as tools for high-fidelity simulations, the development of implicit time integration methods and the challenges encountered in unsteady simulations. After that, the objective and main contributions of the thesis are summarized in Section 1.3 followed by the outlines of the thesis in Section 1.4.

1.1 The need for implicit schemes

There is a growing need for a better understanding of complex flow phenomena in areas such as turbo-machinery [67, 139], which includes noise generation mechanisms, flow interactions between different compressor stages and shock boundary-layer interactions. These phenomena are intrinsically unsteady, anisotropic and across

multiple scales [139, 22]. The prediction accuracy of traditional models like Reynolds averaged Navier-Stokes (RANS) models is generally not sufficient [71]. To fulfill the demand for enhanced accuracy, high-fidelity methods are developing rapidly in the computational fluid dynamics (CFD) community for the simulations of these phenomena. High-fidelity methods directly resolve the flow structures that are difficult to model [61]. This means more flow details need to be resolved directly, which poses new challenges to the numerical methods.

High-order numerical methods are good candidates for high-fidelity simulations because they are able to sustain smaller scale flow structures for longer [153, 67]. With lower dispersion and dissipation errors, it has been shown that high-order methods are more efficient than traditional low-order ones when very accurate simulations are required [153, 126, 67]. This thesis will focus on high order spectral/*hp* element discontinuous Galerkin methods. The compactness of the DG methods makes them easy to extend to high-order and applicable to unstructured meshes [77]. After decades of development, several DG methods with high-order accuracy, high resolution, and good shock-capturing ability have been developed, which make them very promising for industrial high-fidelity simulations [45, 67, 22].

However, further increasing the efficiency of high order DG methods is still highly needed considering that there are still many challenging areas out of reach because of high computational costs [154, 153, 22]. Therefore, improving efficiency is one of the key issues to be addressed before the DG methods can be widely used in industrial high-fidelity simulations [122, 6]. There are many challenging stiff problems involving high Reynolds numbers, in which very large grids across multiple scales are needed for their high-fidelity simulations [140]. One of the main obstacles limiting the efficiency of high-order simulations with explicit time integration methods is the severe time step restrictions imposed to maintain stability [123, 77]. In this case, developing

implicit time integration methods, which are effective in relaxing the time step restrictions, to accelerate the simulations has the potential to largely improve their efficiency.

1.2 Literature review

Section 1.2.1 reviews the state-of-the-art of high-order DG methods and their applications in high-fidelity simulations. Section 1.2.2 then summarizes the development of implicit time integration methods. Details about temporal discretization schemes, nonlinear system solvers, preconditioning methods, parameter choice strategies, developments in the open-source community and new challenges encountered are also discussed.

1.2.1 High-order DG methods

This section reviews the development of DG methods themselves and specifically the attempts to adjust DG methods in high fidelity simulations in this section.

Basic development of the DG methods

Spectral element DG methods [134, 77] use simplex elements to divide the computation domain and place multiple degrees of freedom (DoFs) in each element. They use the DoFs in each element to construct high-order polynomials and exchange only the flow variables and their derivatives at the element interfaces, which means that DG methods are very compact in design. This compactness gives the DG methods advantages in extending them to higher-order, in large scale parallelization

and in incorporating adaption based on mesh size (h) and spatial accuracy (p). Ever since their inception in 1973 [130], DG methods have achieved a significant degree of maturity. Cockburn et al. [34, 35] proposed a framework to use the DG methods in hyperbolic equation systems. Based on this framework, detailed studies of various aspects of DG methods have been performed, which include stability [119], convergence rates [26, 32], influences of various basis functions [77] and quadrature rules [8, 97], and applications to different mesh shapes [30, 77]. To capture discontinuities in hyperbolic systems, various limiting and artificial viscosity methods have also been proposed [9, 124, 165], some of which enable DG methods to capture strong shocks with good stability and sub-cell resolution [9, 124, 137]. To predict the turbulent fluctuations in high-fidelity simulations, DG methods that preserve the kinetic energy were proposed [54, 57], which effectively lower the dissipation in turbulence simulations while maintaining stability.

Another important progress is the development of DG methods applicable to parabolic systems. Reference [134] illustrated that a straight forward extension of the DG methods to parabolic systems leads to inconsistencies. To fix this problem, an interior penalty method (IP) was proposed to give consistent and stable solutions [46, 5]. However, the IP method has free parameters and is still inconsistent for first-order DG methods. Bassi and Rebay [12] proposed a way of exchanging information between elements that leads to consistent, stable and parameter-free schemes. Cockburn and Shu [33] generalized the idea of Bassi and Rebay and proposed the local DG scheme. There were other similar attempts [87, 120, 141] to further improve the DG methods for parabolic problems, most of which could be formulated in a general framework [5].

The studies mentioned above and the references therein propose high-order numerical schemes with good spectral properties, high resolution, applicable to

complex unstructured meshes and to different governing equations for industrial applications.

DG methods in high-fidelity simulations

These improved, mature DG methods have become good candidates for high-fidelity simulations. Nevertheless, high-fidelity simulations such as LES are usually highly under-resolved, and achieving stable simulations in such cases is challenging for factors such as aliasing errors which are more significant in high-order methods [80]. This topic has attracted numerous studies [147, 1, 72]. Various approaches have been explored to further enhance the stability of the DG methods, such as spectral vanishing viscosity [81, 103], consistent integration [98, 99], skew-symmetric or split methods [57, 119], and filtering methods [7, 128]. These attempts significantly improve the stability of the schemes especially in highly under-resolved simulations. Another challenge arises due to the complex coupling of numerical truncation terms and physical models. Both of them take place near the Nyquist wavenumber [54, 51] of the computational setting, which is the upper limit in wavenumber that the numerical methods can capture. This is especially relevant for implicit LES because the numerical truncation terms can significantly influence the physical modelling [61]. Studies have been conducted in understanding of the interplay between numerical truncation terms and physical models based on numerical experiments and theoretical analysis [105, 51, 106, 61, 101, 146, 164, 154]. The results have shown that the numerical truncation terms of the DG methods are similar to a sub-grid scale module and implicit LES modeling can give accurate predictions of turbulent flows.

Consider the studies of high-fidelity simulations on turbo-machinery as an example, significant progress has been made both in developing suitable numerical methods

and in studying the underlying flow physics. Bassi and his co-workers investigated the accuracy and efficiency of the DG methods on turbine simulations [11, 13, 60]. However, most of their work is limited to steady or unsteady RANS simulations. Jia et al. [74, 73] simulated high Reynolds turbine vanes to assess the efficiency, accuracy and stability of high-fidelity simulations based on the flux reconstruction method (FR), which has close similarities to nodal DG methods [100]. Lu et al. [90, 89, 91] published a series of papers on developing a FR based high-fidelity simulation software for turbo-machinery simulations. They emphasized the efficient parallel implementations on many-core computational systems. Cassinelli et al. [27] presented high-fidelity simulations of a high Reynolds number low-pressure turbine blade using high-order p -adaptive DG-kernel methods. Hillewaert et al. have been constantly focusing on applying the DG methods to turbine simulations [56, 68, 67, 40]. Various aspects of turbine high-fidelity simulations, such as wall-modeling [41], efficient data structures [67], and influences of mesh quality [22] have been investigated. A summary of a systematic evaluation of different solvers (also different methods) for high-fidelity turbo-machinery simulations can be consulted in reference [22]. Besides problems on turbulence generation, laminar-to-turbulent transition and grid generation, the authors emphasized the necessity to further improve simulation efficiency. Overall, most published papers still focus on single blade or single compressor stage simulations, while only a few high-fidelity simulation results of multistage compressors are available because of their large simulation costs [22]. As a result, further improving the efficiency is necessary for high-fidelity turbo-machinery simulations and likewise for high-fidelity simulations in other research areas .

1.2.2 Implicit time integration methods

To improve the efficiency of high Reynolds number simulations, it is quite natural to turn to implicit time integration methods, which can relax or even remove the serious time step constraint of explicit time integration methods [78, 67, 121]. An implicit time integration method for the nonlinear governing equations of compressible flows mainly consists of a temporal scheme for discretizing the temporal derivative term and a method for solving the nonlinear system arising from the spatial discretization.

Temporal discretization scheme

Instead of using full discretization scheme, the temporal and spatial discretization are usually considered separately using the method of lines. In this case, the temporal discretization scheme can be considered simply as an ODE (ordinary differential equation) system solver. In the CFD community, both multi-step [154, 15, 109] and multi-stage [15, 78, 18, 84, 117, 151] ODE solvers are widely adopted, which can be expressed in general using Butcher tableau. Various of them have been implemented in the Nektar++ framework using a general linear method framework [150].

For the multi-step schemes, the second order backward differentiation formulation (BDF2) scheme is the most widely adopted one because of the difficulties in maintaining good stability and in starting up for higher order BDF schemes [154, 15, 109]. The most attractive property of BDF schemes is that they can obtain high-order approximations by solving only one nonlinear system.

Implicit Runge-Kutta schemes are one of the most popular choice for the multi-stage time stepping schemes [15, 78, 18, 84, 117, 151]. Here, we mainly focus on diagonally implicit Runge-Kutta (DIRK) schemes, which avoid solving much

larger nonlinear systems coupling different implicit stages. We further limit our attention to singly DIRK (SDIRK) schemes with constant diagonal coefficients. It helps to reduce the complexity in freezing the preconditioning matrices as will be shown in Section 3.2. DIRK schemes have attracted significant attention in topics such as linear and nonlinear stability [36, 84], order reduction in stiff problems [79, 78, 84, 63], optimization on accuracy and efficiency [20, 108], step size control [111, 18, 136], embedded scheme construction [84, 78, 111] and their applications [110, 116]. Comprehensive reviews and comparisons of different DIRK methods are available in references [78, 15, 79]. It is important to note that DIRK schemes with an embedded paired scheme provide a good approach for estimating temporal errors at a relatively low cost, which could be utilized for time step adaption. Recently, fully implicit Runge-Kutta schemes [117] which solve implicit stages as a coupled system, and Rosenbrock schemes [10, 18, 55], which can be regarded as linearized Runge-Kutta methods, have also shown some good properties in CFD simulations. However, we will not go into details of these methods and interested readers can refer to the literature on these topics [15, 78, 63] and references therein.

Development of nonlinear system solver

Although the temporal discretization schemes (BDF or SDIRK) have different accuracy, stability and efficiency properties, they all lead to nonlinear systems when applied to compressible flow simulation. The common choice in numerically solving these nonlinear systems is to use Newton-like iterative methods. At each step of the iterative process, a linear system is formed and solved. In Newton's method, the Jacobian matrix of the nonlinear residual vector with respect to the solution vector is the coefficient matrix. For large-scale simulations of compressible flows, both constructing and solving the linear system is costly. A way around this difficulty

is to use instead an approximate coefficient matrix, which is easier to form and solve. However, this approximation will introduce errors that could deteriorate the super-linear convergence properties of the Newton's method.

In low-order (below third order) CFD simulations, constructing an approximate coefficient matrix was the most popular approach and implicit methods like the LU-SGS (Lower-upper Symmetric-Gauss-Seidel) achieved great success [162]. For instance, the renowned LU-SGS method uses the Jacobian matrix of a first-order upwind scheme together with a spectral radius approximation for the viscous Jacobian, which leads to a diagonalized system. The system is trivial to solve and has a cost similar to the residual evaluation, but introduces a large linearization error compared with the Newton's method. The resulting implicit methods can successfully relax the time step constraint whilst keeping the cost of each implicit iteration low and have achieved great success in low-order simulations.

However, a different trend is clearly seen in the development of implicit methods for high-order simulations. Linearization errors have a much larger influence on high-order simulations than on low-order ones leading to much more implicit steps for the high-order simulations to converge [152]. As a result, many efforts are being made to reduce linearization errors, such as the development of Jacobian-free Newton Krylov (JFNK) methods [82, 160], instead of reducing the cost of each implicit step. The JFNK uses the Jacobian-free method (JF) to numerically calculate the Jacobian matrix, which avoids the explicit storage of a very large matrix while keeping the linearization error low [142, 49, 82]. Together with the Newton's method for solving the nonlinear system and the Krylov method for solving the linear system in each Newton iteration, the JFNK methods offer a general framework for solving various kinds of nonlinear equation systems resulting from implicit time integrations. Various aspects of the JFNK method have been studied, such as the influence of the

Jacobian errors, parallelization of the solver and applications to different problems, to name but a few [142, 49, 82]. Numerical results have shown that the solver can obtain the super-linear convergence property of the Newton's method with an accurate Jacobian matrix and converge within tens of iterations even for some stiff problems [159, 158]. Assuming one has reasonable preconditioners, the resulting implicit schemes can largely speed up the simulations especially for steady-state simulations both for high-order and low-order simulations [162, 159, 163, 82, 158]. In steady-state simulations, these implicit methods can speed up the simulations by orders of magnitude and show notable advantages in efficiency compared with explicit time integration methods [159, 163].

Challenges in unsteady simulations

However, it becomes controversial whether implicit time integration methods have obvious advantages over explicit ones for unsteady simulations. Many researchers turn to the explicit time integration methods for unsteady simulations. This can likely be associated with the following challenges encountered by implicit time integration methods: Ensuring temporal accuracy, improving efficiency and identifying robust parameter choices:

1. *Ensuring temporal accuracy.* Achieving satisfactory temporal accuracy is a key issue for unsteady simulations. Although temporal accuracy [53, 110] can be confirmed by convergence, it can not be obtained before a very accurate solution is obtained. Therefore, the users still face the uncertainty about temporal accuracy when simulating a new problem. The uncertainty becomes stronger facing the freedom to choose the specific temporal scheme and time step size in implicit solvers especially when comparing the orders of magnitude

larger time steps adopted by implicit methods with the corresponding very small time steps of explicit time integration methods. Identifying the edge for temporal accuracy is also important from the point view of efficiency since, as will be discussed, the extra costs of maintaining temporal accuracy is one important reason why implicit solvers cannot achieve as large speed up in unsteady simulations as in steady-state simulations.

2. *Improving efficiency.* Implicit methods are reported to achieve smaller speed up ratio in unsteady simulations [22, 159] relative to the overwhelming speed up of the implicit schemes in steady-state simulations [158, 163]. This reduction in efficiency is caused by the following reasons. Firstly, the need for temporal accuracy limits the time step size and requires the convergence of Newton iterations at each time step, which largely increases the costs of implicit solvers. This requirement also makes the use of accelerating techniques such as the local time stepping more difficult in implicit solvers. Secondly, when close to convergence in steady-state simulations, the current solution naturally provide a good initial guess to the next iteration and greatly help the convergence, which is however not the case in unsteady simulations. Thirdly, it may be more difficult for complex implicit solvers to benefit from some modern HPCs (High Performance Computers) platforms such as GPUs, which may further counterbalance the advantage of implicit methods in efficiency. The implicit methods, required to solve coupled systems, are by design more difficult to parallelize and consume a large amount of memory. Therefore, it is challenging to achieve high performance in memory bandwidth bottle-necked modern HPCs [91, 148].

3. *Identifying robust parameter choices.* Since most of explicit time integration methods are limited by the stability condition, users almost only need to

properly choose a time step as large as possible for stability. However, in implicit time integration methods, there are several parameters influencing the accuracy, robustness and efficiency of the solver. A naive choice of the set of parameters may significantly reduce the efficiency of unsteady simulations. Up to now, a systematic and robust way to choose these parameters is still an open problem, which makes the parameter choice not user friendly especially for general purpose solvers with a wide range of application areas.

To overcome these challenges, we should identify the requirement of temporal accuracy for the implicit time integration, which not only influences accuracy itself but also efficiency. Low memory implicit time integration methods and more efficient data movement techniques should be explored to adjust the implicit methods to modern HPC platforms to achieve peak performances. Moreover, influences of different parameters should be explored and strategies in identifying appropriate parameter choices should be developed for the development of robust and user friendly CFD solvers. Although new methods to accelerate unsteady simulations considering these features have been explored [67, 16, 6], this research area is still relatively new and there are still many open questions to explore.

Efficient low memory preconditioning

Since the convergence of Krylov methods is highly dependent on the distribution of the eigenvalues of the system, adopting appropriate preconditioners to cluster the eigenvalues is very important for improving the efficiency of implicit solvers. A good preconditioner should be able to effectively cluster the eigenvalues at a relatively low cost. Besides effectively clustering the eigenvalues, reducing the ratio of computational cost between the assembly and calculation of the preconditioner

and the residual evaluation is also important to the final efficiency. A good counter example of this is using the inverse of the Jacobian matrix itself as the preconditioning matrix.

Here we emphasize the importance of developing preconditioners with low memory consumption because this is a key in solving the problems of implementation in modern HPC and extending to higher order. In modern HPCs, efficiently loading data becomes a bottleneck problem in most CFD simulations since computational speed of HPCs has developed much faster than their memory and cache speed [149]. In these circumstances, using a preconditioner with large memory consumption will inevitably cause more data loading and longer overall computation time. On the other side, because the size of matrices in DG methods increases very fast with the polynomial order, the costs of matrix-based preconditioners will also increase much faster than those of residual evaluations. This means implicit methods with matrix-based preconditioners will have difficulties extending to high order [14, 25]. Notable preconditioners are the approximate factorization of the Jacobian matrix like LU-SGS [158, 93, 129] and ILU (incomplete lower-upper factorization) [121, 42], iterative methods like the block Gauss-Seidel [121], multigrid methods [14, 67] and preconditioner constructed based on the Jacobian of simplified of residual operators [133, 17]. Among these preconditioners, ILU has proved to be very efficient in various simulations [59, 82, 121, 28]. However, the ILU preconditioner needs to form and store the Jacobian matrix to perform the factorization, which is expensive in memory especially for large scale 3D simulations. Another widely used preconditioner is the block-Jacobi preconditioner, which only keeps the diagonal part of the Jacobian matrix and could be several times lower in memory consumption compared to the ILU preconditioner. Because of its simplicity and relatively low memory consumption, the block-Jacobi preconditioner is a common choice for comparison of different

preconditioners [42, 121, 67] and in large-scale simulations [22, 37].

To reduce memory consumption, Birken et al. [16] proposed a reduced off-diagonal block order SGS (symmetric Gauss-Seidel) preconditioner. The proposed preconditioner shares similar memory cost as the block-Jacobi but provides a better preconditioning effect. The preconditioner can, at most, speed up the simulation by 30% in their tests. Diosady and Murman [43] developed block-Jacobi preconditioners with the elemental blocks solved approximately by taking advantage of the tensor-product formulation of their finite-element scheme for solving a scalar advection-diffusion equation, which utilizes a diagonalized alternating-direction-implicit method and a fast diagonalization method. The preconditioner was extended to compressible Navier-Stokes equations with the diffusion Jacobian approximated by a diagonal matrix with values of its spectral radius. The results show that the preconditioner can speed up the simulation by an order of magnitude when compared to a simple mass-matrix preconditioner. Pazner and Persson [118] proposed approximate tensor-product preconditioners based on a shuffled singular-value decomposition of the diagonal blocks of the Jacobian matrix. The preconditioner successively makes computational complexity increase much slower as polynomial order increases, which is verified up to $P = 30$ and achieved speed up at polynomial orders higher than 4. However, the problem of treating the viscous terms and off-diagonal terms is still not solved. This idea was further investigated by Diosady and Murman [44] to further reduce the computational complexity by constructing a tensor-product approximation of the elemental block-Jacobi preconditioner. However, the preconditioner is difficult to extend to Euler and Navier-Stokes systems. Birken et al. [17] explored a different strategy which divides the DG cells into subcells and constructs a preconditioner based on the operator of first-order finite volume discretizations on the subcells. Promising results have been achieved in 1D Euler systems. A similar idea was

explored in reference [115] by constructing preconditioners based on anisotropic subcells of low-order finite element schemes. Special subcell multigrid methods with smoother were developed. Various examples with viscosity are tested to illustrate the performance of the preconditioner. Bastian et al. [14] compared matrix-based, partially matrix-based, and matrix-free implementations of their preconditioner. The matrix-free is obtained by iteratively inverting the diagonal blocks of the Jacobian matrix. A clear advantage is obtained for high-order polynomials, but developing an efficient iterative solver for the diagonal blocks for various problems is still challenging. Jourdan and Wang [53] completely removed the preconditioner in their JFNK implicit solver and achieved speed up of more than 10 for high Reynolds number simulations.

There have been other attempts in reducing the memory consumption of implicit time integrations but not within the Krylov framework. Luo et al. [92] adopted a p-multigrid method as a preconditioner, where a matrix-based smoother is only utilized at the first-order level while explicit Runge-Kutta schemes are used as smoother at higher order levels. This strategy successively reduces the storage by avoiding matrix-based operations at high-order levels and achieves significant speed up compared with explicit methods in steady-state simulations. Liang et al. [86] proposed an implicit solver based on dual time stepping and a p-multigrid method that uses an explicit scheme as smoother and achieves a speed up higher than 2 compared to the corresponding single-level method in unsteady simulations. This method is further studied with adaptive local time stepping and optimized explicit Runge-Kutta pseudo-time stepping schemes [88, 145], and achieves a speed up of more than 10 compared to a global pseudo-time stepping with an RK4 scheme for high Reynolds number unsteady simulations.

Although significant progress has been made in developing low memory preconditioners, most of them are still based on the simple block-Jacobi preconditioner or

are still limited to simple equation systems. Developing robust and more efficient low-memory preconditioners is still highly needed for solving the bottleneck problems faced by implicit time integrations methods.

Robust parameter choices

Implicit time integration schemes 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 [10, 143, 110, 111, 159]. However, implicit time integration schemes are generally more complex and introduce many parameters, which usually have large influences on the performance of the solver. Consider the widely used implicit time integration schemes based on Runge-Kutta (RK) temporal discretization scheme and Jacobian-free Newton Krylov (JFNK) method as an example, there are parameters such as the order of the Runge-Kutta scheme, the time step size, the Newton iteration convergence tolerance, the linear iteration convergence tolerance and possibly other parameters introduced by preconditioners [78, 82, 159]. These parameters are usually highly problem-dependent and have large influences on the accuracy, efficiency and robustness in specific simulations, the choices of which actually become a complex multi-objective optimization problem. Therefore, a reliable approach for determining these parameters is essential especially for computational fluid dynamics (CFD) softwares with a wide range of application areas and for the automatic simulations of massive cases in a design pipeline.

The majority of studies on steady-state simulations mainly focus on accelerating the convergence to steady-state. Reference [144] proposes an expert system for choosing an efficient CFL, in which the convergence history is manually separated

into three different stages and a different adaptive strategy is used in each stage. Reference [85] develops a solution-limiting method to determine the CFL. Bücker et al. compared some of the existing adaptive methods and concluded that there is no clear winner [23]. Reference [75] develops an error based adaptive CFL method to accelerate the late stage of convergence but still need to manually divide the convergence history into different stages. Much fewer studies have focused on robustness compared with the studies on efficiency. Lian et al. [85] addressed this problem with the solution-limiting method and claimed that the robustness is largely improved. Other attempts on this topic are techniques such as rolling back to the previous time step and recomputing the solution with a smaller time step if the solution is not satisfactory [161]. These studies have highlighted the importance of these parameter choices on different aspects of the solver. Although significant progress has been made, *optimal CFL evolution is still an open problem* [23].

Unsteady simulations require a larger number of parameters and the additional complication of maintaining temporal accuracy. The idea of using adaptive time-stepping in unsteady simulations dates back to step adaptation studies of ODEs (ordinary differential equations) [135, 136, 3]. Various methods for choosing the step size (or step size controllers) as well as other topics, such as the requirement of the temporal error estimator and the stability properties in stiff problems, were proposed and compared, comprehensive reviews of which can be found in references [135, 136]. Birken adopted a similar idea to those 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 is adopted in Rosenbrock methods and explicit first stage singly-diagonally implicit Runge-Kutta (ESDIRK) time integration methods [18]. Noventa et al. developed a similar time step adaptation strategy in unsteady incompressible turbulent flow

simulations [110], which is further tested in compressible simulations focusing on the comparison of different step size controllers [111]. The adaptive method based local temporal error was further developed for goal oriented time step adaptation and compared with a method based on global error estimate [96]. However most of studies such as these in references [16, 18, 111, 110], 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 and spatial discretization employed. 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 of CPU time without obviously improving the results as shown in reference [110].

Open-source development

The development of implicit high-order spectral element compressible flow solvers in open-source software is still quite limited especially for unsteady simulations. A few exceptions are the following. Based on deal.II, Hartmann and Houston [64] developed a solver for the compressible NS equations using implicit time integration and the DG scheme. However, their solver was only tested on steady-state problems and only supports quadrilateral and hexahedral meshes. A fully implicit FR solver for compressible NS equations has been developed in the CoolFluid framework [143]. This implicit solver was only tested in steady-state problems and only first and second-order curved quadrilateral and hexahedral meshes are supported. Nek5000 provides an implicit solver based on Jacobian-free Newton Krylov method [31], but it only supports weakly compressible simulations through some modifications of the incompressible NS equations. An implicit solver based on MOOSE has been

reported recently, but it currently only focuses on incompressible flow systems [125]. In summary, only a few implicit high-order solvers with limited capabilities are available in the open-source community.

Nektar++ is a C++ based cross-platform open-source framework that aims at making high-order spectral/*hp* element methods accessible to a wider range of researchers and engineers [24, 107]. Similar to many packages, the Nektar++ framework is composed of libraries and solvers. The six core libraries provide basic functions for different aspects of the implementation of the spectral/*hp* element method, including elemental basis functions, extraction of geometric information, and numerical schemes for specific types of equations. All these libraries provide the basic building blocks for the numerical methods, which makes the construction of new solvers easier and more transparent. The Nektar++ framework supports various kinds of high-order curved meshes including hexahedral, prismatic, pyramidal, tetrahedral, quadrilateral and triangular meshes. Continuous Galerkin (CG) [77], discontinuous Galerkin (DG) [97], and flux reconstruction (FR) schemes (currently only support hexahedral and quadrilateral meshes) [100] are supported for spatial discretizations. Up to 12 built-in solvers have been developed to date and it provides the capability of multi-solver coupling [107]. These solvers make the Nektar++ framework applicable to a wide range of simulations [24, 107].

The development of the implicit solver for the compressible Navier-Stokes (NS) equations in Nektar++ has been based on the explicit version of the solver. The explicit solver has potentially significant limitations on the time step because of the stability restrictions of high-order spectral/*hp* element schemes, see for instance reference [66]. These time step restrictions can significantly reduce the convective and diffusive time step near the boundaries, e.g. to resolve boundary layers with stretched cells at increasing Reynolds numbers [77] or near badly shaped cells in

complex meshes. The unconditional stability of implicit solvers can not only speed-up the simulations by allowing much larger time steps but also reduce the risk of run time *blow-up* resulting from local instability in the nonlinear evolution of the flow field.

However, the development of the implicit solver presents some challenges both in numerical algorithms and in software design. Firstly, the coupling of different equations in the compressible NS equations means the equations cannot be solved component by component as is the case for existing implicit solvers for decoupled equation systems in Nektar++. This not only makes the coupled implicit system much larger in size but also stiffer due to the scattering of eigenvalues for different equations. Secondly, the problem naturally leads to a nonlinear system requiring highly optimized set up operators since the implicit system of equations needs to be updated along with the simulations.

1.3 Objectives

The objective of this thesis is to develop an efficient implicit in time solver for high order simulations of compressible flows within the Nektar++ framework. In the implicit solver, diagonally implicit Runge-Kutta methods and Jacobian-free Newton Krylov methods are adopted. To accelerate the convergence, a partially matrix-free iterative preconditioner is developed based on studies of the influences of different parts of the Jacobian matrix on preconditioning performance. The solver is then designed and implemented in the open-source framework with techniques such as hybrid analytical and numerical Jacobian calculation to facilitate feasibility. As a framework with different users and wide application areas, a proper strategy for choosing parameters of the implicit solver is developed based on theoretical analysis

of the error relations. Finally, the solver is applied to high-fidelity simulations to show the performance of the solver.

1.4 Outline

The governing equations and numerical methods for spatial and temporal discretization are described in Chapter 2. Chapter 3 discusses Jacobian and preconditioning matrices approximation methods, which is vital for efficiency especially for the nonlinear compressible flow system. Implementation related topics are discussed in Chapter 4 where performance of the solver is profiled and verified using canonical cases. A strategy for choosing simulation parameters in the solver, which significantly influences the performance, is presented in Chapter 5. The solver is applied to the simulations of turbulent flows in Chapter 6. Finally, conclusions are drawn in Chapter 7.

Chapter 2

Governing equations and numerical methods

The governing compressible Navier-Stokes equations are presented in Section 2.1. Section 2.2 the spatial discretization methods in detail including the weak DG method for the advection term, symmetric interior penalty Galerkin (SIPG) method for the diffusion term, and the artificial diffusion method for shock-capturing. The temporal discretization methods are presented in Section 2.3. Finally, the errors of the discretization are estimated in Section 2.4.

2.1 Compressible Navier-Stokes equations

The non-dimensional governing compressible Navier-Stokes equations, representing conservation of mass, momentum and energy, can be written in abridged form as

$$\frac{\partial \mathbf{U}}{\partial t} = L(\mathbf{U}) = -\nabla \cdot \mathbf{H}; \quad \mathbf{H}_i = \mathbf{F}_i(\mathbf{U}) - \mathbf{G}_i(\mathbf{U}, \nabla \mathbf{U}), i = 1, 2, \dots, d, \quad (2.1)$$

where L is the analytical nonlinear spatial operator, $\mathbf{U} = [\rho, \rho u_1 \dots \rho u_d, E]^T$ is the vector of conservative variables, d is 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

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

Here, the summation convention is used for repeated indexes, δ_{ij} is the Kronecker delta, ρ is the density, p is the pressure, T is the temperature, u_i is the i th velocity component, $E = p/(\gamma - 1) + \rho u_k u_k/2$ is the total specific energy, $\gamma = C_p/C_v = 1.4$ is the ratio of the specific heats with constant pressure (C_p) and constant volume (C_v), and the stress tensor is

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

where x_i is the i th coordinate and μ is the dynamic viscosity. Finally, the i th component of heat flux is given by

$$q_i = - \frac{\mu}{\text{Ma}_\infty^2 \text{Pr}(\gamma - 1)} \frac{\partial T}{\partial x_i}, \quad (2.4)$$

where Ma_∞ is the Mach number based on the reference flow state, and Pr is the Prandtl number. The Euler equations of inviscid compressible flow are recovered by ignoring the diffusion flux vector $\mathbf{G}_i$. Together with appropriate initial and boundary conditions, Eq. (2.1) is closed through the equation of state (EoS) for a perfect gas

and the Sutherland's law. The EoS is

$$p = \rho RT, \quad (2.5)$$

where the R is

$$R = \frac{1}{\gamma \text{Ma}_\infty^2}, \quad (2.6)$$

after scaling by u_∞^2/T_∞ . Scaling the viscosity by μ_∞ and the temperature by T_∞ leads to the non-dimensional Sutherland's law

$$\mu = \frac{T^{3/2}}{\text{Re}_\infty} \frac{1 + T_S/T_\infty}{T + T_S/T_\infty}, \quad (2.7)$$

where T_S is the Sutherland temperature and a value $T_S = 110K$ is used. In the above equations, Re_∞ is the Reynolds number of the reference flow state referred to a characteristic length and T_∞ is the reference temperature in degrees Kelvin.

2.2 Spatial discretization

The spatial operator in Eqs. (2.1) is discretized under the framework of spectral/ hp element method. To be specific, the DG method with an artificial diffusion shock regulation is adopted.

2.2.1 Discontinuous Galerkin formulation

In the discontinuous Galerkin method, the computational domain Ω is divided into N_e non-overlapping elements, Ω_e . The space of test functions is defined as

$$V^P = \left\{ \phi : \phi|_{\Omega_e} \in \mathcal{P}^P(\Omega_e), e = 1, \dots, N_e \right\}, \quad (2.8)$$

where $\mathcal{P}^P(\Omega_e)$ is the polynomial space of degree P in Ω_e . The weak form of Eqs. (2.1) is obtained by multiplying them by the test function ϕ_p and performing integration by parts in Ω_e ,

$$\int_{\Omega_e} \frac{\partial \mathbf{U}}{\partial t} \phi_p d\Omega_e = \int_{\Omega_e} \nabla \phi_p \cdot \mathbf{H} d\Omega_e - \int_{\Gamma_e} \phi_p \mathbf{H}^n d\Gamma_e, \quad (2.9)$$

where Γ_e is the element boundary, $\mathbf{H}^n = \mathbf{F}^n - \mathbf{G}^n$, $\mathbf{F}^n = \mathbf{F} \cdot \mathbf{n}$, and $\mathbf{G}^n = \mathbf{G} \cdot \mathbf{n}$ are boundary fluxes on the elemental outward normal direction ($\mathbf{n}$).

The vector of conservative variables $\mathbf{U}$ is approximated in the same polynomial space as the test functions and it is expressed as

$$\mathbf{U}_\delta(\mathbf{x}, t) \simeq \sum_{q=1}^{N_P} \mathbf{u}_q(t) \phi_q(\mathbf{x}), \quad (2.10)$$

where $\mathbf{u}_q(t)$ is the q th coefficient of the basis (or trial) function $\phi_q(\mathbf{x})$ and N_P is the total degrees of freedom (DoFs) in the element. The flow variable values on some quadrature points are calculated using the backward transformation

$$\mathbf{U}_{\delta,i} = \sum_{q=1}^{N_P} \mathbf{u}_q(t) \phi_q(\mathbf{x}_i) = \sum_{q=1}^{N_P} \mathbf{B}_{iq} \mathbf{u}_q, \quad (2.11)$$

where $\mathbf{B}$ is the backward transformation matrix. Here $\mathbf{x}_i$ is the vector of coordinates of the i th quadrature point. The fluxes are calculated at these quadrature points and quadrature rules with N_Q interior quadrature points and N_Q^Γ quadrature points on element boundaries are adopted to calculate the integration. This leads to

$$\begin{aligned} \sum_{i=1}^{N_Q} \sum_{q=1}^{N_P} \phi_p(\mathbf{x}_i) w_i J_i \phi_q(\mathbf{x}_i) \frac{d\mathbf{u}_q}{dt} &= \sum_{i=1}^{N_Q} w_i J_i \nabla \phi_p(\mathbf{x}_i) \cdot \mathbf{H}(\mathbf{U}_{\delta,i}) \\ &\quad - \sum_{i=1}^{N_Q^\Gamma} \phi_p(\mathbf{x}_i^\Gamma) w_i^\Gamma J_i^\Gamma \hat{\mathbf{H}}_i^n, \end{aligned} \quad (2.12)$$

where w_i and J_i are the quadrature weights and grid metric Jacobian on the i th quadrature point respectively, $\hat{\mathbf{H}}_i^n = \hat{\mathbf{F}}_i^n - \hat{\mathbf{G}}_i^n$, $\hat{\mathbf{F}}_i^n$ and $\hat{\mathbf{G}}_i^n$ are the numerical normal fluxes on the i th quadrature point $\mathbf{x}_i^\Gamma$ of the element boundaries, and w_i^Γ and J_i^Γ are the quadrature weights and grid Jacobian of the lower dimensional integrations on element boundaries, which are usually not equal to the w_i and J_i even if the element and element boundary quadrature points are at the same position.

A semi-discrete equation for the vector $\mathbf{U}_\delta$ can be written as

$$\frac{\partial \mathbf{U}_\delta}{\partial t} = L_\delta(\mathbf{U}_\delta), \quad (2.13)$$

where

$$L_\delta(\mathbf{U}_\delta(\mathbf{u})) = \mathbf{B}\mathcal{L}_\delta(\mathbf{u}). \quad (2.14)$$

The spatial truncation error can be expressed as

$$\mathbf{E}_s(\mathbf{U}_\delta) = L_\delta(\mathbf{U}_\delta) - L(\mathbf{U}) = C_s D_s(\mathbf{U}) h^{P+1} + O(h^{P+2}), \quad (2.15)$$

where C_s is a coefficient related to the discretization scheme but independent of $\mathbf{U}$, $D_s(\mathbf{U})$ is a spatial derivative term dependent on $\mathbf{U}$, and P is the polynomial order of the basis functions in Eq. (2.11).

The weak DG scheme for the advection term is complete as long as a Riemann numerical flux is used to calculate the normal flux, $\hat{\mathbf{F}}^n(\mathbf{Q}^+, \mathbf{Q}^-, \mathbf{n})$, in which $\mathbf{Q}^+$ and $\mathbf{Q}^-$ are variable values on the exterior and interior sides of the element boundaries, respectively.

2.2.2 Symmetric interior penalty Galerkin method

The interior penalty (IP) method is adopted to discretize the diffusion term. To simplify the complexity of the expressions for the multi-dimensional matrix operations in this section, all the indexes of matrices and vectors follow the summation convention. The diffusion flux in the IP method is expressed in the equivalent form

$$\mathbf{G}_{ik} = \mathbf{K}(\mathbf{Q})_{ijkl} \nabla_j \mathbf{Q}_l, \quad (2.16)$$

where i and j are indexes of different spatial directions, and k and l are indexes of different flow variables. The expression of $\mathbf{K}(\mathbf{Q})_{ijkl}$ can be found in [64]. For clarity, only the diffusion terms are shown in the derivation. The IP method can be expressed in the following primal form

$$\begin{aligned} \int_{\Omega} \frac{\partial \mathbf{U}_k}{\partial t} \phi_p d\Omega &= \sum_{e=1}^{N_e} \int_{\Gamma_e} \llbracket \phi_p \rrbracket_i \left\{ \mathbf{K}(\{\mathbf{Q}\})_{ijkl} \nabla_j \mathbf{Q}_l \right\} d\Gamma_e - \sum_{e=1}^{N_e} \int_{\Omega_e} \nabla_i \phi_p \mathbf{G}_{ik} d\Omega_e \\ &\quad + \beta_s \sum_{e=1}^{N_e} \int_{\Gamma_e} \llbracket \mathbf{Q}_l \rrbracket_i \left\{ \mathbf{K}(\{\mathbf{Q}\})_{ijkl} \nabla_j \phi_p \right\} d\Gamma_e \\ &\quad + \sum_{e=1}^{N_e} \beta_p \int_{\Gamma_e} \llbracket \phi_p \rrbracket_i \mathbf{K}(\{\mathbf{Q}\})_{ijkl} \llbracket \mathbf{Q}_l \rrbracket_j d\Gamma_e, \end{aligned} \quad (2.17)$$

with $\beta_s = 1$, $\beta_p = (P + 1)^2 / h_T$ for simulations with quadrilateral and hexahedral meshes, $\llbracket w \rrbracket_i = w^+ \mathbf{n}_i^+ + w^- \mathbf{n}_i^-$ and $\{w\} = (w^+ + w^-)/2$, where the superscripts $+$ and $-$ indicate values from the two sides of the element interface. Here, h_T is a characteristic length scale of the elements. We calculate it by an averaging of element lengths on both sides of the trace, which are obtained by the grid derivatives in the trace normal direction. Compared with the IP method proposed in reference [65], $\mathbf{K}(\mathbf{Q})_{ijkl}$ is replaced here by $\mathbf{K}(\{\mathbf{Q}\})_{ijkl}$ for implementation purposes. The choice $\beta_s = 1$ makes the IP method symmetric and adjoint consistent. Values of β_p for

other mesh types can be found in reference [67].

Using the relation between primal form and its flux form given in reference [5], Eq. (2.17) can be expressed in its flux form. As a result, the whole discretization can be written in the matrix form

$$\begin{aligned} \mathbf{M} \frac{d\mathbf{u}}{dt} = & \sum_{j=1}^d \mathbf{B}^T \mathbf{D}_j^T \mathbf{\Lambda} (wJ) \mathbf{H}_j(\mathbf{Q}) \\ & - (\mathbf{B}^T \mathbf{M}_c)^T \mathbf{\Lambda} (w^\Gamma J^\Gamma) \hat{\mathbf{H}}^n \\ & - \sum_{j=1}^d \mathbf{B}^T \mathbf{D}_j^T \mathbf{J}^T \mathbf{\Lambda} (w^\Gamma J^\Gamma) \hat{\mathbf{S}}_j^n, \end{aligned} \quad (2.18)$$

where d is the spatial dimension, $\mathbf{M} = \mathbf{B}^T \mathbf{\Lambda} (wJ) \mathbf{B}$ is the mass matrix, $\mathbf{\Lambda}$ represents a diagonal matrix, $\mathbf{D}_j$ is the derivative matrix in the j th direction, $\mathbf{B}^\Gamma$ is the backward transformation matrix of ϕ^Γ , $\mathbf{M}_c$ is the mapping matrix between ϕ^Γ and ϕ , $\mathbf{J}$ is the interpolation matrix from quadrature points of a element to quadrature points of its element boundaries, $\hat{\mathbf{H}}^n = \hat{\mathbf{F}}^n - \hat{\mathbf{G}}^n$, $\hat{\mathbf{F}}^n$ is the Riemann flux, $\hat{\mathbf{G}}_k^n = \mathbf{n}_i \left(\left\{ \mathbf{K}(\{\mathbf{Q}\})_{ijkl} \nabla_j \mathbf{Q}_l \right\} + \beta_p \mathbf{K}(\{\mathbf{Q}\})_{ijkl} \llbracket \mathbf{Q}_l \rrbracket_j \right)$, and $\hat{\mathbf{S}}_{j,k}^n = \beta_s \llbracket \mathbf{Q}_l \rrbracket_i \left\{ \mathbf{K}(\{\mathbf{Q}\})_{ijkl} \right\}$. The $\hat{\mathbf{G}}_k^n$, which represents the flux integration of the first and last terms on the right-hand side of Eq. (2.17), can be calculated together with the advection flux, while the symmetric flux $\hat{\mathbf{S}}_{j,k}^n$ cannot. Some approximations of the preconditioning matrices are made based on these properties of $\hat{\mathbf{G}}_k^n$ and $\hat{\mathbf{S}}_{j,k}^n$ which will be discussed in Section 3.2.

Both the IP and the local DG (LDG) methods were implemented in Nektar++ for the compressible flow solver. Currently the preferred implicit solver is based on the IP method since it is easier to derive its Jacobian matrix. Moreover, the IP method leads to a more compact stencil than the LDG in general, which is beneficial not only for lowering the memory consumption but also for simplifying the calculation of

the Jacobian matrix needed when preconditioning (see Section 3.2).

2.2.3 Shock-capturing method

The ability to capture flow discontinuities (such as shocks) is an important asset of the numerical simulation of compressible flows. Different approaches to deal with Gibbs-type oscillations arising in the numerical approximation of discontinuities have been proposed in literature in the context of DG methods, including limiters, filtering procedures, h/p adaptation, and artificial viscosity (AV). Nektar++ implements a sensor-based artificial viscosity [29].

This approach augments the right-hand side of the governing equations Eq. (2.1) by a diffusion operator, $\nabla \cdot (\mu_{AV} \nabla \mathbf{U})$, and uses a sensor to detect discontinuities. When the sensor detects a discontinuity, the diffusion operator is activated, which adds an artificial viscosity in proximity to the discontinuity to smooth Gibbs-type oscillations. In the diffusion operator, μ_{AV} is a spatially variable but constant element-wise term defined as

$$\mu_{AV} = \mu_0 \begin{cases} 0, & s_e \leq s_k - k, \\ \frac{1}{2} \left(1 + \sin \frac{\pi(s_e - s_k)}{2k} \right), & ||s_e - s_k|| \leq k, \\ 1, & s_e \geq s_k + k, \end{cases} \quad (2.19)$$

The modal resolution-based shock sensor proposed in reference [124] and Ducros' sensor [47] are implemented. Here s_e is a shock indicator based on the one introduced in [124]

$$s_e = \log_{10} \left(\frac{\langle q - \tilde{q}, q - \tilde{q} \rangle}{\langle q, q \rangle} \right), \quad q(x) = \sum_{i=1}^{N_P} \hat{q}_i \phi_i, \quad \tilde{q}(x) = \sum_{i=1}^{N_{P-1}} \hat{q}_i \phi_i, \quad (2.20)$$

where $\langle \cdot, \cdot \rangle$ represents the L^2 inner product, q and $\tilde{q}$ are the full and truncated expansions of a state variable, $\hat{q}_i$ is the associated coefficients, and N_P is the size of the expansion of order P . In Nektar++, the state variable q is chosen to be the density ρ . In Eq. (2.19), we note that consistency of the numerical method requires $\epsilon_0 \sim h/P$; further dimensional considerations suggest that $\epsilon_0 \sim h/P V_{\text{std}}$ where V_{std} is a characteristic velocity of the system. By analogy with the decay rate of Fourier coefficients for smooth functions ($1/P^2$) [124], $s_k \sim \log_{10}(1/P^4)$. Finally, k needs to be sufficiently large to obtain a smooth shock profile. Following the work of Barter and Darmofal [9], we define

$$\epsilon_0 = \mu_0 \frac{h}{P} \lambda_{\max}, \quad s_k = - \left[4 + 4.25 \log_{10} \left(\frac{1}{P} \right) \right], \quad k = 0.5, \quad (2.21)$$

where $\lambda_{\max}$ is the local maximum wave speed of the system, and the cell characteristics length h is chosen as the minimum edge length and typically $\mu_0 = 1$. This method gives a simple and consistent shock-capturing method for which the parameters are relatively not problem-dependent.

2.3 Implicit time discretization schemes

Section 2.3.1 presents the singly diagonally implicit Runge-Kutta schemes (SDIRK) to approximate the temporal derivatives. Combining the spatial discretization in Section 2.2 and the temporal discretization in Section 2.3.1, the discrete formulation of the governing equations is given in Section 2.3.2, which leads to nonlinear systems for implicit Runge-Kutta (RK) stages. Section 2.3.3 presents the Newton-Krylov method for solving the nonlinear system.

2.3.1 Diagonally implicit Runge-Kutta schemes

Starting with an initial guess

$$\mathbf{U}^{(0)} = \mathbf{U}^n, \quad (2.22)$$

the i th stage of the Runge-Kutta is calculated by

$$\mathbf{S}^{(i)} = \mathbf{U}^n + \Delta t \sum_{j=1}^{i-1} a_{ij} L(\mathbf{U}^{(j)}), i = 1, 2, \dots, S \quad (2.23)$$

$$\mathbf{U}^{(i)} = \mathbf{S}^{(i)} + \Delta t a_{ii} L(\mathbf{U}^{(i)}), i = 1, 2, \dots, S \quad (2.24)$$

Finally, the solution at the new time step $(n + 1)$ is calculated by

$$\mathbf{U}^{n+1} = \mathbf{U}^n + \Delta t \sum_{i=1}^S b_i L(\mathbf{U}^{(i)}), \quad (2.25)$$

To estimate temporal errors, some Runge-Kutta schemes provide an extra embedded scheme,

$$\hat{\mathbf{U}}^{n+1} = \mathbf{U}^n + \Delta t \sum_{i=1}^{S+1} \hat{b}_i L(\mathbf{U}^{(i)}), \quad (2.26)$$

and the temporal error is given by the difference between $\mathbf{U}^{n+1}$ and $\hat{\mathbf{U}}^{n+1}$. In Eqs. (2.22)-(2.26), a_{ij} , b_i and $\hat{b}_i$ are coefficients of the Butcher tableaux on the i th Runge-Kutta (RK) stage.

Various temporal discretization schemes have been implemented and tested in our study. The coefficients of these SDIRK schemes are summarized in Appendix A. We adopt the embedded version of the explicit first stage SDIRK schemes from reference [84] that provides an approximation $\hat{\mathbf{U}}^{n+1}$ which is an order higher than $\mathbf{U}^{n+1}$ at the cost of an extra implicit stage. Thus the leading term of the temporal

error of $\mathbf{U}^{n+1}$ can be accurately estimated as

$$\begin{aligned}
\mathbf{E}_t^{n+1} &= \mathbf{U}^{n+1} - \hat{\mathbf{U}}^{n+1} \\
&= \Delta t \sum_{i=1}^{S+1} (b_i - \hat{b}_i) L(\mathbf{U}^{(i)}) \\
&= C_t D_t(\mathbf{U}) \Delta t^{N+1} + O(\Delta t^{N+2}),
\end{aligned} \tag{2.27}$$

where b_{s+1} is always zero in the ESDIRK schemes adopted, C_t is a coefficient related to the ESDIRK scheme adopted but independent of $\mathbf{U}$, $D_t(\mathbf{U})$ is a problem-dependent temporal derivative term, N is the order of accuracy of the ESDIRK scheme. In the derivation of Eq. (2.27), the analytical Eqs. (2.1) have been adopted to replace the spatial with temporal derivatives, thus the spatial error is not included. The estimation also excludes the temporal errors from previous time steps and will be referred to as the *local temporal error* here.

The selection of a proper time step is vital for both of the explicit and implicit schemes. A commonly adopted method is based on the Courant-Friedrichs-Lewy (CFL) number, of which we consider two different definitions. A *standard* CFL number is defined as

$$\text{CFL} = \frac{c_\lambda \Delta t (c + \sqrt{u_k u_k}) P^2}{d_{\text{CFL}} \Delta x}, \tag{2.28}$$

where $c_\lambda = 0.2$ and c is the speed of sound. The parameter d_{CFL} is designed to reflect the stability property of a specific time discretization scheme so that the stable CFL number has similar value for different time discretization schemes [77]. However, we choose to use the same value $d_{\text{CFL}} = 2$ for different time discretization schemes to ensure that their CFL number ratios directly reflect the corresponding ratios in time steps. The time step of the explicit solver will be chosen based on Eq. (2.28), which ensures the maximum stable CFL is almost constant for polynomial order P [77]. For implicit time discretization schemes, we define a different CFL number to help

determine the time step, which is

$$\text{CFL}_c = \frac{\Delta t P \sqrt{u_k u_k}}{\Delta x}. \quad (2.29)$$

A value $\text{CFL}_c = 1$ means the flow field convects a distance $\Delta x/P$ in one time step, where the distance $\Delta x/P$ in Eq. (2.29) is an estimate of the average length between two DoFs in the element.

For explicit time integration schemes, the stability limitation determines that the CFL number could not be large, which highly limits the size of the time steps especially for simulations with large differences in Δx . Implicit methods can successfully relax this limitation thus potentially improving efficiency.

2.3.2 Discrete formulation

The discrete formulation of the governing equations is obtained by replacing $\mathbf{U}$ and L in Eqs. (2.22)-(2.26) with their discrete counterparts, $\mathbf{U}_\delta$ and L_δ . To reduce the size of the nonlinear system for implicit stages ($a_{ii} \neq 0$), we choose to rewrite the discrete counterparts of Eqs. (2.23)-(2.24) as

$$\mathbf{s}^{(i)} = \mathbf{u}^n + \Delta t \sum_{j=1}^{i-1} a_{ij} \mathcal{L}_\delta(\mathbf{u}^{(j)}), i = 1, 2, \dots, S \quad (2.30)$$

$$\mathbf{u}^{(i)} = \mathbf{s}^{(i)} + \Delta t a_{ii} \mathcal{L}_\delta(\mathbf{u}^{(i)}), i = 1, 2, \dots, S \quad (2.31)$$

where $\mathbf{U}_\delta^{(i)} = \mathbf{B}\mathbf{u}^{(i)}$ and $\mathbf{S} = \mathbf{B}\mathbf{s}$, since the vector length of $\mathbf{u}$ is usually smaller than that of $\mathbf{U}_\delta$. However, for implicit stages, Eq. (2.31) becomes a nonlinear system, which requires an efficient solver for time integration.

2.3.3 Newton-Krylov method

Eq. (2.31) of the implicit stages ($a_{ii} \neq 0$) can be written as the nonlinear system

$$\mathbf{N}(\mathbf{u}^{(i)}) = \mathbf{u}^{(i)} - \mathbf{s}^{(i)} - \Delta t a_{ii} \mathcal{L}_\delta(\mathbf{u}^{(i)}) = \mathbf{0}. \quad (2.32)$$

This system is iteratively solved by Newton's method [82] with the Newton step, $\Delta \mathbf{v} = \mathbf{v}^{k+1} - \mathbf{v}^k$, updated through the solution of the linear system

$$\frac{\partial \mathbf{N}(\mathbf{v}^k)}{\partial \mathbf{v}} \Delta \mathbf{v} = -\mathbf{N}(\mathbf{v}^k). \quad (2.33)$$

where $\mathbf{v}^0 = \mathbf{s}^{(i)}$ is adopted as an initial guess. When the Newton residual is smaller than a convergence tolerance of the Newton iterations τ , i.e.

$$\|\mathbf{N}(\mathbf{v}^k)\| = \|\mathbf{R}(\mathbf{v}^k)\| \leq \tau, \quad (2.34)$$

$\mathbf{u}_{it}^{(i)} = \mathbf{v}^k$ is regarded as the approximate solution of the nonlinear system, where $\mathbf{R}$ is the remaining Newton residual vector $\mathbf{N}(\mathbf{u}_{it}^{(i)})$ after convergence. A proper choice of τ 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 [111]. Assuming $\mathbf{u}^{(i)}$ is the exact solution of the nonlinear system and $\mathbf{u}_{it}^{(i)}$ is the iterative approximation, we can get $\mathbf{N}(\mathbf{u}^{(i)}) = \mathbf{0}$ and $\mathbf{N}(\mathbf{u}_{it}^{(i)}) = \mathbf{R}^{(i)}$. Therefore,

$$\mathbf{N}(\mathbf{u}_{it}^{(i)}) - \mathbf{N}(\mathbf{u}^{(i)}) = (\mathbf{u}_{it}^{(i)} - \mathbf{u}^{(i)}) - \Delta t a_{ii} (\mathcal{L}_\delta(\mathbf{u}_{it}^{(i)}) - \mathcal{L}_\delta(\mathbf{u}^{(i)})) = \mathbf{R}^{(i)}. \quad (2.35)$$

The contribution of the iterative error to the total discretization error will be discussed in Chapter 5.

The linear problem (2.33) is solved using a restarted generalized minimal residual method (GMRES) [132], which is restarted when the number of GMRES iterations exceeds 30 unless otherwise specified. Currently, a standard version of GMRES proposed in [132] is used. Other optimized linear system solvers, such as the SNESANDERSON and SNESNGMRES solvers [21] in PETSc, can also be adopted to further improve the performance of the linear system solver. The GMRES iteration is terminated after the residual drops by 5×10^{-2} unless otherwise specified. This choice will be further discussed in Section 3.1.

The system of Eq. (2.33) can be very stiff simulations with multiscale structures. The use of good preconditioners in GMRES is very important for efficiently solving stiff linear systems. Instead of solving the system of Eq. (2.33) directly, one can solve instead the preconditioned linear system

$$\left(\frac{\partial \mathbf{N}}{\partial \mathbf{u}} \mathbf{P}^{-1} \right) (\mathbf{P} \triangle \bar{\mathbf{u}}^l) = -\mathbf{N}(\bar{\mathbf{u}}^l), \quad (2.36)$$

where $\mathbf{P}$ is the preconditioning matrix. A good preconditioner should be able to effectively cluster the distribution of eigenvalues of the linear system. The preconditioner $\mathbf{P}$ is usually an approximate matrix of the Jacobian matrix $\partial \mathbf{N} / \partial \mathbf{u}$. It should be relatively easy to invert since it has to be used repeatedly in every GMRES iteration. Usually it is not necessary to calculate $\partial \mathbf{N} / \partial \mathbf{u} \mathbf{P}^{-1}$ explicitly. Only the inner product of the matrix $\mathbf{P}^{-1}$ with some vectors is required in a practical implementation. The approximations of the Jacobian matrix, $\partial \mathbf{N} / \partial \mathbf{u}$, and the preconditioning matrix $\mathbf{P}$ will be discussed separately in Chapter 3.

2.4 Discretization error estimate

In this section, the local discretization errors introduced at time step $n + 1$ are estimated, which mainly consists of the spatial error, temporal error and iterative error. The accumulation of these errors, which we refer to as *global errors*, is more difficult to estimate and will not be considered in this paper. Studies considering the global errors can be found in reference [96].

Following Sections 2.2 and 2.3 the numerical solution at the time step $n + 1$ is calculated by

$$\mathbf{U}_{it}^{n+1} = \mathbf{U}^n + \Delta t \sum_{i=1}^S b_i L_\delta \left(\mathbf{U}_{it}^{(i)} \right), \quad (2.37)$$

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

$$\begin{aligned} \mathbf{U}(t^{n+1}) &= \hat{\mathbf{U}}^{n+1} + O(\Delta t^{N+2}) \\ &= \mathbf{U}^n + \Delta t \sum_{i=1}^{S+1} \hat{b}_i L \left(\mathbf{U}^{(i)} \right) + O(\Delta t^{N+2}), \end{aligned} \quad (2.38)$$

where $O(\Delta t^{N+2})$ represents all the high-order truncation terms and is not detailed here. Subtracting Eq. (2.37) by Eq. (2.38) and ignoring the high-order truncation terms of $O(\Delta t^{N+2})$, the leading terms of the local discretization error can be expressed as

$$\mathbf{U}_{it}^{n+1} - \mathbf{U}(t^{n+1}) \approx \Delta t \sum_{i=1}^S b_i L_\delta \left(\mathbf{U}_{it}^{(i)} \right) - \Delta t \sum_{i=1}^{S+1} \hat{b}_i L \left(\mathbf{U}^{(i)} \right). \quad (2.39)$$

If we now add and subtract $\Delta t \sum_{i=1}^{S+1} b_i L \left(\mathbf{U}^{(i)} \right)$ and $\Delta t \sum_{i=1}^S b_i L_\delta \left(\mathbf{U}_\delta^{(i)} \right)$ to the right-hand side of Eq. (2.39), we obtain

$$\begin{aligned} \mathbf{U}_{it}^{n+1} - \mathbf{U}(t^{n+1}) &\approx \Delta t \sum_{i=1}^{S+1} (b_i - \hat{b}_i) L \left(\mathbf{U}^{(i)} \right) + \Delta t \sum_{i=1}^S b_i \left[L_\delta \left(\mathbf{U}_\delta^{(i)} \right) - L \left(\mathbf{U}^{(i)} \right) \right] \\ &\quad + \Delta t \sum_{i=1}^S b_i \left[L_\delta \left(\mathbf{U}_{it}^{(i)} \right) - L_\delta \left(\mathbf{U}_\delta^{(i)} \right) \right], \end{aligned} \quad (2.40)$$

where b_{s+1} is always zero in the ESDIRK schemes adopted. The expression $L_\delta(\mathbf{U}_{it}^{(i)}) - L_\delta(\mathbf{U}_\delta^{(i)})$ in the last term of Eq. (2.40) is related to Eq. (2.35) and can be considered as the iterative error $\mathbf{E}_{it}^{(i)}$. Using Eqs. (2.15) and (2.27) and ignoring high-order terms of $O(h^{P+2})$, we can now rewrite each term as

$$\begin{aligned} \mathbf{U}_{it}^{n+1} - \mathbf{U}(t^{n+1}) &\approx C_t D_t(\mathbf{U})(\Delta t)^{N+1} + \Delta t C_s \bar{D}_s(\mathbf{U}) h^{P+1} + \Delta t \bar{\mathbf{E}}_{it} \\ &= \mathbf{E}_t^{n+1} + \Delta t \bar{\mathbf{E}}_s + \Delta t \bar{\mathbf{E}}_{it} \\ &= \mathbf{B}(\mathbf{e}_t^{n+1} + \Delta t \bar{\mathbf{e}}_s + \Delta t \bar{\mathbf{e}}_{it}), \end{aligned} \tag{2.41}$$

where we have used the overbar to denote summations of the form $\bar{\psi} = \sum_{i=1}^S b_i \psi^{(i)}$, which is a weighted average of ψ because $\sum_{i=1}^S b_i = 1$.

Eq. (2.41) suggests that the error generated in one time step is the sum of the local temporal error, the averaged spatial error and the averaged iterative error. This indicates that to decrease the total error one should decrease these three errors simultaneously since the error will be dominated by the largest value of the three terms.

Chapter 3

Approximating Jacobian and preconditioning matrices

This Chapter mainly discusses two closely related topics: the approximation of the Jacobian and preconditioning matrices. In Section 3.1, the Jacobian-free approximation of the Jacobian matrix is presented with theoretical and numerical analysis of the errors introduced by the Jacobian-free approximation. In Section 3.2, an iterative preconditioner is constructed based on the Jacobian matrix. Moreover, proper approximations of the preconditioning matrices are explored to achieve an efficient implementation of the preconditioner.

3.1 Matrix-free Jacobian matrix approximations

The Jacobian matrix, $\partial \mathbf{N} / \partial \mathbf{u}$, in Eq. (2.33) is a large sparse matrix, the calculation and storage of which is usually excessively expensive compared with those of the explicit solver. What makes the problem worse is that $\partial \mathbf{N} / \partial \mathbf{u}$ is dependent on the

flow solution $\mathbf{u}$, and needs to be updated along with the simulations. Since $\partial\mathbf{N}/\partial\mathbf{u}$ is only used to calculate its inner product with a vector $(\partial\mathbf{N}/\partial\mathbf{u} \cdot \mathbf{q})$ in the GMRES, the Jacobian-free method [82] is widely adopted to avoid explicitly calculating and storing the Jacobian matrix. The Jacobian-free approximation of the Jacobian matrix are presented in Section 3.1.1. Numerical experiments are conducted to study the influences of the errors of Jacobian matrix on the implicit solver in Section 3.1.2.

3.1.1 Jacobian-free method

The construction of the Krylov space in Krylov methods is achieved by successively multiplying the Jacobian matrix to vectors. As a result, the Jacobian matrix itself does not need to be formed explicitly but only the operator of matrix-vector inner product $(\partial\mathbf{N}/\partial\mathbf{u} \cdot \mathbf{q})$ is required. This operator can be approximated by the finite difference formula

$$\frac{\partial\mathbf{N}}{\partial\mathbf{u}}(\mathbf{u}) \cdot \mathbf{q} \simeq \frac{\mathbf{N}(\mathbf{u} + \epsilon\mathbf{q}) - \mathbf{N}(\mathbf{u})}{\epsilon}, \quad (3.1)$$

where ϵ is the step size of the finite difference approximation. By storing $\mathbf{N}(\mathbf{u})$, only one evaluation of the nonlinear residual $\mathbf{N}(\mathbf{u} + \epsilon\mathbf{q})$ is required for each matrix-vector inner product operation in Eq. (3.1).

The step size ϵ in Eq. (3.1) has a large influence on the approximation error of the matrix-vector inner product operator. The first-order finite difference operation leads to an error of $O(\epsilon)$, therefore ϵ should be as small as possible from the point view of minimizing the finite difference truncation error. Another error to consider is the rounding error. Since the vector $\epsilon\mathbf{q}$ is added to $\mathbf{u}$, too small a ϵ would lead to large errors caused by rounding errors. As a result, the whole approximation error in Eq. (3.1) will decrease first but increase afterward as ϵ becomes smaller and smaller

as shown in reference [49]. As suggested in reference [82], we adopt

$$\epsilon = \|\mathbf{u}\|_2 \sqrt{\hat{\epsilon}} / \|\mathbf{q}\|_2, \quad (3.2)$$

where $\hat{\epsilon} = 2.22 \times 10^{-16}$ is the machine epsilon for our computation system. Note that since the vector $\mathbf{q}$ is always normalized first in GMRES, $\|\mathbf{q}\|_2$ is always one in Eq. (3.2).

3.1.2 Jacobian approximation errors

The Jacobian approximation errors associated with Eq. (3.1) is analyzed here to study their influence on the convergence properties. Define the exact Jacobian matrix as $\mathbf{A}$ and its numerical approximation implicitly defined in Eq. (3.1) as $\hat{\mathbf{A}}$. In the following, all the subscript indexes of matrices and vectors follow the summation convention in Eq. (3.3) and Eq. (3.4). Expanding the finite difference approximation (3.2), we get

$$\begin{aligned} \hat{\mathbf{A}}_{ij} \mathbf{q}_j^k &= \left(\mathbf{N}_i(\mathbf{u} + \epsilon \mathbf{q}^k) - \mathbf{N}_i(\mathbf{u}) \right) / \epsilon \\ &= \left[\frac{\partial \mathbf{N}_i}{\partial \mathbf{u}_j} + \frac{\epsilon}{2} \mathbf{q}_l^k \frac{\partial^2 \mathbf{N}_i}{\partial \mathbf{u}_l \partial \mathbf{u}_j} + O(\epsilon^2) \right] \mathbf{q}_j^k \\ &= \left[\delta_{ij} + \lambda_{RK} \Delta t \left(\frac{\partial \mathcal{L}_{\delta,i}}{\partial \mathbf{u}_j} + \frac{\epsilon}{2} \mathbf{q}_l^k \frac{\partial^2 \mathcal{L}_{\delta,i}}{\partial \mathbf{u}_l \partial \mathbf{u}_j} \right) + O(\epsilon^2) \right] \mathbf{q}_j^k \\ &= \left[\mathbf{A}_{ij} + \mathbf{A}_{ij}^{e,k} \right] \mathbf{q}_j^k, \end{aligned} \quad (3.3)$$

where

$$\mathbf{A}_{i,j}^{e,k} = \mathbf{A}_{ij}^e(\epsilon, \mathbf{q}^k) = \frac{\epsilon \lambda_{RK} \Delta t}{2} \mathbf{q}_l^k \frac{\partial^2 \mathcal{L}_{\delta,i}}{\partial \mathbf{u}_l \partial \mathbf{u}_j} + O(\epsilon^2), \quad (3.4)$$

is the error matrix associated with the Jacobian approximation in Eq. (3.1) and λ_{RK} is the constant diagonal coefficient of singly RK methods. Note that the error matrix

depends not only on ϵ but also on $\mathbf{q}^k$. This means even within the GMRES process, the approximate Jacobian matrix is different in generating each basis vector in the Krylov space because of the change of $\mathbf{q}^k$.

The choice of ϵ is important because it determines the approximation error and then influences convergence rate of the inexact Newton's method. This has been discussed in detail in references [82, 142, 49]. Here, we keep the relative good choice of Eq. (3.2) and studies the influences of Jacobian error to highlight the importance of using preconditioner and in guiding the choice of linear system convergence tolerance. To study this, a model linear system with symmetric coefficient matrix $\mathbf{C}$ and an error matrix $\mathbf{C}^e$ Eq. (3.3) is constructed. The linear system is solved using GMRES with the coefficient matrix vector inner product calculated as

$$\left[\frac{\delta_{ij}}{\Delta t} + \mathbf{C}_{ij} + \epsilon \mathbf{q}_i^k \mathbf{C}_{lij}^e \right] \mathbf{q}_j^k, \quad (3.5)$$

for the k th GMRES iteration. As a model system of Eq. (3.3), Eq. (3.5) is constructed to obtain a desired condition number of $\mathbf{C}$. Here the condition number is the ratio between the largest and smallest eigenvalues of the symmetric system. The exact solution of the linear system is set to be a vector with ones on every entity with the corresponding right hand side vector calculated simply by matrix-vector inner product with $\epsilon = 0$.

In constructing the symmetric coefficient matrix $\mathbf{C}_{ij}$ with a specific condition number ζ , the eigenvalues of the system $\hat{\lambda}_k$ are firstly generated with random numbers of $(0, 1]$, with maximum and minimum eigenvalues $\hat{\lambda}_M$ and $\hat{\lambda}_m$, respectively. The eigenvalues are then scaled as

$$\tilde{\lambda}_k = \hat{\lambda}_k^{\log(\zeta)/\log(\hat{\lambda}_M/\hat{\lambda}_m)}, \quad (3.6)$$

to maintain that the ratio between the maximum and minimum values of $\tilde{\lambda}_k$ is ζ . Then, the eigenvalues $\tilde{\lambda}_k$ are normalized to make sure their multiplication product is one. At last, the symmetric coefficient matrix $\mathbf{C}_{ij}$ is constructed by multiplying orthonormal eigenvectors generated using random numbers and the Gram-Schmidt orthogonalization. The three dimensional tensor $\mathbf{C}^e$ is constructed by random numbers that belong to $(0, 1]$ and divided by the vector size of $\mathbf{q}^k$. The L_2 norm of the error vectors normalized by the solution vector norm is calculated as the metric of the influences of errors.

Fig. 3.1 presents the error norms of GMRES solutions with system size of 100 and different condition numbers. A time step $\Delta t = 10^{100}$ is adopted to rule out the the influences of the term of $\delta_{ij}/\Delta t$ first, which mimics the case of steady-state simulations with very large time steps. The GMRES is adopted to solve the linear system with a convergence tolerance of 10^{-16} and a maximum GMRES iteration number equal to the system size.

To reduce specificity of the system caused by a specific combination of random numbers, 200 different linear systems generated by random numbers are solved and their algebraic averaged, maximum and minimum error norms are presented. This clearly illustrates that the solution error norms increase as the condition number increases. For one hand, this highlights the necessity for developing good preconditioners, which not only accelerate the convergence of GMRES but also decrease the error caused by the finite difference approximation of Jacobian matrix. For another, the conclusion that the influence of the finite difference approximation is minor [142, 49] may need to be reevaluated in very stiff problems. There are possible ways to minimize this influence such as using an exact Jacobian matrix [158, 121] or using a complex-step derivative approximation [95].

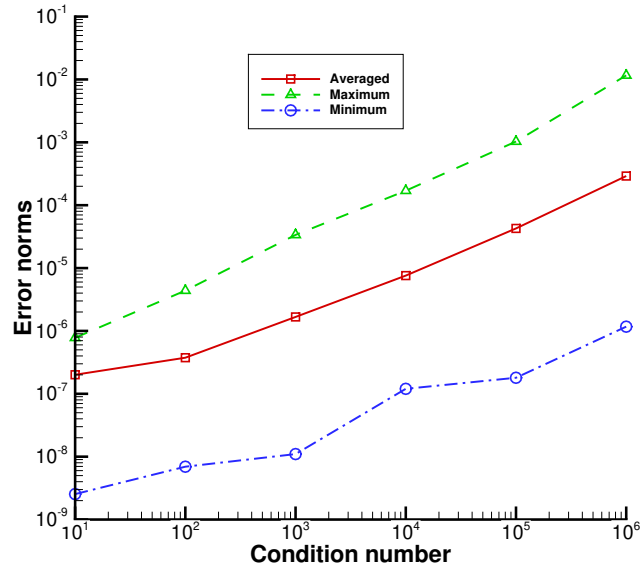


Figure 3.1: Influence of finite difference Jacobian approximation errors with different condition numbers. The system is generated using random numbers. The *Averaged*, *Maximum*, *Minimum* indicate the algebraic, maximum and minimum errors of 200 different random linear systems to reduce the specificity of the system.

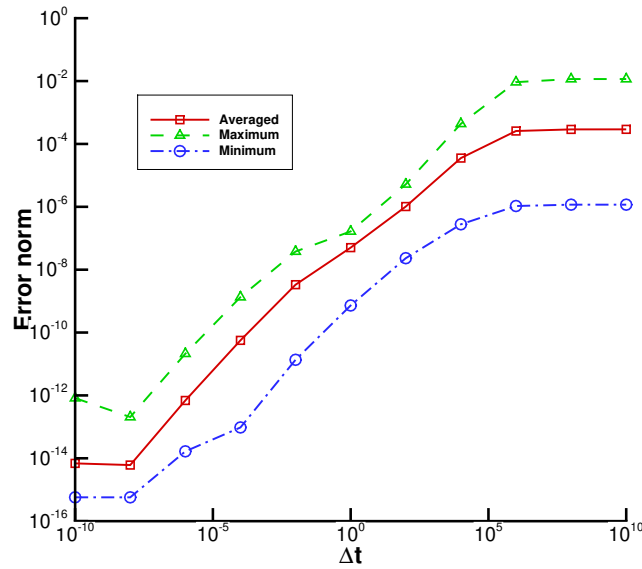


Figure 3.2: Influence of finite difference Jacobian approximation errors with different time steps. The *Averaged*, *Maximum*, *Minimum* indicate the algebraic, maximum and minimum errors of 200 different random linear systems to reduce the specificity of the system.

Fig. 3.2 shows the influence of Δt on the solution errors obtained with a condition number of 10^6 for the **C**. When the Δt is larger than 10^6 the influence of the Δt becomes negligible. The solution error norm will decrease as the time step Δt becomes smaller. These results illustrate that the influence of the finite difference error become smaller and smaller as the time step Δt decreases. Therefore, it is unlikely that the finite difference approximation of the Jacobian matrix seriously degrades the convergence in unsteady simulations with small time step.

The tests on Jacobian error also suggest that using a very small GMRES convergence tolerance may not be reasonable, especially in stiff and steady-state simulations since it makes no sense to require the GMRES to converge to a level smaller than the error caused by approximation. Based on this observation and numerical tests on efficiency, we choose to terminate the GMRES iteration after the residual drops by 5×10^{-2} .

3.2 Constructing efficient preconditioners

Here we choose to decouple the Jacobian matrix into diagonal and off-diagonal parts and use a simple relaxed Jacobi iteration to apply the preconditioning, which is named the block relaxed Jacobi iterative (BRJ) preconditioner. We choose to use an iterative preconditioner mainly because it is easier to achieve a low memory implementation. Jacobi iteration is adopted because the corresponding subroutines in Nektar++ are not implemented element by element, which makes the implementation of Gauss-Seidel like iterations not easy.

The BRJ preconditioner is presented in Section 3.2.1. The influences of the advection and diffusion Jacobian contributions on the BRJ preconditioner are studied

in Section 3.2.2 using eigenspectra analysis and numerical experiments.

3.2.1 Block relaxed Jacobi iterative (BRJ) preconditioner

The whole Jacobian matrix $\partial \mathbf{N} / \partial \mathbf{u}$ can be divided into $N_e \times N_e$ small blocks, where N_e is the number of elements. The matrix $\partial \mathbf{N} / \partial \mathbf{u}$ can be decoupled into

$$\frac{\partial \mathbf{N}}{\partial \mathbf{u}} = \hat{\mathbf{L}} + \hat{\mathbf{D}} + \hat{\mathbf{U}}, \quad (3.7)$$

where $\hat{\mathbf{L}}$, $\hat{\mathbf{D}}$ and $\hat{\mathbf{U}}$ correspond to the lower ($e_1 < e_2$), diagonal ($e_1 = e_2$) and upper ($e_1 > e_2$) block parts of $\partial \mathbf{N} / \partial \mathbf{u}$, respectively. Then the product $\mathbf{P}^{-1} \mathbf{q}$ is implicitly calculated through the relaxed (or weighted) Jacobi iteration,

$$\begin{aligned} \hat{\mathbf{q}}^0 &= \mathbf{0}, \\ \hat{\mathbf{q}}^j &= \omega \hat{\mathbf{D}}^{-1} [\mathbf{q} - (\hat{\mathbf{L}} + \hat{\mathbf{U}}) \hat{\mathbf{q}}^{j-1}] + (1 - \omega) \hat{\mathbf{q}}^{j-1}, \quad j = 1, 2, \dots, J, \end{aligned} \quad (3.8)$$

and $\hat{\mathbf{q}}^J$ is the preconditioned vector with J the number of iterations for the BRJ preconditioner. Currently the only case with $\omega = 1$ is studied.

The e_1 -th row and e_2 -th column block is $\partial \mathbf{N}_{e_1} / \partial \mathbf{u}_{e_2}$ where $\mathbf{N}_{e_1}$ is the nonlinear system residual of the e_1 -th element and $\mathbf{u}_{e_2}$ is the independent variables of the e_2 -th element. The matrix $\partial \mathbf{N}_{e_1} / \partial \mathbf{u}_{e_2}$ can be derived from Eqs (2.18) and (2.32) as

$$\frac{\partial \mathbf{N}_{e_1}}{\partial \mathbf{u}_{e_2}} = \mathbf{I} \delta_{e_1 e_2} - \Delta t a^{mm} \mathbf{M}^{-1} \frac{\partial \mathcal{L}_{\delta, e_1}}{\partial \mathbf{u}_{e_2}}, \quad (3.9)$$

$$\begin{aligned}
\frac{\partial \mathcal{L}_{\delta, e_1}}{\partial \mathbf{u}_{e_2}} = & \sum_{j=1}^d \mathbf{B}_{e_1}^T \mathbf{D}_{j, e_1}^T \Lambda \left[w_{e_1} J_{e_1} \left(\frac{\partial \mathbf{H}_{j, e_1}}{\partial \mathbf{Q}_{e_1}} \right)_{\nabla \mathbf{Q}} \right] \mathbf{B}_{e_1} \delta_{e_1 e_2} + \\
& \sum_{i=1}^d \sum_{j=1}^d \mathbf{B}_{e_1}^T \mathbf{D}_{j, e_1}^T \Lambda \left[w_{e_1} J_{e_1} \left(\frac{\partial \mathbf{H}_{j, e_1}}{\partial \nabla_i \mathbf{Q}_{e_1}} \right)_{\mathbf{Q}} \right] \mathbf{D}_{i, e_1} \mathbf{B}_{e_1} \delta_{e_1 e_2} + \\
& (\mathbf{B}_{e_1}^T \mathbf{M}_c)^T \Lambda \left[w^\Gamma J^\Gamma \left(\frac{\partial \hat{\mathbf{H}}_{e_1}^n}{\partial \hat{\mathbf{Q}}^{+/-}} \right)_{\nabla \mathbf{Q}} \right] \frac{\partial \hat{\mathbf{Q}}^{+/-}}{\partial \mathbf{Q}_{e_2}} \mathbf{B}_{e_2} + \\
& \sum_{j=1}^d (\mathbf{B}_{e_1}^T \mathbf{M}_c)^T \Lambda \left[w^\Gamma J^\Gamma \left(\frac{\partial \hat{\mathbf{H}}_{e_1}^n}{\partial \nabla_j \hat{\mathbf{Q}}^{+/-}} \right)_{\mathbf{Q}} \right] \frac{\partial \nabla_j \hat{\mathbf{Q}}^{+/-}}{\partial \nabla_j \mathbf{Q}_{e_2}} \mathbf{D}_{j, e_2} \mathbf{B}_{e_2} \\
& + \sum_{j=1}^d \mathbf{B}^T \mathbf{D}_j^T \mathbf{J}^T \Lambda \left[w^\Gamma J^\Gamma \left(\frac{\partial \hat{\mathbf{S}}_{e_1}^n}{\partial \hat{\mathbf{Q}}^{+/-}} \right) \right] \frac{\partial \hat{\mathbf{Q}}^{+/-}}{\partial \mathbf{Q}_{e_2}} \mathbf{B}_{e_2},
\end{aligned} \tag{3.10}$$

where $\partial \hat{\mathbf{Q}}^{+/-} / \partial \mathbf{Q}_{e_2}$ is the interpolation matrix from $\mathbf{Q}_{e_2}$ to $\hat{\mathbf{Q}}^{+/-}$ which is independent of $\mathbf{Q}$ itself, the superscript $+/-$ indicates that only one of them is a non-zero matrix for a specific e_2 . The term $\partial \nabla_j \hat{\mathbf{Q}}^{+/-} / \partial \nabla_j \mathbf{Q}_{e_2}$ is similarly an interpolation matrix. The first two terms on the right-hand side of Eq. (3.10) are element integration evaluations, and the remaining three terms are element boundary terms. Owing to the compact properties of the DG scheme, the off-diagonal blocks ($\delta_{e_1 e_2} = 0$) only come from the element boundary terms, while both the element and element boundary terms have contributions to the diagonal blocks ($\delta_{e_1 e_2} = 1$). Another good property is that only the terms of fluxes Jacobian, which are related to $\hat{\mathbf{H}}$ and $\mathbf{H}$, are nonlinear. In addition, these nonlinear flux Jacobian terms are decoupled from each other. We will utilize these properties in Section 3.2.2 to design the implementation to reduce the memory and computational costs.

3.2.2 Approximations of matrices in preconditioning

Although it may lead to fewer number of iterations to converge if using a preconditioning matrix closer to the Jacobian matrix, it may not be the most efficient choice in terms of computational time considering the overhead of computational costs in

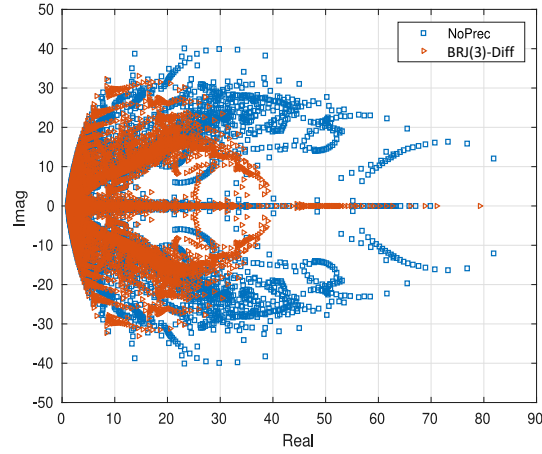
performing the preconditioning operations. Since the real metric in approximating of the preconditioning matrices is to find the balance between decreasing the cost of preconditioner and decreasing the number of iteration to converge.

The preconditioning matrix of the BRJ preconditioner is actually implicitly constructed by the iteration in Eq. (3.8). As the preconditioning iteration number J tends to ∞ the preconditioning matrix becomes closer and closer to the Jacobian matrix and the eigenvalues of the preconditioned system will become closer and closer to one. However, this does not necessarily lead to a good preconditioner because of the large costs in preconditioning. To reduce the preconditioning costs, two approximations could be introduced by using a limited iteration number J and using inexact matrices of $\hat{\mathbf{D}}$, $\hat{\mathbf{L}}$ and $\hat{\mathbf{U}}$. We will explore efficient ways in these approximations to reduce the computational costs without obviously degrading the preconditioning effect.

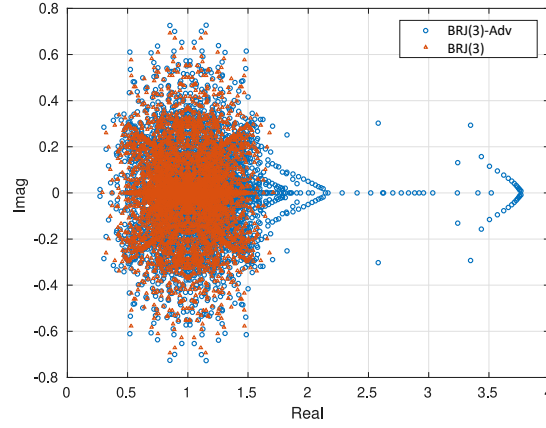
Influence of different parts on preconditioning effect

Firstly, the influence of different terms, the advection Jacobian and the diffusion Jacobian in $\hat{\mathbf{D}}$, $\hat{\mathbf{L}}$ and $\hat{\mathbf{U}}$, on the iteration number to converge are studied to identify which part could be further approximated. This study is performed numerically based on a flat plate boundary layer test case with $\text{Ma} = 0.1$, $\text{Re} = 1.6 \times 10^6$ and highly stretched meshes, which is detailed in Section 5.4.2. Both the eigenvalue distributions and the number of iteration needed to converge are studied here.

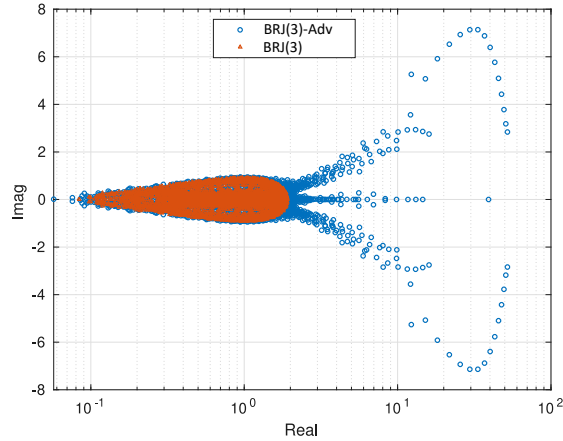
Fig. 3.3 presents the eigenspectra distributions of the preconditioned system with different approximations of the $\hat{\mathbf{D}}$, $\hat{\mathbf{L}}$ and $\hat{\mathbf{U}}$ in Eq. (3.8). The corresponding numbers of iteration for the GMRES method to converge by 5 orders of magnitude with these preconditioners are presented in Tab. 3.1. Although not proved in theory, the



(a) Eigenspectra of Jacobian matrix and BRJ(3)-Diff preconditioned matrix with $\Delta t = 5 \times 10^{-5}$.



(b) Eigenspectra of BRJ(3) and BRJ(3)-Adv preconditioned matrix with $\Delta t = 5 \times 10^{-5}$.



(c) Eigenspectra of BRJ(3) and BRJ(3)-Adv preconditioned matrix with $\Delta t = 10^{-3}$.

Figure 3.3: Eigenspectra distributions based on different approximations of the preconditioning matrices. *NoPrec* represents the one without preconditioning, *BRJ(3)* represents the one preconditioned by the BRJ with $J = 3$, the suffixes *Diff* and *Adv* represents the matrix BRJ constructed using only the advection and diffusion flux Jacobian.

Time step	BRJ(3)	BRJ(3)-Adv	NoPrec
5.0×10^{-5}	26	37	216
1.0×10^{-3}	126	277	/

Table 3.1: The number of iteration for the GMRES method to converge by 5 orders of magnitudes with different time steps and different preconditioners.

results in Fig. 3.3 and Tab. 3.1 indicate that more clustered eigenspectra distributions indicate a better preconditioner in reducing the number of iterations. Therefore, the clustering of eigenspectra is studied to reflect the effect of preconditioning. The suffix *Adv* represents a BRJ preconditioner based on the time discretization and advection terms (ignoring the diffusion term) and the suffix *Diff* represents the one based on the time discretization and diffusion terms (ignoring the advection term). Compared with the eigenspectra without preconditioning, the one with the BRJ(3) preconditioner is largely clustered for the case of $\Delta t = 5.0 \times 10^{-5}$. The corresponding number of iteration for GMRES to converge is reduced from 216 to 26 because of the preconditioner, as shown in Tab. 3.1. If the advection flux Jacobian is ignored, the set of eigenvalues is only slightly clustered compared with the one without preconditioning. A possible explanation is that advection dominates in most regions of the problem. Thus, advection contributions should not be ignored since this would lead to a very poor preconditioner. Further evidence of this is that the GMRES can not converge within 300 iterations (not reported in Tab. 3.1) using a preconditioner ignoring the advection Jacobian. As for the BRJ(3)-Adv, most eigenvalues are well clustered without obvious difference from the BRJ(3) results but there are still some of them obviously away from others. The iteration number becomes 37, which is larger than the one with BRJ(3). This indicates that the contribution from the diffusion flux is important at least for clustering some eigenvalues. This is more clearly illustrated in the case with a large time step of $\Delta t = 1.0 \times 10^{-3}$. The eigenvalues not well clustered are further away and the corresponding iteration

number is more than twice larger compared with BRJ(3) preconditioner (from 126 to 277). This indicates that both the advection and diffusion flux play important roles especially for simulations with large Δt . Therefore, completely ignoring the diffusion contribution could lead to a poor preconditioner especially for simulations with large Δt .

We conduct further numerical studies of the contributions from different terms on the right-hand side of Eq. (3.10). It was found that the last two terms on the right-hand side of Eq. (3.10) have a small impact on the preconditioning performance. In our test, neglecting the last two terms in the compressible Poiseuille flow test case of Section 4.4.2 with $P = 2$ and 20^2 meshes only increases the total number of required GMRES iterations in the first 10 time steps from 1136 to 1144. Similar conclusion can also be drawn based from the tests in Fig. 3.4. One concludes that, although completely ignoring the diffusion contributions degrades the performance of the preconditioner, we can still simplify the Jacobian approximation by ignoring other terms while keeping the preconditioning effect almost not influenced.

Efficient implementation of the preconditioner

Eqs. (3.9) and (3.10) show that all the nonlinear parts of the Jacobian matrix involve evaluation of the flux Jacobian at quadrature points inside the elements and on the element interfaces. All other operations are linear and can be obtained analytically. The analytical expressions of the flux Jacobian inside the elements, $(\partial \mathbf{H}_{j,e_1} / \partial \mathbf{Q}_{e_1})_{\nabla \mathbf{Q}}$ for example, are also available. However, the numerical flux Jacobian matrices on element interfaces, such as $\partial \hat{\mathbf{H}}_{e_1}^n / \partial \hat{\mathbf{Q}}^\pm$, are much more complicated. The advection terms are usually calculated by approximate Riemann fluxes, which are sometimes not differentiable. In addition, on physical boundaries, there is a complex relation

between $\hat{\mathbf{Q}}^+$ and $\hat{\mathbf{Q}}^-$, which are implicitly defined by the physical boundary conditions. Considering the significant amount of choices for numerical Riemann fluxes and boundary conditions [159], it would be difficult and time consuming account for and implement the corresponding Jacobian matrix for each of these choices.

To simplify the coding complexity, a hybrid analytical and numerical strategy is adopted in calculating the $\hat{\mathbf{D}}$, $\hat{\mathbf{L}}$ and $\hat{\mathbf{U}}$ matrices. Numerical flux Jacobians on element interfaces, such as $\partial \hat{\mathbf{H}}_{e_1}^n / \partial \hat{\mathbf{Q}}^\pm$, are calculated using finite difference approximation

$$\left(\frac{\partial \hat{\mathbf{H}}_{e_1}^n}{\partial \hat{\mathbf{Q}}^\pm} \right)_{\nabla \mathbf{Q}} \simeq \frac{\hat{\mathbf{H}}_{e_1}^n(\hat{\mathbf{Q}}^\pm + \chi \mathbf{e}_j, \hat{\mathbf{Q}}^\mp, \nabla \hat{\mathbf{Q}}) - \hat{\mathbf{H}}_{e_1}^n(\hat{\mathbf{Q}}^\pm, \hat{\mathbf{Q}}^\mp, \nabla \hat{\mathbf{Q}})}{\chi}, \quad (3.11)$$

where $\mathbf{e}_j$ is the vector with one on the j th column but zeros on others. The parameter χ is the multiplication of $\sqrt{\hat{\epsilon}}$ with the geometric average of density, momentum magnitude and energy for the corresponding variables. Specific physical boundary conditions are imposed on physical boundaries in the $\hat{\mathbf{H}}_{e_1}^n$ evaluation so that the Jacobian matrices of the boundary conditions are already included in Eq. (3.11). Because $\hat{\mathbf{H}}^n$ on each quadrature point only depends on the local independent variable and independent of each other, the evaluation of Eq. (3.11) on each quadrature point can be performed in parallel. In total of $2N_{var}$ numerical flux evaluations on all the quadrature points of element boundaries and N_{var} boundary treatment operations are required to calculate $\partial \hat{\mathbf{H}}_{e_1}^n / \partial \hat{\mathbf{Q}}^\pm$ without the need for parallel communications. The computational cost and storage consumption of the matrix $\partial \hat{\mathbf{H}}_{e_1}^n / \partial \hat{\mathbf{Q}}^\pm$ is small compared with those in other parts of the implicit solver, as discussed in Section 4.2.

In the following, a partially matrix-free implementation of the BRJ preconditioner is adopted to reduce the computational cost and memory required. The diagonal term, $\hat{\mathbf{D}}$, is explicitly formed, inverted and stored. Following the ideas described in reference [67], single-precision data type, continuous memory operations and matrix

padding are used to reduce the computational cost and optimize cache performance in the implementation of these evaluations. The $(\hat{\mathbf{L}} + \hat{\mathbf{U}}) \hat{\mathbf{q}}^{l-1}$ operator in Eqs. (3.8) is implemented in an (elemental) matrix-free fashion with only the $\partial \hat{\mathbf{H}}_{e_1}^n / \partial \hat{\mathbf{Q}}^\pm$ calculated by Eq. (3.11). The matrices of $\partial \hat{\mathbf{H}}_{e_1}^n / \partial \hat{\mathbf{Q}}^\pm$ are stored for the evaluation of the matrix-free operator $(\hat{\mathbf{L}} + \hat{\mathbf{U}}) \hat{\mathbf{q}}^{l-1}$. The remaining numerical flux Jacobians on element interfaces, $(\partial \hat{\mathbf{H}}_{e_1}^n / \partial \nabla_j \hat{\mathbf{Q}}^{+/-})_{\mathbf{Q}}$ and $(\partial \hat{\mathbf{S}}_{e_1}^n / \partial \hat{\mathbf{Q}}^{+/-})$ in Eq. (3.10) could also be calculated in a similar way.

The performance of the BRJ preconditioner is numerically verified in the case of a turbulent flow past a cylinder, which is detailed in Section 5.4.4. The number of GMRES iterations and computational time are compared in Fig. 3.4 for two different time steps: $\Delta t = 0.001$ and $\Delta t = 0.01$. The system will become stiffer with a larger time step of $\Delta t = 0.01$. In this case, the number of GMRES iterations is decreased by more than 5 times when using BRJ(5) or BRJ(7) compared with that using BRJ(1). Please note that the BRJ(1) preconditioner is equivalent to the block-Jacobi preconditioner [67]. The computational time for the simulation with BRJ(5) is about 2.9 times smaller than that with BRJ(1). Based on the above tests, $J = 7$ is used in this paper unless otherwise specified.

The curves with *VisPrec* means the last two terms on the right-hand side of Eq. (3.10) are included in the BRJ preconditioner. Compared with the one without these two terms, the GMRES iteration number is almost identical, which means neglecting these terms has very small impact on the preconditioning effect. However, including the last two terms obviously increases the overall computational costs by at most 2.7 times for BRJ(3) and BRJ(5), which almost counterbalances the possible speedup by increasing the BRJ iteration number. This is mainly because in considering the last two terms $\nabla \mathbf{Q}$ has to be calculated, which significantly increases the costs. Therefore, the last two terms on the right-hand side of Eq. (3.10) are by

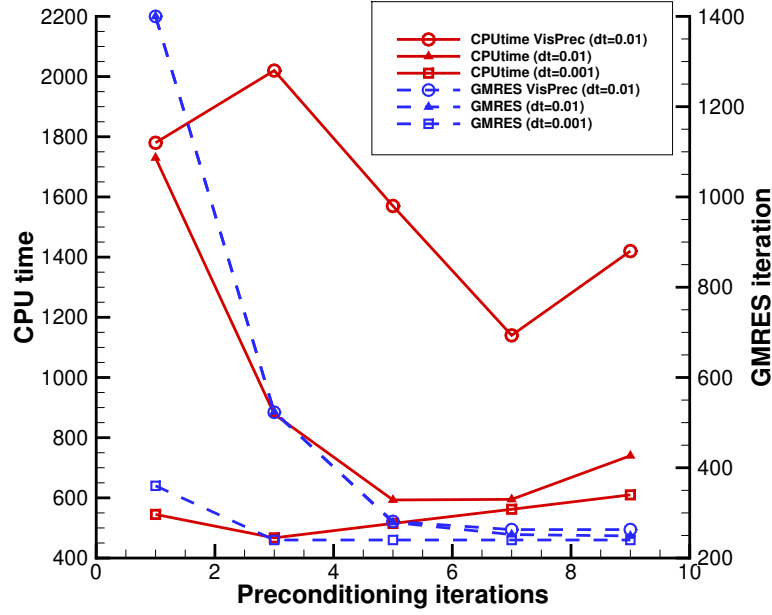


Figure 3.4: Performance comparison of the BRJ preconditioner with different iteration numbers and implementations.

default neglected in the BRJ preconditioner unless otherwise specified.

In summary, a block relaxed Jacobi iterative preconditioner is proposed with a hybrid numerical and analytical method in constructing the preconditioning matrices. The memory consumption and computational costs of the preconditioner are optimized by techniques including adopting a partially matrix-free implementation, using single data precision and proper approximations of the preconditioning matrices. The optimization significantly reduces the computational cost and enables using more preconditioning iterations to effective clustering eigenspectra. The resulting preconditioner is able increase the efficiency by about 3 times compared with a simple diagonal preconditioner while keeping the memory consumption low.

Chapter 4

Implementation and verification

Section 4.1 describes the software design and an efficient implementation of the implicit solver, which is based on an existing explicit in time solver with DG and LDG spatial discretizations. Section 4.2 assesses the performance of the solver. The memory requirements of the solver are then estimated in Section 4.3. At last, some basic test cases are adopted to verify the implementation of the solver in Section 4.4.

4.1 Software design in Nektar++

This section presents some details of the software implementation of the implicit solver. The structure of the Nektar++ libraries and the general procedure followed in constructing a flow solver are introduced in Section 4.1.1. Section 4.1.2 describes the software design of the implicit solver in detail.

4.1.1 Introduction to Nektar++

Nektar++ mainly consists of the libraries and several built-in solvers. The Nektar++ libraries provide a structured hierarchy of C++ classes, which offers the major functions needed for spectral/*hp* element methods. The six core libraries are:

- **LibUtilities**: polynomial basis functions, basic data classes such as array and matrix, linear algebra functions, time integration schemes, parallel communication and I/O.
- **StdRegions**: standard (or reference) elements, polynomial approximation of the solution using basis functions, operations on standard elements such as derivation and integration operations.
- **SpatialDomains**: mapping from standard elements to physical elements, grid metrics and grid Jacobian.
- **LocalRegions**: extension of operations on real space physical elements, inheriting from **StdRegions** and **SpatialDomains**.
- **MultiRegions**: collection of physical elements comprising Ω , adjacent relations between elements, elemental and global variable mapping.
- **SolverUtils**: high level building blocks of solvers, drivers controlling simulation procedures, general equation systems, specific numerical scheme for different terms (WeakDG for advection terms for example).

In the latest version of Nektar++, four additional libraries have been included to enrich the libraries with functionalities encompassing efficient operator evaluation, quasi-3D simulations, high-order curved mesh generation and post-processing [107].

The Nektar++ libraries provide main functions related to data storage (such as matrix and array), linear/nonlinear algebra (such as GMRES), geometry related calculations and numerical projection (such as DG), but nothing related to specific equations.

The main procedures for building a solver are briefly introduced using the explicit solver of the NS equations as an example. Besides the main solver class (`CompressibleFlowSolver`) which controls the solving procedure, an equation system class is required. The `EquationSystem` classes (`EulerCFE` or `NavierStokesCFE`) are instantiated and initialized dynamically based on user inputs using a factory method pattern [24], which is also extensively used for the dynamic object creation of classes in the solver. The equation system class inherits instantiations of classes related to geometry information, solution approximation, time integration and others from `EquationSystem` classes in the libraries such as `UnsteadySystem` in `SolverUtils`. Thus the main functionality of the `EquationSystem` class is to offer equation system related functions and to form spatial discretization operators using numerical methods from the libraries such as, `AdvectionWeakDG`. Fig. 4.1 illustrates the class structure of the explicit and implicit solvers. The equation of state (`EquationOfState`), boundary conditions (`CFSBndCond`), Riemann flux (`RiemmanSolver`), shock capturing method (`ArtificialDiffusion`), forcing term (`Forcing`), advection flux (function pointer `F`) and diffusion flux (function pointer `G`) are instantiated or implemented in the equation system class. Specific numerical schemes like the weak DG scheme for advection terms (`AdvectionWeakDG`) and the interior penalty method for diffusion terms (`DiffusionIP`) are instantiated to calculate $\nabla \cdot \mathbf{F}$ and $\nabla \cdot \mathbf{G}$ in Eq. (2.1) provided with all these equation system related functions (`F` and `G`). Finally the spatial discretization operator $L_\delta(\mathbf{u}, t)$ is linked to the time integration class using the pointer-to-function `A` and the explicit solver is complete.

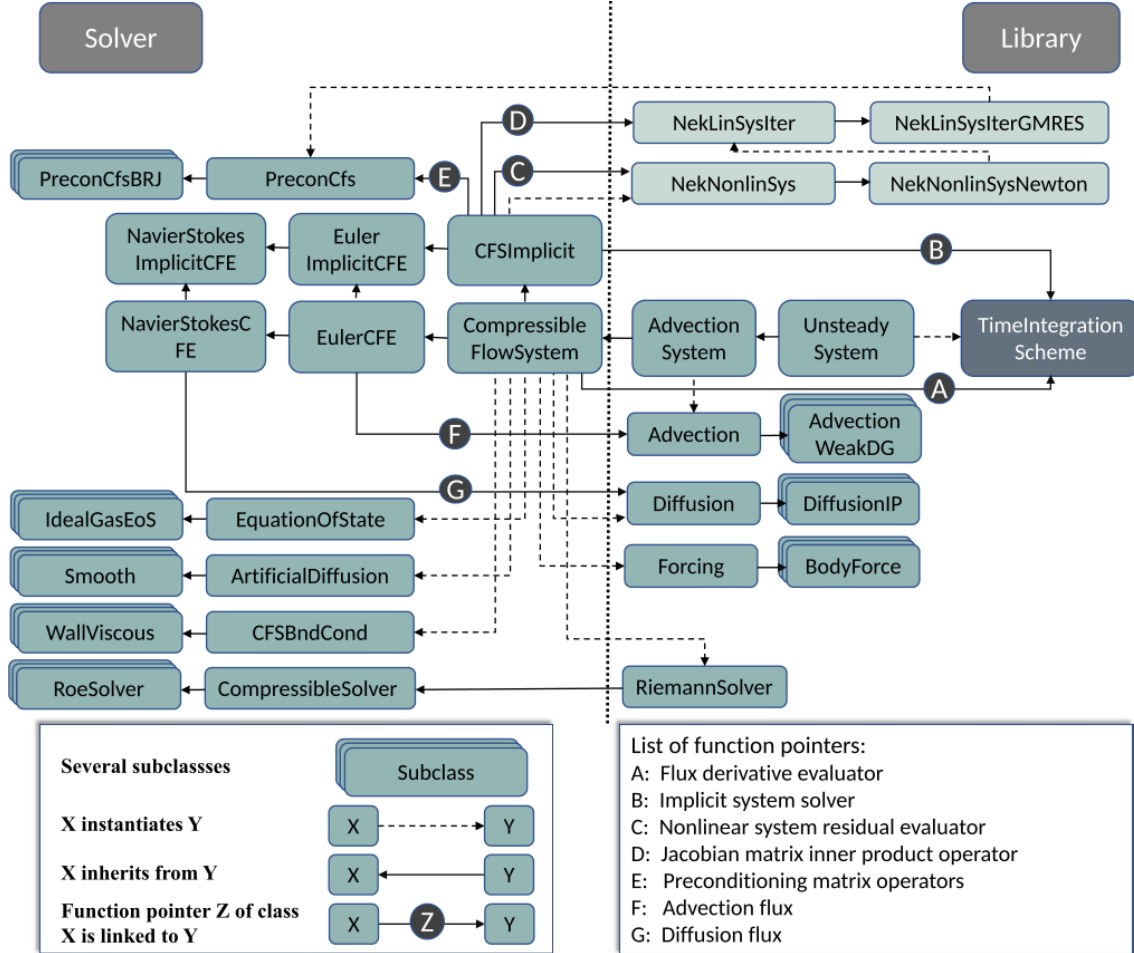


Figure 4.1: Class structure of the explicit and implicit solvers. The equation system classes (`EulerCFE` or `NavierStokesCFE`) contain access to the main functionalities of the solver, such as time integration, and the solution fields. They make use of numerical methods from the libraries, such as `AdvectionWeakDG`, and equations system related functions, such as the advection flux (F) and diffusion flux (G), to form the spatial discretization operator A. The spatial operator A and the time integration method form the explicit solver using the method of lines. For the implicit solver, additional classes like the Newton solver (`NewtonSolver`), GMRES solver (`GMRES`) and linear algebra solver of preconditioners (e.g. `PrecondBRJ`) are instantiated. Together with operators related to the nonlinear system (C), the Jacobian matrix (D) and preconditioning matrix (E), an implicit system solver (B) is constructed, which is linked to the implicit time integration scheme to form the implicit solver.

4.1.2 Implicit solver architecture and implementation

In the following the analysis of different building blocks of the implicit solver and their hierarchical structures are presented. Then, the implementation of the operations related to the NS equations is discussed.

Analysis of the implicit solver architecture

We first analyze the different building blocks of the solver from the point of view of library and software design and place these blocks into the right hierarchical levels in the libraries and the solver. The Newton's method, GMRES method and, in general, any mathematical methods independent of the type of equations, are placed in the `LibUtilities` as shown on the right upper corner of Fig. 4.1. Meanwhile operations that depend on the equations and the numerical schemes, e.g. the implicit system solver (function pointer B in Fig. 4.1), nonlinear system residual evaluator (function pointer C in Fig. 4.1), Jacobian matrix operation (function pointer D in Fig. 4.1) and preconditioning matrix operations (function pointer E in Fig. 4.1), are coded at the solver level in the `CompressibleFlowSystem` class (or their children classes) and they are linked to the corresponding classes from the libraries using pointers to these functions.

This structure provides a general framework for developing implicit solvers for different equation systems. Provided with equation system related functions, an implicit solver can also be constructed for other equations systems using the classes programmed in the libraries.

Implementation of the NS equations related operations

The top level class structure of the implicit solver is illustrated in Fig. 4.1. Even though only a nonlinear system solver is added to the Nektar++ libraries, the nonlinearity and coupling of the NS equations makes the design of the Jacobian matrix and preconditioning matrix related operations much more difficult than that of the existing implicit solvers of linear decoupled equation systems for the following reasons: a) the nonlinearity means the Jacobian and preconditioning matrices should be updated along with the simulation, which makes the computational speed of these operations vital for the overall efficiency; b) The coupling of eigenvalues of different equations increases the stiffness of the system; c) The coupling of different equations leads to a much larger Jacobian matrix, especially in large-scale 3D simulations. We will discuss the design of equation related operations in the following.

Nonlinear system residual The nonlinear system residual, $\mathbf{N}(\mathbf{u})$, is easily available from an existing explicit solver. However, for the sake of efficiency it is important to optimize its evaluation since the implicit solver requires repeated calculations of the residual. Techniques such as sum-factorization [25, 113] and collections of similar linear algebra operations [107], have been studied and adopted in Nektar++ to speed-up the evaluation of $\mathbf{N}(\mathbf{u})$.

Jacobian matrix calculation Explicitly calculating and storing the Jacobian matrix $\partial\mathbf{N}/\partial\mathbf{u}$ accurately is expensive both in computational cost and memory consumption, as will be discussed in Section 4.3.

As highlighted by in Eq. (3.1), a Jacobian-free technique is adopted to calculate the inner product of the Jacobian matrix with a vector, which is the only Jacobian

matrix related operation required in the Krylov methods. From the point of view of software design, this replaces a numerical method and a set of operations that are highly dependent of the type of equation solved by a more general finite difference scheme and an existing residual evaluator $\mathbf{N}(\mathbf{u})$. The Jacobian-free code will be generally applicable to other equations systems if provided with the corresponding $\mathbf{N}(\mathbf{u})$.

Calculation of preconditioning matrix Constructing a good preconditioning matrix $\mathbf{P}$ is highly dependent on appropriately approximating the Jacobian matrix $\partial\mathbf{N}/\partial\mathbf{u}$. Although some explorations on completely matrix-free preconditioners have been reported in the literature, the most efficient and most widely used preconditioners still require explicitly evaluating at least part of $\partial\mathbf{N}/\partial\mathbf{u}$ for low to medium order simulations [55, 82]. Here a J step block relaxed Jacobi iterative preconditioner, BRJ(J), is developed instead of the widely adopted incomplete LU factorization method (ILU) mostly due to its lower memory requirements. The BRJ preconditioner uses a simple relaxed Jacobi iteration to implicitly perform the preconditioning. It is implemented in a partially matrix-free way, with only the diagonal block constructed and stored, while the off-diagonal operations are calculated on the fly. As discussed in Section 3.2, a hybrid analytical and numerical method is adopted to calculate the Jacobian matrix, in which the Jacobian matrices of the numerical fluxes on element boundaries are calculated numerically while the flux Jacobian matrices inside the element and other linear operations are calculated analytically. This offers an efficient and accurate way of calculating the preconditioning matrices. In addition, it greatly reduces the complexity in programming. For simulations with shocks, the terms with $\partial\mu_{av}/\partial\mathbf{u}$ and $\partial\kappa_{av}/\partial\mathbf{u}$ are neglected in the preconditioning matrices calculations. As will be shown in Section 4.4.4, the implicit solver is able to stably and efficiently simulate discontinuous flows neglecting these two terms.

The main simulation flowchart of the implicit solver is given in Fig. 4.2, which also shows the flow between different classes.

Initialization	in CompressibleFlowSolver
Time step loop n	in UnsteadySystem
Runge-Kutta loop $m = 1, \dots, M$	in TimeIntegrationScheme
Calculate source term $\mathbf{S}_m$	in TimeIntegrationScheme
Newton iteration l	in NewtonSolver
Calculate residual $\mathbf{N}(\bar{\mathbf{u}}^l)$	in CompressibleFlowSystem
GMRES iteration k	in NekLinSysGMRES
Calculate search vector $\mathbf{q}_k$	in NekLinSysGMRES
BRJ iteration $j = 1, \dots, J$	in PrecondBRJ
Calculate $\hat{\mathbf{q}}^j = \hat{\mathbf{D}}^{-1}(\mathbf{q}_k - (\hat{\mathbf{L}} + \hat{\mathbf{U}})\hat{\mathbf{q}}^{j-1})$	in CompressibleFlowSystem
Calculate $\partial\mathbf{N}/\partial\mathbf{u} \cdot \hat{\mathbf{q}}^J$	in CompressibleFlowSystem
Calculate linear system residual	in NekLinSysGMRES
Calculate $\bar{\mathbf{u}}^{l+1}$ by linear combination of $\mathbf{q}_k$	in NekLinSysGMRES
Calculate $\mathbf{u}^{n+1}$	in TimeIntegrationScheme
Output and finalization	in CompressibleFlowSolver

Figure 4.2: Nassi–Shneiderman diagram of the implicit solver with the corresponding class names. Brown: library class ; cyan: solver class.

4.2 Performance analysis

The implicit solver can significantly relax the constraint on the time step. However, it also requires much larger computational cost and memory consumption. The computational performance of the implicit solver is analyzed in this section, in which the performances based on different linear algebra libraries, for different the orders of accuracy and for different parallel scales are studied. The evaluations are performed on a CentOS 7.7.1908 using standard -O3 compiler optimizations with the gcc 5.4.0 C++ compiler. The evaluations are run on the HPC cluster of Imperial College London, with each node is equipped with two Intel Xeon E5-2620 CPUs and 120

GB RAM. There are in total 12 physical cores on each node but are virtualized into 24 cores. The detailed computational time for some components of the solver is obtained using Oracle Developer Studio 12.6 in Section 4.2.1 and Section 4.2.2.

4.2.1 Performance based on different linear algebra libraries

There are many matrix related operations in setting up and applying the preconditioner, the efficient implementation of which is vital for the overall solver. Using more preconditioning iterations may not be the most efficient choice since the preconditioner may cause obvious overhead in computational costs. Therefore, reducing the ratio of computational costs between a preconditioning iteration and the system residual evaluation operation $\mathbf{N}(\mathbf{u})$ is a vital premise that enables more preconditioning iterations. These matrix related operations in preconditioning are implemented with the BLAS operations. As discussed in [67], the specific implementation of the BLAS libraries has large difference in performance, which is studied here.

It is shown that all the terms on the right-hand side of Eq. (3.10) have contributions to the $\hat{\mathbf{D}}^{-1}$. The first two terms on the right-hand side of Eq. (3.10) are contributions inside the elements, while the remaining three terms are contributions from the element boundaries.

Using the following equality

$$\sum_{j=1}^d \mathbf{B}^T \mathbf{D}_j^T = \sum_{j=1}^d \mathbf{B}^T \mathbf{D}_{\xi_j}^T \mathbf{\Lambda} \left[\sum_{m=1}^d \frac{\partial \xi_j}{\partial x_m} \right], \quad (4.1)$$

the first term of Eq. (3.10) can be transformed into the standard elements, which

has the following general form

$$\sum_{k=1}^{N_Q} \sum_{j=1}^d \sigma_{k,j} \mathbf{B}_{e_1}^T \mathbf{D}_{\xi_j, e_1}^T \mathbf{B}_{e_1}, \quad (4.2)$$

where $\mathbf{D}_{\xi_j, e_1}$ is the ξ_j direction derivative matrix for the standard element of the e_1 th element, N_Q is the number of quadrature points. In Eq. (4.2),

$$\sigma_{k,j} = \left(\sum_{m=1}^d \frac{\partial \xi_j}{\partial x_m} w_{e_1} J_{e_1} \left(\frac{\partial \mathbf{H}_{m, e_1}}{\partial \mathbf{Q}_{e_1}} \right)_{\nabla \mathbf{Q}} \right)_k, \quad (4.3)$$

is the only term related to the flow and grid derivatives. On the contrary, the matrices of $\mathbf{B}_{e_1}^T \mathbf{D}_{\xi_j, e_1}^T \mathbf{B}_{e_1}$ in Eq. (4.2) are linear matrices of the corresponding standard elements, which only need to be formed once for elements with the same type, the same number of polynomial order and the same number of quadrature points. Forming the $\mathbf{B}_{e_1}^T \mathbf{D}_{\xi_j, e_1}^T \mathbf{B}_{e_1}$ and $\sigma_{k,j}$ in advance and storing them in arrays, the calculation of Eq. (4.2) is implemented using the `BLAS:Ger`. To further reduce the memory consumption and computational cost, single data precision is used with the arrays padded following the idea in reference [67]. The second term of Eq. (3.10) can also be implemented in a similar way.

As for remaining element boundaries terms, some of the inverse operators as those in calculating the system residual $\mathbf{N}(\mathbf{u})$ have to be implemented since the calculation now starts from the flux Jacobian matrix on the quadrature points on the element boundaries. As discussed in Section 3.2, the last two terms in Eq. (3.10) are neglected here in forming $\hat{\mathbf{D}}^{-1}$, which significantly reduce the cost of the BRJ preconditioner. After forming the $\hat{\mathbf{D}}$ in single precision data, it is inverted using the `BLAS:Getrf` and the `BLAS:Getri`. The off-diagonal operations of the BRJ preconditioner is implemented with a matrix-free strategy as discussed in Section 3.2. Therefore, most of the matrix related operations are those in forming the $\hat{\mathbf{D}}^{-1}$ and the $\hat{\mathbf{D}}^{-1}$ vector

inner-product operations. Three different BLAS libraries are considered:

- Naive local compiled Netlib BLAS library of version 3.7.0;
- OpenBlas library of version 0.3.7;
- Intel MKL library of version 2019.5.281.

The comparison of their difference in computational time using the 3D (in Section 5.4.4) cylinder with 4 nodes on the HPC cluster of Imperial College London is summarized in Tab. 4.1. The simulations were run for 20 time steps with $\Delta t = 0.01$. There is not significant difference in computational costs for the GMRES related precesses including the Jacobian-free operation and preconditioning process. However, for calculating the $\hat{\mathbf{D}}^{-1}$, the computational cost implemented with the MKL library is almost one order of magnitude smaller than those with the naive BLAS library. The intel MKL library is not only implemented more efficiently, but can also benefit more with the single precision data. Thus, the Intel MKL library will be adopted in the following tests if not other wise specified. However, please keep in mind that the performances of these libraries are highly related to the specific hardware adopted and should be assessed case by case on specific hardware.

	GMRES			Matrix calculation		
Operation	JF	PREC-DIAG	PREC-OFFDIAG	VOL	BND	INV
Naive	875.0	553.9	645.2	724.8	132.1	71.4
OpenBlas	806.5	425.4	439.7	310.9	68.8	13.0
MKL	772.5	416.5	637.3	47.6	40.1	7.2

Table 4.1: Comparison of computational time with different BLAS libraries. JF represents the Jacobian-free operation in the GMRES. PREC-DIAG represents the $\hat{\mathbf{D}}^{-1}$ vector inner product operation. PREC-OFFDIAG represents the $\hat{\mathbf{L}} + \hat{\mathbf{U}}$ vector inner product operator. VOL, BND and INV present the element volume, element boundary and matrix inversion operations in calculating $\hat{\mathbf{D}}^{-1}$.

4.2.2 Performance for different orders of accuracy

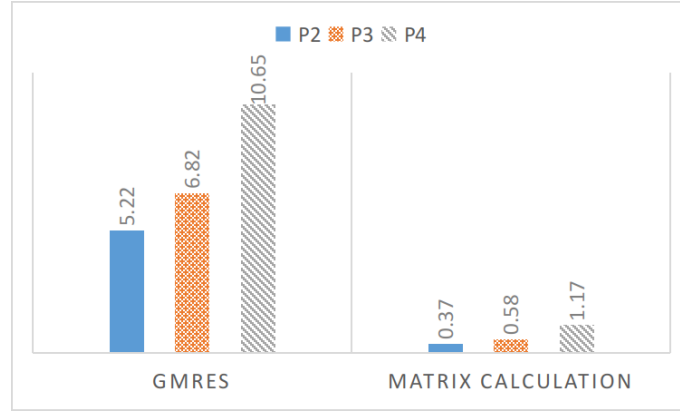
In order to provide a deeper insight of the implicit solver implementation, the computational costs of the different parts of the implicit solver are evaluated in the following. One node and four nodes are adopted for the 2D and 3D cases, respectively. The basic iteration information of the 3D (in Section 5.4.4) cylinder and 2D cylinder (in Section 4.4.3) cases adopted in this section is summarized in Tab. 4.2.

Cases	3D			2D		
Order	$P = 2$	$P = 3$	$P = 4$	$P = 2$	$P = 3$	$P = 4$
Time step (10^{-2})	1.0	0.50	0.23	1.5	0.70	0.38
Newton iteration	3.0	3.0	5.0	2.1	2.3	2.0
GMRES iteration	3.0	3.3	3.9	3.5	3.9	4.0
Computational time (s)	42.0	100.3	198.2	0.18	0.34	0.68

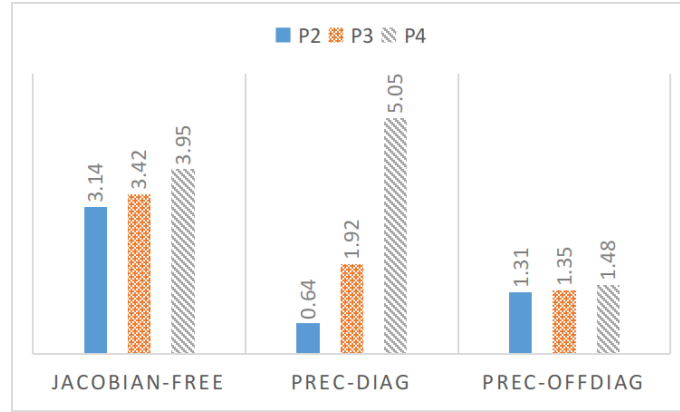
Table 4.2: Iteration information of 3D and 2D cylinder cases. The Newton iteration is the averaged number per implicit stage. The GMRES iteration is the averaged number per Newton iteration. The computational time is the average time per time step.

3D case

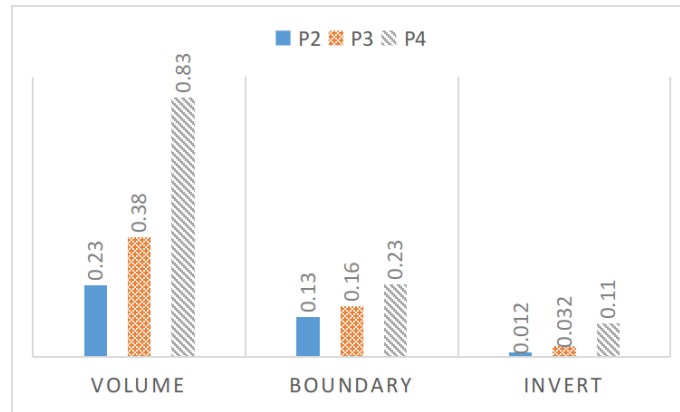
The 3D cylinder test case described in more detail in Section 5.4.4 is adopted here for the performance analysis. The simulations are carried out at polynomial orders $P = 2$, $P = 3$ and $P = 4$, using seven preconditioning iterations ($J = 7$) and a time step corresponding to CFL=40. The Newton iteration tolerance is $\alpha = 10^{-3}$. In total 10 steps are simulated, with the preconditioning matrices calculated at the beginning of the simulations and frozen in the following 10 time steps. Fig. 4.3 shows the computational time spent in different parts of the implicit solver, which is normalized by the total computational time spent in evaluating $\mathbf{N}(\mathbf{u})$ the same number of times as the total number of Newton iterations. Fig. 4.3a shows that the GMRES procedure, which calls the Jacobian-free operation and the preconditioners,



(a) The GMRES and calculations of preconditioning matrices in the implicit solver.



(b) Operations in GMRES: the JACOBIAN-FREE procedure corresponds to the operation of Eq. (3.1), the PREC-OFFDIAG corresponds to the $(\hat{\mathbf{L}} + \hat{\mathbf{U}})$ vector inner product in Eq. (3.8), while the PREC-DIAG corresponds to the $\hat{\mathbf{D}}^{-1}$ vector inner product in Eq. (3.8).



(c) Operations in the calculations of preconditioning matrices: VOLUME represents the volume integration in forming the preconditioning matrices, BOUNDARY represents the element boundary intergration, INVERT represents the costs inverting $\hat{\mathbf{D}}$.

Figure 4.3: Performance analysis of the implicit solver in a 3D cylinder case with polynomial orders of $P = 2, 3, 4$.

makes up the majority of computational time. The calculation of preconditioning matrices is much cheaper because of the freezing of the matrices. However, if the preconditioning matrices are updated every time step, their computational cost will be similar to that of the GMRES procedure and the implicit solver will be about twice as expensive as the current simulations. This observation leads to a simple strategy in updating the preconditioning matrices using the ratio between the cost of one GMRES iteration and the cost of preconditioning matrices calculation. The preconditioning matrices could be updated after the total number of GMRES iterations are larger than 200. Based on the data in Fig. 4.3 and Tab. 4.2, this will maintain that the costs of preconditioning matrix calculation is within 10% for $P = 2$ and $P = 3$ and within 20% for $P = 4$.

Fig. 4.3b shows the three most expensive operations in the GMRES procedure: the JACOBIAN-FREE, PREC-OFFDIAG and PREC-DIAG. The JACOBIAN-FREE procedure corresponds to the operation of Eq. (3.1), in which the majority of computational time is spent on one $\mathbf{N}(\mathbf{u})$ evaluation. As a result, the value of the normalized computational time of the JACOBIAN-FREE correctly indicates the averaged number of GMRES iterations in each Newton iteration, as seen from Fig. 4.3b and Tab. 4.2. The PREC-OFFDIAG corresponds to the $(\hat{\mathbf{L}} + \hat{\mathbf{U}})$ vector inner product in Eq. (3.8), while the PREC-DIAG corresponds to the $\hat{\mathbf{D}}^{-1}$ vector inner product in Eq. (3.8). Since the BRJ(7) is used with zero initial guess, six PREC-OFFDIAG and seven PREC-DIAG are needed for each JACOBIAN-FREE operation. Although much more preconditioning operations are performed, their costs are smaller than that of the JACOBIAN-FREE operations for $P = 2$ and $P = 3$. Even though we have already adopted the single precision data type to reduce the cost of the matrix-vector inner product operations, the $\hat{\mathbf{D}}^{-1}$ vector inner product still grows much faster than the other two parts when P increases. This is mainly

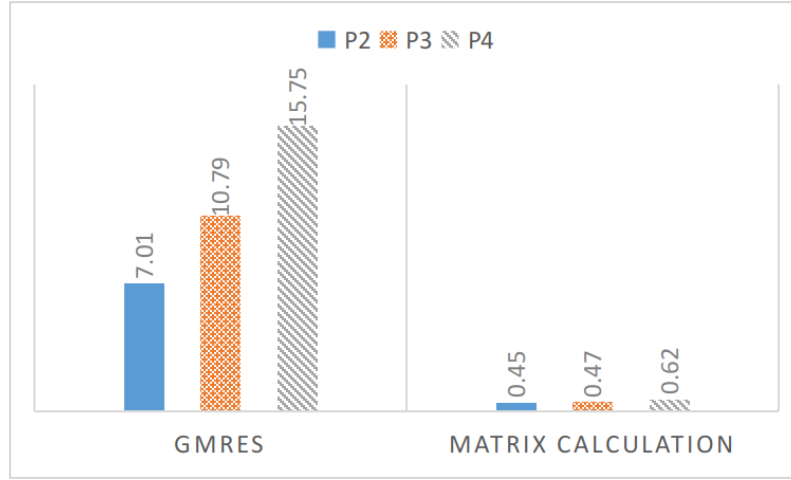
because it uses a matrix-based implementation while most operations in the other parts use a sum-factorization implementation. This also indicates that matrix-based preconditioners will easily lose efficiency for high-order 3D simulations. Techniques proposed in reference [14] may be used to implement a matrix-free version of the BRJ preconditioner to avoid the fast growth of the $\hat{\mathbf{D}}^{-1}$ vector inner product. As for the off-diagonal operations, the simplifications and implementations make sure they are relatively cheap compared with other parts.

Fig. 4.3c shows the most expensive parts in the calculations of the preconditioning matrices. For the element volume part, techniques such as using single precision, maintaining memory continuous and padding the matrices as discussed in reference [67], have been adopted in our implementation to speed-up the calculation. However, this is still the most expensive part. In our implementation, the most expensive part of the volume term is implemented using `BLAS::sger`, but it still grows quite fast as P increases. Inverting $\hat{\mathbf{D}}$, which is implemented using `Lapack::dgetrf` and `Lapack::dgetri`, only makes up a small proportion in the test cases. However, it grows the fastest, which indicates it will become the dominant part when the polynomial order is high enough. In this case, techniques such as approximate matrix inversion [14, 118, 44] may be adopted to reduce the cost related to matrix inversion.

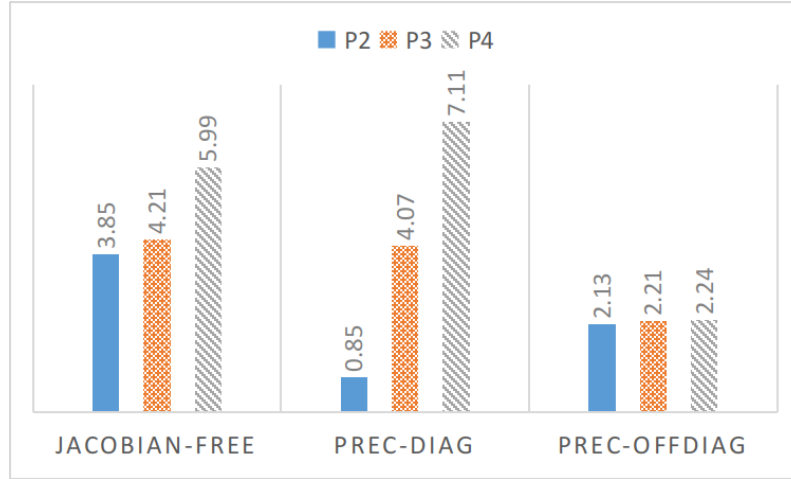
Fig. 4.3 shows that a large proportion of computation costs in the implicit solver is from operations shared with the explicit solver. Therefore, the implicit solver will also greatly benefit from improvements in efficiency of the explicit solver.

2D case

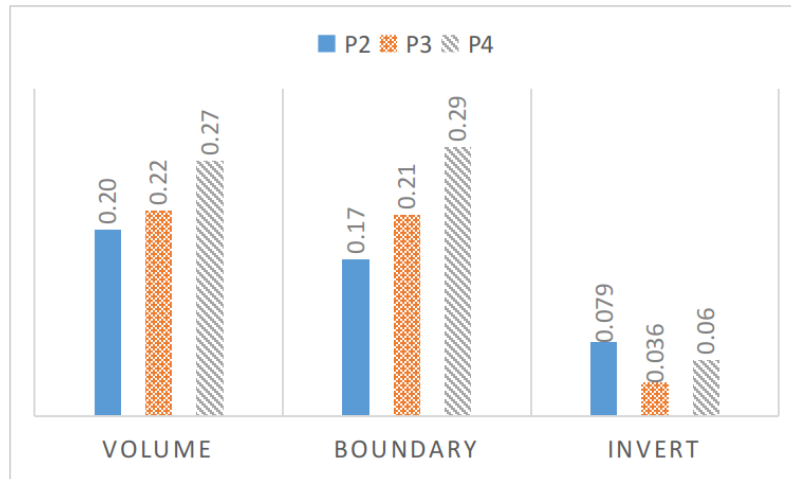
The 2D cylinder test cases of Section 4.4.3 are adopted for the performance analysis of the solver in 2D. The simulations are run at CFL=20 for 200 steps with precondi-



(a) The GMRES and calculations of preconditioning matrices in the implicit solver.



(b) Operations in GMRES.



(c) Operations of matrix calculation of preconditioner.

Figure 4.4: Performance analysis of the implicit solver in a 2D cylinder case with polynomial orders of $P = 2, 3, 4$. The meanings of different operations can be found in Fig. 4.3.

tioning matrices updated every 10 time steps. The other parameters are the same as those of the 3D tests. Fig. 4.4 shows the computational time of different parts of the implicit solver, which are normalized in the same way. Very similar trends as in the 3D case are observed. The $\hat{\mathbf{D}}^{-1}$ vector inner product still grows much faster than the other parts, which means matrix-based preconditioners may also lose efficiency because of the large computational cost. However, the growth of computational time in the calculations of the preconditioning matrices is much slower than that in 3D case. The computational time in inverting the $\hat{\mathbf{D}}$ matrix is not increasing monotonously as that of the 3D case. A possible reason is that the functions in intel MKL library are sensitive to the size of the matrix when the matrix size is small, as discussed in reference [67].

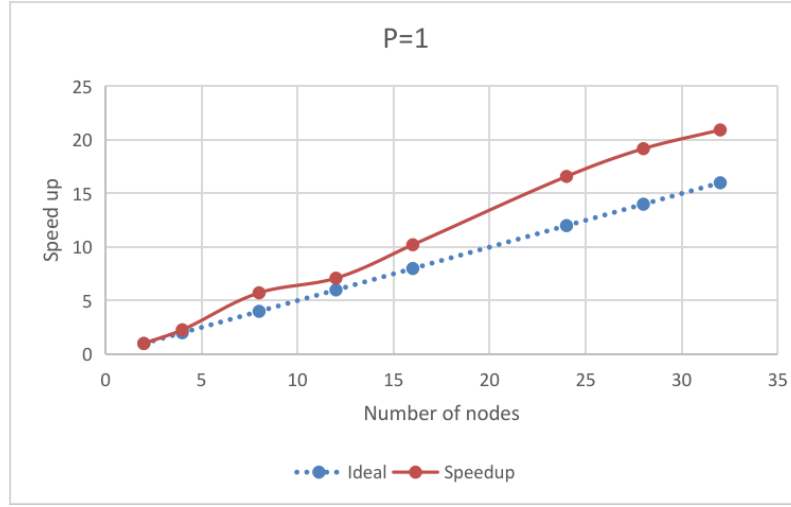
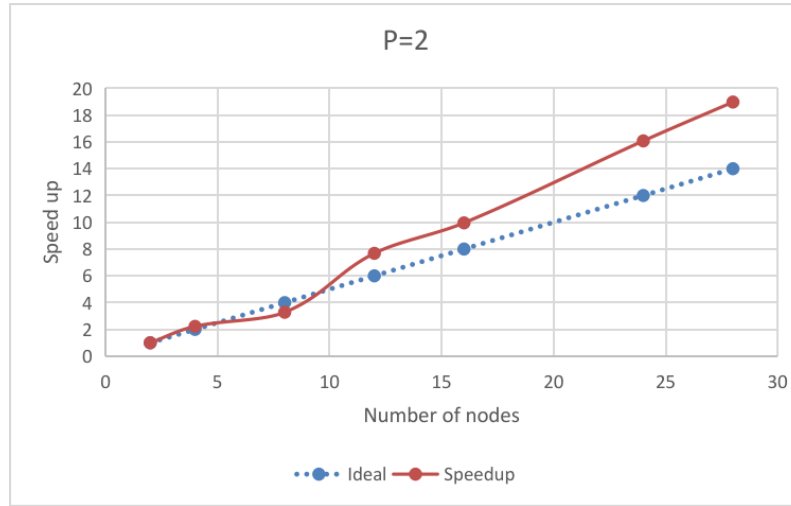
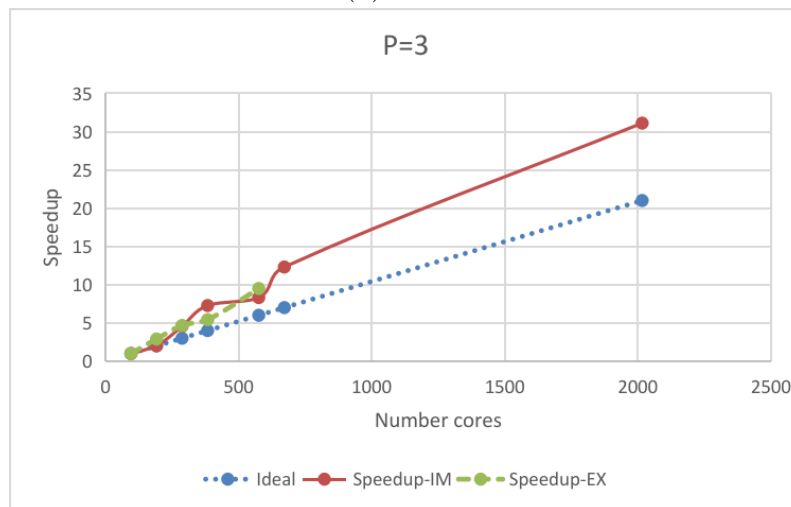
4.2.3 Scalability performance

Parallel scalability is an important metric reflecting the efficiency in parallel computing [62]. The strong parallel scalability of the implicit solver for $P = 1$, $P = 2$ and $P = 3$ is tested based on the 3D cylinder case adopted in Section 4.2.2. The simulations are run using 2 to 72 nodes, which corresponds to 2016 virtual cores at most. At the largest parallel scale tested, there are about 61 elements on each core. The relative speed-up are summarized in Fig. 4.5, in which the perfect speed-up are also provided for comparison. The results indicate that very good scalability properties are obtained by the implicit solver. It even achieves higher speed-up than the perfect one. The reason is probably because of the better cache or memory performance with smaller memory consumption per node. With more nodes adopted, less memory are needed in each node, which seems maintain that less time is required in loading the data. This also reflects that the solver is probably cache or memory performance limited instead of CPU performance limited especially for simulations

with large amount of meshes per node.

The scalability of the explicit solver is also provided. Fig. 4.5 shows that the implicit solver has similar scalability to the explicit solver. This is because the JFNK is highly dependent on the explicit solver in calculating the nonlinear system residuals and in implicitly calculating the Jacobian matrix. In addition, the BRJ preconditioner also uses the same subroutine as the explicit solver to exchange data between different cores. The additional amount of communications from the GMRES is small since only reductions of a few vectors are needed in each GMRES iteration. These communications are relatively small compared with the communications of flow variables at least for the parallel scales considered currently. As a result, the efficiency and scalability of the explicit solver have large influence on both the efficiency and parallel scalability of the implicit solver. It is of great importance to optimize the explicit solver in improving efficiency and parallel scalability.

The performance of the implicit solver based on different linear algebra libraries, for different orders of accuracy and for different parallel scales has been numerically studied. The results show that the linear algebra libraries have a large influence of the computation time especially for the operations in calculating the preconditioning matrices. Among the libraries tested, the intel MKL library performs the best. The performance for different orders of accuracy demonstrates the change trend of different parts of the implicit solver when the polynomial order changes. The data highlights the importance of reducing the costs of the preconditioner in maintaining high efficiency. Moreover, the need for matrix-free preconditioner is highlighted for very high order simulations since matrix-based ones grow too fast in cost as polynomial order increases. The scalability tests shows that almost perfect parallel scaling are obtained, which implies a good performance of the solver in large-scale parallel computations.

(a) $P = 1$.(b) $P = 2$.(c) $P = 3$.Figure 4.5: Strong parallel scalability performance of the implicit solver for polynomial orders $P = 1, 2, 3$.

4.3 Memory consumption analysis

Tab. 4.3 compares the memory requirements of the preconditioning matrices and Jacobian matrix $\partial \mathbf{N} / \partial \mathbf{u}$ using the 3D cylinder case of Section 4.2.2. The memory consumption of $\mathbf{Jac}_S$ is that of the Jacobian matrix considering the sparsity patterns of the off diagonal blocks of $\partial \mathbf{N} / \partial \mathbf{u}$, which is calculated based on the estimation in reference [120]. The memory consumption of $\mathbf{Jac}_D$ is that storing the diagonal and off diagonal blocks as dense matrices. Note that the ILU preconditioner with zero fill-in should have the same memory consumption as $\mathbf{Jac}_D$ since the matrices will generally lose sparsity after the factorization. The memory consumption of the Jacobian matrix becomes large in 3D. Only approximately 90 elements can be afforded per Gbyte of memory for simulations with $P = 4$ and hexahedral meshes if the $\mathbf{Jac}_D$ matrix is stored or the ILU preconditioner is used. The element number will become 45 if stored in double precision data. Moreover, frequently loading these large matrices into the cache will seriously influence the computational speed since most modern high-performance computers (HPCs) are memory bandwidth limited [156]. The memory consumption can be further reduced if using other schemes like the CDG [120] by exploring the sparsity patterns, but it is still of the same order of magnitude as $\mathbf{Jac}_S$.

$\mathbf{BRJ}_D$ and $\mathbf{BRJ}_O$ represent memory consumptions required for storing $\hat{\mathbf{D}}^{-1}$ and $\partial \hat{\mathbf{H}}^n / \partial \hat{\mathbf{Q}}^\pm$, both of which are stored in single precision to reduce memory and computational costs. The storage of $\partial \hat{\mathbf{H}}^n / \partial \hat{\mathbf{Q}}^\pm$ is small compared with that of $\hat{\mathbf{D}}^{-1}$ especially for high order approximations. The total storage of BRJ is much smaller than that of the Jacobian matrix or the ILU preconditioner. The results show that that the proposed BRJ preconditioner can largely reduce the memory consumption. Moreover, it leaves the possibility to further develop completely

matrix-free preconditioners by exploring techniques such as those in references [43, 118, 17, 14].

	BRJ_D	BRJ_O	BRJ_T	Jac_D^D	Jac_S^D	Jac_D	Jac_S
$P = 2$	72 900	30 000	102 900	1 020 600	712 800	510 300	356 400
$P = 3$	409 600	43 200	452 800	5 734 400	3 328 000	2 867 200	1 664 000
$P = 4$	1 562 500	76 800	1 639 300	21 875 000	11 000 000	10 937 500	5 500 000

Table 4.3: Memory consumption of BRJ in bytes for each hexahedral mesh. **BRJ_D**, **BRJ_O** and **BRJ_T** represent the storage for $\hat{\mathbf{D}}^{-1}$, $\partial \hat{\mathbf{H}}^n / \partial \hat{\mathbf{Q}}^\pm$ and their sum. **Jac_S** and **Jac_D** represent the storage for the Jacobian matrix considering the sparsity patterns and not. The superscript D represent the storage is for double precision data while all others without the superscript represent the storage for single precision data.

4.4 Verification of the implicit solver

In the following, the implementations of spatial discretization methods are verified using accuracy tests described in Section 4.4.1 and Section 4.4.2. Section 4.4.3 discusses the temporal accuracy and efficiency of the solver in unsteady laminar flows over a circular cylinder. Finally, a shock wave boundary layer interaction case is studied in Section 4.4.4 to demonstrate the shock-capturing ability of the compressible flow solver.

4.4.1 Isentropic vortex convection

Testing the convergence order of accuracy (OoA) of the solver is a very effective way of code verification [131]. Here two different test cases are used to verify two different aspects of the solver. An isentropic vortex convection problem is used to verify the advection scheme in this section and the treatment of the diffusion terms is verified by means of a manufactured compressible Poiseuille flow in Section 4.4.2.

This test case is adopted because its analytical solution is available, which makes it easier for spatial and temporal error estimates. In the computational domain $[0, 10] \times [-5, 5]$, an inviscid isentropic vortex is convected down stream, the exact solution of which at time t is

$$\begin{aligned}\rho &= \left(1 - \frac{\varphi^2 (\gamma - 1)}{16\gamma\pi^2} e^{2(1-r^2)}\right)^{\frac{1}{\gamma-1}}, \\ u &= u_0 - \frac{\varphi (y - y_0 - v_0 t)}{2\pi} e^{(1-r^2)}, \\ v &= v_0 + \frac{\varphi (x - x_0 - u_0 t)}{2\pi} e^{(1-r^2)}, \\ p &= \rho^\gamma,\end{aligned}\tag{4.4}$$

where $(u_0, v_0) = (1.0, 0.5)$ is the velocity vector of the mean flow, $(x_0, y_0) = (5.0, 0.0)$ is the initial vortex center position, the vortex centre at time t is located at $(x_0 + u_0 t, y_0 + v_0 t)$, r is the distance from the vortex centre, and $\varphi = 0.5$ is a parameter that controls the strength of the vortex. Periodic boundary conditions are applied to the boundaries in both directions. The uniform mesh and the density distribution with $(u_0, v_0) = (1.0, 0.0)$ at $t = 1.0$ are illustrated in Fig. 4.6.

Mesh convergence is analyzed through the use of progressively refined quadrilateral meshes. Periodic boundary conditions are applied to the boundaries in both directions. A fourth-order explicit Runge-Kutta with CFL=0.01 is adopted to guarantee the errors are dominated by the spatial discretization. The error distributions are depicted in Fig. 4.7 with the order of accuracy (OoA) between the two finest mesh levels given in the legend. All simulations with polynomial orders P from 1 to 5 reach the designed order of accuracy, which verifies the implementation of the advection discretizations.

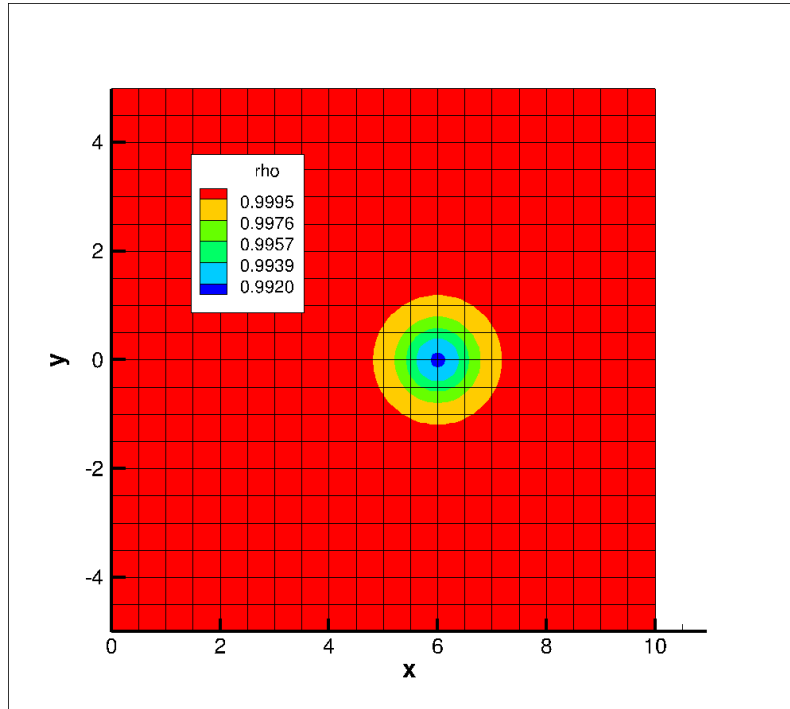


Figure 4.6: Isentropic vortex problem: Mesh and density distribution.

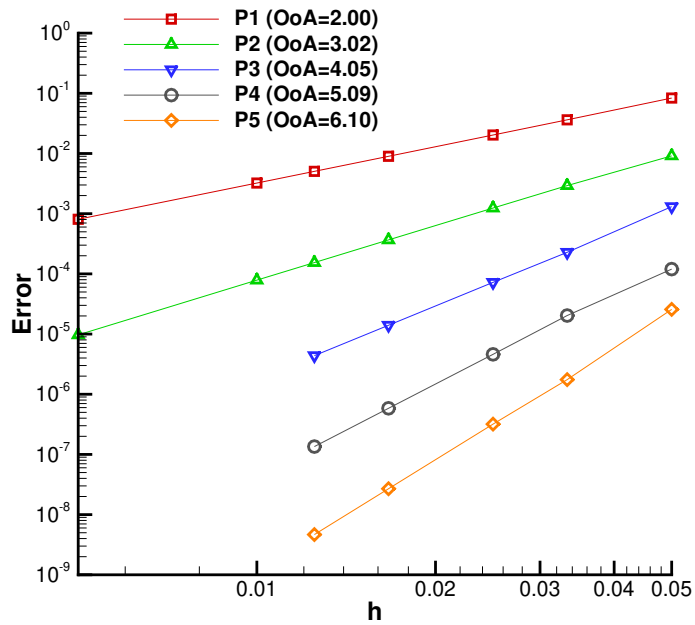


Figure 4.7: Convergence order of accuracy (OoA) tests of the advection scheme with polynomial orders $P = 1, 2, \dots, 5$. The L2 error norms at $t=0.01$ are calculated using the analytical solution as the reference. The OoA between the two finest mesh levels is given in the legend.

4.4.2 Manufactured compressible Poiseuille flow

The method of manufactured solution (MMS) permits us to design specific test cases with analytical solutions for the study of different aspects of the solvers [131]. Using the MMS, a test case similar to Poiseuille flow [112] is designed to verify the DG methods for the diffusion terms. This problem is suitable for the verification of the IP method since it is dominated by the diffusion term. By design, the analytical solution is

$$\begin{aligned}\rho &= 1, \\ u &= -\frac{1}{2\mu} \frac{dp}{dx} \left[y(L-y) + \theta y^2(L-y)^2 \right], \\ v &= 0, \\ p &= \frac{1}{\gamma \text{Ma}^2} + \frac{dp}{dx} x,\end{aligned}\tag{4.5}$$

where $L = 1$ is the height of the computational domain in the y direction. The corresponding forcing terms are given in Eqs. (B.1) in the Appendix and this corresponds to a free-stream flow with $\text{Ma} = 0.1$ and $\text{Re} = 100$ based on the maximum velocity, which is normalized to 1. The corresponding pressure gradient is $dp/dx = -8\mu/L^2$. Compared to incompressible Poiseuille flow, an extra term $\theta y^2(L-y)^2$ with $\theta = 0.01$ is added to the horizontal velocity so that the analytical solution of the energy E , shown in Fig. 4.8, has a polynomial order of 8. This ensures the polynomial order of the solution is higher than that of the basis functions and thus avoids the high-order coefficients becoming excessively small. A series of quadrilateral meshes with 3^2 , 4^2 , 8^2 , 12^2 , 16^2 and 20^2 elements are used with Dirichlet boundary conditions based on the analytical solution. The L_2 and L_∞ norm error distributions are shown in Fig. 4.9. The results show that the designed order of accuracy is achieved for the IP method from $P = 2$ to $P = 5$. Simulations with

triangular meshes can also achieve the designed order of accuracy but are not shown here.

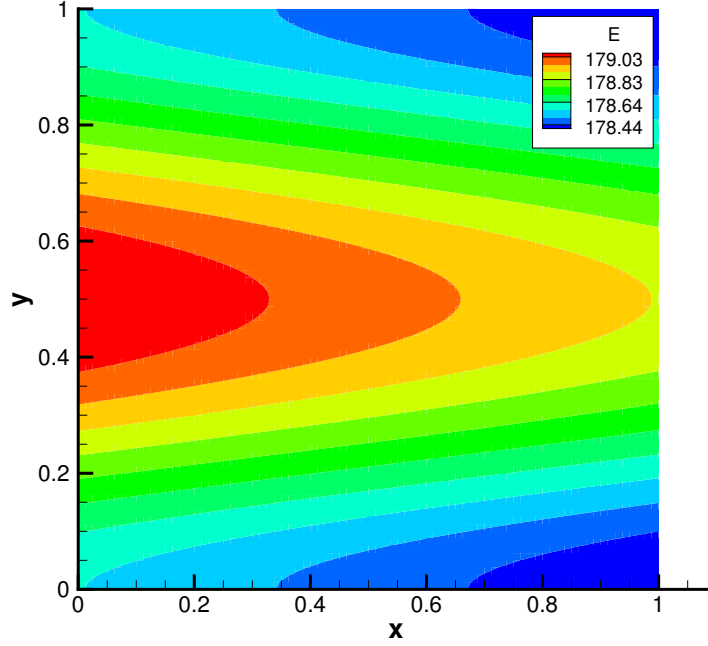


Figure 4.8: Non-dimensional total energy (E) distribution of the manufactured compressible Poiseuille flow.

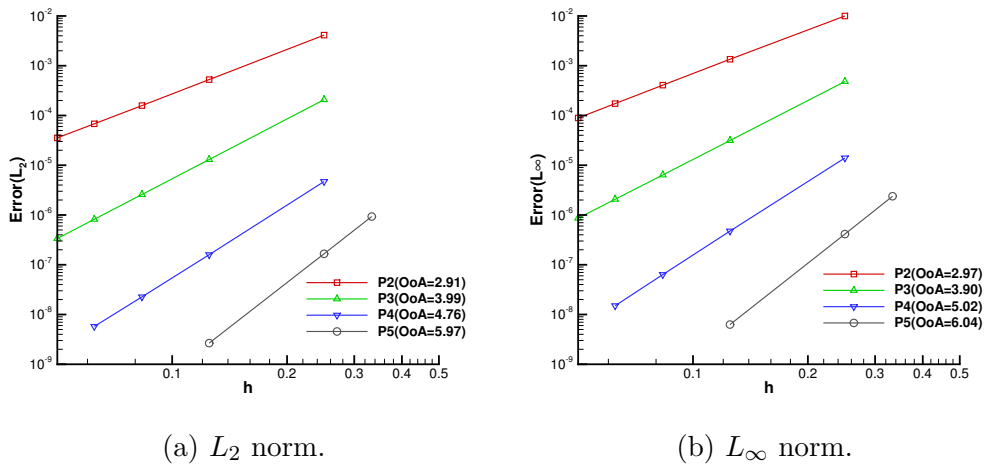
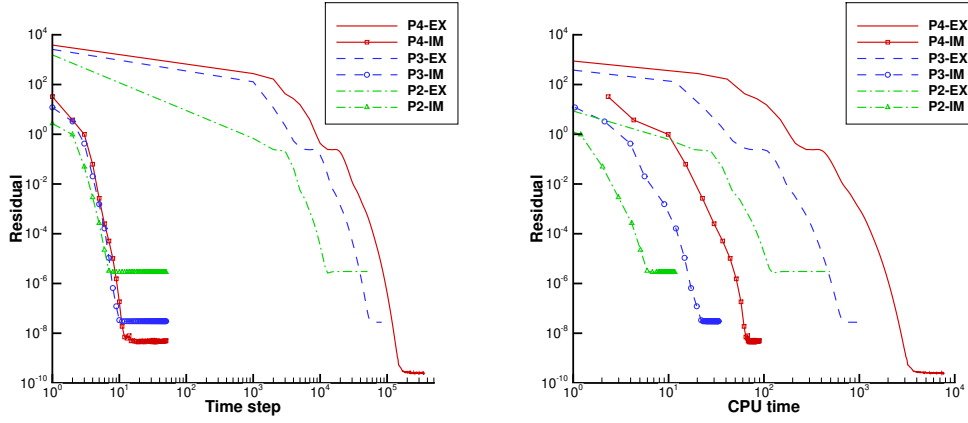


Figure 4.9: Convergence order of accuracy (OoA) of the IP methods in compressible Poiseuille flow where the OoA between the two finest mesh levels is given in the legend.



(a) Convergence history with respect to time steps. (b) Convergence history with respect to computational time.

Figure 4.10: Residual distribution of implicit and explicit simulations in compressible Poiseuille flow.

The efficiency of the explicit and implicit solvers is also compared in this steady problem. The initial condition is similar to that used in Eqs. (4.5) but with uniform velocity $u = -(dp/dx)L^2/(8\mu)$. The time integration used by the implicit solver is SDIRK2 whilst the explicit solver uses ERK2. The CFL number of the implicit solver grows from 500 to 5000 within the first 10 steps. Here the preconditioning matrices are updated in every other time step. The explicit solver uses its maximum stable CFL numbers of 0.08, 0.06 and 0.06 for $P = 2$, $P = 3$ and $P = 4$, respectively. Fig. 4.10 shows the convergence history of the residual norm $\|L_\delta\|_2$. All the implicit simulations reach steady-state within 20 steps, and the explicit solver takes about 13000, 60000 and 170000 steps for $P = 2$, $P = 3$ and $P = 4$ to converge, respectively. Fig. 4.10 shows that even with no grid stretching the implicit solver is about 18.2, 33.1 and 53.4 times faster than the explicit solver for $P = 2$, $P = 3$ and $P = 4$, respectively.

Following references [112, 131], Dirichlet boundary conditions based on the analytical solution are applied weakly in the DG methods in this subsonic problem.

The difference between analytical solutions and numerical solutions will lead to large errors near boundaries, which prevents the simulations from converging to machine epsilon levels. However, the errors caused by boundary treatment does not invalidate the OoA tests since the difference between the numerical and analytical solutions on the boundaries tends to zero as the resolution increases.

Through this test case, the implementation of the diffusion discretization is verified. Meanwhile, it is also demonstrated that the implicit solver can largely accelerate the convergence of this steady-state case.

4.4.3 Laminar flow over a circular cylinder

The temporal accuracy and efficiency of the implicit solver is tested next using unsteady simulations of flow over a circular cylinder. The parameters and meshes employed are similar to those presented in reference [15]. The flow conditions correspond to $Ma = 0.3$, $Re = \rho u D / \mu = 1200$ and we use 64×60 quadrilateral meshes. Starting from the same flow field, different time integration schemes and time steps are used to get the solutions after approximately 1.7 vortex shedding periods. The integrated lift and drag forces are the chosen metrics for the accuracy test with reference solutions obtained by the SDIRK4 in Tab. 16 of reference [78] with a very small time step $\Delta t = 1 \times 10^{-5}$. A smaller Newton iteration tolerance $\alpha = 10^{-6}$ is used for SDIRK3 and SDIRK4. The convergence rates are presented in Fig. 4.11, which shows that the implicit solver achieves the desired order of accuracy.

The efficiency of the implicit solver is compared with that of the explicit solver in this test case. Starting from the same flow field, the solution is obtained after around 30 vortex shedding periods using both the explicit and implicit solver. The details of the simulations are summarized in Tab. 4.4. The implicit solver can run with a time

step 330 times larger than that of the explicit solver. The corresponding CFL_c is about 2.5 for this time step. The implicit solver is around 16.8 times quicker than the explicit solver. The implicit solver is also ran with a larger time step ($CFL_c \simeq 5$), which achieves a speed-up of 26.4. By observing the evolution history of the flow field, it can be observed that the temporal errors mainly lead to a dispersion in the vortex shedding process. Thus the Strouhal number $Sr = Df/u$, where f is the vortex shedding frequency, is a good metric to compare the temporal accuracy. Tab. 4.4 shows that all the simulations give almost the same prediction of Strouhal number (0.2419). The results show that the implicit solver can largely improve efficiency while keeping good temporal accuracy.

This test case verifies the temporal discretization using accuracy test and illustrate the advantage of the implicit solver in efficiency in unsteady simulations.

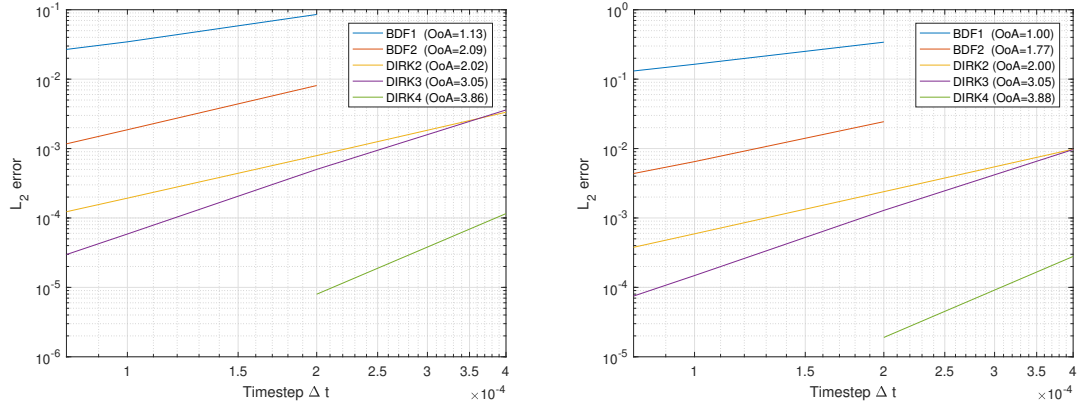


Figure 4.11: Convergence order of accuracy (OoA) of BDF1, BDF2, DIRK2, DIRK3 and DIRK4 implicit time integration schemes in the laminar cylinder case.

4.4.4 Shock boundary layer interaction

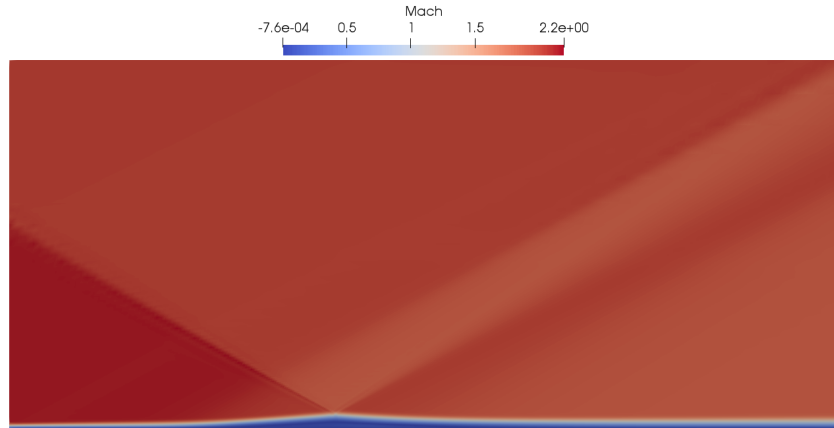
Finally, the shock-capturing ability of the implicit solver is tested using the shock wave boundary-layer interaction (SWBLI) problem [19]. The computational domain

Method	ERK2	SDIRK2	
Δt	4.5×10^{-5}	1.5×10^{-2}	3.0×10^{-2}
CFL	0.06	20	40
CFL_c	7.5×10^{-3}	2.5	5.0
CPU time (s)	9.1×10^{-3}	0.18	0.23
Strouhal number	0.2419	0.2419	0.2419
Newton iterations per RK stage	\	2.1	2.5
GMRES iterations per RK stage	\	3.5	5.0
Speed-up	1.0	16.8	26.4

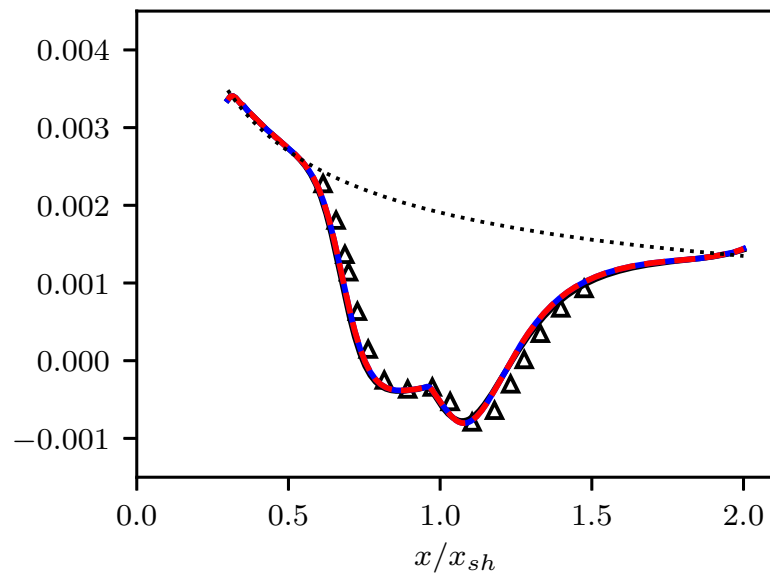
Table 4.4: Efficiency comparison between explicit and implicit time integration schemes in the laminar cylinder case. All simulations are conducted using one HPC node as discussed in Section 4.2.

Polynomial order	ERK2	SDIRK2		
CFL	0.08	2.5	5.0	100.0
Δt	1.3×10^{-4}	4.0×10^{-3}	8.0×10^{-3}	1.6×10^{-2}
J in BRJ	\	3	5	7
Newton iterations per RK stage	\	2.0	2.0	3.0
GMRES iterations per Newton iteration	\	3.0	3.2	8.9
Speed-up	1.0	2.7	4.2	22.8

Table 4.5: Efficiency comparison between explicit and implicit time integration schemes of shock wave boundary-layer interaction.



(a) Mach number distribution.



(b) Skin friction coefficient distribution. Blue line: SDIRK-2; Red line: ERK2; triangles: simulation from [19]; dotted line: engineering relation [48].

Figure 4.12: Mach number and skin friction coefficient distributions in shock wave boundary-layer interaction.

and Mach number distribution are displayed in Fig. 4.12a. The viscous wall starts from $x = 0$ and the shock impingement position is x_{sh} . The computational domain starts from $x = 0.3x_{sh}$ where the analytical compressible boundary layer solution [155] is imposed. Rankine-Hugoniot relations are added to impose inflow boundary conditions consistent with the shock. Flow conditions corresponding to $Ma = 2.15$, $Re_{x_{sh}} = 10^5$ and shock impingement angle of 30.8° are used. The simulations run with 119×39 quadrilateral elements and $P = 3$. The skin friction coefficient distributions are shown in Fig. 4.12b. The results of the explicit and implicit solvers coincide and are in good agreement with the simulation results presented in reference [19]. The efficiency of the solvers is compared in Tab. 4.5. Compared with the explicit solver, larger speed-up can be achieved when the CFL number of the implicit solver increases from 2.5 to 100. However, further increasing the CFL number to 1000, the implicit solver will have difficulty converging. The optimal numbers of preconditioning iterations are also given in Tab. 4.5. With a smaller time step, the system is less stiff and the optimal preconditioning number is smaller. Since the test case is regarded as a steady-state problem, a large CFL number can be used without influencing the accuracy. A speed-up of 22.8 is achieved with CFL=100.

Chapter 5

Robust solver parameter choice

A robust parameter choice strategy, which can maintain basic stability, rational accuracy and efficiency of the solver, is important for a software with a large number of heterogenous users and that is employed in a wide variety of application areas. This is especially true for the implicit solver since there are many free parameters greatly influencing the accuracy, efficiency and stability of the simulations. For the implicit solver, a naive parameter choice could make the simulations several times slower compared with the best one. Tab. 5.1 summarizes the parameters, their influences and their ranges of choice in the current implicit solver. These parameters are nonlinearly coupled with each other and influence the accuracy, efficiency and stability of the solver. In theory, the choice of these parameters becomes a multi-objective optimization problem, which would be challenging for users of the software.

In this Chapter, systematic ways of choosing these parameters are studied based on the requirement of maintaining accuracy of the simulations. Section 5.1 develops a balanced adaptive time stepping strategy, which also helps relax the difficulty in

Parameter	Influence	Range
Order of accuracy in time	A, E, S	$\{1, 2, 3, 4\}$
Time step	A, E, S	$(0, \infty)$
Newton tolerance	A, E, S	$(0, 1)$
GMRES tolerance	E, S	$(0, 1)$
Preconditioning iteration number	E, S	$\{0, 1, 2, \dots\}$
Frequency to update preconditioning matrices	E, S	$\{1, 2, 3, \dots\}$

Table 5.1: Parameters of the implicit solver. A , E and S represent that the parameter has influence on the accuracy, efficiency and stability of the implicit solver, respectively.

choosing the order of accuracy of the temporal discretization scheme. In Section 5.2, an adaptive Newton tolerance is developed to maintain convergence accuracy of the temporal scheme. A suitable choice of adaptive time stepping and adaptive Newton tolerance together would be able to maintain the temporal accuracy of the solver. Together with the GMRES tolerance presented in Section 3.1, relative good choices of the parameters are provided for the implicit solver, which will be summarized in Section 5.3 and numerically tested in Section 5.4.

5.1 Balanced adaptive time-stepping

Recall that Eq. (2.41) suggests that the error generated in one time step is the sum of the local temporal error, averaged spatial error and averaged iterative error. This indicates that to decrease the total error one should decrease these three errors simultaneously and it will not help much if one of them is decreased to a much smaller value than the largest of the three terms. Based on this observation, an adaptive time-stepping strategy and an adaptive Newton tolerance are developed in this section and in Section 5.2, respectively. The basic idea is to choose a time step and a Newton tolerance such that the temporal and iterative errors are smaller than

the spatial error. The iterative error is assumed to be negligible compared with the spatial and temporal errors. This can be achieved as long as Eq. (2.32) converges to a sufficiently small level. The method to control the iterative error will be discussed in Section 5.2.

Based on the error estimate of Eq. (2.41), a time step adaptation method is proposed based on the idea that the time step should be as large as possible without obviously influencing the total discretization error. If the spatial error is fixed in a specific simulation, this implies that

$$\|\mathbf{E}_t^{n+1}\| = \beta \Delta t \|\bar{\mathbf{E}}_s\|, \quad (5.1)$$

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

To ensure Eq. (5.1) is satisfied, spatial and temporal errors need to be estimated. The embedded ESDIRK scheme offers an accurate estimate of temporal error $\mathbf{E}_t^{n+1}$ using Eq. (2.27). The spatial truncation error $\bar{\mathbf{E}}_s$ is estimated using a one order higher approximation at the end of each time step, that is

$$\bar{\mathbf{E}}_s \approx L_\delta^{P+1}(\mathbf{U}_{it}^{n+1}) - L_\delta^P(\mathbf{U}_{it}^{n+1}) = C_s D_s(\mathbf{U}) h^{P+1} + O(h^{P+2}). \quad (5.2)$$

The leading term of this equation is the same as that in Eq. (2.15). Eq. (5.1) implies that the desired time step ($\Delta \hat{t}$) should satisfy that

$$\|\mathbf{E}_t^{n+1}(\Delta \hat{t})\| \approx \|C_t D_t(\mathbf{U}) \Delta \hat{t}^{N+1}\| = \beta \Delta \hat{t} \|\bar{\mathbf{E}}_s\|. \quad (5.3)$$

Dividing Eq. (5.3) by Eq. (2.27), time step $\Delta\hat{t}$ can be calculated as

$$\Delta\hat{t} = \Delta t^n \left(\frac{\beta \Delta t \|\bar{\mathbf{E}}_s\|}{\|\mathbf{E}_t\|} \right)^{1/N}, \quad (5.4)$$

which can be used as the value of the next time step. The simplest elementary controller [136] is adopted here in determining the time step, but other choices are also possible [136]. In the derivation of Eqs. (5.2), (5.3) and (5.4), the change of $D_t(\mathbf{u})$ is assumed to be small throughout the time step.

In the design of adaptive time-stepping methods presented in references [111, 18], a L_2 norm of the $\mathbf{E}_t^{n+1}$ 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 by large-scale unsteady structures of the unsteady flow. As a result, the adapted time step may be too large for small unsteady structures of interest.

To avoid these problems, the norm is defined in each element here. 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

$$\Delta t_{e,m}^{n+1} = \Delta t^n \left(\frac{\beta \Delta t \|\bar{\mathbf{E}}_s\|_{e,m} + 1.5^N \epsilon_m}{\|\mathbf{E}_t\|_{e,m} + \epsilon_m} \right)^{1/N}. \quad (5.5)$$

Here ϵ_m a very small value 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 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,

$$\epsilon_m = 10^{-12} \mathbf{U}_{it,m}^{\text{rms}}, \quad (5.6)$$

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 a noisy elemental time step 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. (5.1) is satisfied in every element. Here we choose to calculate it using a weighted average of $\Delta t_{e,m}^{n+1}$ as

$$\Delta t_m^{n+1} = \sum_{e=1}^{N_e} \|\mathbf{E}_t\|_{e,m}^r \Delta t_{e,m}^{n+1} / \sum_{e=1}^{N_e} \|\mathbf{E}_t\|_{e,m}^r. \quad (5.7)$$

The temporal error $\|\mathbf{E}_t\|_{e,m}^r$ is used as the weights such that elements with larger temporal error 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. Here $r = 1$ is used. This weighted averaging helps maintain a relatively smooth change of the time step. Finally, the time step Δt^{n+1} is calculated by

$$\Delta t^{n+1} = \min(\Delta t_m^{n+1}). \quad (5.8)$$

5.2 Adaptive convergence tolerances

A similar idea to that used in the time step adaptation can be adopted here in choosing the Newton tolerance so that the iterative error is smaller than the temporal error to avoid affecting the temporal accuracy. References [110, 18] propose a Newton tolerance τ in Eq. (2.34) by directly relating the Newton tolerance to the temporal error of the form

$$\tau = \eta \|\mathbf{e}_t\|, \quad (5.9)$$

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. (2.41) is not trivial. In the following this relation is estimated to illustrate the assumptions made in choosing the Newton tolerance of Eq. (5.9).

Considering that

$$\Delta t \|\bar{\mathbf{e}}_{it}\| = \Delta t \left\| \sum_{i=1}^S b_i \mathbf{e}_{it}^{(i)} \right\| \leq \Delta t \sum_{i=1}^S b_i \|\mathbf{e}_{it}^{(i)}\|, \quad (5.10)$$

with $\sum_{i=1}^S b_i = 1$, a sufficient condition for

$$\Delta t \|\bar{\mathbf{e}}_{it}\| \leq \eta \|\mathbf{e}_t\|, \quad (5.11)$$

is that

$$\Delta t \|\mathbf{e}_{it}^{(i)}\| \leq \eta \|\mathbf{e}_t\|, \quad (5.12)$$

which ensures that the iterative error is smaller than the temporal error in Eq. (2.41) with $\eta < 1.0$.

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

expressed as

$$\mathbf{R}^{(i)} = \mathbf{N}(\mathbf{u}_{it}^{(i)}) - \mathbf{N}(\mathbf{u}^{(i)}) = \mathbf{N}'(\mathbf{u}_{it}^{(i)}) (\mathbf{u}_{it}^{(i)} - \mathbf{u}^{(i)}) + O(\Delta \mathbf{u}^2), \quad (5.13)$$

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

$$\begin{aligned} \mathbf{u}_{it}^{(i)} - \mathbf{u}^{(i)} &= (\mathbf{N}')^{-1} (\mathbf{R}^{(i)} - O(\Delta \mathbf{u}^2)) \\ &= (\mathbf{I} - \Delta t a_{ii} \mathcal{L}'_{\delta}(\mathbf{u}_{it}^{(i)}))^{-1} (\mathbf{R}^{(i)} - O(\Delta \mathbf{u}^2)). \end{aligned} \quad (5.14)$$

Combining Eqs. (5.14) and (2.35) and ignoring high-order terms of $O(\Delta \mathbf{u}^2)$, we can further deduce a relation between $\Delta t \mathbf{e}_{it}$ and $\mathbf{R}$ as

$$\begin{aligned} \Delta t \mathbf{e}_{it}^{(i)} &= \Delta t (\mathcal{L}_{\delta}(\mathbf{u}_{it}^{(i)}) - \mathcal{L}_{\delta}(\mathbf{u}^{(i)})) \\ &= \Delta t \mathcal{L}'_{\delta}(\mathbf{u}_{it}^{(i)}) (\mathbf{N}')^{-1} \mathbf{R}^{(i)}. \end{aligned} \quad (5.15)$$

Substituting Eq. (5.15) into Eq. (5.12), we get

$$\|\mathcal{L}'_{\delta}(\mathbf{I}/\Delta t - a_{ii} \mathcal{L}'_{\delta})^{-1} \mathbf{R}^{(i)}\| \leq \eta \|\mathbf{e}_t\|. \quad (5.16)$$

The presence of $\mathcal{L}'_{\delta}(\mathbf{I}/\Delta t - a_{ii} \mathcal{L}'_{\delta})^{-1}$ makes the relation between $\Delta t \mathbf{e}_{it}$ and $\mathbf{R}$ problem-dependent and difficult to evaluate or to evaluate accurately. As Δt becomes larger $\mathcal{L}'_{\delta}(\mathbf{I}/\Delta t - a_{ii} \mathcal{L}'_{\delta})^{-1}$ becomes closer and closer to a diagonal matrix with diagonal values $-1/a_{ii}$, and at the limit $\Delta t \rightarrow \infty$, Eq. (5.16) becomes

$$\|\mathbf{R}^{(i)}\| \leq \eta a_{ii} \|\mathbf{e}_t\|, \quad (5.17)$$

with $0.25 < a_{ii} < 0.45$ in the ESDIRK schemes adopted. When $\Delta t \rightarrow 0$, the entries of the matrix $\mathcal{L}'_{\delta} \mathbf{u}_{it}^{(i)} (\mathbf{I}/\Delta t - a_{ii} \mathcal{L}'_{\delta} \mathbf{u}_{it}^{(i)})^{-1}$ will tend to zero. Therefore, a even

larger Newton tolerance will ensure that Eq. (5.16) is satisfied. Overall, Eq. (5.17) appears to provide a good estimate of the convergence upper bound at least at the limiting cases $\Delta t \rightarrow \infty$ and $\Delta t \rightarrow 0$. Based on numerical experiments, we observe that the Newton tolerance of Eq. (5.9) seems good enough to maintain temporal convergence accuracy, as will be shown in Section 5.4. However, if the assumptions made in deriving of the Newton tolerance are invalid, $\mathcal{L}'_{\delta} \mathbf{u}_{it}^{(i)} (\mathbf{I}/\Delta t - a_{ii} \mathcal{L}'_{\delta} \mathbf{u}_{it}^{(i)})^{-1}$ can be estimated using the Jacobian matrix or simply by a calibration for a specific problem.

Eq. (5.9) is less problem dependent than other Newton tolerances commonly used such as,

$$\begin{aligned}\tau_1 &= \theta_1 \|\mathbf{v}^0\|, \\ \tau_2 &= \theta_2 \|\mathbf{N}(\mathbf{v}^0)\|,\end{aligned}\tag{5.18}$$

as is shown in references [110, 18, 111] and will be further shown in Section 5.4. It will also vary in a more consistent manner with the temporal error when Δt changes. In the following, a value $\theta_2 = 10^{-3}$ in Eq. (5.18) is adopted unless otherwise specified for simulations not using the adaptive Newton tolerance.

5.3 Summary of the adaptive strategy and discussion

The basic idea behind the time step adaptation strategy proposed in Section 5.1 is that the relation between temporal and spatial errors 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 to that behind the convergence tests in CFD, which is also a key point in guaranteeing high

efficiency implicit time integration methods.

The implicit time integration methods could be more efficient than the explicit methods in some stiff flows such as problems with low Mach number, high Reynolds number and highly stretched grids [10, 143, 110, 111, 159]. 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 accuracy of the simulations. In other words, 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 integration methods [15, 69], but is based on the computational 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 it is not known before an accurate simulation result is obtained.

Similarly, the error condition behind the time adaptation strategy discussed in Section 5.1, is to achieve the maximum value of Δt without obviously degrading the numerical accuracy of the results of the discrete PDE (partial differential equation) system. Although convergence tests are still needed to ensure the physical accuracy, the strategy avoids unsuitable error relations, which may seriously lower the efficiency without obviously improving the results. In a case presented in reference [110], the unsuitable relations of the spatial and temporal errors can increase the CPU time by

¹These 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 achieve 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.

about a factor of 20 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 reference [111]. However, Eqs. (2.41) and (5.1) 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 advisable to refine the mesh instead of limiting the time step to improve the computation accuracy in our time adapting strategy when the adaptive time step is too large to capture the unsteady flow.

As a by-product of this observation, the adapted time step can also serve as an indicator of mesh refinement if the temporal scale of interest is well defined and the properties of the ODE solver (ESDIRK in this paper) is well understood. For a specific ODE solver, such as DIRK, we can assess its ability to predict a single wave at a specific time interval with a given time step [69]. If the unsteady flow scale (unsteady frequency) of interest is clearly defined in our simulations, this allows us to determine if the unsteady flow scale is satisfactorily captured with the ODE solver and the adapted Δt . If the temporal error is too large to be satisfactory, relation (5.1) tells us that decreasing the time step cannot improve the results, and that a more efficient way to increase accuracy is to refine the spatial mesh instead. Therefore, we can use the adaptive time steps as a criteria for spatial mesh refinement.

In the adaptative strategy, extra computational costs are required to estimate spatial and temporal errors. The embedded scheme of Eq. (2.26) requires an extra implicit RK stage, which will approximately increase the computational cost by $1/(S - 1)$, where S is the number of stages of the main ESDIRK scheme, and

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. (5.2) requires the calculation of a higher-order spatial discretization operator $L_\delta^{P+1}(\mathbf{u}_{it}^{n+1})$. The extra cost from this calculation is relatively lower than those in temporal error estimation for stiff problems since up to an order of magnitude or more L_δ operations are needed in the extra implicit RK stage. For example, in simulations with $P = 2$ and $P = 3$, the extra cost of evaluating $L_\delta^{P+1}(\mathbf{u}_{it}^{n+1})$ is only about 1.5 times the cost of evaluating $L_\delta^P(\mathbf{u}_{it}^{n+1})$, while the extra costs in temporal error estimating is more 10 times of that. Therefore, we can roughly estimate the extra cost to be approximately $100/(S - 1)\%$. For simulations with small time scale changes, a freezing of the adapted time step can be adopted to reduce the additional costs. Currently the adaptive time step is implemented to updated simultaneously with the updating of preconditioning matrices as discussed in Section 4.2.2.

Compared to the adaptive time-stepping strategies discussed in references [110, 111, 18, 16], 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 define temporal error level. Therefore, it avoids problem-dependent user-defined tolerances.
- An embedded ESDIRK scheme with an order of $N + 1$ is employed instead of schemes of $(N - 1)$ th order, which predicts the temporal error more accurately, and also helps avoiding problem-dependent calibration parameters in the estimation of errors.
- The adaptation is determined element by element first instead of using the L_2 norm of the temporal error in the whole flow field. Therefore, it has the

potential to avoid the time step being determined primarily by the large-scale unsteady structures and thus avoid small-scale unsteady structures being under-resolved.

5.4 Numerical tests of the parameter choices

Numerical experiments are conducted in this section to demonstrate the performance of the adaptive methods proposed. Section 5.4.1 presents the simulation of an isentropic vortex to validate the Newton tolerance choice and illustrate some basic properties of the adaptive time-stepping. A steady-state flow past a flat plate is adopted in Section 5.4.2 to demonstrate the performance of the adaptive method in steady-state simulations. Sections 5.4.3 and 5.4.4 study the Taylor-Green vortex problem and the turbulent flow over a circular cylinder at $\text{Re} = 3900$ to illustrate the performance of the methods in freely decaying and wall-bounded turbulent flow simulations, respectively.

5.4.1 Isentropic vortex

The basic simulation settings for this test case have been given in Section 4.4.1. Here, $(u_0, v_0) = (1.0, 0.0)$ is adopted. To evaluate errors, various 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 (4.4). As in reference [15], the difference between the solution and a solution with a very small time step is representative of 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 can be regarded as the spatial error, since the

error here is dominated by the spatial error.

Analysis of the adaptive Newton tolerance

This section compares the performance of the proposed Newton tolerance, given by Eq. (5.9) and $\eta = 0.1$, with commonly adopted tolerances given by Eq. (5.18) using values $\theta_1 = (10^{-6}, 10^{-10})$, and $\theta_2 = (10^{-3}, 10^{-7})$. The main purpose is to show that Eq. (5.9) can give a reasonable tolerance and still maintain temporal accuracy. In this case, we employ $P = 2$ and $N = 3$ with $h = 1/3$ and three different time steps: $\Delta t = 0.05, 0.1$, and 0.2 .

Fig. 5.1 shows that large values of the parameters θ_1 and θ_2 in the Newton tolerances (5.18) lead to a degradation of the order of temporal accuracy. Small values of these parameters achieve the desired temporal order of accuracy but are not necessarily the best 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. (5.18) does not scale simultaneously with the temporal error. On the other hand and as shown in Fig. 5.1, the adaptive Newton tolerance (5.9) 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 ensures the 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 to maintain temporal accuracy but not too small to maintain efficiency. Tab. 5.2 shows that the ratios between the adaptive Newton tolerance and the τ_{max} are of the same order of 0.1 for different time steps. This indicates that the iterative error scales simultaneously with the temporal error and $\eta = 0.1$ is close to the maximum value that can maintain temporal accuracy in

this test.

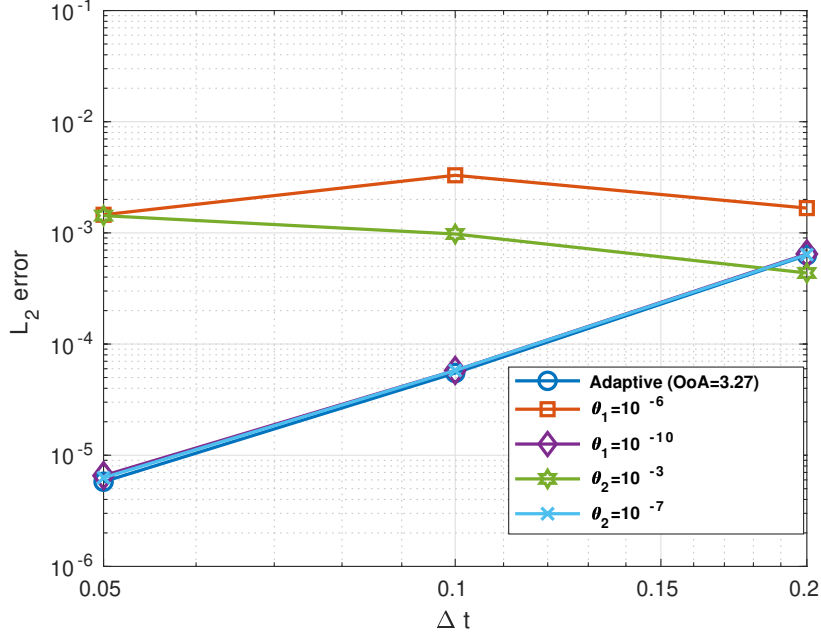


Figure 5.1: Isentropic vortex problem: Effect of Newton tolerances in Eqs. (5.18) and (5.9) on temporal accuracy.

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

Analysis of errors in the adaptive time stepping

We have described in Section 5.4.1 how to choose the tolerances in Newton iteration to ensure that the iterative error is smaller than the temporal error. 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, and CFL numbers $\text{CFL}=(10.0, 1.0, 0.2)$ using different mesh sizes $h = (0.2, 0.1, 0.067, 0.005)$ are presented in Fig. 5.2. The corresponding temporal errors are shown in Fig. 5.3. 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 $\text{CFL}=10.0$, the total errors are dominated by the temporal errors and converge at a lower order of 3 instead. For simulations with $\text{CFL}=0.2$, the total errors are dominated by spatial errors. However, the temporal errors are much smaller than the spatial errors, which may lead to extra computational costs without obviously improving the results. For simulations with $\text{CFL}=1.0$, the 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 references [111, 18], the temporal error curve should be a horizontal line because the temporal error tolerance in these methods is user-defined and independent of h .

The CPU time of the above simulations is compared in Fig. 5.4. 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. 5.4 also shows the corresponding curves of GMRES iteration number and residual evaluation number. These numbers are summed over all the time steps during the time interval. The GMRES iteration number and the residual evaluation number of the adaptive method are also close to

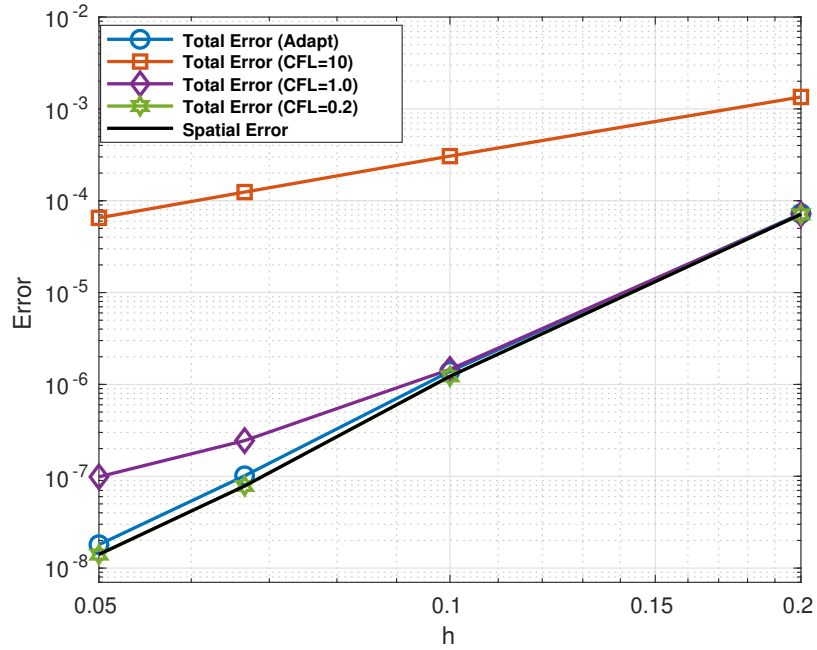


Figure 5.2: Isentropic vortex problem: Effect of time-stepping method on total errors.

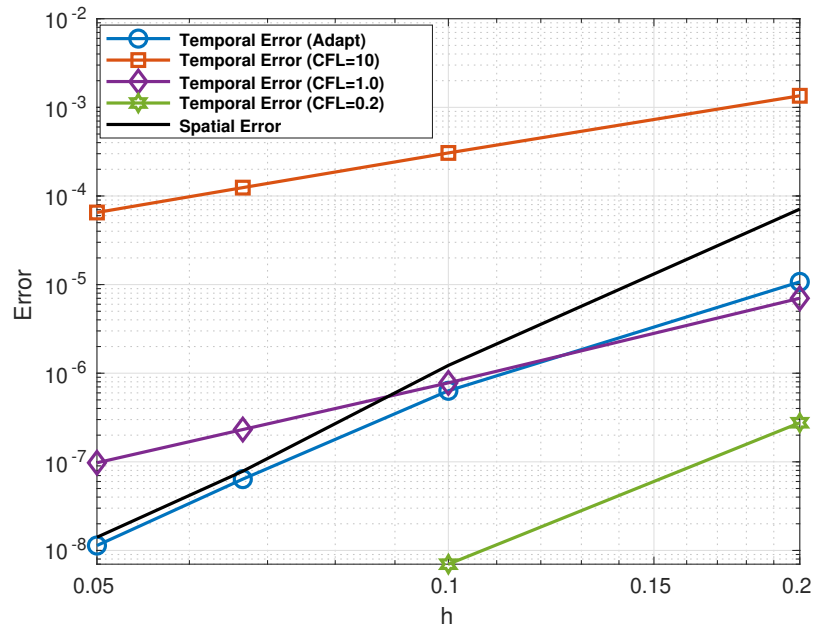
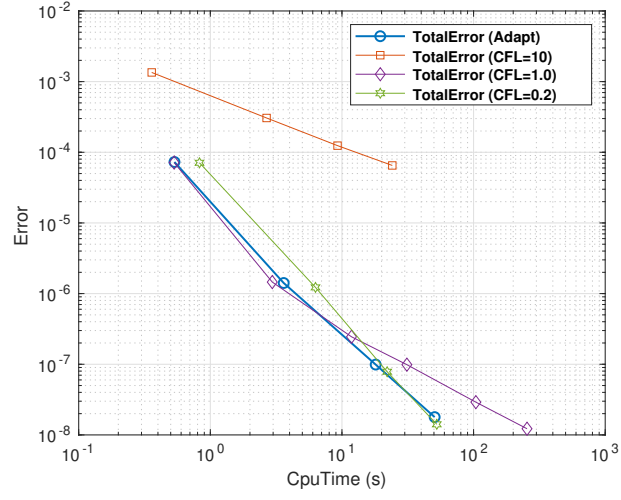
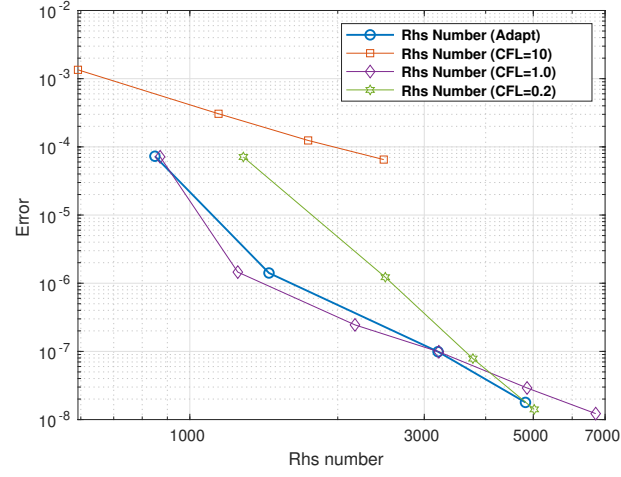


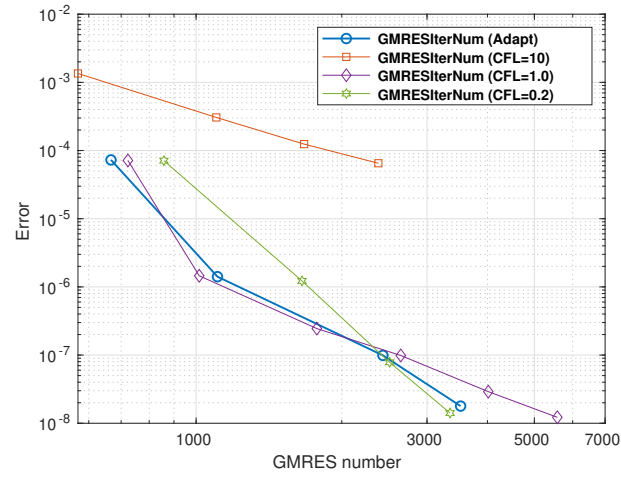
Figure 5.3: Isentropic vortex problem: Effect of time-stepping method on temporal errors and spatial errors.



(a) CPU time to compare efficiency.



(b) Residual evaluation numbers.



(c) GMRES iteration numbers.

Figure 5.4: Isentropic vortex problem: Effect of time-stepping method on efficiency.

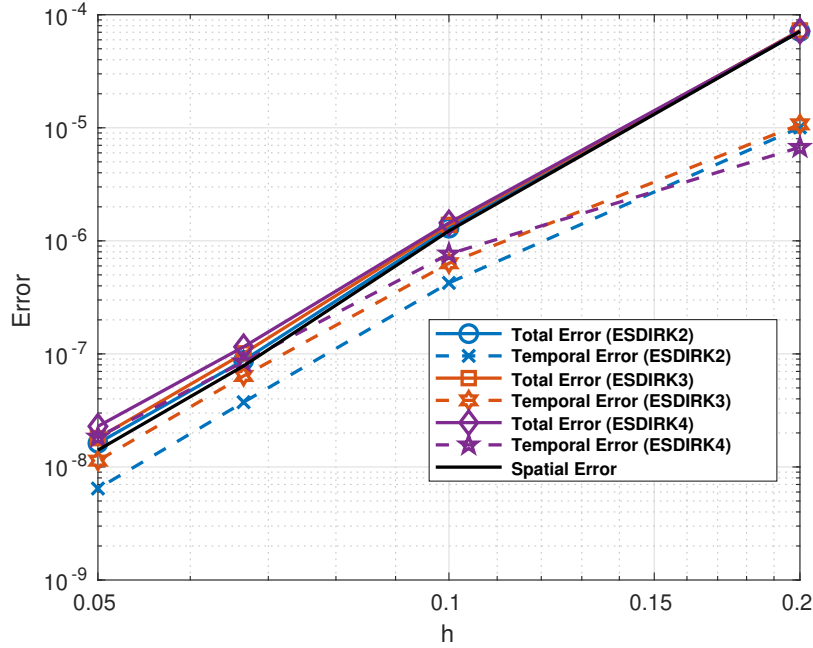


Figure 5.5: Isentropic vortex problem: Performance of adaptive time-stepping for different ESDIRK schemes.

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 higher at the same error level than that of 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.

The errors of ESDIRK schemes with $N = 2, 3, 4$ are compared in Fig. 5.5. 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 should also be achieved by the adaptive methods proposed in references [111, 18], but it requires an additional calibrating process which is problem-dependent

5.4.2 Flat plate boundary layer

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 case considered here is the boundary-layer flow past a flat plate at a Mach number $\text{Ma} = 0.1$ and a Reynolds number $\text{Re} = 1.6 \times 10^6$. The geometry and boundary conditions are depicted in Fig. 5.6. Details of these boundary conditions can be found in references [159, 97]. The simulations are performed using $P = 2$ and $N = 3$ with the highly stretched mesh shown in Fig. 5.7(a). The computed profile of the horizontal velocity is shown in Fig. 5.7(b), which is in good agreement with the analytical Blasius profile. 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 ensure 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 in order to show that the proposed adaptive strategy can adjust itself based on the unsteadiness properties of the problem. For comparison, a time-stepping strategy based on a growing CFL, which starts from $\text{CFL}=0.1$ and grows by rates of 1.5 and 2.0 up to $\text{CFL}=5 \times 10^4$, is also presented. A relative Newton tolerance $\theta_2 = 10^{-6}$ in Eq. (5.18) is adopted since the default choice $\theta_2 = 10^{-3}$ leads to blowing up at such a large CFL number. 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 allows to accelerate the convergence when the flow field is close to the steady-state solution [23, 144].

The convergence history of the normalized residual norm $\|L_\delta(\mathbf{U})\|$ is presented in Fig. 5.8. The simulation with the adaptive time-stepping proposed in Section 5.1

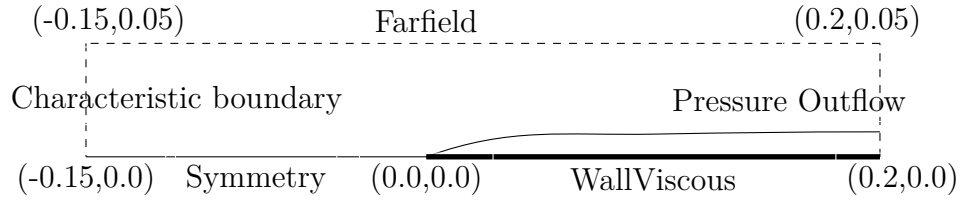


Figure 5.6: Boundary-layer flow past a flat plate: Geometry and boundary conditions.

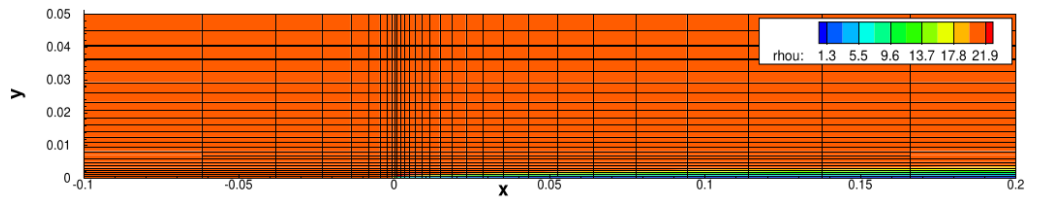
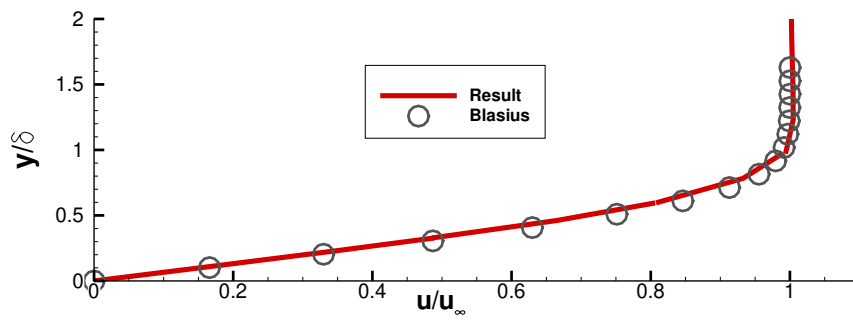
(a) Mesh and x -momentum.(b) Velocity profile, where δ is the boundary layer thickness.

Figure 5.7: Boundary-layer flow past a flat plate: mesh and velocity profile.

converges almost as fast as the simulation with a CFL growth rate of 2.0. The residual, Δt and CPU time at different steps are illustrated in Fig. 5.9. It shows that the Δt of the adaptive method grows much slower during the first 100 time steps. However, the extra computational cost due to the extra steps is low since it is cheap to converge at a small Δt . Fig. 5.9 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. (2.27) 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 Eqs. (5.1) and (2.27). The adaptive time-stepping adjusts the Δt based on 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 multigrid, it is general and is capable of efficiently simulating a wide range of steady and unsteady flows with a fixed set of parameters.

5.4.3 Taylor-Green vortex

Here we apply the adaptive time-stepping strategy to the simulation of the Taylor-Green vortex (TGV), which represents an unsteady flow of decaying vortices. The evolution of the TGV includes the breakdown of the initially large vortices, transition to turbulence and decay of turbulence. With a very simple set-up but a complex flow evolution, it is a very good test case for transitional and turbulent simulations. Initially, large vortices are placed in a cubic domain $[-\pi, \pi]^3$ with periodic boundary

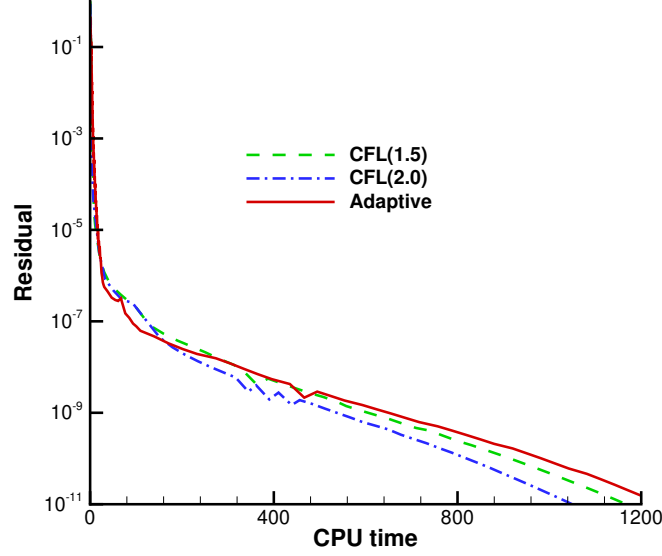
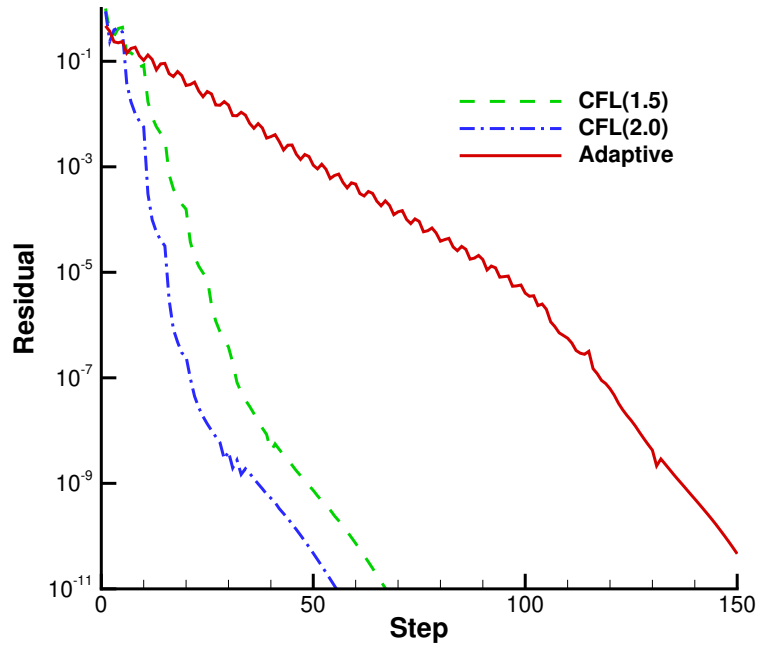


Figure 5.8: Boundary-layer flow past a flat plate: Residual convergence history. The number in the parenthesis indicates the growth rate of the CFL number.

conditions. The expression of the initial flow field is [39]

$$\begin{aligned}
 u &= V_0 \sin\left(\frac{x}{L}\right) \cos\left(\frac{y}{L}\right) \cos\left(\frac{z}{L}\right), \\
 v &= -V_0 \cos\left(\frac{x}{L}\right) \sin\left(\frac{y}{L}\right) \cos\left(\frac{z}{L}\right), \\
 w &= 0, \\
 p &= p_0 + \frac{\rho_0 V_0^2}{16} \left(\cos\left(\frac{2x}{L}\right) + \cos\left(\frac{2y}{L}\right) \right) \left(\cos\left(\frac{2z}{L}\right) + 2 \right),
 \end{aligned} \tag{5.19}$$

where $V_0 = 1.0$, $\rho_0 = 1.0$ and $L = 1.0$ in our non-dimensional computational setting and p_0 is calculated based on the $\text{Ma} = 0.1$ and the equation of state (2.5). A $\text{Re} = 1600$ is used to calculate the viscosity coefficient. The simulations are run on a 32^3 uniformly distributed mesh with $P = 4$ and $N = 4$. Note that, in order to simulate nearly incompressible conditions, the flow is taken to be compressible but at a low Mach number of $\text{Ma} = 0.1$.



(a) Residual.

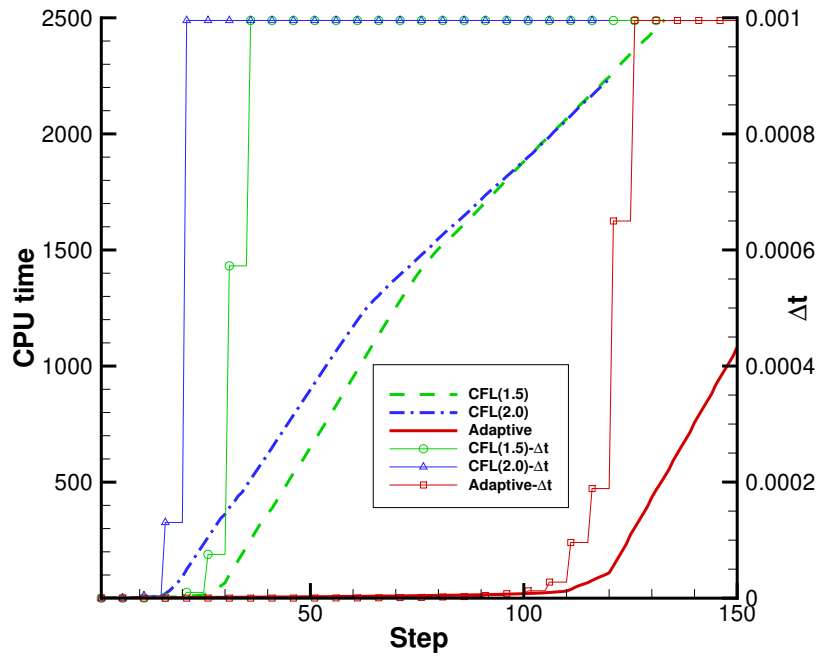
(b) CPU time and Δt .

Figure 5.9: Boundary-layer flow past a flat plate: residual, CPU time and time step evolution curves as a function of steps. The number in the parenthesis indicates the growth rate of the CFL number.

Fig. 5.10 presents the evolution of $2\mu\varepsilon/\rho$, where ε denotes the enstrophy. According to reference [39], this variable is a good estimate of the kinetic energy dissipation rate in the incompressible limit. Fig. 5.11 presents the errors of the dissipation rate as defined by Eq. (18) of reference [39]. 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 fixed time steps of $\Delta t = 0.1$ 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. 5.11. The error with adaptive time-stepping and $\beta = 0.1$ is slightly larger than other errors. The error with $\beta = 0.01$ almost coincides with the error of with $\Delta t = 0.1$. Both simulations with $\beta = 0.1$ and $\beta = 0.01$ have very good temporal accuracy especially when compared with the spatial error, which can be roughly estimated by the deviation from the DNS result [39].

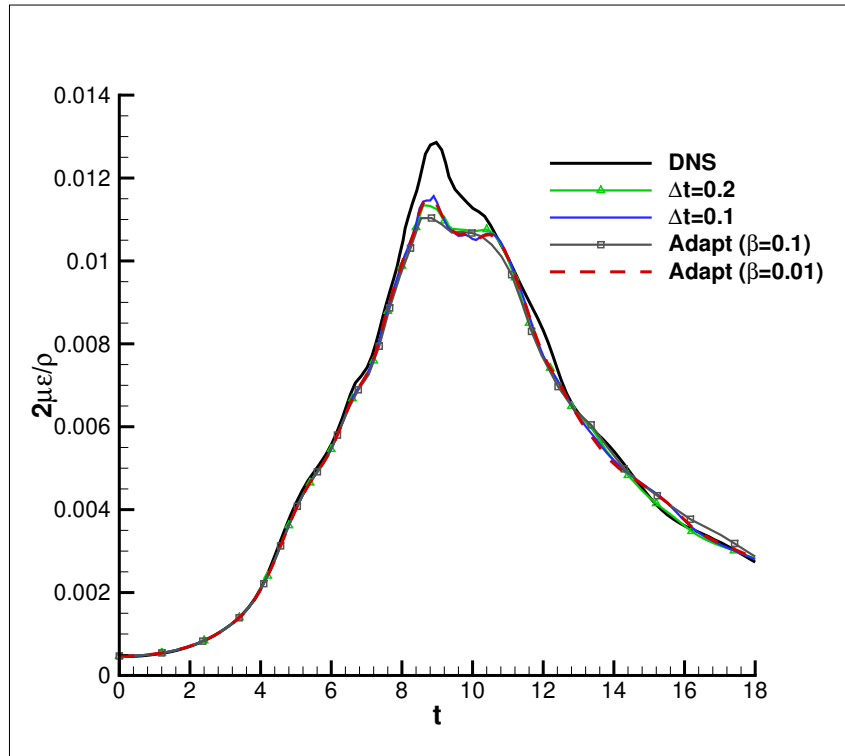


Figure 5.10: Taylor-Green vortex: Evolution of the enstrophy.

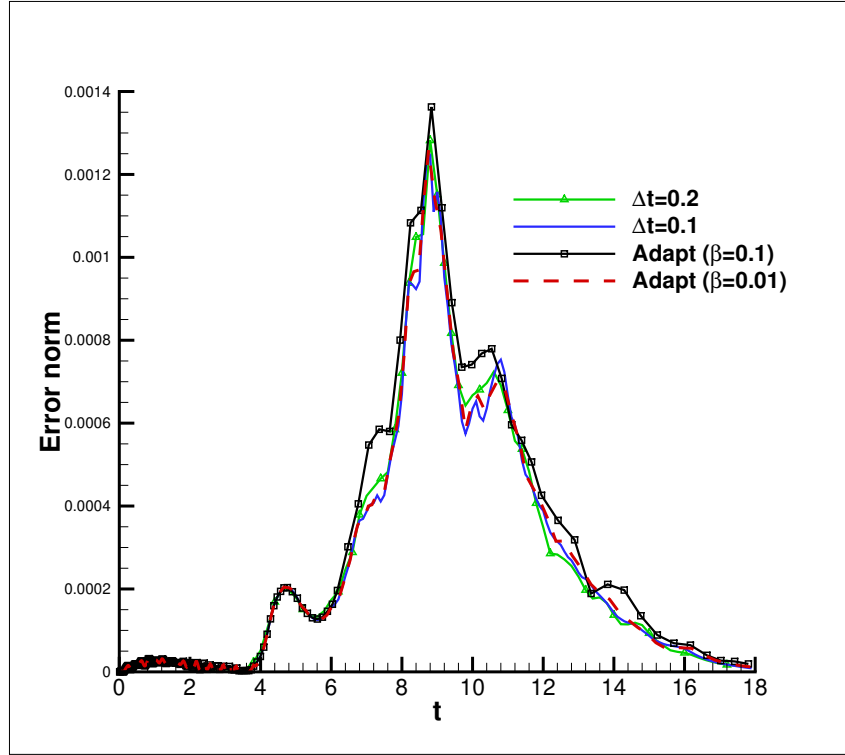


Figure 5.11: Taylor-Green vortex: Error norms of the kinetic energy dissipation rate.

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 serves as an implicit modelling of the under-resolved scales, which is referred to as the implicit large eddy simulation (LES) model [61].

To avoid the temporal error from influencing the modelling of the under-resolved scales, a smaller temporal error compared with the spatial error, $\beta = 0.01$, is adopted in implicit LES of turbulent flows in our study. This approach gives reasonable results as shown in Fig. 5.11 and in Section 5.4.4. The computational CPU time of the simulations with $\beta = 0.01$ is slightly larger than that of the simulation with $\Delta t = 0.1$ (1.27 times) partly because of the additional costs required for the error

estimation.

5.4.4 Turbulent flow over a circular cylinder at $\text{Re} = 3\,900$

The flow over a circular cylinder at $\text{Re} = 3900$ is another widely used test case for turbulence simulations. Various experimental and numerical results can be found for this test case, e.g. [114, 83]. Compared with the TGV, there are extra complexities because of grid stretching, the need for simulating boundary layers, and the instability of shear layers. The computational domain boundaries are located $40D$ away down stream and $20D$ away on other directions. Riemann invariant based boundary conditions are applied at the outer boundaries. The mesh near the cylinder is shown in Fig. 5.12. The mesh consists of 123 360 curved unstructured hexahedral elements in total, which corresponds to about 3.3M DoFs for $P = 2$. The smallest mesh size is approximately $0.006D$, which is located on the cylinder surface and is chosen based on the estimate advocated by reference [83]. We use $P = 2$ with consistent integration [100]. The computed Q-criteria iso-surface, which is a good representation of vortex structures [70], using $P = 2$ is illustrated in Fig. 5.12, which shows the complex turbulent flow structures in the wake of the cylinder. Reference [159] 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 reference [159]. Here we simulate the same case using the ESDIRK2, ESDIRK3 and ESDIRK4, together with the adaptive time-stepping strategy to demonstrate their performance.

The adaptive time steps for a duration corresponding to approximately 19 vortex shedding periods are shown in Fig. 5.13. Tab. 5.3 presents the corresponding averages, $\Delta \bar{t}$. The averaged time step of ESDIRK2 is approximately 0.027, which is 2.7 times

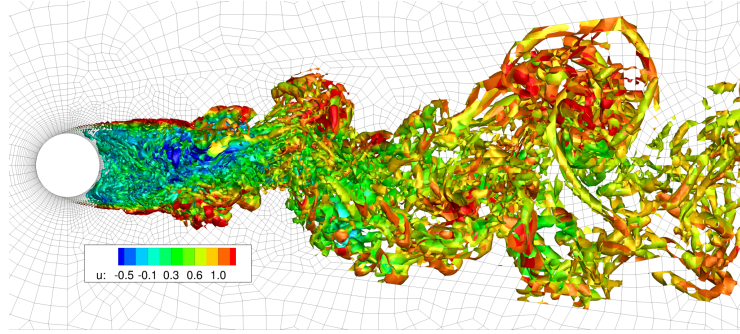


Figure 5.12: Turbulent flow over a circular cylinder: Q-criteria iso-surface ($Q = 5$) and mesh from reference [159].

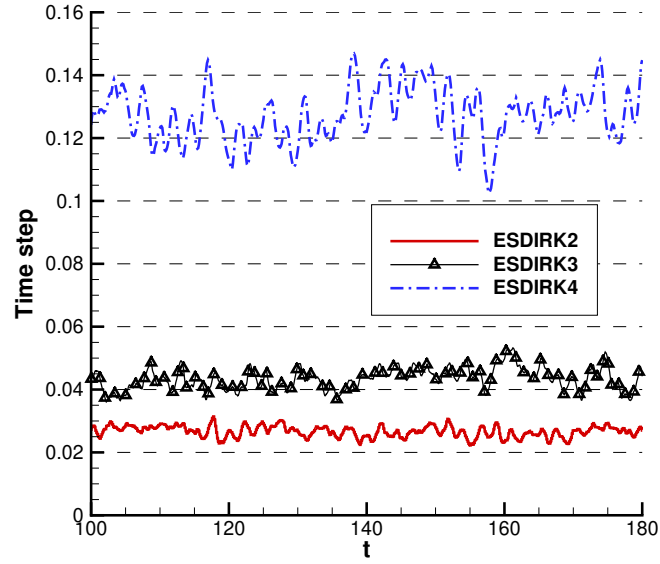


Figure 5.13: Turbulent flow over a circular cylinder: Adaptive time steps (Δt) of ESDIRK2, ESDIRK3 and ESDIRK4.

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 5.3: 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.

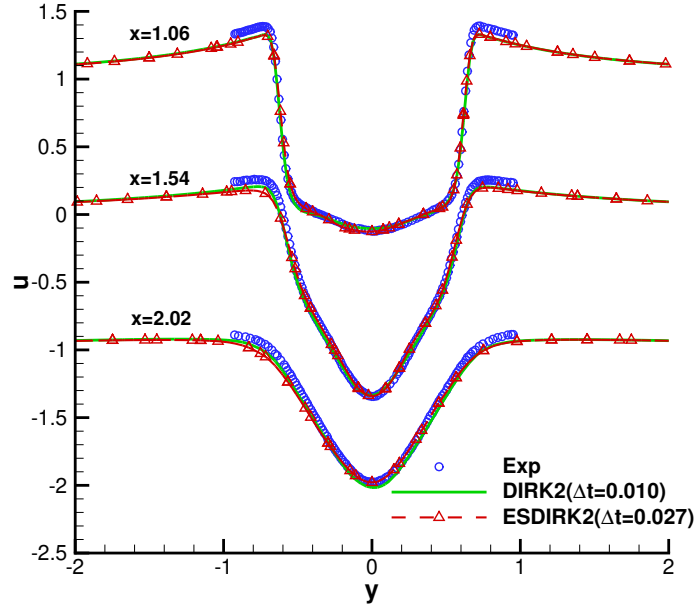


Figure 5.14: Turbulent flow over a circular cylinder: Comparison of time-averaged velocity distribution. The velocity profiles at $x = 1.54$ and $x = 2.02$ are shifted down by increments of $u = -1$ and $u = -2$, respectively.

larger than the $\Delta t = 0.01$ of the DIRK2. 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 higher-order ESDIRK schemes.

Tab. 5.3 summarizes the averaged time steps and CPU time of the simulations. The results show that the simulations with 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. 5.14. The result of ESDIRK2 with $\Delta t = 0.027$ almost coincides with the result of DIRK2 with $\Delta t = 0.01$, and both are in a good agreement with experimental results [114]. The simulations with adaptive time step give accurate predictions of the time-averaged quantities.

Fig. 5.15 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 colouring is cut below the global time step and these elements are presented in white. One important observation is that the relation of Eq. (5.1) is satisfied in the majority of elements because the elemental time step is smaller than the global time step in fewer than 1% of the elements. The influence of ϵ_m in Eq. (5.5) is negligible in the global time step in this test case, which is only effective in elements with very small error magnitudes.

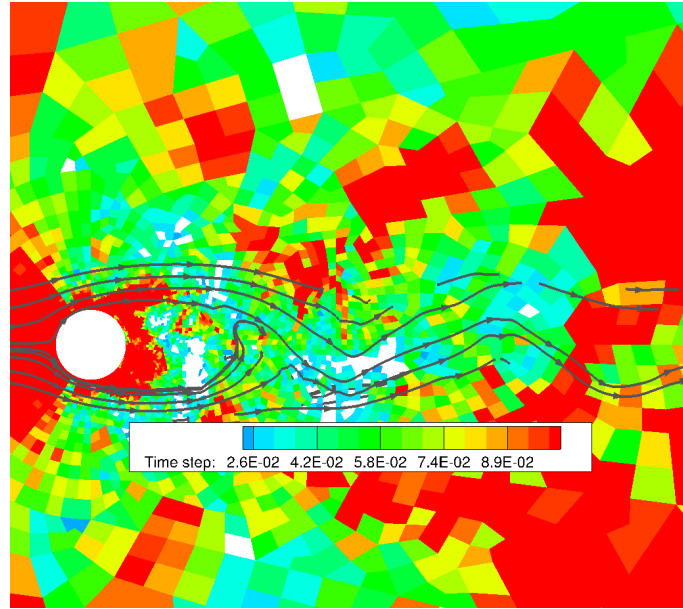


Figure 5.15: Turbulent flow over a circular cylinder: Spatial distributions of the values of the elemental time steps.

5.4.5 Summary of parameters

The tunable parameters in the adaptive strategy are listed in Tab. 5.4. The parameter β controls the relation between the temporal error and spatial error, which is set to 0.1 for general well-resolved simulations, but to 0.01 for implicit under-resolved LES as discussed in Section 5.4.4. Similarly, η controls the relation between the temporal

error and iterative error, which is set to 0.1. The value r in Eq. (5.7) 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. (5.5) avoids division by zero and also serves as a threshold values for the error level of interest. The current choice of Eq. (5.6) is motivated by the truncation error of the time integration process. In determining β and η , we assume the spatial error is dominating. However, no optimization on the values of β and η is performed, since the optimal values are usually problem-dependent.

Parameter	Position	Range	Current choice
β	Eq. (5.1)	$0 < \beta < 1$	$\beta = 0.1$ or $\beta = 0.01$
η	Eq. (5.9)	$0 < \eta < 1$	$\eta = 0.1$
r	Eq. (5.7)	$r \in [0, \infty)$	$r = 1$
ϵ_m	Eq. (5.5)	Small relative to $\mathbf{U}$	Eq. (5.6)

Table 5.4: Summary of tunable paramters in 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 5.5: 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.

On the other hand, in the simulations without 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 Tab. 5.5. These case by case choices are obtained at the cost of repeated numerical experients a priori, which could be expensive especially when starting the simulations of a new test case. The comparison of

computational efficiency in Tab. 5.5 shows that the adaptive strategy is close to the most efficient choices in all the cases without case by case optimised parameters, which demonstrates the remarkable generality of the adaptive strategy.

Chapter 6

Application to high-fidelity simulations

Any high-fidelity CFD framework, especially if used in industry, should satisfy minimum requirements in terms of numerical stability or robustness, and physical accuracy. The first requirement guarantees that the simulation will not unexpectedly crash, thereby undermining the computational pipeline. The second assures that the results obtained represent the underlying physics faithfully.

In the following, we detail the stabilization techniques incorporated into the Nektar++ library of the compressible flow solver (Section 6.1), and discuss the use of the DG method as a means to construct an implicit LES (also referred to as under-resolved DNS) setting, by tuning the size of the element and polynomial order space, thus modifying the dispersion and diffusion properties of the discretization. Application of the implicit solver to the Taylor-Green vortex, flow over a circular cylinder at $Re = 3900$, and turbulent channel flow are presented in Sections 6.2, 6.3 and 6.4, respectively. The performance of the implicit solver is summarized in

Section 6.5.

6.1 Stabilization techniques

SEMs are known to suffer from a lack of numerical stability in some situations, and the associated simulations can fail without notice [80]. Several solutions have been proposed in the literature to address this robustness issue, ranging from efficient consistent-integration techniques [97], to entropy-stable split forms [58, 57], and their interplay [157]. In Nektar++, we implement primarily consistent-integration schemes that can address the aliasing-driven instabilities due to the under-integration of the nonlinear terms that arise in both the compressible Euler and Navier-Stokes equations. Although it is impossible to completely eliminate the aliasing error in the polynomial functional space due to the existence of rational functions of the polynomial approximations in the flux evaluations in Eq. (2.9), the consistent-integration methods are able to effectively improve their robustness [97, 157]. Through consistent-integration, both geometrically- and PDE-induced aliasing can be tackled.

In under-resolved simulations like implicit LES, a large proportion of energy is distributed near or even beyond the Nyquist wavenumber [54, 51]. Thus, proper dealiasing becomes crucial for stabilizing simulations. The consistent-integration is tested in implicit LES simulations [157], which shows that it can maintain stable simulations for DG of low-to-medium order ($P < 5$).

When simulating challenging compressible viscous flow configurations, especially in the context of non-canonical problems typically found in industrial applications, it is common to face under-resolution issues. In these cases, a portion of the features characterizing the flow contains scales that are smaller than the grid Nyquist

wavenumber [54, 51] and thus cannot be captured by the numerical methods. In order to correctly reflect the physics without resolving all the scales, there are two approaches to model the sub-grid scale (SGS) contributions: (i) explicit filtering and SGS modelling and (ii) implicit LES SGS modelling approaches [4, 61]. The first explicitly filters out the SGS waves and uses a SGS model to recover the physical mechanisms of the under-resolved scales in turbulence simulations [76]. The second relies on the numerical schemes to filter the under-resolved scales and to reflect the influence of SGS to large scales [127], which is often referred to as under-resolved DNS (uDNS) or implicit LES (iLES)(see e.g. [105, 104, 51]).

The use of the underlying numerics as an implicit sub-grid scale model offers some advantages over more traditional explicit LES modelling techniques. Firstly, it does not require complex SGS models [127]. Secondly, explicit LES models are commonly expensive, thereby allowing a lower number of degrees of freedom per given computational resources than implicit LES techniques. Therefore, by using implicit LES techniques, the turbulence physics can be more accurately captured for a broader range of scales by using a higher grid resolution. And finally, the development of an explicit filter itself is not straightforward on complex unstructured meshes, which also relies on the dissipation and dispersion properties of the numerics just as the implicit modelling. In addition, the widths of the explicit filters are usually larger than those of the implicit filter provided by the numerical schemes to avoid the numerical errors from influencing the explicit SGS model. This in turn makes the difference of effective resolution between implicit and explicit LES larger.

In the compressible flow solver, the implicit SGS modelling strategy is used for under-resolved turbulence simulations. Three important aspects of numerical methods have been studied to construct an effective implicit SGS modelling:

- Stability of under-resolved simulations. Ensuring stability is the prerequisite for successive LES simulations. As stated above, using consistent-integration is effective in solving the instability problems arising from aliasing error. It was also revealed that using Roe-like numerical fluxes can prevent over-upwinding and thus avoid energy pileups near Nyquist wavenumber [157, 98], which is a sign of insufficient de-aliasing.
- Equivalent filtering width of the numerical scheme. Accurately estimating the equivalent filtering width of the numerical scheme is important in maintaining the SGS cutoff is located in the inertial sub-range. In [105], a 1% threshold (also referred to as 1% rule [105]) is a good estimate of the scale where the dissipation in the energy spectral is significant. Thus, it is possible to define equivalent Gaussian filters of different numerical schemes. Combining with the *hp*-space tuning, desired dispersion and dissipation properties can be achieved to filter out the SGS waves in the inertial sub-range.
- Similarity to physics-based SGS models. The dispersion and dissipation properties of DG methods are shown to mimic some explicit SGS models [38]. This is mainly because of the low dispersion error and the sharp dissipation concentrated near the Nyquist wavenumber [54, 51]. Although a non-modal analysis shows that the dissipation may not be sufficient in highly under-resolved simulations for the hybridized DG method with $P > 4$ [51], the DG methods based iLES has been shown to give good predictions of turbulent flows [10, 38, 52], which are no worse than those of the state-of-the-art explicit LES simulations [38, 52].

It should be pointed out that a better knowledge of the dispersion and dissipation properties of the numerical discretization as well as their relation to the physical

SGS models also allows better explicit LES modelling, as one can ensure that the explicit filter width adopted is well separated from the Nyquist wavenumber [54, 51] provided by the numerics.

6.2 Taylor-Green vortex

The detailed computational settings for this case have been provided in Section 5.4.3 and will not repeat here. The value $2\mu\varepsilon/\rho$, where ε is the enstrophy, is a good estimate of the energy dissipation rate at the incompressible limit [39]. Fig. 6.1a displays evolution curves of $2\mu\varepsilon/\rho$ obtained with 32^3 meshes and using resolutions at different polynomial orders. Dealiasing via consistent integration with $3(P+1)/2$ quadrature points in each spatial direction [157] is used in the under-resolved simulations of the TGV. The curves gradually tend to the DNS result [39] as the resolution increases, which shows the implicit solver can correctly resolve the main flow phenomena. All these simulations are run using SDIRK2 with a CFL_c smaller than 2.5. The details of the implicit solver and their comparisons with ERK2 scheme are presented in Tab. 6.1. Although there is no grid stretching in this test case, the implicit solver can still be as efficient as, or more efficient than, the explicit solver. Fig. 6.1b compares results calculated using SDIRK2 with different time steps with the result obtained with ERK2, which is very accurate in time because of the very small time step used ($CFL = 0.08$). Three different time steps corresponding to CFL values of 0.05, 0.1 and 0.2 are used for SDIRK2. Although the corresponding temporal errors of SDIRK2 could change by 16 times for different time steps, all these results agree quite well with the results of ERK2. This means the temporal error is very small for the implicit solver. Given the large differences for different spatial resolutions, the implicit solver with the current parameter choice is accurate enough in the sense that

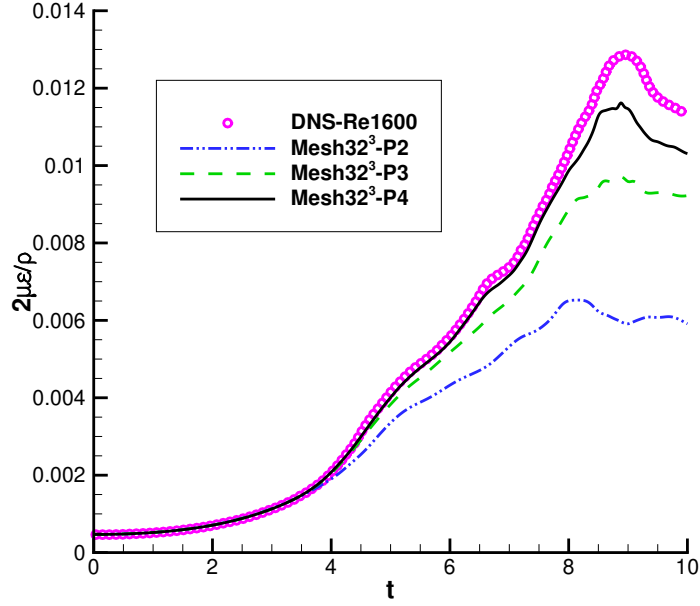
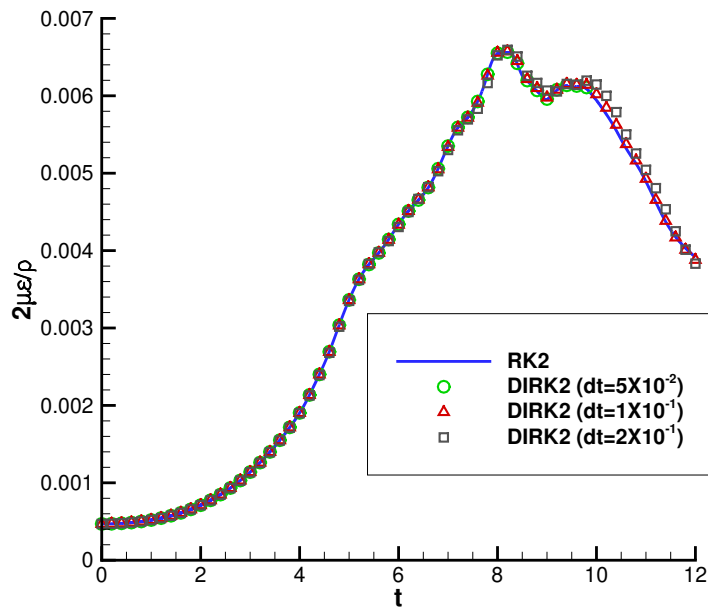
the error of the solution is dominated by the spatial error as discussed in Section 5.3. Finally, Fig. 6.2 shows instantaneous vorticity distributions obtained using ERK2 and SDIRK2. They are in good agreement with each other which further verifies the accuracy of the implicit solver.

Polynomial order	$P = 2$		$P = 3$	
Method	ERK2	SDIRK2	ERK2	SDIRK2
Δt	1.1×10^{-3}	0.1	6.1×10^{-4}	0.046
CFL	0.08	7.5	0.10	7.5
CFL_c	2.6×10^{-2}	2.5	1.8×10^{-2}	1.4
CPU time(s)	0.41	12.8	0.74	42.3
Newton iterations per RK stage	\	4	\	5
GMRES iterations per Newton iteration	\	3.7	\	4.6
Speed-up	1.0	2.94	1.0	1.3

Table 6.1: Efficiency comparison between explicit and implicit time integration schemes of TGV. All simulations are conducted using four HPC nodes, the information of which is discussed in Section 4.2.

6.3 Flow over a circular cylinder at $Re = 3900$

Details of the simulation settings for this test case have been presented in Section 5.4.4. As shown in Tab. 6.2, a value $CFL = 40$ is employed in the simulations. The CFL_c value is around 2.5 at this CFL number, which ensures the temporal accuracy of the simulations. The corresponding time step is about 270 times larger than the explicit one with $CFL = 0.15$. On average, the implicit solver converges using 3 Newton iterations per implicit RK stage and 3.0 GMRES iterations per Newton iteration. The implicit solver is 14.3 times faster than the explicit one. The $P = 3$ case is also run for efficiency comparison and, as shown in Tab. 6.2, a speed-up of 12.2 is achieved compared with the explicit solver. The seven-stage third-order low-storage

(a) Different resolutions (variable P).

(b) Different temporal methods.

Figure 6.1: Evolution of the dissipation rate in Taylor-Green vortex.



Figure 6.2: Taylor-Green vortex case: Comparison of contours of surface normal vorticity on the plane $y = \pi$ at $t = 8$ calculated using ERK2 (red dashed lines) and SDIRK2 with $\Delta t = 0.1$ (blue solid lines).

ERK scheme (ERK3-7) of Tab. A.15 of reference [138] is also compared, which runs at a much larger CFL number ($CFL = 0.6$). The implicit solver is 11.5 times faster than the ERK3-7 for the simulation with $P = 2$.

The influence of the Newton iteration tolerance α is also studied here. With $\alpha = 10^{-4}$, $\alpha = 10^{-5}$, $\alpha = 10^{-6}$ and $\alpha = 10^{-7}$, the CPU time normalized by CPU time with $\alpha = 10^{-3}$ is 1.0, 1.3, 1.6 and 1.7, respectively. One can adopt a smaller α if needed at a cost of more CPU time as shown above.

The instantaneous and mean flow field are illustrated in Fig. 5.12 and Fig. 6.3, respectively. The instantaneous Q-criteria [70] distribution in Fig. 5.12 clearly shows the complex turbulent flow structures in the wake of the cylinder. The mean streamwise velocity contour is shown in Fig. 6.3. The statistics is first averaged by 5000 samples during the non-dimensional time interval of 50 and then further averaged using 40 evenly spaced slices in the homogenous direction. The length of

Polynomial order	$P = 2$		$P = 3$	
Method	ERK2	SDIRK2	ERK2	SDIRK2
Δt	3.5×10^{-5}	1.0×10^{-2}	1.56×10^{-5}	5.0×10^{-3}
CPU time(s)	2.1	42.0	3.8	100.3
CFL	0.15	40	0.15	48
CFL_c	9.0×10^{-3}	2.5	6.0×10^{-3}	1.7
Newton iterations per RK stage	\	3	\	3
GMRES iterations per Newton iteration	\	3.0	\	3.3
Speed-up	1.0	14.3	1.0	12.2

Table 6.2: Efficiency comparison between explicit and implicit time integration schemes of turbulent circular cylinder at $Re = 3900$. All simulations are conducted using four HPC nodes, the information of which is discussed in Section 4.2.

the recirculation bubble, which is sensitive to the prediction accuracy, is evaluated to be 1.49 in our simulation, while the experimental result reported in reference [114] is 1.51. The mean velocity and velocity fluctuation distributions are quantitatively compared with other experimental and numerical results [83, 114, 94] in Figs. 6.4 and 6.5. The mean velocities in Fig. 6.4 are in good agreement with the experimental results of reference [114] on the streamwise and crossflow directions at least in the wake region $x < 2.5$. Beyond the position $x = 2.5$ a clear discrepancy is seen in Fig. 6.4a which could be due to a too quick mesh coarsening. The mean fluctuations however coincide well with previous experimental and numerical results as seen in Fig. 6.5.

6.4 Turbulent channel flow

The turbulent channel flow is used to assess the solver accuracy in implicit LES of fully-developed turbulent boundary layer. The test case has been extensively studied using DNS and LES [38, 50, 2, 102], which makes it suitable for validation. Driven

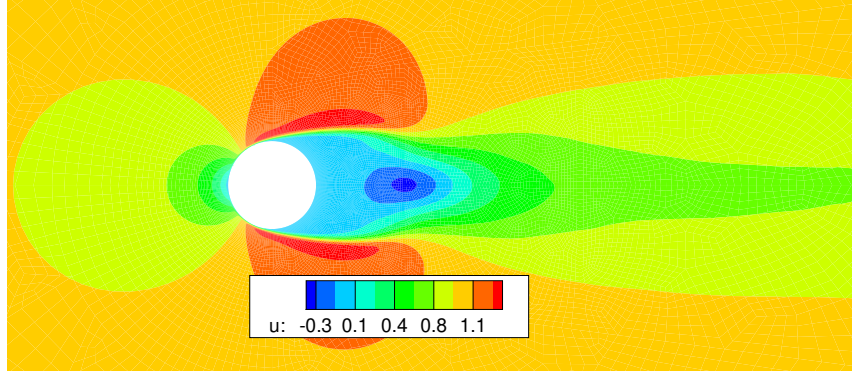
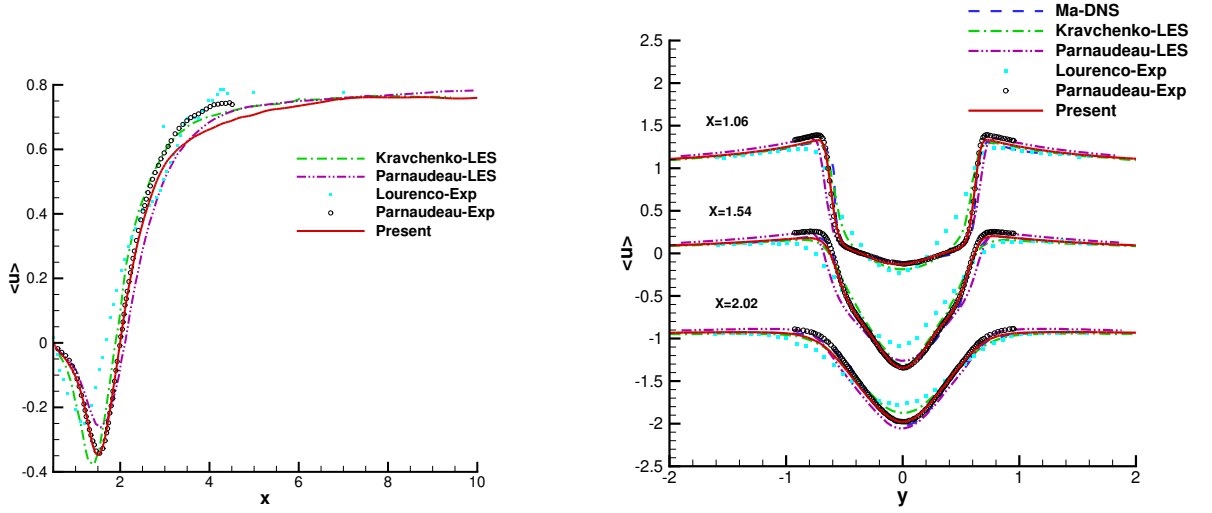


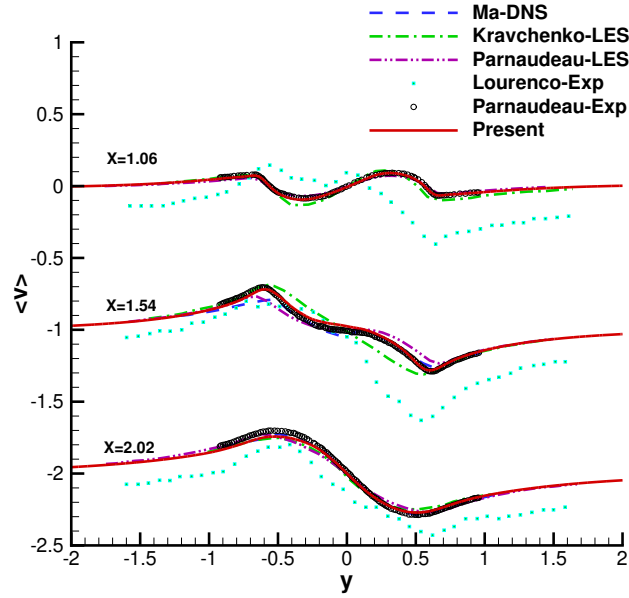
Figure 6.3: Non-dimensional mean u velocity of turbulent flow over a circular cylinder at $Re = 3900$.

by a body force, the flow between two flat plates is simulated with periodic boundary conditions in the streamwise (x) and spanwise (z) directions. At a high enough Reynolds number, the boundary layers near these flat plates become fully developed turbulent boundary layers. The computation domain is $[0, 2\pi\delta] \times [0, 2\delta] \times [0, \pi\delta]$ in the x , y and z directions, respectively, where $\delta = 1$ is used. A forcing vector $\mathbf{f} = [0, \rho u_\tau^2/\delta, 0, 0, 0]^T$ is added to the Navier-Stokes equations to sustain the flow at a friction Reynolds number $Re_\tau = \rho u_\tau \delta / \mu = 550$ where u_τ is the friction velocity. To mimic incompressible flow, a Mach number based on the mean centre-line velocity is set to 0.1. The centre-line mean velocity is $21.0 \times u_\tau$ for $Re_\tau = 550$, which is known from existing DNS results [2]. The coarse, baseline and fine meshes, which are provided by the 5th International Workshop on High-Order CFD Methods (HiOCFD5) website, are adopted for the $P = 3$ simulations. Two other similar meshes are generated for the simulations with $P = 2$. The mesh information is summarized in Tab. 6.3.

Fig. 6.6a shows the complex vortex structures in the fully-developed turbulent flow using the Q-criteria, which is a good indicator of vortices structures [70]. The mean streamwise velocity and mean turbulent fluctuation distributions are given in Figs. 6.6b, 6.6c and 6.6d. The simulation results on fine meshes are in good agreement



(a) Mean streamwise velocity in the centerline. (b) Mean streamwise velocity at three x slices.



(c) Mean crossflow velocity at three x slices

Figure 6.4: Mean streamwise and crossflow velocity distributions in the wake of the circular cylinder. Kravchenko-LES in [83], Ma-DNS in [94], and Parnaudeau-LES, Parnaudeau-Exp and Lourenco-Exp in [114].

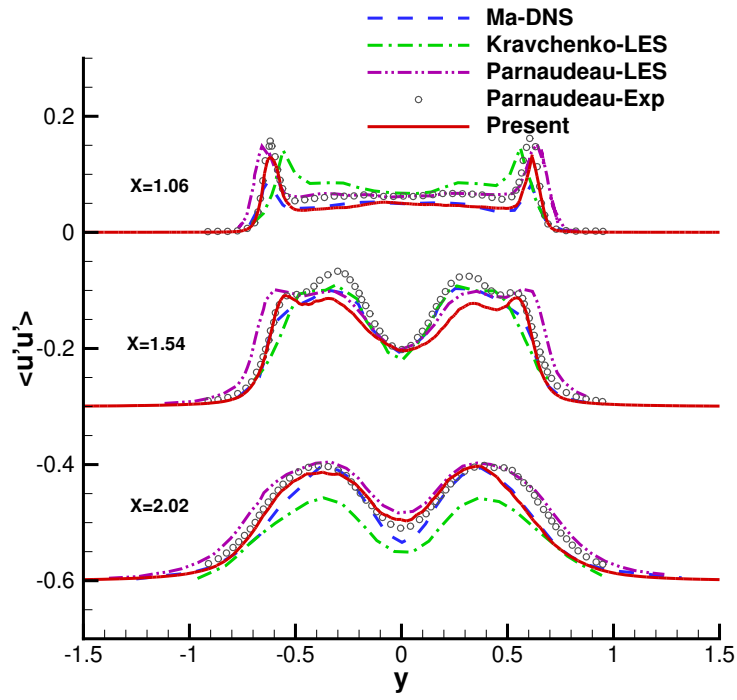
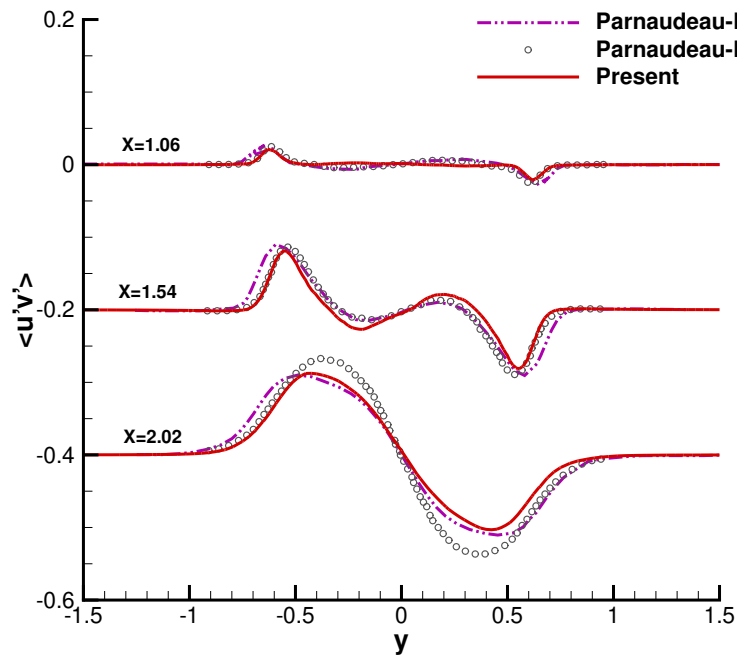
(a) Mean $u'u'$ (b) Mean $u'v'$

Figure 6.5: Mean velocity fluctuations in the wake of the circular cylinder.

Order	Mesh	Number of DoFs	Number of elements	$\Delta x^+ \times \Delta y^+ \Delta z^+$
$P = 3$	Fine	$256 \times 96 \times 128$	$64 \times 24 \times 32$	$14.5 \times \sim 0.7 \times 14.5$
	Baseline	$192 \times 96 \times 96$	$48 \times 24 \times 24$	$19.3 \times \sim 0.7 \times 19.3$
	Coarse	$128 \times 96 \times 64$	$32 \times 24 \times 16$	$29.0 \times \sim 0.7 \times 29.0$
$P = 2$	Baseline	$240 \times 120 \times 120$	$80 \times 40 \times 40$	$15.4 \times \sim 0.7 \times 15.4$
	Coarse	$156 \times 120 \times 78$	$52 \times 40 \times 26$	$23.7 \times \sim 0.7 \times 23.7$

Table 6.3: Meshes of turbulent channel flow simulations.

with the DNS results [2]. For the simulation with a coarse mesh, the discrepancy is larger and there are obvious differences in the mean turbulent fluctuations. The results with the baseline mesh are very close to the results of the fine mesh in the viscous sub-layer and the buffer layer, but are in between the results of fine and coarse meshes in the logarithmic area of the boundary layer. Fig. 6.7 compares the results of $P = 2$ and $P = 3$ simulations with different DoFs, which are presented in the legend. The result with $P = 2$ and $240 \times 120 \times 120$ DoFs is in a good agreement with the result with $P = 3$ and $128 \times 96 \times 64$ DoFs, which means about 4.39 times more DoFs are needed for the $P = 2$ scheme to give similar results to the $P = 3$ scheme. The computational time of the simulations is given in Tab. 6.4, which shows that the $P = 3$ method is much more efficient in computational time to give results of similar accuracy.

6.5 Summary

An implicit time integration method has been developed to accelerate the above simulations. The computational efficiency is summarized in Tab. 6.4. In the TGV test case, the implicit solver can only accelerate the simulations by fewer than 3 times since there is no grid stretching. However, a speed-up from 8.0 to 14.7 is obtained compared with the explicit solver in the unsteady simulations of turbulent channel

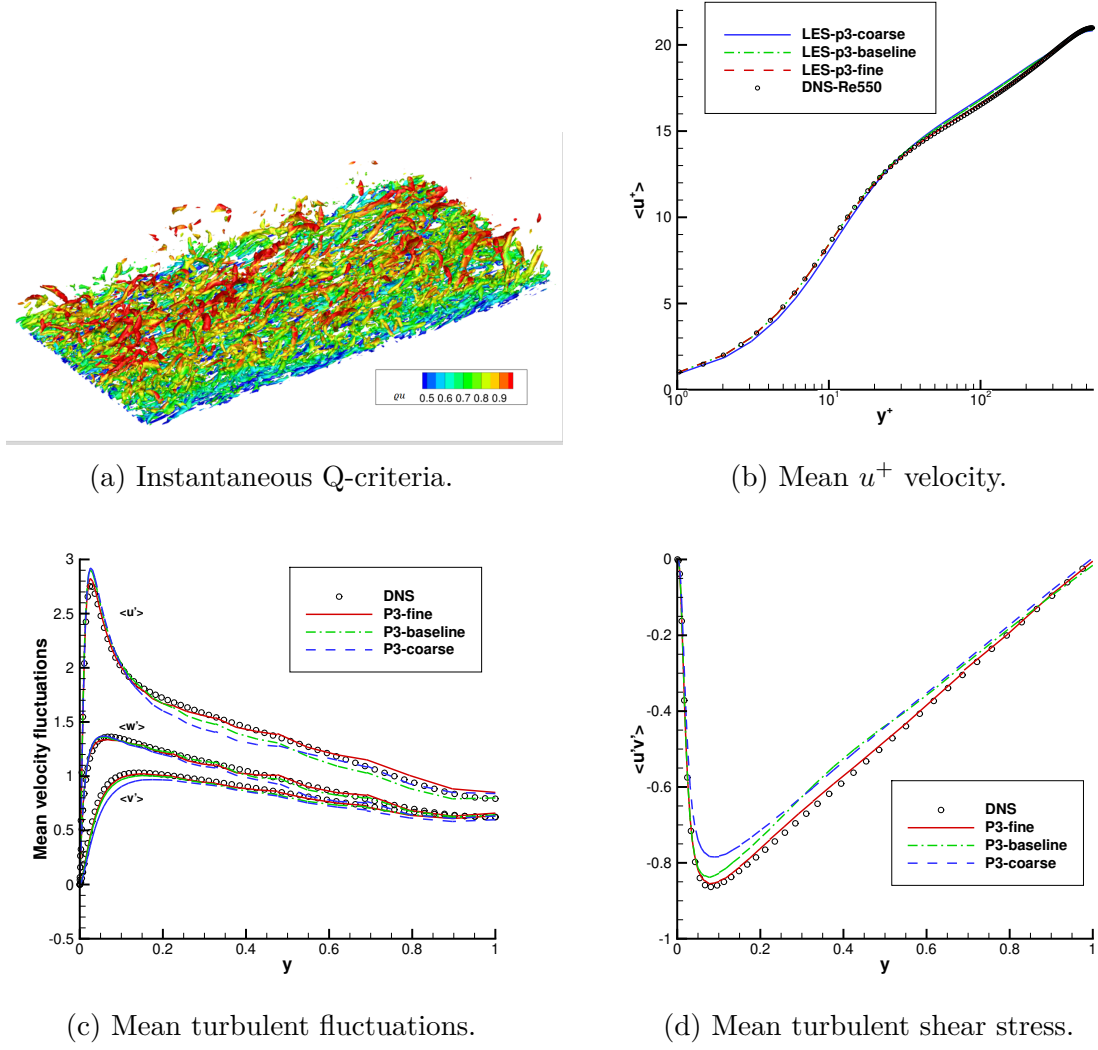


Figure 6.6: Flow field, mean velocity and turbulent fluctuation distributions of turbulent channel flow.

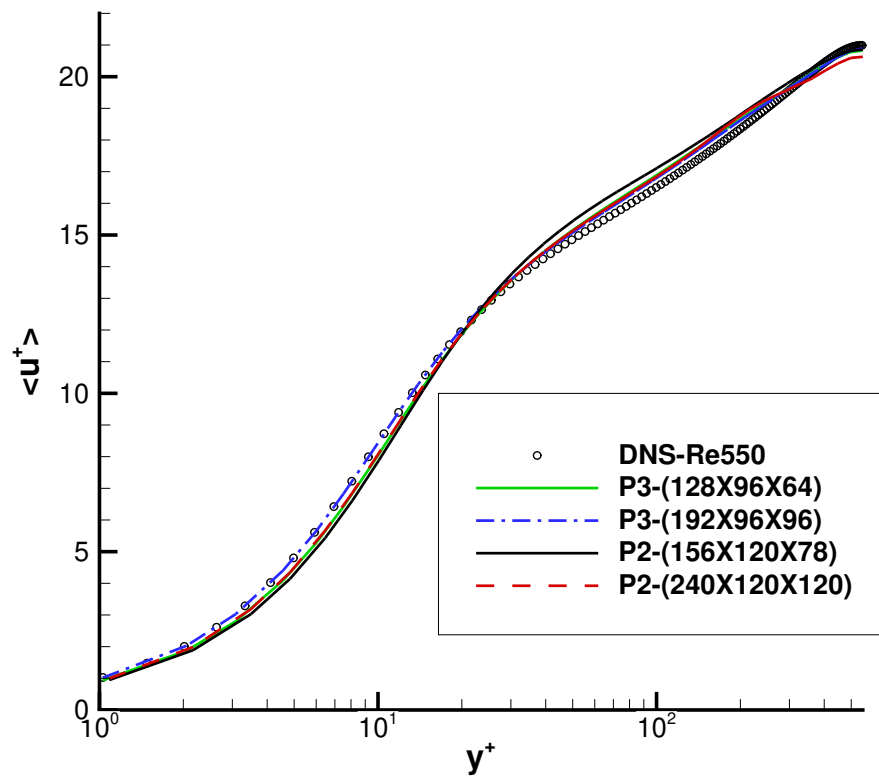


Figure 6.7: Comparison of turbulent channel flow simulation results at $Re_\tau = 550$ for different polynomial orders and mesh resolutions.

flow and flow over a circular cylinder. The simulations of TGV and turbulent channel flow with $P = 2$ and $P = 3$, which give similar results with different mesh resolutions, are compared in CPU time. The overall efficiency is higher for simulations with $P = 3$ than those with $P = 2$.

Cases	Order	Elements	Δt	CPU time (s)	Speed-up
Taylor-Green	$P = 2$	$64 \times 64 \times 64$	5.1×10^{-2}	163.2	3.0
	$P = 3$	$32 \times 32 \times 32$	4.6×10^{-2}	42.3	1.2
Channel	$P = 2$	$80 \times 40 \times 40$	2.0×10^{-2}	95.6	9.9
	$P = 3$	$32 \times 24 \times 16$	1.2×10^{-2}	7.2	8.0
Cylinder	$P = 2$	123360	1.0×10^{-2}	42.0	14.7

Table 6.4: Summary of computational efficiency. All tests are ran using 48 cores. The CPU time is the averaged time for one time step. The speed-up in the last column is relative to the explicit solver with second-order Runge-Kutta scheme.

Chapter 7

Conclusions and future work

Section 7.1 presents some conclusions and highlights the main contributions of this thesis. Section 7.2 proposes possible future work. Finally, a list of publications and conference presentations is provided in Section 7.3.

7.1 Concluding remarks

An implicit flow solver for the compressible Navier-Stokes equations has been developed in Nektar++ to improve simulation efficiency using singly diagonally implicit Runge-Kutta and Jacobian-free Newton Krylov methods. To improve the efficiency, a block relaxed Jacobi (BRJ) iterative preconditioner is proposed with hybrid analytical and numerical methods for calculating the flux Jacobian matrices and a partially matrix-free implementation. The implicit solver is designed and implemented in the open-source framework of Nektar++ with verification tests. Based on this implementation, the problem of parameter choice is studied to improve the robustness of the solver in various kinds of problems. At last, the developed solver is applied to

iLES of compressible turbulent flows.

The main contributions of the thesis can be summarized as follows:

- An efficient block relaxed Jacobi (BRJ) iterative preconditioner has been developed which speeds up the solver by about 3 times in stiff problems when compared with the block-Jacobi (or block diagonal) preconditioner. The BRJ preconditioner is constructed using a Jacobi iteration based on the diagonal and off-diagonal blocks of the Jacobian matrix. The influences of different parts of the Jacobian matrix on the preconditioning performance have been analyzed and this analysis has shown that the diffusion flux Jacobian play an important role in clustering the eigenspectra especially in stiff problems. Further this analysis has highlighted that some terms of the diffusion flux Jacobian can be neglected without obviously influencing the preconditioning effect. By neglecting these terms and implementing the off-diagonal terms in a matrix-free way, the memory consumption of the preconditioner is reduced by about one order of magnitude compared with matrix-based implementations. Finally, it has shown that the proposed BRJ preconditioner is low in computational cost at least for $P = 2$ and $P = 3$, which enables the solver to use more preconditioning iterations and achieve a larger speedup.
- An efficient implicit compressible flow solver has been implemented in the open-source framework Nektar++ based on an existing explicit solver with DG and LDG spatial schemes. The implicit solver not only extends the capabilities of the Nektar++ framework but also provides a good candidate for unsteady and steady compressible flow simulations to the open-source community. The software design and implementation details have been described with emphasis on providing a general pattern for the development of implicit solvers in

Nektar++, which will be used as a template for developing new solvers based on Nektar++.

- We have proposed an adaptive strategy for choosing time steps and Newton convergence tolerances based on the idea of balancing the discretization errors generated within one time step. This has been implemented in an implicit DG high-order spectral/*hp* element solver. It maintains temporal accuracy in the sense that the total error is dominated by the spatial error and further decreasing the temporal and iterative error will not obviously improve the results of the discrete PDE system, which is verified by numerical simulations of isentropic vortex, Taylor-Green vortex and turbulent flow over a circular cylinder. Moreover, this adaptive time step is a relatively efficient choice because it avoids excessively small time steps and too conservative Newton tolerances. Its efficiency is tested in a variety of steady-state and unsteady problems, including isentropic vortex, Taylor-Green vortex, flat plate boundary layer and turbulent flow over a circular cylinder. The adaptive methods in all these tests are close to the most efficient one possibly with only a small overhead partly due to extra costs in error estimating. This adaptive strategy performs well in a variety of problems without the need of tuning the 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 developed implicit solver is applied to high-fidelity implicit LES of turbulent flows. The results show that the implicit solver can improve the efficiency in these simulations by up to one order of magnitude. It is more efficient in these simulations to use higher-order ($P = 3$) methods. The results also illustrate

that accurate turbulent predictions can be obtained using the implicit LES strategy based on high-order spectral/*hp* element methods.

7.2 Future work

Further studies will focus on the following aspects:

- *Exploring complete matrix-free preconditioning techniques to further reduce memory consumptions.* The BRJ preconditioner is currently implemented partially matrix-free, which significantly reduces memory consumption and computational cost of the preconditioner. However, as shown in Section 4.2, the matrix-related costs grow very fast as the polynomial order increases. Therefore, exploring completely matrix-free versions of the BRJ implementation offers the promise to improve the performance of the implicit solver for higher-order simulations.
- *Developing more efficient preconditioners.* The current BRJ preconditioner is based on a simple Jacobi iteration, which is known to be effective for damping high-frequency error but not for the low-frequency ones. Thus, it cannot maintain a scale independent performance as the scale of the simulation increases. Multigrid methods is on the contrary proved to be effective in damping the low-frequency waves. Thus, incorporating the multigrid methods with the current methods is one way worth exploring especially for larger scale simulations.
- *Extending the solver to higher value of Ma .* Up to now, we have mainly focused on simulations without shocks. Studies on developing an implicit solver for

transonic and supersonic simulations are being undertaken considering the high nonlinearity and highly anisotropic properties of shocks.

- *Applying the developed solver in more complex flow simulations.* The solver has not only been verified in spatial and temporal accuracy but it has also proved to be efficient for high fidelity simulations. The solver will be adopted to study compressible flow problems, such as stability of transonic flows and shock/boundary-layer interactions.

7.3 List of publications and conference presentations

- [1] **Zhen-Guo Yan**, Yu Pan, Giacomo Castiglioni, Joaquim Peiró, and Spencer Sherwin. Development of efficient preconditioners for Newton-Krylov method in spectral/hp element, DG compressible flow simulation. In SIAM Conference on Computational Science and Engineering (CSE19), Spokane, Washington, USA, March 2019.
- [2] 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.
- [3] **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.

-
- [4] **Zhen-Guo Yan**, Yu Pan, Joaquim Peiró, and Spencer Sherwin. Development of implicit compressible flow solver in Nektar++. In Nektar++ Workshop 2019, Exeter, UK, June 2019.
- [5] **Zhen-Guo Yan**, Hui Xu, Yu Pan, Giacomo Castiglioni, Joaquim Peiró, and Spencer Sherwin. Development of an efficient DG solver for high-fidelity simulations of compressible flows. In International Symposium on High Fidelity Computational Methods and Applications (HiFiCOMA), Shanghai, China, December 2019.
- [6] **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.
- [7] 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.
- [8] **Zhen-Guo Yan**, Yu Pan, Joaquim Peiró, and Spencer Sherwin. Development of a balanced adaptive time-stepping strategy based on an implicit JFNK-DG compressible flow solver. In 3rd PRISM Workshop on Application of Time-Stepping Techniques, Online, 2021.
- [9] **Zhen-Guo Yan**, Yu Pan, Joaquim Peiró, and Spencer Sherwin. Design of an adaptive time-stepping by balancing discretization errors in an implicit compressible flow solver. In International Conference on Spectral and High

Order Methods 2020 (ICOSAHOM 2020), Vienna, Austria, 12th-16th July 2021.

- [10] Spencer Sherwin, **Zhen-Guo Yan**, Yu Pan, Ganlin Lyu, and Joaquim Peiró. Development and application of a spectral/hp element, implicit compressible solver using the JFNK approach. In 7th International Conference on Numerical Methods for Hyperbolic Problems (Dedicated to the 75th birthday of Prof. Eleuterio F. Toro OBE), Trento, Italy, 26-30 July 2021.

Appendix A

Butcher Tableau of ESDIRK schemes

The coefficients of ESDIRK schemes adopted in this paper can be expressed in the form of the general Butcher tableau shown in Tab. A.1 [84], 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 $a_{ii} = \alpha$; $i = 2, \dots, S + 1$. The coefficients of the matrix A for the ESDIRK2, ESDIRK3 and ESDIRK4 schemes are presented in Tab. A.2, Tab. A.3 and Tab. A.4, respectively.

$$\begin{array}{c|c} c & A \\ \hline & b^T \\ & \hat{b}^T \end{array} = \begin{array}{c|ccccccc} 0 & 0 & 0 & \dots & \dots & \dots & 0 \\ c_2 & a_{21} & \alpha & 0 & \dots & \dots & 0 \\ c_3 & a_{31} & a_{32} & \alpha & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & & \ddots & \ddots & \vdots \\ c_S & a_{S1} & a_{S2} & \dots & \dots & \alpha & 0 \\ c_R & a_{R1} & a_{R2} & \dots & \dots & \dots & \alpha \\ \hline & b_1 & \dots & \dots & \dots & b_S & 0 \\ & \hat{b}_1 & \dots & \dots & \dots & \dots & \hat{b}_R \end{array}$$

Table A.1: General form of the Butcher tableau of the embedded ESDIRK. Here R is shorthand for $R = S + 1$.

Table A.2: Butcher array of embedded ESDIRK2 with $S + 1 = 4$.

	0	0	0	0
A =	0.292893218800000	0.2928932188000000	0	0
	0.35353390567523	0.35353390632477	0	0
	0.215482203122508	0.68686723913539	0.292893218800000	0
		-0.195262145836047	0.292893218800000	0

Table A.3: Butcher array of embedded ESDIRK3 with $S + 1 = 5$.

	0	0	0	0	0
A =	0.43586652150846	0.43586652150846	0	0	0
	0.14073777472471	-0.10836555138132	0.43586652150846	0	0
	0.10239940061991	-0.37687845225556	0.83861253012719	0	0
	0.15702489786032	0.11733044137044	0.61667803039212	0.43586652150846	0
			-0.32689989113134	0.43586652150846	0

Table A.4: Butcher array of embedded ESDIRK4 with $S + 1 = 7$.

	0	0	0	0	0	0	0
A =	0.2700000000000000	0.2700000000000000	0	0	0	0	0
	0.1350000000000000	0.87265371804359686	0	0	0	0	0
	0.24814211234447322	0.13282088522859322	0.2700000000000000	0	0	0	0
	0.25494479822150471	0.13106196422347200	-0.0388686658917771	0.2700000000000000	0	0	0
	0.17549975523182941	0	-0.04522093930235708	0.03389121682051642	0.2700000000000000	0	0
	0.15847612643670410	0	-0.01641725931492383	3.59357175290010625	-3.02265424881701182	0.2700000000000000	0
			-0.07384703732094983	5.26056776397634893	-4.83946947758407500	0.22427262449197180	0.270

Appendix B

Forcing terms of manufactured compressible Poiseuille flow

The analytical solution of the manufactured compressible Poiseuille flow has already been given in Eq. (4.5). The corresponding forcing terms are given by

$$f = \begin{pmatrix} 0 \\ \frac{dp}{dx} \theta (L^2 - 6yL + 6y^2) \\ 0 \\ f_4 \end{pmatrix}, \quad (\text{B.1})$$

where

$$\begin{aligned} f_4 = & \frac{1}{2\mu} \frac{dp^2}{dx} (y(L-y) (1 - \theta(L^2 - 6yL + 6y^2) - \gamma/(\gamma-1))) \\ & - \frac{1}{4\mu} \frac{dp^2}{dx} (L-2y)^2 (2\theta yL - 2\theta y^2 + 1)^2. \end{aligned} \quad (\text{B.2})$$

Bibliography

- [1] Abe, Y., Morinaka, I., Haga, T., Nonomura, T., Shibata, H., Miyaji, K.: Stable, non-dissipative, and conservative flux-reconstruction schemes in split forms. *Journal of Computational Physics* **353**, 193–227 (2018)
- [2] del Álamo, J.C., Jiménez, J.: Spectra of the very large anisotropic scales in turbulent channels. *Physics of Fluids* **15**(6), L41–L44 (2003)
- [3] Arévalo, C., Söderlind, G., Hadjimichael, Y., Fekete, I.: Local error estimation and step size control in adaptive linear multistep methods. *Numerical Algorithms* **86**, 537–563 (2020)
- [4] Argyropoulos, C.D., Markatos, N.C.: Recent advances on the numerical modelling of turbulent flows. *Applied Mathematical Modelling* **39**(2), 693–732
- [5] Arnold, D., Brezzi, F., Cockburn, B., Marini, L.: Unified analysis of discontinuous Galerkin methods for elliptic problems. *SIAM Journal on Numerical Analysis* **39**(5), 1749–1779 (2002)
- [6] Asada, H., Kawai, S.: High-order implicit discontinuous Galerkin scheme for unsteady turbulent flows. In: 55th AIAA Aerospace Sciences Meeting. AIAA 2017-0739, Grapevine, Texas (2017)

- [7] Asthana, K., López-Morales, M.R., Jameson, A.: Non-linear stabilization of high-order flux reconstruction schemes via Fourier-spectral filtering. *Journal of Computational Physics* **303**, 269–294 (2015)
- [8] Atkins, H.L., Shu, C.W.: Quadrature-free implementation of discontinuous Galerkin method for hyperbolic equations. *AIAA Journal* **36**(5), 775–782 (1998)
- [9] Barter, G.E., Darmofal, D.L.: Shock capturing with PDE-based artificial viscosity for DGFEM: Part I. Formulation. *Journal of Computational Physics* **229**(5), 1810–1827 (2010)
- [10] Bassi, F., Botti, L., Colombo, A., Crivellini, A., Ghidoni, A., Massa, F.: 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 (2016)
- [11] Bassi, F., Crivellini, A., Rebay, S., Savini, M.: Discontinuous Galerkin solution of the Reynolds-averaged Navier-Stokes and $k-\omega$ turbulence model equations. *Computers & Fluids* **34**(4), 507–540 (2005)
- [12] Bassi, F., Rebay, S.: 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 (1997)
- [13] Bassi, F., Rebay, S.: A high order discontinuous Galerkin method for compressible turbulent flows. *Discontinuous Galerkin Methods* **11**, 77–88 470 (2000)

- [14] Bastian, P., Müller, E.H., Müthing, S., Piatkowski, M.: Matrix-free multigrid block-preconditioners for higher order discontinuous Galerkin discretisations. *Journal of Computational Physics* **394**, 417–439 (2019)
- [15] Bijl, H., Carpenter, M.H., Vatsa, V.N., Kennedy, C.A.: Implicit time integration schemes for the unsteady compressible Navier–Stokes equations: Laminar flow. *Journal of Computational Physics* **179**(1), 313–329 (2002)
- [16] Birken, P., Gassner, G., Haas, M., Munz, C.D.: Preconditioning for modal discontinuous Galerkin methods for unsteady 3D Navier-Stokes equations. *Journal of Computational Physics* **240**, 20–35 (2013)
- [17] Birken, P., Gassner, G.J., Versbach, L.M.: Subcell finite volume multigrid preconditioning for high-order discontinuous Galerkin methods. *International Journal of Computational Fluid Dynamics* **33**(9), 353–361 (2019). Publisher: Taylor & Francis
- [18] Blom, D.S., Birken, P., Bijl, H., Kessels, F., Meister, A., van Zuijlen, A.H.: A comparison of Rosenbrock and ESDIRK methods combined with iterative solvers for unsteady compressible flows. *Advances in Computational Mathematics* **42**(6), 1401–1426 (2016)
- [19] Boin, J.P., Robinet, J.C., Corre, C., Deniau, H.: 3D steady and unsteady bifurcations in a shock-wave/laminar boundary layer interaction: A numerical study. *Theoretical and Computational Fluid Dynamics* **20**(3), 163–180 (2006)
- [20] Boom, P.D., Zingg, D.W.: Optimization of high-order diagonally-implicit Runge-Kutta methods. *Journal of Computational Physics* **371**, 168–191 (2018)
- [21] Brune, P.R., Knepley, M.G., Smith, B.F., Tu, X.: Composing scalable nonlinear algebraic solvers. *SIAM Review* **57**(4), 535–565 (2015)

- [22] Brunet, V., Croner, E., Minot, A., Laborderie, J.d., Lippinois, E., Richard, S., Boussuge, J.F., Dombard, J., Duchaine, F., Gicquel, L., Poinot, T., Puigt, G., Staffelbach, G., Segui, L., Vermorel, O., Villedieu, N., Cagnone, J.S., Hillewaert, K., Rasquin, M., Lartigue, G., Moureau, V., Couaillier, V., Martin, E., Plata, M.d.l.L., Gouez, J.M.L., Renac, F.: Comparison of various CFD codes for LES simulations of turbomachinery: From inviscid vortex convection to multi-stage compressor. pp. V02CT42A013–V02CT42A013. American Society of Mechanical Engineers (2018)
- [23] Bücker, H.M., Pollul, B., Rasch, A.: On CFL evolution strategies for implicit upwind methods in linearized Euler equations. *International Journal for Numerical Methods in Fluids* **59**(1), 1–18 (2009)
- [24] Cantwell, C.D., 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.M., Sherwin, S.J.: Nektar++: An open-source spectral/hp element framework. *Computer Physics Communications* **192**, 205–219 (2015)
- [25] Cantwell, C.D., Sherwin, S.J., Kirby, R.M., Kelly, P.H.J.: From h to p efficiently: Strategy selection for operator evaluation on hexahedral and tetrahedral elements. *Computers & Fluids* **43**(1), 23–28 (2011)
- [26] Cao, W., Zhang, Z.: Some recent developments in superconvergence of discontinuous Galerkin methods for time-dependent partial differential equations. *Journal of Scientific Computing* pp. 1–22 (2018)
- [27] Cassinelli, A., Montomoli, F., Adami, P., Sherwin, S.J.: High fidelity spectral/hp element methods for turbomachinery. In: *Proceedings of ASME Turbo Expo 2018*, p. V02CT42A020. ASME, Oslo, Norway (2018)

- [28] Chapman, A., Saad, Y., Wigton, L.: High-order ILU preconditioners for CFD problems. *International Journal for Numerical Methods in Fluids* **33**(6), 767–788 (2000)
- [29] Chaudhuri, A., Jacobs, G.B., Don, W.S., Abbassi, H., Mashayek, F.: Explicit discontinuous spectral element method with entropy generation based artificial viscosity for shocked viscous flows. *Journal of Computational Physics* **332**, 99–117 (2017)
- [30] Cheng, J., Yang, X., Liu, X., Liu, T., Luo, H.: A direct discontinuous Galerkin method for the compressible Navier-Stokes equations on arbitrary grids. *Journal of Computational Physics* **327**, 484–502 (2016)
- [31] Chudanov, V., Aksenova, A., Goreinov, S., Makarevich, A., Pervichko, V.: Validation of a new method for solving of CFD problems in nuclear engineering using petascale HPC. pp. V004T10A009–V004T10A009. American Society of Mechanical Engineers (2014)
- [32] Cockburn, B., Dong, B., Guzmán, J.: A superconvergent LDG-hybridizable Galerkin method for second-order elliptic problems. *Mathematics of Computation* **77**(264), 1887–1916 (2008)
- [33] Cockburn, B., Shu, C.: The local discontinuous Galerkin method for time-dependent convection-diffusion systems. *SIAM Journal on Numerical Analysis* **35**(6), 2440–2463 (1998)
- [34] Cockburn, B., Shu, C.W.: TVB Runge-Kutta local projection discontinuous Galerkin finite element method for conservation laws. II. General framework. *Mathematics of Computation* **52**(186), 411–435 (1989)

- [35] Cockburn, B., Shu, C.W.: The Runge-Kutta discontinuous Galerkin method for conservation laws V: Multidimensional systems. *Journal of Computational Physics* **141**(2), 199–224 (1998)
- [36] Cong, N.h.: A-stable diagonally implicit Runge-Kutta-Nyström methods for parallel computers. *Numerical Algorithms* **4**(2), 263–281 (1993)
- [37] Crivellini, A., Bassi, F.: An implicit matrix-free discontinuous Galerkin solver for viscous and turbulent aerodynamic simulations. *Computers & Fluids* **50**(1), 81–93 (2011)
- [38] De Wiart, C.C., Hillewaert, K., Bricteux, L., Winckelmans, G.: Implicit LES of free and wall-bounded turbulent flows based on the discontinuous Galerkin/symmetric interior penalty method. *International Journal for Numerical Methods in Fluids* **78**(6), 335–354 (2015)
- [39] De Wiart, C.C., Hillewaert, K., Duponcheel, M., Winckelmans, G.: 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 (2014)
- [40] De Wiart, C.C., Hillewaert, K., Geuzaine, P.: DNS of a low pressure turbine blade computed with the discontinuous Galerkin method. pp. 2101–2111. American Society of Mechanical Engineers (2012)
- [41] De Wiart, C.C., Hillewaert, K., Lorriaux, E., Verheylewegan, G.: Development of a discontinuous Galerkin solver for high quality wall-resolved/modelled DNS and LES of practical turbomachinery flows on fully unstructured meshes. pp. V02BT39A035–V02BT39A035. American Society of Mechanical Engineers (2015)

- [42] Diosady, L.T., Darmofal, D.L.: Preconditioning methods for discontinuous Galerkin solutions of the Navier-Stokes equations. *Journal of Computational Physics* **228**(11), 3917–3935 (2009)
- [43] Diosady, L.T., Murman, S.M.: Tensor-product preconditioners for higher-order space-time discontinuous Galerkin methods. *Journal of Computational Physics* **330**, 296–318 (2017)
- [44] Diosady, L.T., Murman, S.M.: Scalable tensor-product preconditioners for high-order finite-element methods: Scalar equations. *Journal of Computational Physics* **394**, 759–776 (2019)
- [45] Dolejší, V., Feistauer, M.: Discontinuous Galerkin Method: Analysis and Applications to Compressible Flow, *Springer Series in Computational Mathematics*, vol. 48. Springer (2015)
- [46] Douglas, J., Dupont, T.: Interior penalty procedures for elliptic and parabolic Galerkin methods. In: R. Glowinski, J.L. Lions (eds.) *Computing Methods in Applied Sciences, Lecture Notes in Physics*, pp. 207–216. Springer Berlin Heidelberg (1976)
- [47] Ducros, F., Ferrand, V., Nicoud, F., Weber, C., Darracq, D., Gacherieu, C., Poinso, T.: Large-eddy simulation of the shock/turbulence interaction. *Journal of Computational Physics* **152**(2), 517–549 (1999)
- [48] Eckert, E.: Engineering relations for friction and heat transfer to surfaces in high velocity flow. *Journal of the Aeronautical Sciences* **22**(8), 585–587 (1955)
- [49] Ezertas, A., Eyi, S.: Performances of numerical and analytical jacobians in flow and sensitivity analysis. In: *19th AIAA Computational Fluid Dynamics*. American Institute of Aeronautics and Astronautics, San Antonio, Texas (2009)

- [50] Fehn, N., Wall, W.A., Kronbichler, M.: A matrix-free high-order discontinuous Galerkin compressible Navier-Stokes solver: A performance comparison of compressible and incompressible formulations for turbulent incompressible flows. *International Journal for Numerical Methods in Fluids* **89**(3), 71–102 (2019)
- [51] Fernandez, P., Moura, R.C., Mengaldo, G., Peraire, J.: Non-modal analysis of spectral element methods: Towards accurate and robust large-eddy simulations. *Computer Methods in Applied Mechanics and Engineering* **346**, 43–62 (2019)
- [52] Fernandez, P., Nguyen, N.C., Peraire, J.: Subgrid-scale modeling and implicit numerical dissipation in DG-based large-eddy simulation. In: 23rd AIAA Computational Fluid Dynamics Conference. American Institute of Aeronautics and Astronautics, Denver, Colorado (2017)
- [53] Filho, E.J.d.A.J., Wang, Z.J.: A matrix-free GMRES algorithm on GPU clusters for implicit large eddy simulation. In: AIAA Scitech 2021 Forum. American Institute of Aeronautics and Astronautics (2021)
- [54] Flad, D., Gassner, G.: On the use of kinetic energy preserving DG-schemes for large eddy simulation. *Journal of Computational Physics* **350**, 782–795 (2017)
- [55] Franciolini, M., Crivellini, A., Nigro, A.: On the efficiency of a matrix-free linearly implicit time integration strategy for high-order discontinuous Galerkin solutions of incompressible turbulent flows. *Computers and Fluids* **159**, 276–294 (2017)
- [56] Frère, A., Sørensen, N.N., Hillewaert, K., Winckelmans, G.: Discontinuous Galerkin methodology for large-eddy simulations of wind turbine airfoils. *Journal of Physics: Conference Series* **753**(2), 022037 (2016)

- [57] Gassner, G.: A skew-symmetric discontinuous Galerkin spectral element discretization and its relation to SBP-SAT finite difference methods. *SIAM Journal on Scientific Computing* **35**(3), A1233–A1253 (2013)
- [58] Gassner, G.J., Winters, A.R., Kopriva, D.A.: Split form nodal discontinuous Galerkin schemes with summation-by-parts property for the compressible Euler equations. *Journal of Computational Physics* **327**, 39–66 (2016)
- [59] Gatsis, J.: Preconditioning techniques for a Newton-Krylov algorithm for the compressible Navier-Stokes equations. Ph.D. thesis, University of Toronto (2013)
- [60] Ghidoni, A., Colombo, A., Rebay, S., Bassi, F.: Simulation of the transitional flow in a low pressure gas turbine cascade with a high-order discontinuous Galerkin method. *Journal of Fluids Engineering* **135**(7), 071101 (2013)
- [61] Grinstein, F.F., Margolin, L.G., Rider, W.J.: *Implicit Large Eddy Simulation: Computing Turbulent Fluid Dynamics*. Cambridge University Press (2007)
- [62] Gropp, W.D., Kaushik, D.K., Keyes, D.E., Smith, B.F.: High-performance parallel implicit CFD. *Parallel Computing* **27**(4), 337–362 (2001)
- [63] Hairer, E., Wanner, G.: *Solving Ordinary Differential Equations II. Stiff and Differential-Algebraic Problems*, *Springer Series in Computational Mathematics*, vol. 14. Springer-Verlag, Berlin (1996)
- [64] Hartmann, R., Houston, P.: Symmetric interior penalty DG methods for the compressible Navier-Stokes equations I: Method formulation. *International Journal of Numerical Analysis and Modeling* **3**(1), 1–20 (2005)

- [65] Hartmann, R., Houston, P.: An optimal order interior penalty discontinuous Galerkin discretization of the compressible Navier–Stokes equations. *Journal of Computational Physics* **227**(22), 9670–9685 (2008)
- [66] Hesthaven, J.S., Warburton, T.: *Nodal Discontinuous Galerkin Methods: Algorithms, Analysis, and Applications*. Springer Science & Business Media (2007)
- [67] Hillewaert, K.: Development of the discontinuous Galerkin method for high-resolution, large scale CFD and acoustics in industrial geometries. Ph.D. thesis, Université Catholique de Louvain (2013). Google-Books-ID: IBJij3jtUCQC
- [68] Hillewaert, K., de Wiart, C.C., Verheylewegen, G., Arts, T.: Assessment of a high-order discontinuous Galerkin method for the direct numerical simulation of transition at low-Reynolds number in the T106c high-lift low pressure turbine cascade. In: *Proceedings of ASME Turbo Expo 2014*, p. V02BT39A034. American Society of Mechanical Engineers, Düsseldorf, Germany (2014)
- [69] Holst, K.R., Glasby, R.S., Bond, R.B.: On the effect of temporal error in high-order simulations of unsteady flows. *Journal of Computational Physics* **402**, 108989 (2020)
- [70] Hunt, J.C.R., Wray, A.A., Moin, P.: Eddies, streams, and convergence zones in turbulent flows. NTRS Author Affiliations: Cambridge Univ. (England)., NASA Ames Research Center, Stanford Univ. NTRS Document ID: 19890015184 NTRS Research Center: Legacy CDMS (CDMS)
- [71] James, S., Zhu, J., Anand, M.S.: Large-eddy simulations as a design tool for gas turbine combustion systems **44**(4), 674–686

- [72] Jameson, A., Vincent, P.E., Castonguay, P.: On the non-linear stability of flux reconstruction schemes. *Journal of Scientific Computing* **50**(2), 434–445 (2012)
- [73] Jia, F., Ims, J., Wang, Z.J., Kopriva, J., Laskowski, G.M.: An evaluation of a commercial and a high order FR/CPR flow solvers for industrial large eddy simulation. In: 2018 AIAA Aerospace Sciences Meeting. AIAA 2018-0827, Kissimmee, Florida (2018)
- [74] Jia, F., Wang, Z.J., Bhaskaran, R., Paliath, U., Laskowski, G.M.: Scalability and efficiency of explicit and implicit FR/CPR schemes in large eddy simulations. In: 23rd AIAA Computational Fluid Dynamics Conference. AIAA 2017-3096, Denver, Colorado (2017)
- [75] Kalkote, N., Assam, A., Eswaran, V.: Acceleration of later convergence in a density-based solver using adaptive time stepping. *AIAA Journal* **57**(1), 352–364 (2019)
- [76] Karamanos, G., Karniadakis, G.: A spectral vanishing viscosity method for large-eddy simulations. *Journal of Computational Physics* **163**(1), 22–50 (2000)
- [77] Karniadakis, G., Sherwin, S.: *Spectral/hp Element Methods for Computational Fluid Dynamics*, second edn. Oxford Science Publications (2013)
- [78] Kennedy, C.A., Carpenter, M.H.: Diagonally implicit Runge-Kutta methods for ordinary differential equations. A review. NASA report TM-2016-219173 (2016)
- [79] Kennedy, C.A., Carpenter, M.H.: Diagonally implicit Runge-Kutta methods for stiff ODEs. *Applied Numerical Mathematics* **146**, 221–244 (2019)
- [80] Kirby, R.M., Sherwin, S.J.: Stabilisation of spectral/*hp* element methods through spectral vanishing viscosity: Application to fluid mechanics modelling.

- Computer Methods in Applied Mechanics and Engineering **195**(23), 3128–3144 (2006)
- [81] Kirby, R.M., Sherwin, S.J.: Stabilisation of spectral/hp element methods through spectral vanishing viscosity: Application to fluid mechanics modelling. Computer Methods in Applied Mechanics and Engineering **195**(23), 3128–3144 (2006)
- [82] Knoll, D.A., Keyes, D.E.: Jacobian-free Newton-Krylov methods: A survey of approaches and applications. Journal of Computational Physics **193**(2), 357–397 (2004)
- [83] Kravchenko, A.G., Moin, P.: Numerical studies of flow over a circular cylinder at $Re_D=3900$. Physics of Fluids **12**(2), 403–417 (2000)
- [84] Kværnø, A.: Singly diagonally implicit Runge-Kutta methods with an explicit first stage. BIT Numerical Mathematics **44**(3), 489–502 (2004)
- [85] Lian, C., Xia, G., Merkle, C.L.: Solution-limited time stepping to enhance reliability in CFD applications. Journal of Computational Physics **228**, 4836–4857 (2009)
- [86] Liang, C., Chan, A.S., Jameson, A.: A p-multigrid spectral difference method for two-dimensional unsteady incompressible Navier-Stokes equations. Computers & Fluids **51**(1), 127–135 (2011)
- [87] Liu, H., Yan, J.: The direct discontinuous Galerkin (DDG) methods for diffusion problems. SIAM Journal on Numerical Analysis **47**(1), 675–698 (2009)
- [88] Loppi, N.A., Witherden, F.D., Jameson, A., Vincent, P.E.: Locally adaptive pseudo-time stepping for high-order flux reconstruction. Journal of Computational Physics **399**, 108913 (2019)

- [89] Lu, Y., Dawes, W.N.: High order large eddy simulations for a transonic turbine blade using hybrid unstructured meshes. In: Proceedings of ASME Turbo Expo 2015, p. V02CT44A005. ASME, Montréal, Canada (2015)
- [90] Lu, Y., Liu, K., Dawes, W.N.: Large eddy simulations for 3D turbine blades using a high order flux reconstruction method. In: Proceedings of ASME Turbo Expo 2013, p. V06BT37A014. ASME, San Antonio, Texas, USA (2013)
- [91] Lu, Y., Liu, K., Dawes, W.N.: Fast high order large eddy simulations on many core computing systems for turbomachinery. In: Proceedings of ASME Turbo Expo 2016, p. V02DT44A029. Seoul, South Korea (2016)
- [92] Luo, H., Baum, J.D., Löhner, R.: A p-multigrid discontinuous Galerkin method for the Euler equations on unstructured grids. *Journal of Computational Physics* **211**(2), 767–783 (2006)
- [93] Luo, H., Sharov, D., Baum, J.D., Lohner, R.: Parallel unstructured grid GMRES+LU-SGS method for turbulent flows. *AIAA* 2003-273 (2003)
- [94] Ma, X., Karamanos, G.S., Karniadakis, G.E.: Dynamics and low-dimensionality of a turbulent near wake. *Journal of Fluid Mechanics* **410**, 29–65 (2000)
- [95] Martins, J., Sturdza, P., Alonso, J.: The complex-step derivative approximation. *ACM Transactions on Mathematical Software* **29**, 245–262 (2003)
- [96] Meisrimel, P., Birken, P.: Goal oriented time adaptivity using local error estimates. *Algorithms* **13**(5), 113 (2020). Number: 5 Publisher: Multidisciplinary Digital Publishing Institute
- [97] Mengaldo, G.: Discontinuous spectral/*hp* element methods: development, analysis and applications to compressible flows. Ph.D. thesis, Imperial College London (2015)

- [98] Mengaldo, G., De Grazia, D., Moura, R.C., Sherwin, S.J.: Spatial eigensolution analysis of energy-stable flux reconstruction schemes and influence of the numerical flux on accuracy and robustness. *Journal of Computational Physics* **358**, 1–20 (2018)
- [99] Mengaldo, G., De Grazia, D., Moxey, D., Vincent, P., Sherwin, S.: Dealiasing techniques for high-order spectral element methods on regular and irregular grids. *Journal of Computational Physics* **299**, 56–81 (2015)
- [100] Mengaldo, G., De Grazia, D., Vincent, P.E., Sherwin, S.J.: On the connections between discontinuous Galerkin and flux reconstruction schemes: Extension to curvilinear meshes. *Journal of Scientific Computing* **67**(3), 1272–1292 (2016)
- [101] Mengaldo, G., Moura, R.C., Giralda, B., Peiró, J., Sherwin, S.J.: Spatial eigensolution analysis of discontinuous Galerkin schemes with practical insights for under-resolved computations and implicit LES. *Computers & Fluids* **169**, 349–364 (2018)
- [102] Morinishi, Y., Tamano, S., Nakabayashi, K.: Direct numerical simulation of compressible turbulent channel flow between adiabatic and isothermal walls. *Journal of Fluid Mechanics* **502**, 273–308 (2004)
- [103] Moura, R.C., Aman, M., Peiró, J., Sherwin, S.J.: Spatial eigenanalysis of spectral/hp continuous Galerkin schemes and their stabilisation via DG-mimicking spectral vanishing viscosity for high Reynolds number flows. *Journal of Computational Physics* **406**, 109112 (2020)
- [104] Moura, R.C., Mengaldo, G., Peiró, J., Sherwin, S.J.: An LES setting for DG-based implicit LES with insights on dissipation and robustness. In: M.L. Bittencourt, N.A. Dumont, J.S. Hesthaven (eds.) *Spectral and High Order*

- Methods for Partial Differential Equations ICOSAHOM 2016, Lecture Notes in Computational Science and Engineering, pp. 161–173. Springer International Publishing (2017)
- [105] Moura, R.C., Mengaldo, G., Peiró, J., Sherwin, S.J.: On the eddy-resolving capability of high-order discontinuous Galerkin approaches to implicit LES / under-resolved DNS of Euler turbulence. *Journal of Computational Physics* **330**, 615–623 (2017)
- [106] Moura, R.C., Sherwin, S.J., Peiró, J.: Linear dispersion–diffusion analysis and its application to under-resolved turbulence simulations using discontinuous Galerkin spectral/*hp* methods. *Journal of Computational Physics* **298**(695–710) (2015)
- [107] Moxey, D., Cantwell, C.D., Bao, Y., Cassinelli, A., Castiglioni, G., Chun, S., Juda, E., Kazemi, E., Lackhove, K., Marcon, J., Mengaldo, G., Serson, D., Turner, M., Xu, H., Peiró, J., Kirby, R.M., Sherwin, S.J.: Nektar++: Enhancing the capability and application of high-fidelity spectral/*hp* element methods. *Computer Physics Communications* **249**, 107110 (2020)
- [108] Nazari, F., Mohammadian, A., Charron, M., Zadra, A.: Optimal high-order diagonally-implicit Runge-Kutta schemes for nonlinear diffusive systems on atmospheric boundary layer. *Journal of Computational Physics* **271**, 118–130 (2014)
- [109] Nigro, A., Ghidoni, A., Rebay, S., Bassi, F.: Modified extended BDF scheme for the discontinuous Galerkin solution of unsteady compressible flows: DG-MEBDF unsteady solution of compressible flows. *International Journal for Numerical Methods in Fluids* **76**(9), 549–574 (2014)

- [110] Noventa, G., Massa, F., Bassi, F., Colombo, A., Franchina, N., Ghidoni, A.: A high-order discontinuous Galerkin solver for unsteady incompressible turbulent flows. *Computers & Fluids* **139**, 248–260 (2016)
- [111] Noventa, G., Massa, F., Rebay, S., Bassi, F., Ghidoni, A.: Robustness and efficiency of an implicit time-adaptive discontinuous Galerkin solver for unsteady flows. *Computers & Fluids* **204**, 104529 (2020)
- [112] Oliver, T.A.: Multigrid solution for high-order discontinuous Galerkin discretizations of the compressible Navier-Stokes equations. Thesis, Massachusetts Institute of Technology (2004)
- [113] Orszag, S.A.: Spectral methods for problems in complex geometries. *Journal of Computational Physics* **37**(1), 70–92 (1980)
- [114] Parnaudeau, P., Carlier, J., Heitz, D., Lamballais, E.: Experimental and numerical studies of the flow over a circular cylinder at Reynolds number 3900. *Physics of Fluids* **20**(8), 085101 (2008)
- [115] Pazner, W.: Efficient low-order refined preconditioners for high-order matrix-free continuous and discontinuous Galerkin methods. *SIAM Journal on Scientific Computing* **42**(5), A3055–A3083 (2020). Publisher: Society for Industrial and Applied Mathematics
- [116] Pazner, W., Franco, M., Persson, P.O.: High-order wall-resolved large eddy simulation of transonic buffet on the OAT15A airfoil. In: *AIAA Scitech 2019 Forum*. American Institute of Aeronautics and Astronautics, San Diego, California (2019)

- [117] Pazner, W., Persson, P.O.: Stage-parallel fully implicit Runge-Kutta solvers for discontinuous Galerkin fluid simulations. *Journal of Computational Physics* **335**, 700–717 (2017)
- [118] Pazner, W., Persson, P.O.: Approximate tensor-product preconditioners for very high order discontinuous Galerkin methods. *Journal of Computational Physics* **354**, 344–369 (2018)
- [119] Pazner, W., Persson, P.O.: Analysis and entropy stability of the line-based discontinuous Galerkin method. *Journal of Scientific Computing* **80**(1), 376–402 (2019)
- [120] Peraire, J., Persson, P.: The compact discontinuous Galerkin (CDG) method for elliptic problems. *SIAM Journal on Scientific Computing* **30**(4), 1806–1824 (2008)
- [121] Persson, P., Peraire, J.: Newton-GMRES preconditioning for discontinuous Galerkin discretizations of the Navier-Stokes equations. *Siam Journal on Scientific Computing* **30**(6), 2709–2733 (2008)
- [122] Persson, P.O.: Scalable parallel Newton-Krylov solvers for discontinuous Galerkin discretizations. In: 47th AIAA Aerospace Sciences Meeting, January, pp. 1–12. AIAA 2009-606, Denver, Colorado (2009)
- [123] Persson, P.O., Peraire, J.: An efficient low memory implicit DG algorithm for time dependent problems. In: 44th AIAA Aerospace Sciences Meeting and Exhibit. American Institute of Aeronautics and Astronautics, Reno, Nevada (2006)

- [124] Persson, P.O., Peraire, J.: Sub-cell shock capturing for discontinuous Galerkin methods. In: 44th AIAA Aerospace Sciences Meeting and Exhibit. AIAA 2006-112, Reno, Nevada (2006)
- [125] Peterson, J.W., Lindsay, A.D., Kong, F.: Overview of the incompressible Navier–Stokes simulation capabilities in the MOOSE framework. *Advances in Engineering Software* **119**, 68–92 (2018)
- [126] Pirozzoli, S.: Numerical methods for high-speed flows. *Annual Review of Fluid Mechanics* **43**, 163–194 (2011)
- [127] Pope, S.B.: Ten questions concerning the large-eddy simulation of turbulent flows. *New journal of Physics* **6**(1), 35 (2004)
- [128] Ranocha, H., Glaubitz, J., Offner, P., Sonar, T.: Stability of artificial dissipation and modal filtering for flux reconstruction schemes using summation-by-parts operators. *Applied Numerical Mathematics* **128**, 1–23 (2018)
- [129] Rasetarinera, P., Hussaini, M.Y.: An efficient implicit discontinuous spectral Galerkin method. *Journal of Computational Physics* **172**(2), 718–738 (2001)
- [130] Reed, W.H., Hill, T.R.: Triangular mesh methods for the neutron transport equation. Tech. Rep. LA-UR-73-479; CONF-730414-2, Los Alamos Scientific Lab., N.Mex. (USA) (1973)
- [131] Roy, C.J.: Review of code and solution verification procedures for computational simulation. *Journal of Computational Physics* **205**(1), 131–156 (2005)
- [132] Saad, Y., Schultz, M.H.: GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing* **7**(3), 856–869 (1986)

- [133] Sherwin, S.J., Casarin, M.: Low-energy basis preconditioning for elliptic substructured solvers based on unstructured spectral/hp element discretization. *Journal of Computational Physics* **171**(1), 394–417 (2001)
- [134] Shu, C.W.: Discontinuous Galerkin method for time-dependent problems: Survey and recent developments. In: *Recent Developments in Discontinuous Galerkin Finite Element Methods for Partial Differential Equations, The IMA Volumes in Mathematics and its Applications*, pp. 25–62. Springer, Cham (2014)
- [135] Söderlind, G.: Automatic control and adaptive time-stepping. *Numerical Algorithms* **31**(1), 281–310 (2002)
- [136] Söderlind, G., Wang, L.: Adaptive time-stepping and computational stability. *Journal of Computational and Applied Mathematics* **185**(2), 225–243 (2006)
- [137] Sonntag, M., Munz, C.D.: Efficient parallelization of a shock capturing for discontinuous Galerkin methods using finite volume sub-cells. *Journal of Scientific Computing* **70**(3), 1262–1289 (2017)
- [138] Toulorge, T., Desmet, W.: Optimal Runge-Kutta schemes for discontinuous Galerkin space discretizations applied to wave propagation problems. *Journal of Computational Physics* **231**(4), 2067–2091 (2012)
- [139] Tucker, P.G.: Progress in aerospace sciences computation of unsteady turbomachinery flows: Part 1 progress and challenges. *Progress in Aerospace Sciences* **47**(7), 522–545 (2011)
- [140] Tucker, P.G.: Progress in aerospace sciences computation of unsteady turbomachinery flows: Part 2 LES and hybrids. *Progress in Aerospace Sciences* **47**(7), 546–569 (2011)

- [141] Van Leer, B., Nomura, S.: Discontinuous Galerkin for diffusion. In: 17th AIAA Computational Fluid Dynamics Conference. AIAA 2005-5108, Toronto, Ontario, Canada (2005)
- [142] Vanden, K.J., Orkwis, P.D.: Comparison of numerical and analytical Jacobians. *AIAA Journal* **34**(6), 1125–1129 (1996)
- [143] Vandenhoeck, R., Lani, A.: Implicit high-order flux reconstruction solver for high-speed compressible flows. *Computer Physics Communications* **242**, 1–24 (2019)
- [144] Vanderstraeten, D.: An expert system to control the CFL number of implicit upwind methods. In: E.F. Toro (ed.) *Godunov Methods: Theory and Applications*, pp. 977–984. Springer US, Boston, MA (2001)
- [145] Vermeire, B.C., Loppi, N.A., Vincent, P.E.: Optimal Runge-Kutta schemes for pseudo time-stepping with high-order unstructured methods. *Journal of Computational Physics* **383**, 55–71 (2019)
- [146] Vermeire, B.C., Nadarajah, S., Tucker, P.G.: Implicit large eddy simulation using the high-order correction procedure via reconstruction scheme. *International Journal for Numerical Methods in Fluids* **82**(5), 231–260 (2016)
- [147] Vermeire, B.C., Vincent, P.E.: On the properties of energy stable flux reconstruction schemes for implicit large eddy simulation. *Journal of Computational Physics* **327**, 368–388 (2016)
- [148] Vermeire, B.C., Witherden, F.D., Vincent, P.E.: On the utility of GPU accelerated high-order methods for unsteady flow simulations: A comparison with industry-standard tools. *Journal of Computational Physics* **334**, 497–521 (2017)

- [149] Vincent, P., Witherden, F., Vermeire, B., Park, J.S., Iyer, A.: Towards green aviation with Python at petascale. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC '16, pp. 1:1–1:11. IEEE Press, Piscataway, NJ, USA (2016). Event-place: Salt Lake City, Utah
- [150] Vos, P.E.J., Eskilsson, C., Bolis, A., Chun, S., Robert, M., Sherwin, S.J.: A generic framework for time-stepping partial differential equations (PDEs): General linear methods, object-oriented implementation and application to fluid problems. *International Journal of Computational Fluid Dynamics* **25**(3), 107–125 (2011)
- [151] Wang, C., Cheng, J., Berndt, M., Carlson, N.N., Luo, H.: Application of nonlinear Krylov acceleration to a reconstructed discontinuous Galerkin method for compressible flows. *Computers and Fluids* **163**, 32–49 (2018)
- [152] Wang, G., Jiang, Y., Ye, Z.: An improved LU-SGS implicit scheme for high Reynolds number flow computations on hybrid unstructured mesh. *Chinese Journal of Aeronautics* **25**(1), 33–41 (2012)
- [153] Wang, Z.J., Fidkowski, K., Abgrall, R., Bassi, F., Caraeni, D., Cary, A., Deconinck, H., Hartmann, R., Hillewaert, K., Huynh, H.T., Kroll, N., May, G., Persson, P.O., van Leer, B., Visbal, M.: High-order CFD methods: current status and perspective. *International Journal for Numerical Methods in Fluids* **72**, 811–845 (2013)
- [154] Wang, Z.J., Rahmani, S.: Implicit large eddy simulation of the NASA CRM high-lift configuration near stall. *Computers & Fluids* **220**, 104887 (2021)
- [155] White, F.M.: Viscous fluid flow, second edn. McGraw-Hill series in mechanical engineering. McGraw-Hill, New York (1991)

- [156] Williams, S., Waterman, A., Patterson, D.: Roofline: An insightful visual performance model for multicore architectures. *Commun. ACM* **52**(4), 65–76 (2009)
- [157] Winters, A.R., Moura, R.C., Mengaldo, G., Gassner, G.J., Walch, S., Peiró, J., Sherwin, S.J.: A comparative study on polynomial dealiasing and split form discontinuous Galerkin schemes for under-resolved turbulence computations. *Journal of Computational Physics* **372**, 1–21 (2018)
- [158] Xiaoquan, Y., Cheng, J., Luo, H., Zhao, Q.: Robust implicit direct discontinuous galerkin method for simulating the compressible turbulent flows. *AIAA Journal* **57**(3), 1113–1132 (2019)
- [159] Yan, Z.G., Pan, Y., Castiglioni, G., Hillewaert, K., Peiró, J., Moxey, D., Sherwin, S.J.: Nektar++: Design and implementation of an implicit, spectral/hp element, compressible flow solver using a Jacobian-free Newton Krylov approach **81**, 351–372 (2021)
- [160] Yang, X., Cheng, J., Wang, C., Luo, H., Si, J., Pandare, A.: A fast, implicit discontinuous Galerkin method based on analytical Jacobians for the compressible Navier-Stokes equations. In: 54th AIAA Aerospace Sciences Meeting. AIAA 2016-1326, San Diego, California (2016)
- [161] Yildirim, A., Kenway, G.K.W., Mader, C.A., Martins, J.R.R.A.: A Jacobian-free approximate Newton-Krylov startup strategy for RANS simulations. *Journal of Computational Physics* **397**, 108741 (2019)
- [162] YOON, S., JAMESON, A.: Lower-upper symmetric-Gauss-Seidel method for the Euler and Navier-Stokes equations. *AIAA Journal* **26**(9), 1025–1026 (1988)

- [163] Zhang, L., Li, M., Liu, W., He, X.: An implicit algorithm for high-order DG/FV schemes for compressible flows on 2D arbitrary grids. *Communications in Computational Physics* **17**(1), 287–316 (2015)
- [164] Zhu, H., Fu, S., Shi, L., Wang, Z.J.: Implicit large-eddy simulation for the high-order flux reconstruction method. *AIAA Journal* **54**(9), 2721–2733 (2016). Publisher: American Institute of Aeronautics and Astronautics
- [165] Zhu, J., Zhong, X., Shu, C.W., Qiu, J.: Runge-Kutta discontinuous Galerkin method with a simple and compact hermite WENO limiter. *Communications in Computational Physics* **19**(4), 944–969 (2016)