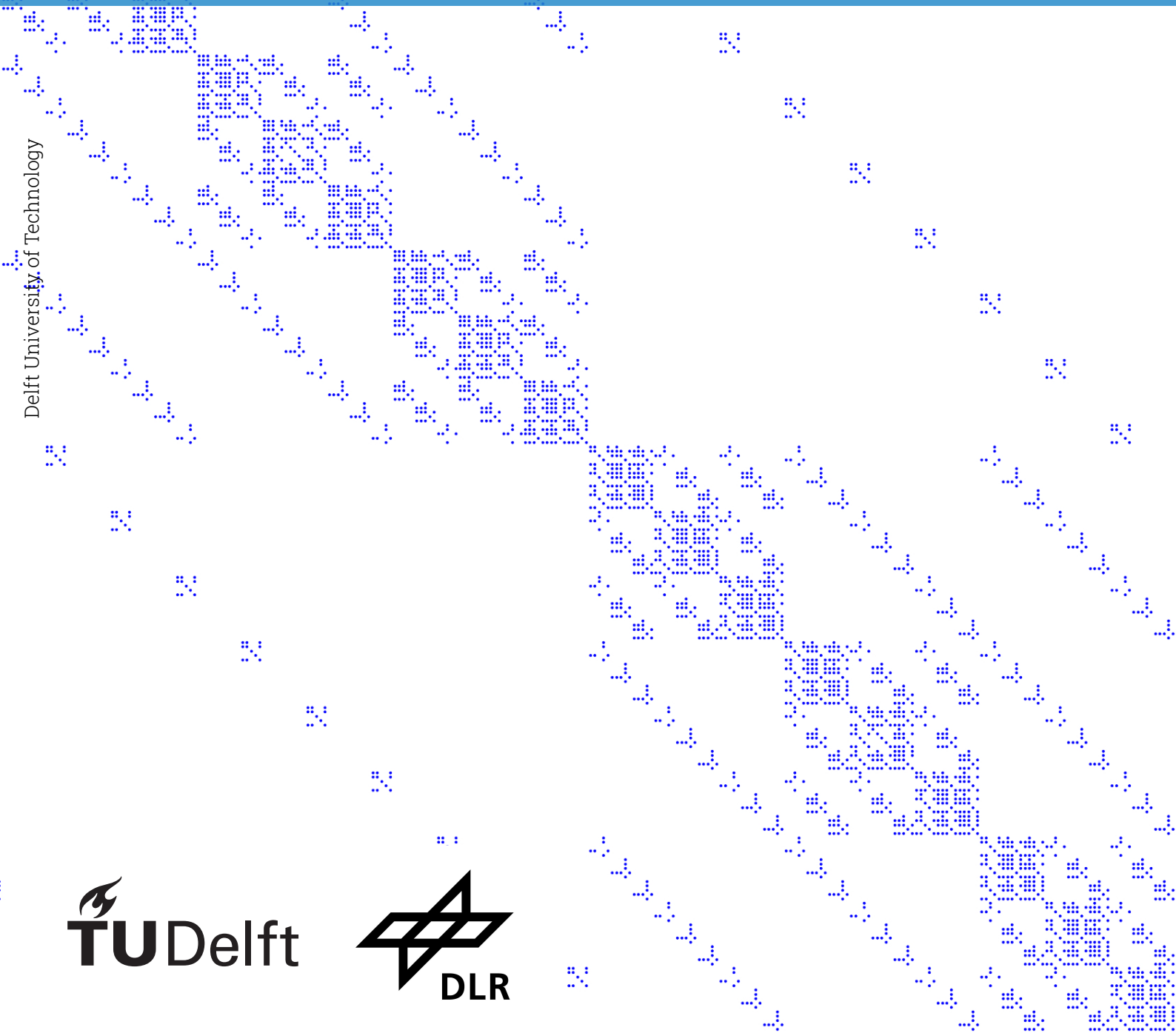
Implementation and Analysis of a Prototype Implicit, Matrix- Free, DGSEM Solver for LES

MSc Thesis in Aerospace Engineering

Jurgen Jacobus de Groot

Delft University of Technology



TU Delft

DLR

Implementation and Analysis of a Prototype Implicit, Matrix- Free, DGSEM Solver for LES

MSc Thesis in Aerospace Engineering

by

Jurgen Jacobus de Groot

to obtain the degree of Master of Science
at the Delft University of Technology.

Student number:	5303133	
Project duration:	17 March 2025 – 18 November 2025	
Thesis committee:	Dr. I. Langella	TU Delft, Chair
	Dr. S.J. Hulshof	TU Delft, Examiner
	Dr.ir. M.I. Gerritsma	TU Delft, Supervisor
	Dr. G.B. Ashcroft	DLR, Supervisor

Cover: Jacobian non-zero pattern created by LGL-DGSEM spatial operator using Euler equations

TU Delft DLR

Preface

I would like to dedicate this space to all the people who have supported me in the making of this work. First, I would like to express my sincere gratitude to Dr. Edmund Kügeler and Dr. Graham Ashcroft for providing me with the opportunity to perform my thesis project at the Institute of Propulsion Technology at DLR in Cologne. Under Dr. Graham Ashcroft his supervision and guidance, I have been able to experience academic freedom in my work, driven by insightful discussions and constant support.

I want to thank all the people in the Numerical Methods department; I have truly felt supported and part of the team from the moment I started at the institute. The shared passion for numerical methods of the entire department made me learn new things every day. In particular, I would like to thank Dr. Jan Backhaus for his help on the implementation of the solver in TRACE. His knowledge and generous support were truly essential to my progress. I am also very grateful to Dr. Björn Klose for his help with setting up the LES test cases. I want to thank Jens Wellner, Christian Berthold and Felix Schwarzenthal for the lunch runs every Tuesday and Thursday. They helped me stay healthy during the thesis, and the positive impact of that on my work shouldn't be underestimated.

I would also like to thank my TU Delft supervisor, Dr.ir. Marc Gerritsma, for providing me with supervision from the university. His guidance and support during the thesis ensured the project stayed on track.

I want to thank my friends, some of whom I met during my stay in Germany, and others I had to temporarily say goodbye to in order to pursue this academic endeavour. Their support and encouragement have meant a lot to me throughout this time. Last but not least, I want to give my special thanks to my parents. Their unconditional support, love, and guidance in life have been the driving force behind this journey. I could not have reached this milestone without them.

Abstract

This master's thesis explores the application of a matrix-free GMRES solver for implicit time integration of unsteady turbulent flows using a high-order DGSEM spatial discretisation. Efficient time-integration methods remain one of the primary challenges towards enabling industrial use of high-order methods for scale-resolving simulations. The focus of this study is on the development of such a prototype implicit solver for the DGSEM solver in the CFD framework TRACE, developed by DLR, which specialises in internal aerodynamics of turbomachinery components. To gain a fundamental understanding of the method and have a lightweight experimentation tool, a one-dimensional DGSEM solver that solves the Burgers equation was first developed from scratch. With this gained knowledge, the implementation of the solver in TRACE could start. The prototype solver developed in TRACE is capable of using both finite-difference and forward automatic differentiation techniques to enable the matrix-free GMRES method. Performance tests show that the automatic differentiation method is approximately twice as slow as the finite-difference differentiation method when comparing them for an equal number of linear iterations. However, the finite-difference method produces inaccurate gradient estimations that could influence the linear solve. Lastly, the solver was applied to perform an LES of a turbulent channel flow. The implicit solver yielded similar results compared to the time-explicit reference simulations at time step sizes greater than the explicit limit. Using the finite-difference differentiation method, the implicit solver was also found to require fewer CPU hours than the explicit reference simulations.

Keywords: DGSEM, JFNK, matrix-free, GMRES, LES, TRACE, CFD, time-implicit, IRK

Contents

List of Figures

List of Tables

Nomenclature

Abbreviations

AD Automatic Differentiation

ASCII American Standard Code for Information Interchange

BDF Backward Differentiation Formula

BE Backward Euler

BR1 Bassi-Rebay 1

BR2 Bassi-Rebay 2

CFD Computational Fluid Dynamics

CFL Courant-Friedrich-Lewy

CPU Central Processing Unit

CPU Graphics Processing Unit

DG Discontinuous Galerkin

DGSEM Discontinious Galerkin Spectral Element Methods

DIRK Diagonally Implicit Runge-Kutta

DLR German Aerospace Center

DNS Direct Numerical Simulation

DOF Degrees Of Freedom

DTS Dual Time-Stepping

EBJ Element Block-Jacobi

ERK Explicit Runge-Kutta

ESDIRK Explicit first stage Singly Diagonally Implicit Runge-Kutta

FD Finite Difference

FE Finite Element

FGMRES Flexible Generalised Minimal Residual Method

FV Finite Volume

FVM Finite Volume Method

GMRES Generalised Minimal Residual Method

HPC High Performance Computing

ILU Incomplete Lower Upper

IMEX Implicit-Explicit

IRK Implicit Runge-Kutta

IVP Initial Value Problem

JFNK Jacobian-Free Newton-Krylov

KG Kennedy-Gruber

LCF Laminar Channel Flow

LES Large Eddy Simulation

LG Legendre-Gauss

LGL Legendre-Gauss-Lobatto

LU-SGS Lower Upper Symmetric Gauss-Seidel

MB Matrix-Based

MF Matrix-Free

MUSCL Monotonic Upstream-Centered Scheme for Conservation Laws

NSE Navier-Stokes Equations

PDE Partial Differential Equation

RANS Reynolds Averaged Navier-Stokes Equations

RK Runge-Kutta

RKDG Runge-Kutta Discontinuous Galerkin

ROBO-SGS Reduced Off Diagonal Order Symmetric Gauss-Seidel

SBP Summation-By-Parts

SDIRK Singly Diagonally Implicit Runge-Kutta

SEM Spectral Element Method

SGS Symmetric Gauss-Seidel

Spliss Sparse Linear System Solver

SRS Scale-Resolving Simulations

SSOR Symmetric Successive Over-Relaxation

SST Shear Stress Transport

SUPG Stream Upwind Petrov Galerkin

TCF Turbulent Channel Flow

TRACE Turbomachinery Research Aerodynamic Computational Environment

URANS Unsteady Reynolds Averaged Navier-Stokes Equations

WENO Weighted Essentially Non-Oscillatory

Latin Symbols

$\mathcal{F}$	Contravariant flux vector	-
$\mathcal{H}$	Numerical flux vector	-
$\mathcal{M}$	Transformation matrix from the physical to the reference coordinate system	-
a^i	Contravariant basis vector in ξ_i direction	-
a_i	Covariant basis vector in ξ_i direction	-
A	Jacobian matrix	-
a	Runge-Kutta stage coefficients	-
B	Boundary operator	-

$\boldsymbol{D}$	Discrete derivative matrix	
$\boldsymbol{F}$	Covariant flux vector	-
$\boldsymbol{h}$	Upper Hessenberg matrix	-
$\boldsymbol{I}$	Identity matrix	
$\boldsymbol{J}$	Jacobian determinant matrix	-
$\boldsymbol{M}$	Preconditioning matrix	-
$\boldsymbol{x}$	Cartesian coordinate vector $\boldsymbol{x} = (x, y, z)^T = (x_1, x_2, x_3)^T$	-
$\boldsymbol{x}$	Cartesian velocity vector $\boldsymbol{x} = (u, v, w)^T = (u_1, u_2, u_3)^T$	-
a	Speed of sound	ms^{-1}
l	Lagrange polynomial	-
M	Mach number	-
N	Number of inner-element degrees of freedom	-
p	Polynomial order	-
p	Pressure	Pa
T	Temperature	K
t_{wall}	Wall-clock time	s
w	Numerical integration (Gauss) weights	-
y^+	non-dimensional wall distance	-
c_p	Specific heat capacity at constant pressure	$\mathrm{J\,kg^{-1}\,K^{-1}}$
E	Energy spectrum function	-
E	Total specific Energy	$\mathrm{J\,kg^{-1}}$
f	Frequency	Hz
H	Total enthalpy	$\mathrm{kg\,m^2 s^{-2}}$
Pr	Prandtl number	-
R	Gas constant	$\mathrm{J\,K^{-1} kg^{-1}}$
Re	Reynolds number	-
t	Time	s

Greek Symbols

β	Initial GMRES residual norm	-
$\boldsymbol{\tau}$	Viscous stress tensor	Pa
$\boldsymbol{\xi}$	Coordinate vector of reference element $\boldsymbol{\xi} = (\xi, \eta, \zeta)^T = (\xi_1, \xi_2, \xi_3)^T$	-
δ	Dirac delta function	-
δ	Half channel height	m
$\delta_{i,j}$	Kronecker delta	-
ϵ	Linear solve convergence criterion	-
ϵ	Perturbation parameter	-
η	Forcing term	-

γ	Heat capacity ratio	-
γ	Inexact Newton method forcing term damping parameter	-
λ	Conductivity	W $m^{-1}K^{-1}$
λ	Eigenvalue	-
$\mathcal{K}_m$	Krylov subspace of dimension m	-
μ	Molecular Viscosity	kg m^{-1} s^{-1}
ν	Kinematic Viscosity	m^2s^{-1}
Ω	Domain	-
ϕ	Basis function	-
ψ	Bassi-Rebay 2 correction factor	-
ρ	Density	Kg m^{-3}
σ	Inexact Newton relative convergence criterion	-
τ_w	Wall shear stress	-

Sub- and superscripts

∞	Free-stream
ijk	Values at node (i,j,k) in the three-dimensional element
0	Initial solution or parameter
a	Advective part
bc	Value at domain boundary
v	Viscous part

Decorations and operators

$\bar{\square}$	Time-averaged quantity
$\square''$	Fluctuating part of a Favre-averaged quantity
$\square^+$	Non-dimensionalised quantity based on friction velocity u_τ
$\square^+$	Right value at element interface
$\square^-$	Left value at element interface
$\Delta\square$	Width, step-size
$\|\|\square\|\|$	Euclidean norm
∇_ξ	Nabla operator in reference space $\nabla_\xi = (\frac{\partial}{\partial\xi}, \frac{\partial}{\partial\eta}, \frac{\partial}{\partial\zeta})^T$
∇_x	Nabla operator in physical space $\nabla_x = (\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z})^T$
$\tilde{\square}$	Favre-averaged quantity

1 Introduction

The ability to numerically simulate turbulent flows with a high resolution and accuracy is in increasing demand by several industries to solve engineering problems related to fluid dynamics. In the context of turbomachinery engineering for aero engines, this increased demand is driven by the need for improved engine designs that meet future climate goals.

To enable these developments, numerical modelling approaches in turbomachinery engineering must be capable of accurately simulating a wide range of flow phenomena in a consistent and reliable manner. With the continuous refinement in aero engine design, the understanding of complex flow configurations in turbomachinery components has become a problem of increasing importance. The current industrial use of computational fluid dynamics (CFD) in turbomachinery design primarily focuses on solving the steady and unsteady Reynolds-averaged Navier-Stokes (RANS) equations. Although very efficient solvers have been produced for these equations, they naturally fail to represent flow features such as separation and turbulent transitions due to their inherent assumptions. Thus, performing simulations of flows with such features requires extensive modelling. In order to simulate such flows with more accurate and consistent results, the Navier-Stokes equations must be solved with higher fidelity. Therefore, the use of scale-resolving simulation (SRS) by means of large eddy simulation (LES) is becoming of increasing interest for the aero engine design process. SRS have been proven to produce more consistent and accurate results for the aforementioned flow configurations [1]. To facilitate the future use of SRS, more efficient numerical methods are sought after. Specifically, high-order methods have been shown to produce higher accuracy results for equal computational costs compared to the traditional second-order Finite Volume (FV) method, leading to a higher computational efficiency of the scheme [2]. The term *high-order* is generally accepted to be a spatial discretisation of third-order or higher [3], again stemming from the common usage of second-order FV methods in the CFD community.

Among the high-order methods, discontinuous Galerkin spectral element methods (DGSEM) have been found to be particularly well-suited for fluid dynamic simulations. As the name suggests, the method makes use of a discontinuous Galerkin (DG) formulation, where solutions between elements are allowed to be discontinuous and a Galerkin method is assumed. This is combined with the spectral element method (SEM), where nodal-based high-order Lagrange polynomials are often used as basis functions in combination with collocation of the quadrature and interpolation nodes. The DGSEM schemes exhibit both properties of a high-order finite element method (elements with internal degrees of freedom) and a finite volume method due to the allowed discontinuity between elements, resulting in formulations of fluxes between the element faces. With this method, elements can be assigned arbitrary spatial orders as they are only coupled through the fluxes at the faces. This allows for the flexible use of h, p-refinement strategies. Compared to other DG methods, the DGSEM allows for the simplification of many numerical operations due to the collocation of quadrature and interpolation nodes combined with the use of Lagrange polynomials. Furthermore, the DGSEM spatial discretisation has very favourable dispersion-dissipation relations, which result in the dissipation of the spatial operator being pushed to high wave numbers. This is a property that is very desirable in SRS, as the resolved scales are thereby

less influenced by the dissipation of the spatial operator. At the same time, smaller scales are dissipated, which is desirable when considering implicit LES. Thus, it can be said that the dissipation of the DGSEM operator possesses desirable properties, making it suitable to act as a built-in sub-grid-scale model [2].

As SRS considers the simulation of unsteady phenomena, time integration must also be considered. Explicit time-integration methods have been favoured for SRS due to their ease of implementation as well as their locality, which fits with the good locality of the DGSEM. On the other hand, high-order DG methods such as DGSEM are characterised by their increased time-step restriction compared to the FV method. This imposes a stability condition that can be excessively restrictive on the time-integration scheme, resulting in time-step sizes that are significantly smaller than what is required to resolve the relevant physical time scales in the SRS accurately. Therefore, possible efficiency gains are being left on the table due to the usage of an explicit time-integration scheme. Furthermore, there is also potential in applying implicit-explicit (IMEX) time-integration schemes to SRS cases that suffer from geometric-induced stiffness [4]. Here, only the stiff part of the domain (e.g. mesh refinement near the wall) is solved implicitly. To make such advanced time-integration schemes possible, efficient implicit solvers are required.

This motivates the exploration of implicit time-integration schemes applied to DGSEM for simulating unsteady turbulent flows. Such methods can lift the time step size constraints, thus enabling time step sizes that are chosen based on the temporal accuracy required by the solution. A promising approach to perform the implicit solving is with the use of so-called matrix-free linear solvers. This class of solver can avoid the explicit formulation and storage of the Jacobian matrix by only making use of matrix-vector product evaluations of the Jacobian matrix in the linear solve. With this, the matrix-vector product can be evaluated in a matrix-free manner by, for instance, the usage of finite differences or automatic differentiation [5]. This approach significantly reduces the memory footprint and computational overhead typically associated with implicit schemes, making it particularly attractive for usage in the DGSEM for SRS.

The goal of this work is therefore to explore the application of implicit time-integration schemes using a matrix-free linear solver with the DGSEM method in the simulation of unsteady turbulent flows governed by the compressible Navier-Stokes equations. With the overall aim of enabling future applications of the method in turbomachinery-related contexts. The research in this thesis is focused on the development of such a method in the Turbomachinery Research Aerodynamic Computational Environment (TRACE) code developed by DLR. TRACE is a CFD solver that is specialised in the simulation of internal aerodynamics for turbomachinery design.

1.1. Research Goals of the Thesis

The DGSEM method is envisioned to be an essential design tool in SRS for the future of turbomachinery engineering. In this thesis, the focus will be on the application of implicit time-integration schemes and the subsequent implicit solving methods required in combination with the DGSEM spatial discretisation method. Such methods, due to their greater complexity compared to explicit time-integration schemes, are still an active area of research. This too is the case for the DGSEM solver developed in TRACE in recent years [2]. This solver is mostly used together with explicit Runge-Kutta (ERK) time-integration schemes. Therefore, the research objective of this thesis is formulated as:

"To evaluate and understand the implementation of matrix-free, implicit time-integration schemes to the DGSEM method for convection-diffusion simulations, and produce a prototype matrix-free implicit solver for the DGSEM method in the CFD framework TRACE."

To answer this research question, the following research goals were generated:

- Evaluate the usage of matrix-free GMRES for unsteady implicit solving.
- Evaluate the performance of simple preconditioning methods for matrix-free GMRES.
- Compare the usage of automatic differentiation against finite-difference differentiation in matrix-free GMRES.
- Analyse the effect of temporal resolution on the flow solution in LES of turbulent flows.

1.2. Structure of the Report

Following the stated goals of the thesis, an overview of the report is given. Firstly, in Chapter 2, the findings of the literature review performed at the start of the thesis are described. The literature review serves as a descriptive overview of the state-of-the-art on implicit solving methods for DGSEM, as well as further motivation for the thesis. After this, in Chapter 3, the derivation of the DGSEM together with the time-discretisation and the matrix-free GMRES method is presented. With the groundwork laid out, Chapter 4 covers the implementation and subsequent analysis of the stand-alone one-dimensional DGSEM solver that was produced in this project. After this, the implementation of the implicit solving scheme for the DGSEM solver in TRACE and the analysis of this implementation are presented in Chapter 5. Finally, Chapter 6 will summarise the outcomes of the thesis, answer the initially posed research questions, and provide a future outlook and recommendations.

2

Literature Review

The research scope of this project is the application of implicit time-integration methods to LES using high-order spatial discretisation methods. The literature review will first provide an overview of the state-of-the-art on high-order methods for scale-resolving simulations in Section 2.1. After this, the focus will shift to implicit time-integration methods applied to scale-resolving simulations in Section 2.2. Following the formulation of the time-implicit scheme, the research on efficient nonlinear and linear solver schemes to solve the implicit system emerging from these schemes is covered in Section 2.3. Finally, the outcomes of the literature review are summarised in Section 2.4.

2.1. High-order Methods for Scale-resolving Simulations

Solving a partial differential equation (PDE) numerically can be done using many different techniques. The spatial discretisation schemes can be classified into three categories, [6]:

- The finite difference (FD) method, where the domain is discretised into nodes and the approximate differential form of the system of equations is satisfied at each node. Spatial derivatives are obtained using truncated Taylor expansions on the nodes.
- The finite volume (FV) method, where the domain is discretised into cells, and the integral form of the system equations is satisfied in each cell, leading to a cell-average solution. The cell-average solutions are coupled through numerical fluxes, which result in a globally defined system.
- The finite element (FE) method, where the domain is discretised into sub-domains (elements) that contain internal degrees of freedom. An approximating function space (e.g. piecewise polynomials) is used to define the solution on each element. The degrees of freedom correspond to coefficients obtained from sampling the solution (e.g. nodal or modal sampling) for which the variational (weak) form of the system of equations is minimised. In order to obtain a unique solution, a set of linearly independent basis functions spanning the approximating solution space is chosen accordingly to the choice of degrees of freedom. The solution between elements can either be continuous or discontinuous. Combining all element local formulations, a global system of equations is constructed.

All three methods allow the formulation of a spatial scheme of arbitrary order. However, for the FD and FV method, a direct application of high-order approximations to the method leads to instabilities related to the Gibbs phenomenon. The most popular class of schemes to overcome these stability issues is the weighted essentially non-oscillatory (WENO) schemes [7]. These schemes use smaller inner stencils to reconstruct the solution on the main stencil as a weighted combination of the sub-stencil reconstructions, where oscillations are suppressed by adjusting the weights of the inner stencils. Although WENO schemes achieve stable high-order FD and FV schemes, they also require significantly large stencils, which reduce the parallel scalability of the method and increase the computational cost. Specifically for implicit methods, the increased stencil size means a denser Jacobian structure, which negatively affects the performance of the linear solver. High-order FV WENO schemes have been improved upon in recent years [8], but their problems of a large stencil size and adaptability to complex grid structures

still remain. For industrial use of (U)RANS and LES, the second-order accurate FV method with the monotonic upstream-centred scheme for conservation laws (MUSCL) is most common. This is because the method has a compact stencil and exhibits more predictable dissipation characteristics when limiters are used. This comes at the cost of reduced efficiency due to the low order of accuracy of the method and the requirement for grid refinement to increase the solution accuracy. Moreover, WENO schemes are typically applied only to structured grids, as extensions to unstructured grids are possible only for certain grid formulations [9].

The FE method, originally developed and still widely used for numerical simulation in structural mechanics, works differently from the FD and FV methods in that the domain is discretised into elements that act as sub-domains with internal degrees of freedom. As said, an approximate solution space is used to construct a solution to the PDE. The solution is formulated within this function space by sampling it. To achieve a unique solution in accordance with the chosen sampling points, a linearly independent *finite* set of basis functions that span the approximate function space is used. Traditionally, FEM was applied not to satisfy the PDE directly, like the FD or FV method, but to minimise a physical functional (e.g. potential energy in structural mechanics). In this minimisation process, test functions naturally arise from a variational analysis, and by performing integration by parts, the weak form of the system is found. More generally, however, the FE method can be seen as a weighted residual method, where the PDE is multiplied by test (weight) functions and integrated to minimise the residual over the element. In structural mechanics, linear piecewise polynomials that are nodally sampled at the end points of the element are often used, leading to a second-order accurate spatial discretisation. In this use case of the FE method, spatial accuracy is controlled through mesh refinement (h-FEM) [10]. Nonetheless, the FE method suits the extension to higher orders of accuracy well due to the local element formulation and inner-element nodes. Due to this, p-FE and hp-FE methods have been developed, showcasing spectral convergence rates [11, 12, 11].

The choice of the test function space can be arbitrary in the FE method. However, when the test function space is chosen to be equal to the solution function space, it is referred to as a Galerkin method. The Galerkin approach is a common method because it can be shown that if the operator is symmetric and positive definite, setting the solution and test functions equal to each other leads to a minimum error [13]. In other words, the Galerkin method results in an orthogonal projection of the exact solution onto the approximate solution space in the case of a symmetric positive definite operator. The Galerkin method can also be applied to operators without these properties, although this may result in instabilities of the method. Specifically, these properties do not hold for fluid mechanics [14] due to convection phenomena not being symmetric. In order to stabilise the solution of these schemes, a different test function space is required, leading to a Petrov-Galerkin scheme. For instance, the stream upwind Petrov Galerkin (SUPG) method [15] has been found to stabilise fluid mechanics problems using the FE method.

A class of FE methods that is more suited for convection-dominated problems are discontinuous Galerkin (DG) finite element methods. Originally developed by Reed and Hill [16] for the neutron transport equation. With the DG method, the same basis and test function space are assumed, but the solution is allowed to be discontinuous between element interfaces. In order to still couple the elements, a Riemann solver is required to determine the fluxes between the elements. This aspect of the method is very similar to the FV method, where Riemann solvers are also required to deal with the discontinuities between the cell average values. The method can thus be seen as a combination of the FE and FV methods, using the most favourable characteristics of both methods for convection-dominated problems. From this perspective, it could even be said that the FV method is simply a zero-th order DG method. The first developments on the DGFEM method applied to nonlinear hyperbolic conservation laws were by Cockburn and Shu [17, 18, 19, 20, 21], who introduced the method in combination with explicit Runge-Kutta time-integration schemes, naming them Runge-Kutta discontinuous Galerkin (RKDG) methods. The RKDG method was first applied to the Navier-Stokes equations by Bassi and Rebay [22], where they introduced methods to treat the viscous terms through the initial BR1 scheme and subsequent BR2 [23]. The method has been of specific interest for scale-resolving simulations because of the favourable dispersion-dissipation properties which push dissipation towards high wave numbers [24]. Not only has the method been increasingly popular for scale-resolving simulations [25, 26, 27], but it also has gained attention to solve RANS equations [28, 29, 30].

One specific variant of the DG method is when it is combined with a spectral element method (SEM).

Firstly, the SEM will be covered shortly. The spectral element method is a form of finite element method where spectral accuracy is achieved, meaning that the convergence rate can be exponential if the (analytical) solution is sufficiently smooth. Spectral methods achieve this by choosing high-order orthogonal polynomial basis functions. Such polynomials are, for instance, Chebyshev or Legendre polynomials. For SEM, besides Galerkin and Petrov-Galerkin approaches, another approach is the *collocation* approach. With collocation, the test functions are chosen to be Dirac deltas at specific collocation points. In practice, Lagrange polynomials are often used for this purpose. This causes the method to enforce the residual of the strong form to zero at the collocation points [31]. Going back to the Galerkin approach, the Lagrange polynomials can also be used to evaluate any orthogonal polynomial basis function in a nodal manner. This means that the polynomial basis functions (e.g. Legendre polynomials) can be represented in Lagrange form by using the roots of the chosen orthogonal polynomials [32, 31]. This can thus be described as a nodal spectral Galerkin method. Interestingly, the Lagrange polynomials are not orthogonal themselves, but because the choice of quadrature and interpolating nodes corresponds to the roots of orthogonal polynomials, the formulation is orthogonal in the discrete sense. The method is different from the collocation approach because it enforces the weak form residual rather than the strong form residual at the nodes. Nonetheless, it can be considered a collocation approach in the weak sense [31].

Combining the nodal spectral Galerkin method with a DG method, we arrive at the discontinuous Galerkin spectral element method (DGSEM). This method uses a nodal spectral Galerkin approach, allowing for discontinuities between element interfaces. Usually, the Legendre polynomials are used as the underlying interpolating orthogonal polynomial basis. When the exact roots of the Legendre Polynomials are used, the nodes are typically referred to as the Legendre-Gauss (LG) nodes or LG quadrature points. Usage of these nodes ensures that the Gauss quadrature is exact for all functions of polynomial order $2p + 1$ or less. Here p is the order of the polynomial approximation. Because, in theory, the orthogonal Legendre polynomial basis functions can be extended to infinite order of accuracy, the order p is limited by the $N = p + 1$ number of interpolating nodes. A downside of using the LG nodes is that no quadrature points are located on the element boundary ($\Omega = [-1, 1]$). Usually, the solution at the element interfaces is required for boundary conditions and numerical flux evaluations in the discontinuous case, and an interpolation of the solution onto the element interface is necessary when using the LG nodes. The Legendre-Gauss-Lobatto (LGL) quadrature points fix this issue by placing the two end nodes at the element boundary (when considering a 1D element). The other inner quadrature points are again based on the roots of the Legendre polynomials. By doing this, the exactness of the Gauss quadrature is reduced to polynomial orders of $2p - 1$ [32]. On the other hand, the LGL nodes have been found to allow for roughly twice the time step sizes compared to LG nodes [33]. This results in the use of LGL nodes being numerically more efficient, at the cost of non-exact integration, which removes the spectral accuracy of the method. Nonetheless, the LGL modes result in *near* spectral accuracy. For the extension of the method to two- or three-dimensional elements (as is required for practical usage of the method), a tensor-product approach has to be taken in the construction of the Lagrange basis functions in order to retain the discrete orthogonality of the method.

The DGSEM has quite a complex theoretical background as described before. However, the resulting numerical method contains many numerical simplifications due to the choices made. Firstly, because of the choice of a *nodal* representation in Lagrange form, the physical variables of interest are directly available at the interpolation points. Secondly, due to the Galerkin approach with Lagrange polynomials, the collocation in the weak sense reduces the integrals to pointwise evaluations at the quadrature nodes. This means that the mass matrix reduces to a diagonal matrix [34] and thus is trivial to invert. Thirdly, the choice of quadrature nodes enables (near) spectral accuracy of the method. Lastly, because of the discontinuous formulation of the solution, the method allows for a local high-order spectral formulation in the element, but the usage of numerical fluxes between elements results in a very compact stencil of the method. This is particularly attractive for the parallel scalability of the method as shown by [35], as only the element interface states have to be communicated between neighbouring elements in order to evaluate the numerical fluxes. The discontinuity also enables the use of Riemann solvers that stabilise the otherwise unstable Galerkin formulation when asymmetric operators, such as convective terms, are present. This makes the DGSEM a very computationally efficient method compared to other DG methods [36].

The DGSEM method, however, does have disadvantages. Namely, although the usage of discontinuity

between elements stabilises the global solution, the formulation of the solution within an element is still unstable due to the usage of a Galerkin approach on an asymmetric operator. It must be noted that these instabilities only arise when the numerical solution is under-resolved. Under-resolving of the solution, however, is exactly the goal of LES, and steep gradients of the solution, such as pressure waves or shocks, also would be expensive to completely resolve. Furthermore, the aliasing errors become more apparent with the use of high-order polynomial approximations due to their lower dissipation [37], and could even lead to the method becoming unstable and crashing [38]. A computationally efficient and robust solution to the aliasing problem is to use split form flux formulations, first developed by Gassner et al. [37] following the work of Carpenter and Fisher [39, 40]. This method utilises the summation-by-parts (SBP) property of the DGSEM method when LGL nodes are used. In this method, the flux functions are rewritten into a split formulation, resulting in an equivalent formulation in the continuous space but with different properties in the discrete formulation. Different split form formulations exist, such as the one from Kennedy and Gruber [41] or Pirozzoli [42]. These different split-forms are described and compared in detail by Gassner et al. [37]. Although other methods, such as over-integration, can also be applied to overcome the aliasing problems, split-form methods have been found to be more efficient while providing similar stabilisation to the method [2]. Another disadvantage of the DGSEM is that, due to the tensor-product formulation of the basis functions, the method is restricted to two-dimensional quadrilateral and three-dimensional hexahedral elements. The tensor-product formulation is a crucial choice for reducing the numerical operations required. However, for practical usage, it currently limits the method to such element types only.

To summarise, traditional CFD methods such as the FV method have been found unsuitable for the realisation of efficient scale-resolving simulations. Finite element methods are much more flexible in terms of high-order formulations. Specifically, the DGSEM in combination with LGL nodes and a split form flux formulation is a highly efficient spatial discretisation method that allows for arbitrary polynomial orders and is robust against aliasing errors. Therefore, the LGL-DGSEM method is analysed in this project in combination with implicit time-integration methods.

2.2. Implicit Time-integration Methods in Scale-resolving Simulations

For LES simulations using DG methods, explicit Runge-Kutta methods have generally been preferred for temporal discretisation. The first reason for this choice is that LES simulations are often large-scale, requiring efficient parallel scalability. A strong point of the DGSEM method is its local formulation and, therefore, great parallel scalability. Because explicit time-integration methods only perform a local solve, they are able to exploit the locality of the DGSEM scheme fully and only communicate face states to neighbouring elements [43] in between Runge-Kutta iterations. Secondly, as unsteady turbulent structures are required to be resolved both spatially and temporally, an upper limit exists on the allowable time step to temporally resolve the flow. This means that the leverage that implicit schemes have over explicit schemes, namely increasing the time-step size, is limited, because a too large time-step size would cause excessive temporal dissipation. Lastly, the implementation effort of explicit schemes is generally lower compared to implicit time-integration methods, as no implicit solving method is required.

On the other hand, high-order methods impose a severe constraint on the time step size when explicit time-integration methods are used. As shown in [44], in general, for DG schemes, the convective Courant number scales with $(2p + 1)$, meaning that the higher the polynomial order, the stricter the CFL condition. Here p again denotes the degree of the polynomial approximation. The efficiency of time-integration methods has been identified as one of the main issues holding back the performance of high-order DG schemes [3]. As stated in [45], in industrial applications, the explicit time step restriction could result in an inefficient overall framework. Thus, it becomes interesting to explore the usage of implicit methods to overcome this stability constraint. Such methods would then allow the time step to be chosen based solely on temporal accuracy [46].

For high-order DG schemes, the search for implicit schemes that can compete with explicit counterparts has been an active area of research. The first usage of implicit schemes to DG formulations of the unsteady Navier-Stokes equations is from Bassi and Rebay [47], who investigated the use of the backward

Euler scheme for the two-dimensional compressible Navier-Stokes equations. They used Newton's method to linearise the system, which was then solved using a left-preconditioned generalised minimal residual iterative method (GMRES). Similarly, Persson and Peraire [48] studied the implementation of the backward differentiation formula of order k (BDF-k), where again emphasis was put on the preconditioning method. Although the backward Euler (which is BDF1) and BDF-k schemes offer ease of implementation, they are only A-stable up to second order.

Implicit schemes of higher order accuracy have also been studied on the DG methods applied to the Navier-Stokes equations. The application of linearly implicit Rosenbrock-type Runge-Kutta schemes [49] has been extensively researched in [50, 51]. Rosenbrock schemes are a linearised version of the explicit first stage singly diagonally implicit Runge-Kutta (ESDIRK) schemes, which reduce the Runge-Kutta stages to only require one linear solve [49]. These methods are of particular interest for high-order DG methods as they only need one Jacobian formulation per time step. Additionally, only one linear solve is required per stage. This is attractive for high-order DG, as the Jacobian formulation can often be costly. Furthermore, if correctly preconditioned, Rosenbrock schemes have been shown to have greater performance compared to similar ESDIRK schemes, especially when considering stiff systems of equations and high-order time-integration schemes [52]. In Section 2 of [53], the same authors compare the usage of the linearly implicit Rosenbrock Runge-Kutta methods with matrix-based and matrix-free formulations applied with different preconditioning methods and time step adaptation on the incompressible NSE. In [54], this work was extended to the compressible NSE. It is noted that if an approximate Jacobian formulation is used, then the choice is restricted to W-methods [49, 55].

Another class of time-integration schemes that have been applied to the DGSEM are the singly diagonally implicit Runge-Kutta (SDIRK) and explicit first stage SDIRK (ESDIRK) schemes. In Section 3 of [53], a 5-stage SDIRK method is used together with a dual time-stepping approach that utilises a block-Jacobi preconditioned GMRES method. In this study, the dual-time stepping approach is used to converge the nonlinear system of equations resulting from the SDIRK formulation. For this dual time-stepping, the backward Euler scheme is used as accuracy is not required to converge the steady-state solution in pseudo-time. With this, they demonstrate the efficiency of a well-designed SDIRK scheme through correct preconditioning. Both [56] and [57] implement ESDIRK schemes to DGSEM and compare them with the ERK4 scheme. Here, speed-ups are found at sufficient accuracy levels, given the implicit solver is well designed (good preconditioning and use of Jacobian lagging). In [46], the low-order BDF1 and BDF2 schemes are compared with an ESDIRK4 scheme, and the fourth-order scheme was found to be consistently more efficient than the low-order schemes.

Fully implicit schemes are, however, not the only interesting time-integration schemes that have been applied to DG. Namely, another family of time-integration schemes that was found to show promising results for simulation of unsteady flows using high-order DG methods are the so-called implicit-explicit (IMEX) schemes [58]. IMEX schemes allow for only partial implicit time integration on the system of equations, solving the remaining part explicitly. An IMEX scheme can be applied to split the system of equations into stiff and non-stiff components, which can be solved implicitly and explicitly, respectively. In the case of highly local stiff systems, this allows for a significant reduction of the implicit system size while still alleviating the time-step size restrictions faced in pure explicit time integration. IMEX can be applied on a source term basis where the hyperbolic (convective) part is treated explicitly and the parabolic (diffusive) part is treated implicitly. The method was originally designed to handle the stiffness of a convection-diffusion-reaction operator [58]. However, for the compressible Navier-Stokes equations, the zonal IMEX method using IMEX-RK schemes, where the computational domain is split into explicit and implicit parts to combat geometric-induced stiffness, has shown promising results [59, 60, 4, 61].

Although these methods appear to be more efficient than full implicit time integration for high-order DG scale-resolving simulations, it must be noted that for these schemes to work properly, efficient explicit and implicit solver routines need to be available. In the TRACE project, implicit schemes applied to the DGSEM method have not been researched in depth yet, which motivates the goal of this project.

2.3. Linear and Nonlinear Solving Methods for Implicit Time Integration on Unsteady Turbulent Flows

The efficiency of the linear and nonlinear solving methods is an important aspect in developing an implicit time-integration method that can compete with explicit schemes. In this section, numerical methods reported in the literature for solving the nonlinear and subsequent linear systems of equations are reviewed and discussed.

2.3.1. Nonlinear Solver Methods

For solving the nonlinear system of equations, one method is to use Picard iterations to iteratively solve the system. Picard iterations are globally convergent with a linear convergence rate, and they are typically used to implicitly solve the steady incompressible Navier-Stokes equations [62]. However, if the initial guess $\boldsymbol{u}^0$ is sufficiently close, the iterative Newton method provides quadratic convergence, thus being faster than Picard iterations [63]. For unsteady problems, the initial guess can be assumed to be close to the solution. Therefore, Newton's method and its variants are more popular for unsteady implicit time advancements. Newton's method has been used for implicit time integration for the DGSEM in several ways. The first is the inexact Newton method, which is a modified version of Newton's method commonly used when tackling large systems of equations [56, 64]. The inexact Newton method allows for inexact linear solves at each Newton step that, specifically for large systems of equations, do not reduce the quadratic convergence of the Newton method. Furthermore, the linear solver tolerance can be reduced over the subsequent Newton steps as the solution is reached [65]. The inexact Newton method paired with an adaptive forcing term, such as proposed by [66], has been found to produce a flexible inexact Newton method that prevents over-solving of the system [56, 66].

Newton's method also allows for relaxation. Namely, the dual time-stepping method (DTS) is a common way of adding a relaxation to the Newton solver. This is also referred to as pseudo-transient continuation [67]. With DTS, the nonlinear problem is treated as a steady-state problem in pseudo-time. This results in time integration in pseudo-time. Commonly, the backward Euler method is used to integrate in pseudo-time, as accuracy is not required in pseudo-time. The method becomes the standard Newton method if the dual time-step size $\tau \to \infty$. Dual time-stepping has been found to be useful when the standard Newton method does not converge. Therefore, the method has been found to be more popular for steady flows than unsteady flows, because for unsteady flows the initial guess is much closer to the solution [68]. In [53], DTS is utilised for implicit time integration with a DG scheme, but the method was found to be slower than its explicit counterpart. Possibly, DTS was not required for the test case considered, and a Newton method would have produced better performance. Combining the DTS with the inexact Newton method is also possible.

2.3.2. Linear Solver Methods

For the linear solver scheme, two types of solvers have been commonly observed to be used for the implicit solving of high-order DG methods.

The first linear solver that is often used is the LU-SGS method. Although the LU-SGS scheme is a linear solver, it is often referred to as a complete implicit solver because it is combined with a dual time stepping approach [69, 70]. Therefore, in practice, the nonlinear solver is a relaxed Newton method arising from the dual time stepping formulation, and LU-SGS is used as the linear solver. Usually, per pseudo time step, only one LU-SGS solve, consisting of a forward and backward sweep, is performed. This solve often reduces the linear residual sufficiently to advance the nonlinear pseudo time step. This explains why the method is often referred to as a nonlinear solver, as it is an iterative method in the nonlinear pseudo-time. More LU-SGS iterations can be used to drop the linear residual further. Still, because of the non-quadratic convergence rate, the LU-SGS solver is most efficient at dropping the residual fast in a couple of iterations rather than being able to perform very tight linear solves. The method has a low memory footprint compared to fully implicit matrix-based solvers (e.g. GMRES) because only diagonal matrices are required to be stored [71]. Due to this, the method is sometimes also called matrix-free, as no global Jacobian matrix is required. The method has been found to be an efficient implicit solver for high-order DG methods [71, 72, 73], achieving lower CPU hours than the explicit reference simulations. The linear solving method has also been successfully applied to GPUs [74]. The downside of the LU-SGS method is that it is less robust in general and also has limited convergence

rates compared to the Krylov subspace methods described next. This is why it is combined with a dual time-stepping approach. When performing unsteady implicit time integration, the linear solve in general is not required to perform very tight solves, but the system considered can be ill-conditioned, thus requiring a more robust solver. Moreover, the LU-SGS method has shown limited robustness for stiff problems, and importantly, cannot be formulated in a fully matrix-free manner. Since the present work focuses on matrix-free implicit methods, LU-SGS is not considered further in this report.

The second linear solving method found to be commonly used is the generalised minimal residual (GMRES) method [75]. The GMRES method belongs to the class of Krylov subspace methods and will be explained in more detail in Subsection 3.4.2. In short, the method works by constructing a growing sequence of basis vectors that span a Krylov subspace of the image of the Jacobian operator in which he residual of the equation is minimised. The subspace is built by repeated application of the Jacobian operator onto the residual vector, which is then orthogonalized against the existing basis vectors. A very detailed description of the GMRES method is given in [76]. The GMRES method is a more robust solver because the solution is improved at every iteration by minimising the residual over an expanding Krylov subspace. While ill-conditioning can still slow convergence, appropriate preconditioning can mitigate this effect. In contrast to LU-SGS, GMRES does not oscillate and only stalls in the worst case. The GMRES method formally requires an explicit formulation of the linearised system operator, more commonly referred to as the Jacobian. However, in GMRES, the Jacobian is only used in a matrix-vector product manner (i.e. the Jacobian operator is repeatedly multiplied by the residual vector). Therefore, construction and storage of the matrix can be avoided as only the resulting matrix-vector product is used in the method. Thus, if an operator can be defined that performs the same operation as the Jacobian matrix-vector product, the GMRES method can be made completely matrix-free. The matrix-free GMRES (MF-GMRES) coupled with an outer Newton solver is also commonly referred to as a Jacobian-free Newton-Krylov (JFNK) [5]. Most implementations found in the literature of high-order DG schemes paired with GMRES make use of the matrix-free formulation [56, 77, 53, 57]. The other option is to analytically linearise the equations and explicitly fill a matrix, as for instance shown in [78], where the matrix is explicitly needed to perform the static condensation. However, these analytical formulations often result in an approximate linearisation of the system because case-specific terms are not included (such as the diffusive term linearisation in [78]).

Matrix-free differentiation methods The matrix-vector product can be evaluated in a matrix-free manner in several ways. The first method is to use a first-order finite-difference (FD) approximation as described in [5]. The FD method is a common method applied to evaluate the directional derivative in the JFNK. The method was, for instance, used in [56] to perform implicit time integration on the DGSEM method applied to the unsteady compressible Navier-Stokes equations. Similarly, the method was also applied in other related use cases found in literature [77, 53, 57]. In both [79] and [77], the FD method is applied to a matrix-free GMRES algorithm running on GPU clusters. The reason the method is favoured is due to its low implementation costs and efficiency. Because the base evaluation can be stored, each matrix-vector product requires only one evaluation of the residual function. The commonly used forward finite-difference method does have a significant drawback. Namely, the result is a first-order approximation. This means that truncation errors are present. Additionally, if the perturbation size is not chosen correctly, round-off errors could also significantly pollute the derivative approximation. As stated by [80], the inexactness of finite-difference approximations can have a considerable negative impact on the convergence of the linear solve for SDIRK methods. The truncation error can be reduced by using finite-difference approximations of higher order. However, such methods, such as the second-order central-difference method, are generally not considered due to the increased evaluation counts that significantly increase the cost of the method [5].

Another method to evaluate the matrix-vector product is by the use of automatic differentiation (AD). AD methods can provide *exact* derivatives up to floating point precision, in contrast to the finite-difference approximation. Further explanation of the method will be given in Subsection 3.4.4. AD has been an active area of research for adjoint solvers, which make use of *reverse* AD [81, 82, 83]. In matrix-free GMRES, the so-called *forward* AD method is used, as only a forward propagation of derivative data is required. For JFNK, the use of AD has been less popular. The reason for this can be explained by the fact that implementing AD in a code base requires a significant amount of implementation effort. Comparing this to the easy implementation of the FD method and the fact that much of the

previous literature suggests that the perturbation size can be chosen such that round-off error effects are minimal, the motivation behind selecting FD is understood. Nevertheless, the fact that AD methods are capable of providing exact derivatives should mean that they can overcome convergence issues that FD-based JFNK methods might encounter. Next to this, AD libraries such as CodiPack [84] have been extensively optimised for the usage of adjoint solvers such that they can be used in large-scale scientific and industrial simulations [82].

2.3.3. Preconditioning Methods

The GMRES method is often paired with a preconditioning method, which means that the linear system to be solved is premultiplied by an approximate inverse of the linear operator. This, in turn, reduces the conditioning number of the system to be solved, making the GMRES solver more efficient. Regarding preconditioning for matrix-free GMRES methods, using a matrix as a preconditioner indeed means that the method technically is not matrix-free any more. Matrix-based as well as matrix-free preconditioning approaches have been used for JFNK applied to DG.

In [53], block-Jacobi, element block-Jacobi, and p-multigrid preconditioners are applied to a matrix-free JFNK solver. The study concludes that the p-multigrid preconditioner offers the best performance in terms of wall time and memory consumption. The advantage of p-multigrid methods is that they naturally suit the DG method rather than h-multigrid methods [85]. They also explore the lagging of the preconditioner over multiple time steps, resulting in further performance increase. Multigrid preconditioners based on simplified finite-volume discretisations on the sub-cell level have also been explored for the LGL-DGSEM method [86, 87]. Overall, the literature has suggested that multigrid preconditioners are most effective for DG operators. At the same time, these preconditioning methods require significant implementation effort.

In [56], an analytically derived element block-Jacobi preconditioner is used. Several different inversion strategies are compared, and in practical use cases, it is concluded that the ILU0 inversion method is most efficient. Here, it is also noted that when small time-steps are used (that are still well above the explicit time-step constraint), freezing of the preconditioner is proven to be an effective strategy. In [64], it is argued that multigrid preconditioners, although promising, often lead to non-optimal preconditioners due to the complexity of the methods. Furthermore, in this study, a new reduced off-diagonal block order SGS-type (ROBO-SGS) preconditioner is compared against other preconditioning methods for a modal DG discretisation. They demonstrate that the ROBO-SGS preconditioner outperforms standard block-Jacobi or ILU0 preconditioners for the considered test case. However, as noted in [64], the ROBO-SGS method would require modification to be applied with the DGSEM.

2.4. Outcomes of the Literature Review

In this section, the main outcomes of the literature review are summarised.

Firstly, although all three classes of spatial discretisation methods allow for high-order (third order or higher) formulations, it was found that specifically discontinuous Galerkin methods have been shown to be most suitable for high-order spatial discretisation of fluid mechanics problems. This is because the method allows for an inner-element high-order formulation while keeping the scheme compact with the discontinuous formulation at the element interfaces. Moreover, the discontinuity allows for the usage of numerical flux schemes traditionally used in the FV method. Out of the DG methods, the DGSEM has been found to be particularly attractive due to the many numerical simplifications of the method that make the scheme significantly more efficient compared to other DG formulations. Next to this, recent work on the DGSEM has introduced efficient split-form methods to stabilise the Galerkin approach that otherwise can exhibit instabilities when the solution is under-resolved.

Going over the time-integration schemes paired with the high-order DG methods, literature suggests that the use of high-order time-integration schemes is most suited for implicit DGSEM of unsteady turbulent flows. Within this space, the diagonal implicit Runge-Kutta methods (ESDIRK and DIRK) as well as their linearised version (Rosenbrock-type Runge-Kutta methods) appear to be most attractive. This outcome follows logically from the fact that the leverage of the implicit scheme is to increase the time step size, and that with an increase in accuracy of the time-integration method, this time step size can be further increased without introducing additional dissipation. The fourth-order Runge-Kutta

schemes have been found to be the most efficient, as for higher orders of accuracy, the increase in Runge-Kutta stages outweighs the allowed increase in time step size. Furthermore, literature also showed that IMEX schemes could offer superior performance to both implicit and explicit schemes when applied to high-order DG scale-resolving simulations. However, as mentioned before, well-designed and optimised implicit and explicit schemes are required for IMEX. Therefore, this study will focus solely on the usage of solving methods to perform fully implicit time-integration schemes in TRACE.

Summarising the literature review regarding the implicit solution methods. It was observed that, in general, two approaches of implicit solving were applied to high-order DG methods. The first being the usage of a dual time stepping method, which can be seen as a relaxed Newton solver, paired with an LU-SGS solver that usually only performs one iteration (forward-backwards sweep) per pseudo time step. Although more linear iterations can also be used if necessary. The second approach that was found in literature is to pair a Newton solver with a matrix-free GMRES linear solver. This is often referred to as a Jacobian-free Newton Krylov (JFNK) method. To speed up the Newton solver, an inexact Newton solver is often used. The usage of the GMRES linear solver results in a more robust scheme that also does not require a pseudo time approach to relax the Newton solve. Furthermore, GMRES itself can work entirely matrix-free. The most common matrix-vector product evaluation method is with a finite-difference approximation, although forward AD could potentially overcome the inaccuracies of approximating the matrix-vector product with a first-order finite-difference approximation. In this report, the JFNK approach with an inexact Newton method will be considered. For the matrix-free GMRES, both the FD and forward AD techniques will be considered, as a direct comparison between the two methods was not found in the literature reviewed.

The literature review on preconditioning methods for matrix-free GMRES has shown that many preconditioning methods have been explored. Although multigrid preconditioners are found to be most suitable for the high-order DG spatial discretisation, the fact that unsteady phenomena are of interest means that small time-step sizes are solved by the implicit solver. This reduces the overall requirements of the preconditioning methods, and thus simpler preconditioning methods have also been found to be efficient. To limit the scope of this project, it was therefore decided to only consider well-established approaches such as ILU0 or block-Jacobi preconditioners.

3

Numerics and Theoretical Background

This chapter will describe the discretisation of a set of governing equations using the Discontinuous Galerkin Spectral Element Method (DGSEM). As part of this project, a 1D prototype solver was made from scratch in order to create a flexible and simple test environment, as well as to completely cover the basics of a DGSEM implementation. In this chapter, the DGSEM method will be derived for the three-dimensional case, and simplifications will be made to connect it to the 1D DGSEM method. Firstly, in Section 3.1, the governing equations considered will be presented. In Section 3.2, the spatial discretisation of the governing equations using the DGSEM will then be explained. After this, the semi-discrete formulation is fully discretised with a time discretisation, as explained in Section 3.3. Finally, the methods to solve the fully discretised nonlinear system in an implicit manner in time will be explained in Section 3.4.

3.1. Governing Equations

In this study, the focus is on solving hyperbolic-parabolic partial differential equations, such as the compressible Navier-Stokes equations. A generalised system of hyperbolic-parabolic conservation laws can be written as

$$\frac{\partial \boldsymbol{u}}{\partial t} + \nabla_x \cdot \mathbf{F}^a(\boldsymbol{u}) - \nabla_x \cdot \mathbf{F}^v(\boldsymbol{u}, \nabla_x \boldsymbol{u}) = 0, \tag{3.1}$$

$$\frac{\partial \boldsymbol{u}}{\partial t} + \nabla_x \cdot \mathbf{F}(\boldsymbol{u}, \nabla_x \boldsymbol{u}) = 0. \tag{3.2}$$

In this set of equations, the advective flux $\boldsymbol{F}^a(u)$ represents the hyperbolic terms of the equations, which are related to transport phenomena such as wave propagation. The diffusive flux $\boldsymbol{F}^v$ represents the dissipative effects such as the viscous terms in the Navier-Stokes equations. The total, or the so-called physical, flux is denoted with $\boldsymbol{F}$ and is a combination of the two flux terms. Nonetheless, the terms are typically treated separately.

Compressible Navier-Stokes equations In the unsteady compressible Navier-Stokes equations the solution state vector $\boldsymbol{u}$ represents the conservative variables $\boldsymbol{u} = (\rho, \rho u_1, \rho u_2, \rho u_3, \rho E)^T$. Where ρ is the fluid density, (u_1, u_2, u_3) are the velocity components in Cartesian space and E is the total specific energy. With these state variables, the flux functions are given by

$$\boldsymbol{F}_i^a(\boldsymbol{u}) = \begin{pmatrix} \rho u_i \\ \rho u_1 u_i + \delta_{1,i} p \\ \rho u_2 u_i + \delta_{2,i} p \\ \rho u_3 u_i + \delta_{3,i} p \\ \rho H u_i \end{pmatrix}, \qquad \boldsymbol{F}_i^v(\boldsymbol{u}, \nabla_x \boldsymbol{u}) = \begin{pmatrix} 0 \\ \tau_{1,i} \\ \tau_{2,i} \\ \tau_{3,i} \\ \sum_{j=1}^{3} \tau_{i,j} u_j + \lambda \frac{\partial T}{\partial x_i} \end{pmatrix}, \qquad i = 1, 2, 3. \tag{3.3}$$

In the formulation of the flux functions, $\delta_{i,j}$ denotes the Kronecker delta product, $\tau_{i,j}$ denotes the viscous stress tensor, and p denotes the pressure. The pressure in this case is computed using the ideal gas law

$$p = \rho RT = (\gamma - 1)\rho \left[E - \frac{1}{2}(\boldsymbol{u} \cdot \boldsymbol{u}) \right]. \tag{3.4}$$

For all test cases considered in this thesis, a specific heat ratio of $\gamma = 1.4$ [-] is assumed. Next to this, the gas constant is also assumed to be $R = 287.15$ [J $K^{-1}kg^{-1}$]. The advective flux uses the total enthalpy H, which is given by $H = E + \frac{p}{\rho}$. The stress tensor $\boldsymbol{\tau}$ is defined as

$$\tau_{i,j} = \mu \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} - \sum_{k=1}^{3} \frac{2}{3} \frac{\partial u_k}{\partial x_k} \delta_{i,j} \right), \qquad i, j = 1, 2, 3. \tag{3.5}$$

In the diffusive flux, a heat flux term $\lambda \frac{\partial T}{\partial x_i}$ is added to the energy equation. Here λ denotes the conductivity, which is computed assuming a constant Prandtl number Pr = 0.72 [-] in all test cases considered in this study

$$\mathrm{Pr} = \frac{c_p \mu}{\lambda}. \tag{3.6}$$

Where $c_p = \frac{R\gamma}{\gamma - 1}$ is the specific heat at constant pressure.

In this study, the inviscid compressible Euler equations are also used. When this system of equations is considered, the diffusive flux is ignored and only the advective fluxes are used. In other words, thermal and viscous effects are ignored in the case of the inviscid compressible Euler equations.

3.2. Discontinuous Galerkin Spectral Element Method

In this section, the spatial discretisation using the DGSEM will be explained. The derivation will follow the three-dimensional case, but simplifications will be given to obtain the one-dimensional form of the derived equations.

3.2.1. Reference Space and Transformation

The first step is to divide the physical domain $\Omega \subset \mathbb{R}^d$ into shape-regular meshes $\mathcal{T}_h = \{K\}$ consisting of non-overlapping elements K where h denotes a piecewise constant mesh function. Because for the DGSEM method the tensor-product property is utilised, the element shapes are restricted in 3D to hexahedral, and in 2D to quadrilateral. Each element is transformed into a reference space $K_{\mathrm{ref}} = [-1, 1]^d$ with coordinates $\boldsymbol{\xi} \in \mathbb{R}^d$. This allows for a generalisation of numerical operations on the different elements, saving computational storage and effort. In order to do this, the governing equations themselves have to be transformed from the physical space to the reference element. For this, a mapping M^k for every element $K \in \mathcal{T}_h$ is defined such that $\mathbf{x} = M^K(\boldsymbol{\xi})$. Where $\mathbf{x} = (x_1, ..., x_d)$ is the physical coordinate system and $\boldsymbol{\xi} = (\xi_1, ..., \xi_d)$ is the reference coordinate system.

In order to transform the governing equations onto the reference element, the gradient and divergence operators have to be transformed. The gradient of a function $g(\mathbf{x})$ defined on the reference space can be computed by means of the chain rule. In the three-dimensional case, this results in

$$\nabla_{\xi} g = \mathbf{J} \nabla_{\mathbf{x}} g, \tag{3.7}$$

Where $\mathbf{J}$ is the Jacobian matrix of the mapping $M^K(\boldsymbol{\xi})$. However, the aim of the reference space transformation is to express physical space derivatives in terms of reference space derivatives ($\nabla_x g = \boldsymbol{J}^{-1} \nabla_{\xi} g$). Therefore, the inverse $\mathbf{J}^{-1}$ of the Jacobian matrix is needed. Following [88], two types of basis vectors are introduced to represent directions in physical space. Firstly, the *covariant* basis $\mathbf{a}_i$ is defined as

$$\mathbf{a}_i = \frac{\partial \mathbf{x}}{\partial \xi_i}, \qquad i = 1, 2, 3. \tag{3.8}$$

The covariant basis vectors can be computed from the known mapping function M^k and are tangent to the coordinate lines. The second type of basis vector is the *contravariant* basis vector, which are normal to the coordinate lines and are defined as

$$\mathbf{a}^i = \nabla \xi_i, \qquad i = 1, 2, 3. \tag{3.9}$$

From (3.9) it appears that the inverse of the transformation is required to obtain the contravariant basis vectors. However, as shown by [88], the contravariant basis vectors can be defined by the covariant basis as

$$J\mathbf{a}^i = \mathbf{a}_j \times \mathbf{a}_k, \qquad i = 1, 2, 3;\ (i, j, k) \text{ cyclic}. \tag{3.10}$$

Cyclic indicates that the indices take the cyclic permutations of (1,2,3), namely (1,2,3), (2,3,1), and (3,1,2). Here, J is the determinant of the Jacobian matrix $\mathbf{J}$ given by

$$J = \mathbf{a}_i \cdot \left(\mathbf{a}_j \times \mathbf{a}_k\right), \quad (i, j, k) \text{ cyclic}. \tag{3.11}$$

Following the derivation by [88], the conservative form of the gradient of a function g and divergence of a vector-valued function $\mathbf{g}$ are given by

$$\nabla_{\mathbf{x}} g = \frac{1}{J} \sum_{i=1}^{3} \frac{\partial}{\partial \xi_i} \left(J\mathbf{a}^i g\right), \tag{3.12}$$

$$\nabla_{\mathbf{x}} \cdot \mathbf{g} = \frac{1}{J} \sum_{i=1}^{3} \frac{\partial}{\partial \xi_i} \left(J\mathbf{a}^i \cdot \mathbf{g}\right). \tag{3.13}$$

With (3.13), the governing equations (3.2) can be transformed to the reference space. A contravariant flux $\boldsymbol{\mathcal{F}}$ is defined based on the covariant flux $\boldsymbol{F}$ as

$$\mathcal{F}^i = J\mathbf{a}^i \cdot \boldsymbol{F}, \qquad i = 1, 2, 3. \tag{3.14}$$

With this, the divergence form with respect to the reference coordinates can be written

$$\nabla_{\boldsymbol{x}} \cdot \boldsymbol{F} = \frac{1}{J} \nabla_{\boldsymbol{\xi}} \cdot \boldsymbol{\mathcal{F}}. \tag{3.15}$$

And thus the governing equations (3.2) are transformed to the reference space, resulting in

$$J\boldsymbol{u}_t + \nabla_{\boldsymbol{\xi}} \cdot \boldsymbol{\mathcal{F}}(\boldsymbol{u}, \nabla_{\mathbf{x}} \boldsymbol{u}) = \mathbf{0}. \tag{3.16}$$

(3.16) Still contains the gradient $\nabla_{\boldsymbol{x}} u$ that is not transformed. The method used to treat second-order terms, commonly found in the diffusive flux operator, is explained in Subsection 3.2.4.

Free-stream preservation of discrete metric terms

As explained by [89], the metric terms satisfy three metric identities on a fixed mesh

$$\sum_{i=1}^{3} \frac{\partial \left(J a_n^i\right)}{\partial \xi_i} = 0, \quad n = 1, 2, 3. \tag{3.17}$$

where $J a_n^i = J\mathbf{a}^i \cdot x_n$ is the component of $J\mathbf{a}^i$ in the n-th physical coordinate direction. These identities can be explained by the fact that if a flux is constant in space, the divergence is zero, and the solution does not change in time. This has to hold under any transformation of the equations. In the context of fluid dynamics, the identities can thus also be understood as a form of *free-stream preservation* such that a uniform flow remains uniform for all time. Consequently, the discrete metric terms that will be used have to satisfy the metric identities

$$\sum_{i=1}^{3} \frac{\partial I^N (J a_n^i)}{\partial \xi_i} = 0, \qquad n = 1, 2, 3, \tag{3.18}$$

where $I^N(g)$ represents the polynomial interpolation of a function g. Following [89], the discrete metric identities are adhered to if the discrete metric terms are computed as

$$Ja_n^i = -\hat{x}_i \cdot \nabla_\xi \times \left[I^N(x_l \nabla_\xi x_m) \right], \qquad i = 1,2,3;\ n = 1,2,3;\ (n,m,l) \text{ cyclic}, \tag{3.19}$$

where $\hat{x}_i = (\delta_{i,1}, \delta_{i,2}, \delta_{i,3})^T$. An interesting alternative way the metric terms can be constructed is through the use of mimetic projections, as shown by Bach et al. [90].

Transformation in one-dimensional case

For the 1D-DGSEM method also considered in this project, the transformation from reference space to physical coordinate space is simplified compared to the two- and three-dimensional cases. Firstly, considering the physical coordinate x, each element is mapped onto the reference element $K_{\text{ref}} = [-1, 1]$ by

$$x(\xi) = x_{k-1} + \Delta x_k \frac{\xi + 1}{2}. \tag{3.20}$$

where $\Delta x_k = x_k - x_{k-1}$ is the width of the element k. As the Jacobian only has one component, its inverse is easily calculated, and we can write

$$J(\xi) = \frac{\partial x}{\partial \xi} = \frac{\Delta x}{2}, \tag{3.21}$$

$$\nabla_\xi g = J \nabla_x g \rightarrow \nabla_x g = \frac{2}{\Delta x} \nabla_\xi g. \tag{3.22}$$

3.2.2. Spatial Discretisation

In this section, the application of the DGSEM approximation to the governing equations will be explained.

Polynomial approximation and numerical quadrature

To construct the solution, it is approximated by polynomials of degree N on every element in the reference space, i.e. $\boldsymbol{u} \in \mathbb{P}^N(K_{\text{ref}})$. To interpolate the polynomials, a set of $(N+1)$ interpolation nodes is defined as

$$\{\xi_i\}_{i=0}^N \subset [-1, 1], \tag{3.23}$$

on the reference element. In the DGSEM, the basis functions in one direction are defined by the one-dimensional Lagrange polynomials l_i of degree N given by

$$l_i(\xi) = \prod_{\substack{k=0 \\ k \neq i}}^{N} \frac{\xi - \xi_k}{\xi_i - \xi_k}, \qquad i = 0, .., N. \tag{3.24}$$

A useful property of the Lagrange polynomials is that

$$l_i(\xi_j) = \delta_{i,j}, \qquad i, j = 0, ..., N. \tag{3.25}$$

Meaning that the polynomial approximation at the i-th interpolation node is fully described by the i-th interpolation coefficient. Due to this, a scalar function f can be approximated using the nodal approach as

$$f(\xi) \approx \sum_{i=0}^{N} f_i l_i(\xi), \tag{3.26}$$

where f_i are the nodal coefficients, which store the value of the solution at ξ_i as $f_i = f(\xi_i)$ due to the Lagrange property. This eliminates the need to interpolate the solution using the interpolation modes, thereby reducing computational complexity and cost. To use this approach in multiple dimensions, the solution is approximated with a tensor-product fashion construction of Lagrange polynomials

$$\boldsymbol{u}(\boldsymbol{\xi}, t)|_K \approx \boldsymbol{u}_h(\boldsymbol{\xi}, t) = \sum_{i,j,k=0}^{N} \boldsymbol{u}_{i,j,k}(t) l_i(\xi) l_j(\eta) l_k(\zeta). \tag{3.27}$$

Where $\boldsymbol{u}_{i,j,k}$ again represents the nodal coefficient of the solution vector in the element K. One disadvantage of the tensor-product polynomial approximation is that the method is restricted to hexahedral elements in three dimensions and quadrilateral elements in two dimensions. Whereas the high locality of the method implies that it would also be suitable to use on (unstructured) meshes with differently shaped elements (i.e. triangular elements).

Weak formulation and discontinuous Galerkin approach

With the previous methods defined, the governing equations can be discretised. Following the DG method, the transformed governing equations given in (3.16) are multiplied by a test function $\phi = \phi(\boldsymbol{\xi}) \in \mathbb{R}$ and integrated over the reference element K.

$$\int_{K_{\text{ref}}} \left(J\boldsymbol{u}_t + \nabla_\xi \cdot \boldsymbol{\mathcal{F}} \right) \phi d\boldsymbol{\xi} = 0. \tag{3.28}$$

This results in the variational form of the equations in the strong form. Strong form here meaning the solution u is assumed to be twice differentiable because $\boldsymbol{\mathcal{F}} = \boldsymbol{\mathcal{F}}(\boldsymbol{u}, \nabla_\xi \boldsymbol{u})$. To reduce the differentiability requirements of the solution, integration by parts is applied, resulting in the so-called weak form

$$\underbrace{\int_{K_{\text{ref}}} J\frac{\partial \boldsymbol{u}}{\partial t}\phi d\boldsymbol{\xi}}_{\text{time integral}} - \underbrace{\int_{K_{\text{ref}}} \boldsymbol{\mathcal{F}} \cdot \nabla_\xi \phi d\boldsymbol{\xi}}_{\text{volume integral}} + \underbrace{\int_{\partial K_{\text{ref}}} (\boldsymbol{\mathcal{F}} \cdot \boldsymbol{n})\, \phi dS}_{\text{surface integral}} = 0. \tag{3.29}$$

For the surface integral, $\boldsymbol{n}$ denotes the outward-pointing surface normal vector and dS is the surface differential of the reference element. As described in Section 2.1, the DGSEM uses a Galerkin approach with Lagrange polynomials sampled at the roots of the Legendre polynomials. With the Lagrange polynomials, the assumed solution and test function space thus becomes $\phi, u \in \mathbb{P}^N(K_{\text{ref}})$. As a three-dimensional element is considered, the solution and test function spaces are constructed in a tensor-product fashion. The solution coefficients, here also referred to as u, then become the nodal values of the solution at the quadrature/interpolation nodes. Using these choices for ϕ and u and inserting them in (3.29) we obtain

$$\begin{aligned}
&\int_{K_{\text{ref}}} J(\xi,\eta,\zeta) \left(\sum_{m,n,o=0}^{N} \frac{\partial \boldsymbol{u}_{m,n,o}}{\partial t} l_m(\xi) l_n(\eta) l_o(\zeta) \right) l_i(\xi) l_j(\eta) l_k(\zeta) d\boldsymbol{\xi} \\
&+ \int_{\partial K_{\text{ref}}} \underbrace{(\boldsymbol{\mathcal{F}} \cdot \boldsymbol{n})}_{\boldsymbol{\mathcal{H}}(\xi,\eta,\zeta,\boldsymbol{n})} l_i(\xi) l_j(\eta) l_k(\zeta) dS \\
&- \int_{K_{\text{ref}}} \left(\sum_{m,n,o=0}^{N} \boldsymbol{\mathcal{F}}_{m,n,o} l_m(\xi) l_n(\eta) l_o(\zeta) \right) \cdot \nabla_\xi \left[l_i(\xi) l_j(\eta) l_k(\zeta) \right] d\boldsymbol{\xi} = 0, \qquad i,j,k = 0,\dots,N.
\end{aligned} \tag{3.30}$$

For the surface integral in (3.30), because the solution is allowed to be discontinuous across element interfaces, the flux $\boldsymbol{\mathcal{F}}$ is not uniquely defined and thus a numerical flux function $\boldsymbol{\mathcal{H}}(\boldsymbol{\xi}, \boldsymbol{\eta}, \boldsymbol{\zeta}, \boldsymbol{n})$ needs to be used. This flux function can be split up into an advective term $\boldsymbol{\mathcal{H}_a}$ and a viscous term $\boldsymbol{\mathcal{H}_v}$ such that

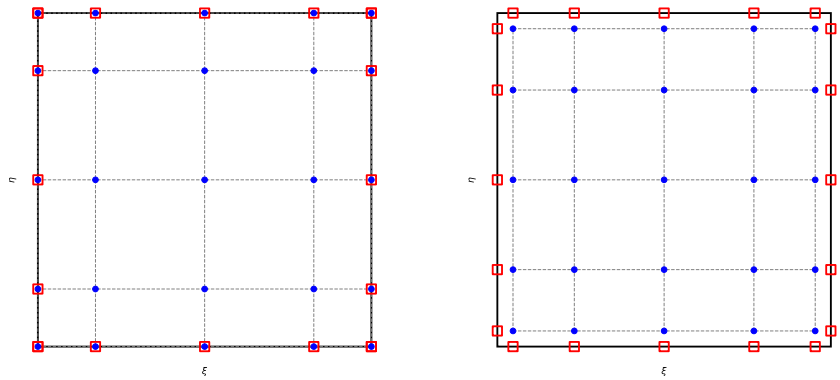
$$\boldsymbol{\mathcal{H}}(\xi,\eta,\zeta,\boldsymbol{n}) = \boldsymbol{\mathcal{H}_a}(\boldsymbol{u}^-, \boldsymbol{u}^+, \boldsymbol{n}) - \boldsymbol{\mathcal{H}}_v(\boldsymbol{u}^-, \nabla_x \boldsymbol{u}^-, \boldsymbol{u}^+, \nabla_x \boldsymbol{u}^+, \boldsymbol{n}). \tag{3.31}$$

As can be seen in (3.31), the viscous flux $\boldsymbol{\mathcal{H}}_v$ requires the gradient on the inner and outer elements $\nabla_x u^\pm$, which has not been formulated yet. The choice of $\boldsymbol{\mathcal{H}}_v$ will be explained in Subsection 3.2.4.

Numerical integration

To compute the integrals defined in the previous section, Legendre-Gauss or Legendre-Gauss-Lobatto quadrature can be used. Both methods work by defining a set of (M + 1) nodes η_i and weights ω_i $\{\eta_i, \omega_i\}_{i=0}^{M}$ on the reference space interval $[-1, 1]$ with which the integral in one dimension can be approximated as

$$\int_{-1}^{1} f(\eta) d\eta \approx \sum_{i=0}^{M} f(\eta_i)\omega_i. \tag{3.32}$$



(a) LGL-DGSEM **(b)** LG-DGSEM

Figure 3.1: Visualisation of the computational domain in reference space for an approximation order of $p = 4$ for LGL- and LG-DGSEM. The blue circles indicate the collocated interpolation and quadrature nodes, the red squares indicate the boundary quadrature nodes used by the Riemann solver.

In this project, it is chosen to have the same number of integration nodes as interpolation nodes ($N = M$) and to collocate the quadrature and interpolation nodes, which decreases computational effort while maintaining spectral accuracy [91]. For the LG nodes, integrals up to a degree of $2p+1$ can be integrated exactly, where p is the polynomial degree. In contrast, the LGL nodes can only integrate integrals up to a degree of $2p-1$. The main difference between the LG and LGL integration nodes is that the LG nodes are not present on the element boundary. This can be seen in Figure 3.1, where the quadrature and solution points together with surface points are shown for a two-dimensional grid and $p = 3$ for the LG and LGL nodes. As will be explained later, the value of the solution on the element boundary is required for surface flux evaluation. Due to the LG nodes not being readily available at the surface, the surface flux evaluation is dependent on all other inner element nodes. This affects the simplifications that can be made for the boundary integral using the LG nodes. In turn, there is also a difference in the Jacobian matrix sparsity pattern between LG-DGSEM and LGL-DGSEM as shown in [78]. With the Gauss quadrature rule, the weak form integral formulation shown in (3.30) can be simplified. Firstly, we start with the time derivative integral

$$
\begin{aligned}
&\int_{K_{\text{ref}}} J(\xi,\eta,\zeta)\left(\sum_{m,n,o=0}^{N}\frac{\partial \boldsymbol{u}_{m,n,o}}{\partial t} l_m(\xi)l_n(\eta)l_o(\zeta)\right) l_i(\xi)l_j(\eta)l_k(\zeta)d\boldsymbol{\xi} \\
&\approx \sum_{\mu,\nu,\omega=0}^{N} \omega_\mu\omega_\nu\omega_\omega J(\xi_\mu,\eta_\nu,\zeta_\omega)\left(\sum_{m,n,o=0}^{N}\frac{\partial \boldsymbol{u}_{m,n,o}}{\partial t}\underbrace{l_m(\xi_\mu)}_{\delta_{m,\mu}}\underbrace{l_n(\eta_\nu)}_{\delta_{n,\nu}}\underbrace{l_o(\zeta_\omega)}_{\delta_{o,\omega}}\right)\underbrace{l_i(\xi_\mu)}_{\delta_{i,\mu}}\underbrace{l_j(\eta_\nu)}_{\delta_{j,\nu}}\underbrace{l_k(\zeta_\omega)}_{\delta_{k,\omega}} d\boldsymbol{\xi} \\
&= w_i w_j w_k J\left(\xi_i,\eta_j,\zeta_k\right)\frac{\partial u_{i,j,k}}{\partial t}.
\end{aligned}
\tag{3.33}
$$

As can be seen, due to the collocation of the quadrature and interpolation nodes as well as the Lagrange property, the sums vanish. This results in a diagonal mass matrix which can be easily inverted.

For the volume integral, a similar approach can be taken

$$\begin{aligned}
&\int_{K_{\text{ref}}} \left(\sum_{m,n,o=0}^{N} \boldsymbol{\mathcal{F}}_{m,n,o} l_m(\xi) l_n(\eta) l_o(\zeta) \right) \cdot \nabla_\xi \left[l_i(\xi) l_j(\eta) l_k(\zeta) \right] d\boldsymbol{\xi} \\
&\approx \sum_{\mu,\nu,\omega=0}^{N} w_\mu w_\nu w_\omega \left(\sum_{m,n,o=0}^{N} \boldsymbol{\mathcal{F}}_{m,n,o} \underbrace{l_m(\xi_\mu)}_{\delta_{m,\mu}} \underbrace{l_n(\xi_\nu)}_{\delta_{n,\nu}} \underbrace{l_o(\xi_\omega)}_{\delta_{o,\omega}} \right) \cdot \nabla_\xi \left[l_i(\xi_\mu) l_j(\eta_\nu) l_k(\zeta_\omega) \right] \\
&= \sum_{m,n,o=0}^{N} w_m w_n w_o \boldsymbol{\mathcal{F}}_{m,n,o} \cdot \nabla_\xi \left[l_i(\xi_m) l_j(\eta_n) l_k(\zeta_o) \right].
\end{aligned} \tag{3.34}$$

To further simplify (3.34), the tensor-product fashion construction of the basis test functions can be used. This results in

$$\begin{aligned}
&\sum_{m,n,o=0}^{N} w_m w_n w_o \boldsymbol{\mathcal{F}}_{m,n,o} \cdot \nabla_\xi \left[l_i(\xi_m) l_j(\eta_n) l_k(\zeta_o) \right] \\
&= \sum_{m,n,o=0}^{N} w_m w_n w_o \left((\boldsymbol{\mathcal{F}}_1)_{m,n,o} \frac{\partial l_i(\xi_m)}{\partial \xi} \underbrace{l_j(\eta_n)}_{\delta_{j,n}} \underbrace{l_k(\zeta_o)}_{\delta_{k,o}} \right. \\
&\left. + (\boldsymbol{\mathcal{F}}_2)_{m,n,o} \frac{\partial l_j(\eta_n)}{\partial \eta} \underbrace{l_i(\xi_m)}_{\delta_{i,m}} \underbrace{l_k(\zeta_o)}_{\delta_{k,o}} + (\boldsymbol{\mathcal{F}}_3)_{m,n,o} \frac{\partial l_k(\zeta_o)}{\partial \zeta} \underbrace{l_i(\xi_m)}_{\delta_{i,m}} \underbrace{l_j(\eta_n)}_{\delta_{j,n}} \right) \\
&= w_j w_k \sum_{p=0}^{N} w_p \left(\boldsymbol{\mathcal{F}}_1 \right)_{p,j,k} D_{p,i} + w_i w_k \sum_{p=0}^{N} w_p \left(\boldsymbol{\mathcal{F}}_2 \right)_{i,p,k} D_{p,j} + w_i w_j \sum_{p=0}^{N} w_p \left(\boldsymbol{\mathcal{F}}_3 \right)_{i,j,p} D_{p,k}
\end{aligned} \tag{3.35}$$

Where $D_{i,j}$ is the differential operator defined as

$$D_{i,j} = \left. \frac{\partial l_j}{\partial \xi} \right|_{\xi=\xi_i}. \tag{3.36}$$

The surface integral term contains the sum of the boundary integrals over the element. Which, for a three-dimensional case, is thus the sum of the six face integrals over the hexahedral element. For example, on the face in the η-ζ plane at $\xi = 0$ the boundary integral can be simplified to

$$\begin{aligned}
&\int_{\eta=-1}^{\eta=1} \int_{\zeta=-1}^{\zeta=1} \boldsymbol{\mathcal{H}} \left(0, \eta, \zeta, \boldsymbol{n}\right) l_i(0) l_j(\eta) l_k(\zeta) d\eta d\zeta \\
&\approx \sum_{n,o=0}^{N} w_n w_o \boldsymbol{\mathcal{H}} \left(0, \eta, \zeta, \boldsymbol{n}\right) l_i(0) \underbrace{l_j(\eta_n)}_{\partial j,n} \underbrace{l_k(\zeta_o)}_{\partial_{k,o}} \\
&= w_j w_k \boldsymbol{\mathcal{H}} \left(0, \eta_j, \zeta_k, \boldsymbol{n}\right) l_i(0).
\end{aligned} \tag{3.37}$$

Here, a difference between the LG and LGL node method can be observed. If LG nodes are used, in order to evaluate the flux at the element interface, the solution has to be interpolated onto the surface nodes where the flux is evaluated, as was shown in Figure 3.1b. Because $l_i(\xi_b) \neq 0 \ \forall i = 1, ..., N$, where ξ_b denotes the reference element boundary nodes denoted with the red squares in Figure 3.1, the surface flux integral contributes to the residual of all nodes. If, however, LGL nodes are used, then the surface integral only contributes to the outer element nodes that are located on the element boundaries ξ_b and

(3.37) becomes

$$\int_{\eta=-1}^{\eta=1}\int_{\zeta=-1}^{\zeta=1} \boldsymbol{\mathcal{H}}\left(0,\eta,\zeta,\boldsymbol{n}\right) l_i(0)l_j(\eta)l_k(\zeta)d\eta d\zeta \approx w_j w_k \boldsymbol{\mathcal{H}}\left(0,\eta_j,\zeta_k\right)\delta_{0,1}. \tag{3.38}$$

As mentioned before, this difference between the LG and LGL nodes causes a difference in the computational cost but also the Jacobian matrix sparsity pattern, as shown by [78], which is important for the implicit solving methods that are considered in this project.

3.2.3. Semi-discrete Formulation

With the time derivative, volume and surface integral defined, the semi-discrete formulation can be formulated. Putting (3.33), (3.35) and (3.37) into (3.30) and dividing by $w_i w_k w_k$ we obtain

$$\begin{aligned}
&J(\xi_i,\eta_j,\zeta_k)\frac{\boldsymbol{u}_{i,j,k}}{\partial t}+\\
&\left(\boldsymbol{\mathcal{H}}(1,\eta_j,\zeta_k,\boldsymbol{n})\frac{l_i(1)}{w_i}+\boldsymbol{\mathcal{H}}(0,\eta_j,\zeta_k,\boldsymbol{n})\frac{l_i(0)}{w_i}\right)-\sum_{p=0}^{N}\frac{w_p}{w_i}(\boldsymbol{\mathcal{F}}_1)_{p,j,k}D_{p,i}+\\
&\left(\boldsymbol{\mathcal{H}}(\xi_i,1,\zeta_k,\boldsymbol{n})\frac{l_j(1)}{w_j}+\boldsymbol{\mathcal{H}}(\xi_i,0,\zeta_k,\boldsymbol{n})\frac{l_j(0)}{w_j}\right)-\sum_{p=0}^{N}\frac{w_p}{w_j}(\boldsymbol{\mathcal{F}}_2)_{i,p,k}D_{p,j}+\\
&\left(\boldsymbol{\mathcal{H}}(\xi_i,\eta_j,1,\boldsymbol{n})\frac{l_k(1)}{w_k}+\boldsymbol{\mathcal{H}}(\xi_i,\eta_j,0,\boldsymbol{n})\frac{l_k(0)}{w_k}\right)-\sum_{p=0}^{N}\frac{w_p}{w_k}(\boldsymbol{\mathcal{F}}_3)_{i,j,p}D_{p,k}=0.
\end{aligned} \tag{3.39}$$

In the case of LGL-DGSEM, the flux contributions from the surface integrals are purely local, and we can write

$$\begin{aligned}
&J(\xi_i,\eta_j,\zeta_k)\frac{\boldsymbol{u}_{i,j,k}}{\partial t}+\\
&\left(\boldsymbol{\mathcal{H}}(1,\eta_j,\zeta_k,\boldsymbol{n})\frac{\delta_{N,i}}{w_i}+\boldsymbol{\mathcal{H}}(0,\eta_j,\zeta_k,\boldsymbol{n})\frac{\delta_{0,i}}{w_i}\right)-\sum_{p=0}^{N}\frac{w_p}{w_i}(\boldsymbol{\mathcal{F}}_1)_{p,j,k}D_{p,i}+\\
&\left(\boldsymbol{\mathcal{H}}(\xi_i,1,\zeta_k,\boldsymbol{n})\frac{\delta_{N,j}}{w_j}+\boldsymbol{\mathcal{H}}(\xi_i,0,\zeta_k,\boldsymbol{n})\frac{\delta_{0,j}}{w_j}\right)-\sum_{p=0}^{N}\frac{w_p}{w_j}(\boldsymbol{\mathcal{F}}_2)_{i,p,k}D_{p,j}+\\
&\left(\boldsymbol{\mathcal{H}}(\xi_i,\eta_j,1,\boldsymbol{n})\frac{\delta_{N,k}}{w_k}+\boldsymbol{\mathcal{H}}(\xi_i,\eta_j,0,\boldsymbol{n})\frac{\delta_{0,k}}{w_k}\right)-\sum_{p=0}^{N}\frac{w_p}{w_k}(\boldsymbol{\mathcal{F}}_3)_{i,j,p}D_{p,k}=0.
\end{aligned} \tag{3.40}$$

One-dimensional semi-discrete formulation From (3.40), the one-dimensional semi-discrete formulation using LGL quadrature can be written as

$$\frac{\partial \boldsymbol{u}_i}{\partial t}-\frac{1}{J(\xi_i)}\sum_{p=0}^{N}\frac{w_p}{w_i}\boldsymbol{\mathcal{F}}_p\cdot D_{p,i}+\frac{1}{J(\xi_i)w_i}\left(\boldsymbol{\mathcal{H}}(1,\boldsymbol{n})\partial_{N,i}+\boldsymbol{\mathcal{H}}(0,\boldsymbol{n})\partial_{0,i}\right). \tag{3.41}$$

Although this formulation is correct. In order to better represent the underlying workings, we revert to the one-dimensional form of (3.30).

$$\int_{-1}^{1}\frac{\Delta x}{2}\sum_{m=0}^{N}\frac{\partial \boldsymbol{u}_m}{\partial t}l_m(\xi)l_k(\xi)d\xi-\int_{-1}^{1}\sum_{m=0}^{N}\boldsymbol{\mathcal{F}}_m l_m(\xi)l_k'(\xi)d\xi=-\left[\boldsymbol{\mathcal{H}}(\xi)l_k(\xi)\right]_{-1}^{1}\qquad k=0,\ldots,N. \tag{3.42}$$

Where $J(\xi)=\frac{\Delta x}{2}$ with Δx being the width of the element in physical coordinate space. Applying the Legendre-Gauss-Lobatto quadrature and using the Lagrange property leads to

$$\frac{\Delta x}{2}w_i\frac{\partial \boldsymbol{u}_i}{\partial t}-\sum_{m=0}^{N}w_m\boldsymbol{\mathcal{F}}_m l_i'(\xi_m)=-\left(\boldsymbol{\mathcal{H}}(1,\hat{n})\delta_{i,N}-\boldsymbol{\mathcal{H}}(0,\hat{n})\delta_{i,0}\right)\qquad i=0,\ldots,N. \tag{3.43}$$

Now, we define the following matrix operators

$$M_{i,j} = \partial_{i,j} w_i, \qquad D_{i,j} = \left.\frac{\partial l_j}{\partial \xi}\right|_{\xi=\xi_i}, \qquad i,j = 0,...,N. \tag{3.44}$$

$$\boldsymbol{B} = \begin{pmatrix} -1 & & & \\ & 0 & & \\ & & \ddots & \\ & & & 0 \\ & & & & 1 \end{pmatrix}. \tag{3.45}$$

And with these operators, (3.43) can be written as

$$\frac{\partial \boldsymbol{u}}{\partial t} = \frac{2}{\Delta x}\left(\underbrace{\boldsymbol{M}^{-1}(\boldsymbol{M}\boldsymbol{D})^T \boldsymbol{\mathcal{F}}}_{\text{volume integral}} - \underbrace{\boldsymbol{M}^{-1}\boldsymbol{B}\boldsymbol{\mathcal{H}}}_{\text{boundary integral}} \right). \tag{3.46}$$

Denoting the semi-discrete weak formulation of the 1D LGL-DGSEM formulation. This formulation can be split into a viscous and advective contribution, leading to

$$\frac{\partial \boldsymbol{u}}{\partial t} = \frac{2}{\Delta x}\left(\boldsymbol{M}^{-1}(\boldsymbol{M}\boldsymbol{D})^T \boldsymbol{\mathcal{F}}_v - \boldsymbol{M}^{-1}\boldsymbol{B}\boldsymbol{\mathcal{H}}_v)\right) + \frac{2}{\Delta x}\left(\boldsymbol{M}^{-1}(\boldsymbol{M}\boldsymbol{D})^T \boldsymbol{\mathcal{F}}_a - \boldsymbol{M}^{-1}\boldsymbol{B}\boldsymbol{\mathcal{H}}_a)\right). \tag{3.47}$$

Where $\boldsymbol{\mathcal{F}}_v$ and $\boldsymbol{\mathcal{H}}_v$ denote the viscous flux and numerical flux function, respectively. Similarly, for the advective case, $\boldsymbol{\mathcal{F}}_a$ and $\boldsymbol{\mathcal{H}}_a$ denote the advective flux and numerical flux function. This can be further simplified to a viscous and advective flux contribution $\boldsymbol{\mathcal{L}}_v$ and $\boldsymbol{\mathcal{L}}_a$.

$$\frac{\partial \boldsymbol{u}}{\partial t} = \frac{2}{\Delta x}\left(\boldsymbol{\mathcal{L}}_a(\boldsymbol{u}) + \boldsymbol{\mathcal{L}}_v(\boldsymbol{u}, \nabla_x \boldsymbol{u})\right). \tag{3.48}$$

As can be observed, the viscous flux contribution requires the gradient of the solution $\nabla_x \boldsymbol{u}$. Methods to calculate this gradient will be discussed in the next section.

3.2.4. Second-order Term Approximation

As mentioned before, the viscous flux depends on the solution and its gradient at both sides of the element interface, i.e. $\boldsymbol{\mathcal{H}}_v(\boldsymbol{u}^-, \nabla_x \boldsymbol{u}^-, \boldsymbol{u}^+, \nabla_x \boldsymbol{u}^+, \boldsymbol{n})$. This poses a problem as second-order derivatives cannot be accommodated directly in a weak variational formulation using a discontinuous function space [22]. To handle the second-order terms, the Bassi-Rebay 1 (BR1) [22] and Bassi-Rebay 2 (BR2) schemes are considered [23]. Other methods have also been created and are summarised in [92]. In this section, an overview of the two methods will be given.

For the BR1 scheme, the gradient of the conservative variables $\nabla_x \boldsymbol{u} = \boldsymbol{\theta}$ is regarded as a set of auxiliary unknowns defined by the first-order differential equation

$$\boldsymbol{\theta} - \nabla_x \boldsymbol{u} = 0. \tag{3.49}$$

This equation can be discretised in the same manner as was done for the governing equations. Firstly, the equation is mapped to the reference space using (3.12) and writing it as

$$J\boldsymbol{\theta} = \sum_{i=1}^{3} J\boldsymbol{a}^i \frac{\partial \boldsymbol{u}}{\partial \xi_i} = \boldsymbol{\mathcal{M}} \nabla_\xi \boldsymbol{u}. \tag{3.50}$$

Where $\boldsymbol{\mathcal{M}}$ is a matrix containing the contravariant coordinate vectors as described by [93]. With this, the same polynomial approximation of degree N as for the solution is assumed, and the same steps taken to discretise the main equation can be applied. Namely, assuming a Galerkin test function space ϕ and integrating over the reference element K_{ref}, the weak form of (3.49) is written as

$$\int_{K_{\text{ref}}} \phi \boldsymbol{\theta} d\xi + \int_{K_{\text{ref}}} \nabla_x \phi \boldsymbol{u} d\xi = \int_{\partial K_{\text{ref}}} \phi \hat{\boldsymbol{u}} \boldsymbol{n} dS. \tag{3.51}$$

Where, again, due to the discontinuity at the element interfaces a numerical flux function $\hat{\boldsymbol{u}}$ is introduced and set to be $\hat{\boldsymbol{u}} = \frac{\boldsymbol{u}^+ + \boldsymbol{u}^-}{2}$ at internal faces and $\hat{\boldsymbol{u}} = \boldsymbol{u}_{\text{bc}}$ at the domain boundary [23]. From the weak form, integration by parts can be performed again on the volume integral term, just as was done for the advective flux term in Subsection 3.2.2

$$\int_{K_{\text{ref}}} \phi \boldsymbol{\theta} d\xi - \int_{K_{\text{ref}}} \phi \nabla_x \boldsymbol{u} d\xi = \int_{\partial K_{\text{ref}}} \phi \left(\hat{\boldsymbol{u}} - \boldsymbol{u}\right) \boldsymbol{n} dS. \tag{3.52}$$

Where the boundary integral term now represents the corrections to the strong internal gradient $\nabla_x \boldsymbol{u}$ due to the discontinuities of u across element interfaces. As suggested by [23], a function $\boldsymbol{R}$ can be introduced defined as

$$\int_{K_{\text{ref}}} \phi \boldsymbol{R} d\xi = \int_{\partial K_{\text{ref}}} \phi (\hat{\boldsymbol{u}} - \boldsymbol{u}) \boldsymbol{n} dS. \tag{3.53}$$

Such that $\boldsymbol{\theta} = \nabla_x \boldsymbol{u} + \boldsymbol{R}$. Again showing that $\boldsymbol{\theta}$ is a corrected gradient, composed of internal gradient and boundary discontinuity correction terms.

For the BR1 method, the gradient corrections from all different faces, as denoted by (3.53), are used in the volume integral and surface integral terms in (3.29). Thus, the viscous flux contribution can be written in integral form as

$$\boldsymbol{\mathcal{L}}_v(\boldsymbol{u}, \boldsymbol{\theta}) = \int_{K_{\text{ref}}} \nabla \phi \cdot \boldsymbol{\mathcal{F}}_v \left(\boldsymbol{u}, \boldsymbol{\theta}\right) d\xi + \int_{\partial K_{\text{ref}}} \phi \boldsymbol{\mathcal{H}}_v(\boldsymbol{u}, \boldsymbol{\theta}, \boldsymbol{u}^+, \boldsymbol{\theta}^+, \boldsymbol{n}) dS, \tag{3.54}$$

For BR1, the weak form of (3.51) can also be used to compute $\boldsymbol{\theta}$ as formulated in the original paper by Bassi and Rebay [22]. The viscous flux thus uses the corrected gradient $\boldsymbol{\mathcal{F}}_v(\boldsymbol{u}, \boldsymbol{\theta})$. For the viscous numerical flux, a central flux is used

$$\boldsymbol{\mathcal{H}}_v(\xi, \eta, \zeta, \boldsymbol{n}) = \frac{1}{2} \left[\boldsymbol{\mathcal{F}}_v(\boldsymbol{u}^-, \boldsymbol{\theta}^-) + \boldsymbol{\mathcal{F}}_v(\boldsymbol{u}^+, \boldsymbol{\theta}^+)\right] \cdot \boldsymbol{n} \tag{3.55}$$

The BR1 scheme follows directly from the derivation of the auxiliary equation. The scheme, in general, is considered consistent but unstable. However, recent work from Gassner et al. [93] proved that the BR1 scheme is stable for the compressible Navier-Stokes equations if LGL quadrature points are used together with a split-form stabilisation. Because the method depends on the DG discretised gradient on both sides of the element interface, the method is said to be non-compact. Meaning that elements may be linked with neighbours of its neighbours [94], leading to a non-compact Jacobian matrix, which is unfavourable for iterative solving. Furthermore, the method also exhibits non-optimal odd-even convergence behaviour depending on the polynomial approximation degree used, as shown in [93].

The BR2 method is a modification of the BR1 method, where the gradient correction term $\boldsymbol{R}$ in the surface integral is replaced by a local gradient correction $\boldsymbol{r}_k$ defined as

$$\int_{K_{\text{ref}}} \phi \boldsymbol{r}_k d\xi = \int_k \phi (\hat{\boldsymbol{u}} - \boldsymbol{u}) \boldsymbol{n} dS \qquad \forall k \in \partial K_{\text{ref}}. \tag{3.56}$$

Where the following property holds

$$\boldsymbol{R} = \sum_{k \in \partial K_{\text{ref}}} \boldsymbol{r}_k. \tag{3.57}$$

Then, the gradient correction for the numerical flux used at an element boundary k is written as

$$\boldsymbol{\theta}_k^{\text{BR2}} = \nabla_x \boldsymbol{u} + \psi \boldsymbol{r}_k \tag{3.58}$$

This results in the corrected gradient used in the surface integral being only dependent on the local element interface considered, subsequently resulting in a more compact stencil. The parameter ψ was considered to be $\psi = 1$ in the original formulation by Bassi and Rebay [23], but has been shown by [94] to be stable for $\psi \geq a$, where a corresponds to the number of surfaces of the element. That is $\psi \geq 6$

in a three-dimensional case with hexahedral elements and $\psi \geq 2$ in a one-dimensional case. The ψ parameter can be seen as a gradient correction term. If ψ is increased, the stability of the scheme is increased, but also more dissipation is added.

Although lower restrictions have been found to work by [95], in this work $\psi = 2$ is considered for the one-dimensional case. Thus, for BR2 we can write (3.54) as

$$\mathcal{L}_v(\boldsymbol{u}, \boldsymbol{\theta}) = \int_{K_{\text{ref}}} \nabla\phi \cdot \boldsymbol{\mathcal{F}}_v(\boldsymbol{u}, \boldsymbol{\theta})\, d\xi + \int_{\partial K_{\text{ref}}} \phi \boldsymbol{\mathcal{H}}_v(\boldsymbol{u}, \nabla_x \boldsymbol{u} + \psi \boldsymbol{r}_k, \boldsymbol{u}^+, \nabla_x \boldsymbol{u}^+ + \psi \boldsymbol{r}_k^+, \boldsymbol{n}) dS. \tag{3.59}$$

Although the BR2 method has commonly been more favoured when used with implicit solving methods due to the more compact stencil and stability, such as also argued by [56] and in the introduction of the BR2 [23], when the BR1 method is used together with LGL quadrature the method is equivalent to the interior penalty method and thus compact [96]. Therefore, if LGL quadrature is used, there is no difference in the Jacobian matrix sparsity pattern for the BR1 and BR2 methods. Next to this, as already said, when applied to the compressible Navier-Stokes equations on LGL nodes, the BR1 scheme has been proven to be stable by Gassner et al. [93]. In this work, it is revealed that the instabilities of the BR1 scheme are caused by an unstable discretisation of the advective terms. By using a stabilising split-form formulation of the advective terms, they then show that the BR1 scheme is stable and does not add artificial dissipation. As a last note, as shown by Theorem four in [97], the BR1 and BR2 schemes on the LGL quadrature nodes with $\psi = 1$ produce the exact same gradient formulation, meaning that the schemes are equivalent in the discrete sense.

3.2.5. Stabilisation Techniques for the DGSEM

In the standard DGSEM, the Galerkin approach is known to be unstable when inexact integration is used on the nonlinear flux functions [14, 37]. These instabilities arise from aliasing errors, which can introduce artificial energy into the solution of the NSE that could lead to diverging simulations [2]. In this work, the specifics of stabilisation techniques to combat the aliasing errors from the DGSEM method are not covered. For the LGL-DGSEM in the TRACE solver shown in Chapter 5, the Kennedy-Gruber (KG) split-form stabilisation scheme [41] is used for all simulations considered. The split-form framework works on the LGL-DGSEM because the operator belongs to the class of diagonal-norm summation-by-parts (SBP) operators. For more details on this stabilisation technique, the reader is referred to [2, 98].

3.3. Time Discretisation

From the spatial discretisation, the semi-discrete form was derived. This semi-discrete form resulted in the discretisation of the spatial DGSEM operator. Writing this DGSEM operator as $R(\boldsymbol{u})$, we can write the problem of unsteady time integration as the following initial value problem (IVP)

$$\begin{aligned} &\frac{\partial \boldsymbol{u}}{\partial t} = \boldsymbol{R}(t, \boldsymbol{u}(t)), \\ &\boldsymbol{u}(0) = \boldsymbol{u}_0. \end{aligned} \tag{3.60}$$

In order to advance the DGSEM discretised equation in time, one-step multistage implicit Runge-Kutta methods will be used, also referred to as diagonally implicit Runge-Kutta (DIRK) methods [99]. For these methods, the following Butcher tableau is defined as shown in Table 3.1. From a Taylor series expansion, it can be shown that the RK method is consistent if

$$\sum_{i=1}^{s} b_i = 1. \tag{3.61}$$

Table 3.1: General structure of a Butcher tableau.

c_1	$a_{1,1}$	0	$\cdots$	0
c_2	$a_{2,1}$	$a_{2,2}$	$\cdots$	0
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
c_s	$a_{s,1}$	$a_{s,2}$	$a_{s,s-1}$	$a_{s,s}$
	b_1	b_2	$\cdots$	b_s

With this table and condition defined, solution at time-step n is advanced to $n+1$ through s stages as

$$\boldsymbol{u^{n+1}} = \boldsymbol{u}^n + \Delta t \sum_{i=1}^{s} b_i \boldsymbol{k}_i, \tag{3.62}$$

$$\boldsymbol{k}_i = \boldsymbol{R}\left(t^n c_i, \boldsymbol{u}^n + \Delta t \sum_{j=1}^{i} a_{i,j} \boldsymbol{k}_j \right) \quad i = 1, ..., s. \tag{3.63}$$

It can already be seen that for every i-th stage, if $a_{i,i} \neq 0$, the equation is implicit. In the group of one-step multistage implicit Runge-Kutta methods, the singly diagonally implicit Runge-Kutta (SDIRK) methods have a fixed diagonal coefficient $a_{i,i} = \alpha$. If the diagonal coefficient $a_{i,i}$ is nonzero for any stage, the i-th stage solution $\boldsymbol{u}_i$ is defined as

$$\boldsymbol{u}_i = \boldsymbol{u}_n + \Delta t \sum_{j=1}^{i-1} a_{i,j} \boldsymbol{k}_j + \Delta t a_{i,i} \boldsymbol{R}(t^n + c_i \Delta t, \boldsymbol{u}_i) \tag{3.64}$$

Thus, the SDIRK scheme starting from the previous time-step solution $\boldsymbol{u}_n$ can be written as

$$\begin{aligned}
&\text{for i = 1,...,s} \\
&\quad \boldsymbol{u}_i = \boldsymbol{u}_n \\
&\quad \boldsymbol{u_i} \mathrel{+}= \Delta t \sum_{j=1}^{i-1} a_{i,j} \boldsymbol{k}_j \\
&\quad \text{if } a_{i,i} \neq 0 \\
&\qquad \text{solve } \boldsymbol{u_i} = \boldsymbol{u}_n + \Delta t \sum_{j}^{i-1} a_{i,j} \boldsymbol{k}_j + \Delta t a_{i,i} \boldsymbol{R}(t^n + c_i \Delta t, \boldsymbol{u}_i) \\
&\quad \boldsymbol{k_i} = \boldsymbol{R}(t^n + c_i \Delta t, \boldsymbol{u}_i) \\
&\boldsymbol{u}_{n+1} = \boldsymbol{u}_n + \sum_{i=1}^{s} b_i \boldsymbol{k}_i
\end{aligned} \tag{3.65}$$

3.3.1. Stability Constraints on Explicit Time Integration with DGSEM

In this thesis, implicit time-integration methods are considered to overcome the stability constraints faced when using explicit time-integration schemes. A common method to normalise the restriction on the time-step size is by using the Courant-Friedrichs-Lewy (CFL) condition [100]. The CFL number acts as a normalisation of the time-step size for a given spatial discretisation and system of equations. Such that the time step size restrictions of explicit time-integration schemes, which are known in terms of the CFL number, can be estimated. In general, the CFL number can be formulated as

$$\mathrm{CFL} = \frac{\Delta t}{\max_{ijk \in K}(\lambda^{\mathrm{a}}_{ijk}, \lambda^{\mathrm{v}}_{ijk})}. \tag{3.66}$$

Where the CFL number is calculated on every node (i, j, k) on every element K. Due to the Navier-Stokes equations being partially hyperbolic, time-step restrictions originating from the advective and diffusive terms have to be taken into account. The advective eigenvalues are estimated using

$$\lambda^{\mathrm{a}}_{ijk} = \frac{f_N}{J_{ijk}} \sum_{n=1}^{3} \left[(J\boldsymbol{a}^{\boldsymbol{n}})_{ijk} \cdot \boldsymbol{u}_{ijk} + a_{ijk} \sqrt{J(\boldsymbol{a}^{\boldsymbol{n}})_{ijk} \cdot (\boldsymbol{a}^{\boldsymbol{n}})_{ijk}} \right], \tag{3.67}$$

which follow from [101]. Here, a_{ijk} represents the local speed of sound at the considered node. The metric terms $J\boldsymbol{a}$ are required to formulate the CFL number on the reference element, and the eigenvalues are taken from the inviscid flux Jacobian. For DG schemes, usually $f_N = 2N + 1$ is used, because it leads to a normalisation of the CFL number if the time order M is equal to the spatial order $M = N + 1$ [33, 44]. However, as shown in [33], when LGL nodes are used, the maximum allowable time-step size is found to be approximately twice as large as for the LG nodes. The diffusive eigenvalues are estimated using

$$\lambda^{\mathrm{v}}_{ijk} = \left(\frac{f_N}{J_{ijk}} \right)^2 \sum_{n=1}^{3} \left[(J\boldsymbol{a}^{n})_{ijk} \cdot (J\boldsymbol{a}^{n})_{ijk} \right] \max \left(\frac{4\mu_{ijk}}{3\rho_{ijk}}, \frac{\gamma \mu_{ijk}}{\mathrm{Pr}\rho_{ijk}} \right). \tag{3.68}$$

Although differently formulated due to the metric terms, the approximations of the eigenvalues for the advective and convective terms in the Navier-Stokes equations follow those commonly used for finite-volume schemes. The difference is in the scaling factor f_N, leading to a more severe restriction on the time step size [33].

3.4. Implicit Solver

In the previous section, the time-discretisation resulted in stages requiring an implicit system of equations to be solved if the stage diagonal coefficient $a_{i,i}$ is non-zero. In this section, the type of nonlinear solver and subsequent linear solver to solve such an implicit stage will be discussed.

3.4.1. Inexact Newton Method

The nonlinear equation appearing in the implicit RK scheme in (3.65) can be considered as the root finding problem

$$\boldsymbol{f}(\boldsymbol{u}_i) = 0. \tag{3.69}$$

Where $\boldsymbol{f}(\boldsymbol{u})$ represents the fully discretised operator resulting from the DG and RK discretisation. In order to solve the problem, the operator is linearised around the current guess of the solution and iteratively solved until the convergence criterion σ is reached

$$\begin{aligned} &\text{While } \|f(\boldsymbol{u}^j)\| \geq \sigma \|f(\boldsymbol{u}^0)\| \\ &\qquad \text{solve } \left. \frac{\partial \boldsymbol{f}(\boldsymbol{u})}{\partial \boldsymbol{u}} \right|_{\boldsymbol{u}^j} \Delta \boldsymbol{u}^j = -\boldsymbol{f}(\boldsymbol{u}^j) \\ &\qquad \boldsymbol{u}^{j+1} = \boldsymbol{u}^j + \Delta \boldsymbol{u}^j. \end{aligned} \tag{3.70}$$

For every Newton step, a linear system of equations must be solved to determine the update $\Delta \boldsymbol{u}^j$. As will be explained later, the linear solver also solves according to a convergence criterion σ. According to (3.70), the linear solve has to be solved exactly, or in practice to machine precision, and this is also what occurs when the exact Newton method is used. However, exact linear solves can be expensive and by modifying Newton's method as

$$\begin{aligned} &\text{While } \|f(\boldsymbol{u}^j)\| \geq \sigma \|f(\boldsymbol{u}^0)\| \\ &\qquad \text{solve } \left. \frac{\partial \boldsymbol{f}(\boldsymbol{u})}{\partial \boldsymbol{u}} \right|_{\boldsymbol{u}^j} \Delta \boldsymbol{u}^j = -\boldsymbol{f}(\boldsymbol{u}^j) + \boldsymbol{r}^j \\ &\qquad \boldsymbol{u}^{j+1} = \boldsymbol{u}^j + \Delta \boldsymbol{u}^j., \end{aligned} \tag{3.71}$$

the restriction to solve the linear system exactly is lifted. Here, the convergence criterion ϵ is defined as

$$\epsilon = \|\boldsymbol{r}^j\| = \eta \|\boldsymbol{f}(\boldsymbol{u}^j)\|. \tag{3.72}$$

Where η is considered a non-negative forcing term that determines the precision to which the linear system will be solved. This method is therefore referred to as the inexact Newton method and proven to be stable if $\eta < 1$ is chosen [102]. As stated in [65], the forcing term η has to be chosen adaptively in order to avoid oversolving. Oversolving occurs when the Newton guess $\boldsymbol{u}^j$ is far away from the solution of the nonlinear equation, which results in the obtained update being large, meaning that it exceeds the space in which the local linear approximation is correct. Because of this, beyond a certain reduction in the linear residual norm, it is not guaranteed that the nonlinear residual norm also decreases further. In order to determine the forcing term at the k-th Newton step η_k, the work from [66] is followed. For this, we define

$$\eta_j^A = \gamma \left(\frac{\|\boldsymbol{f}(\boldsymbol{u}^j)\|}{\|\boldsymbol{f}(\boldsymbol{u}^{j-1})\|} \right)^2, \tag{3.73}$$

with $\gamma \in (0, 1]$. In this case, $\gamma = 0.9$ was chosen, in order to avoid η becoming too large, as well as to avoid η from changing too small too quickly. The following safeguards, as recommended by [66], are used

$$\eta_j^s = \begin{cases} \eta_0, & j = 0 \\ \min\{\eta_{\max}, \eta_j^A\}, & j > 0, \gamma\eta_{j-1}^2 \leq .1 \\ \min\{\eta_{\max}, \max\{\eta_j^A, \gamma\eta_{j-1}^2\}\}, & j > 0, \gamma\eta_{j-1}^2 > .1. \end{cases} \tag{3.74}$$

Where η_0 is the initial forcing term, which can either be set to $\eta_{\max}$ to avoid over-solving in the initial linear solve. However, if it is known that the initial solution guess of the Newton method is close to the final solution, which can occur when unsteady implicit time-stepping is considered (more discussion in Subsection 4.3.1), then choosing an η_0 lower might be beneficial to avoid under-solving in the initial linear solve of the inexact Newton method.

Additionally, as argued by [103], in order to avoid that in the final iterations, the linear solve over-solves past the given convergence criterion of the Newton method, the final choice of η_k is set to

$$\eta_k = \min\{\eta_{\max}, \max\{\eta_j^s, 0.5\frac{\tau}{\|\boldsymbol{f}(\boldsymbol{u}^j)\|}\}\}. \tag{3.75}$$

Which is related to the convergence criterion of Newton's method because

$$\tau = \sigma\|\boldsymbol{f}(\boldsymbol{u}^0)\|. \tag{3.76}$$

3.4.2. Linear Solver Method

In order to solve the linear system of equations appearing in every Newton step, it is common to also use an iterative solution method. This is because direct solving methods scale poorly with increasing matrix size and lead to dense matrices, resulting in large memory costs. Furthermore, using an iterative solving method also allows for inexact linear solves and thus the usage of the inexact Newton method described in Subsection 3.4.1. In this work, the generalised minimal residual (GMRES) method is used [75]. Firstly, the linear system of equations to be solved in (3.71) or (3.70) can be written as

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}. \tag{3.77}$$

Where

$$\boldsymbol{A} := \left. \frac{\partial \boldsymbol{f}(\boldsymbol{u})}{\partial \boldsymbol{u}} \right|_{\boldsymbol{u}^j}, \tag{3.78}$$

$$\boldsymbol{x} := \Delta\boldsymbol{u}^j, \tag{3.79}$$

$$\boldsymbol{b} := -\boldsymbol{f}(\boldsymbol{u}^j). \tag{3.80}$$

Following [104], the GMRES is a Krylov subspace method that extracts an approximate solution $\boldsymbol{x}_m$ from an affine subspace $\boldsymbol{x}_0 + \mathcal{K}_m$ of dimension m by imposing the Petrov-Galerkin condition

$$\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_m \perp \mathcal{L}_m. \tag{3.81}$$

Here $\mathcal{K}_m$ is the Krylov subspace, and $\mathcal{L}_m$ is another subspace of dimension m. Specifically for GMRES $\mathcal{L}_m = A\mathcal{K}_m$ is used. The Krylov subspace is defined as

$$\mathcal{K}_m(\boldsymbol{A}, \boldsymbol{r}_0) = \text{span}\{\boldsymbol{r}_0, \boldsymbol{A}\boldsymbol{r}_0, \boldsymbol{A}^2\boldsymbol{r}_0, ..., \boldsymbol{A}^{m-1}\boldsymbol{r}_0\}, \tag{3.82}$$

with $\boldsymbol{r}_0 = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_0$. In this work, the restarted GMRES method based on Arnoldi's method with Givens rotations is used. In the GMRES algorithm, Arnoldi's method is used with a modified Gram-Schmidt orthogonalisation to construct an orthogonal basis $\{\boldsymbol{v}_1, ..., \boldsymbol{v}_m\}$ of the Krylov subspace $\mathcal{K}_m$. The Arnoldi method was first created to reduce a dense matrix into Hessenberg form with a unitary transformation. However, it was discovered that the eigenvalues of the Hessenberg matrix obtained from a smaller number of steps than the original matrix size already approximated some eigenvalues of the original matrix. This led to the method being used in solving large, sparse linear systems of equations because of its ability to approximate large matrices in the much smaller Hessenberg form. In the Arnoldi-Modified Gram-Schmidt method, the orthonormal basis vector of the Krylov subspace $\mathcal{K}_{j+1}$ is computed from the already constructed orthonormal basis vectors $\{\boldsymbol{v}_1, ..., \boldsymbol{v}_j\}$. This is done by first projecting the previous basis vector $\boldsymbol{v}_j$ onto $\boldsymbol{A}$. After this, the newly projected vector $\boldsymbol{w}_j$ is used to fill in the Hessenberg matrix over all previous basis vectors while also applying the modified Gram-Schmidt orthogonalisation. The full method is shown in Algorithm 1. The algorithm for the restarted GMRES method utilising the

Algorithm 1 Arnolid-Modified Gram-Schmidt

1: **procedure** ARNOLDI_MODIFIED_GRAM_SCHMIDT(j, $\boldsymbol{A}$, $\{\boldsymbol{v_1}, ..., \boldsymbol{v}_j\}$, $\boldsymbol{h}$, $\boldsymbol{w}_j$)
2: $\quad \boldsymbol{w}_j = \boldsymbol{A}\boldsymbol{v}_j$
3: $\quad$ **for** $i = 1$ to j **do**
4: $\quad\quad h_{i,j} = (\boldsymbol{v}_i, \boldsymbol{w}_j)$
5: $\quad\quad \boldsymbol{w}_j = \boldsymbol{w}_j - h_{i,j}\boldsymbol{v}_j$
6: $\quad$ **end for**
7: $\quad h_{j+1,j} = \|\boldsymbol{w}_j\|$
8: $\quad \boldsymbol{v}_{j+1} = \frac{\boldsymbol{w}_j}{h_{j_1,j}}$
9: **end procedure**

Arnoldi-Modified Gram-Schmidt process is given in Algorithm 2. The input for this algorithm are the desired maximum Krylov subspace dimension m, the operator evaluated at the current Newton step $\boldsymbol{f}(\boldsymbol{u}^j)$ as well as a guess for the initial update of the solution vector $\Delta\boldsymbol{u}_g^j$, which can be set to the null vector $\boldsymbol{0}$. As can be seen, the algorithm loops over the Krylov subspace dimensions and calculates the new basis vector as explained earlier. Then, for every basis vector, a Givens rotation is used to transform the Hessenberg matrix into an upper triangular matrix. In [105], the usage of Givens rotation is explained in more detail. As can be seen in line 17 of Algorithm 2, in the construction loop of the Krylov subspace, the current linear residual is evaluated using

$$\|\beta\boldsymbol{v}_1 - \boldsymbol{h}_m y_m\|. \tag{3.83}$$

Where $\beta = \|\boldsymbol{r}_0\|$, $\boldsymbol{h}_m$ is the current upper Hessenberg matrix, $\boldsymbol{v}_1$ is the normalised initial residual vector and $\boldsymbol{y}_m$ is the vector that minimises (3.83) [104].

3.4.3. Preconditioned GMRES

Preconditioning of the linear system of equations is a common (if not necessary) approach to accelerate the convergence behaviour of Krylov subspace methods such as GMRES. Preconditioning aims to modify the linear system of equations to be solved such that the new system of equations has a matrix-operator $\boldsymbol{A}^*$ with a lower condition number than the original matrix-operator $\boldsymbol{A}$. The condition number of a function, or in this case a linear system of equations, is a measure of how sensitive the output of the function is to small input changes. This also explains intuitively why iterative methods struggle to converge when the matrix has a high condition number (ill-conditioned) compared to when the matrix has a low condition number (well-conditioned).

In order to lower the condition number, a preconditioning operator $\boldsymbol{M}$ that approximates the Jacobian operator $\boldsymbol{A}$ needs to be found. The inverse of this preconditioning operator is required to be known.

Algorithm 2 Restarted Generalised Minimal Residual (GMRES) method

procedure GMRES_RESTARTED(m,$-\boldsymbol{f}(\boldsymbol{u}^j)$, $\Delta\boldsymbol{u}_g^j$)
  while r $\leq$ r_{max} **do**
    $\boldsymbol{r}_0 = -\boldsymbol{f}(\boldsymbol{u}^j), \quad \boldsymbol{v}_1 = \frac{\boldsymbol{r}_0}{\|\boldsymbol{r}_0\|}, \quad y_1 = \|\boldsymbol{r}_0\|, \quad \Delta\boldsymbol{u}_0^j = \Delta\boldsymbol{u}_g^j$
    for $j = 1$ to m **do**
      Arnoldi_Modified_Gram_Schmidt$(j, \boldsymbol{A}, \{\boldsymbol{v}_1, ..., \boldsymbol{v}_j\}, \boldsymbol{h}, \boldsymbol{w}_j)$
      if $h_{j+1,j} < \epsilon_m$ **then**
        break
      end if
      for $i = 0$ to $j - 1$ **do**
        /* Apply Previous Givens rotations*/
        $\tilde{\boldsymbol{h}} = \boldsymbol{h}$
        $h_{i+1,j} = c_1(i)\tilde{h}_{i,j} - s_1(i)\tilde{h}_{i+1,j}$
        $h_{i+1,j} = s_1(i)\tilde{h}_{i,j} + c_1(i)\tilde{h}_{i+1,j}$
      end for
      ApplyGivensRotations$(h_{i,j}, h_{i+1,j}, c_1(j), s_1(j))$
      /* Rotate residual vector*/
      $y_{j+1} = s_1(i)y_j, \quad y_j = c_j(i)y_j$
      if $y_{j+1} < \eta\|\boldsymbol{f}(\boldsymbol{u}^j)\|$ or $j = m$ **then**
        for $i = j$ to 1 **do**
          $\alpha_i = \frac{1}{h_{i,i}}\left(y_i - \sum_{k=i+1}^{j} h_{i,k}\alpha_k\right)$
        end for
        $\Delta\boldsymbol{u}_{r+1}^j = \Delta\boldsymbol{u}_r^j + \sum_{i=1}^{j} \boldsymbol{v}_i\alpha_i$
        if $y_{j+1} < \eta\|\boldsymbol{f}(\boldsymbol{u}^j)\|$ **then**
          return $\Delta\boldsymbol{u}_{r+1}^j$
        end if
        $\boldsymbol{r}_0 = -\boldsymbol{f}(\boldsymbol{u}^j) - \boldsymbol{A}\Delta\boldsymbol{u}_{r+1}^j$
        $r = r + 1$
      else
        Continue
      end if
    end for
  end while
end procedure

The effectiveness of the preconditioner depends on how well $\boldsymbol{M}$ approximates $\boldsymbol{A}$ while remaining inexpensive to construct and invert. A perfectly accurate preconditioner, $\boldsymbol{M} = \boldsymbol{A}$, would lead to immediate convergence but would be as costly as solving the system directly. On the other hand, a very cheap preconditioner may offer little improvement in convergence. Therefore, an appropriate balance must be found between preconditioner quality and computational cost, ensuring that the overall solution process is more efficient than solving the unpreconditioned system.

With this, the new linear system of equations can be defined in two different ways [106]

$$\boldsymbol{M}^{-1}\boldsymbol{A}\boldsymbol{x} = \boldsymbol{M}^{-1}\boldsymbol{b}, \tag{3.84}$$

$$\boldsymbol{A}\boldsymbol{M}^{-1}\boldsymbol{y} = \boldsymbol{b}, \qquad \boldsymbol{M}\boldsymbol{x} = \boldsymbol{y} \tag{3.85}$$

Where the system defined in (3.84) and (3.85) are defined as *left* preconditioning and *right* preconditioning, respectively. For GMRES in combination with inexact Newton's method, right preconditioning is preferred because the residual remains identical to the true residual. Therefore, only the right preconditioning will be considered in this work. In order to use right preconditioning, the matrix-vector multiplications in line 2 of Algorithm 1 and line 26 in Algorithm 2 have to be pre-multiplied by $\boldsymbol{M}^{-1}$. Furthermore, as the GMRES method now solves for $\boldsymbol{y} = \boldsymbol{M}\boldsymbol{x}$ and in order to return $\boldsymbol{x}$, the resulting vector out of the GMRES iterations has to be multiplied by $\boldsymbol{M}^{-1}$ in order to retrieve $\Delta\boldsymbol{u}^j$.

3.4.4. Matrix-free GMRES

The previous sections covered the derivation and implementation of the preconditioned restarted GMRES algorithm that is used to solve the linear system of equations. In this process, it was assumed that the Jacobian matrix operator $\boldsymbol{A}$ is defined as

$$\boldsymbol{A} := \left.\frac{\partial \boldsymbol{f}(\boldsymbol{u})}{\partial \boldsymbol{u}}\right|_{\boldsymbol{u}^j} = \boldsymbol{I} - a_{i,i}\Delta t \left.\frac{\partial \boldsymbol{R}(\boldsymbol{u})}{\partial \boldsymbol{u}}\right|_{\boldsymbol{u}^j} \tag{3.86}$$

Where the second formulation comes from differentiating the nonlinear system of equations shown in (3.65). Formally, this matrix thus has size $n_{\mathrm{DOF}} \times n_{\mathrm{DOF}}$, where n_{DOF} is the total degrees of freedom. However, following from Subsection 3.2.2, the LGL-DGSEM operator naturally only considers local DOF on the element and DOF at the element boundaries of its direct neighbours. This thus naturally leads to a block-structured matrix pattern that offers significant benefits to the matrix sparsity pattern and density.

Nonetheless, the cost of storing a sparse matrix also grows with increasing polynomial order and element count. And when looking at the manner in which the Jacobian is used in the GMRES and Arnoldi algorithm, it can be noticed that it is only used to evaluate matrix-vector products $\boldsymbol{A}\boldsymbol{v}$. Nowhere is $\boldsymbol{A}$ explicitly required. Therefore, methods that do not store the matrix explicitly but only apply operators that evaluate the matrix-vector product $\boldsymbol{A}\boldsymbol{v}$ or, more intuitively, evaluate the directional derivative of the linearised operator, have been developed to overcome the memory issues of storing the Jacobian matrix explicitly. This class of methods is typically referred to as Jacobian-Free Newton Krylov Methods (JFNK) or matrix-free GMRES [5].

Intuitively, the matrix-free GMRES method is based on the fact that the traditional GMRES algorithm only evaluates "directional" derivatives of the form $\boldsymbol{A}\boldsymbol{v}$. In this project, two different methods of evaluating the matrix-vector product will be used.

Finite-difference approximation

The first method considered to approximate the matrix-vector product $\boldsymbol{A}\boldsymbol{v}$ is by a first-order finite difference method

$$\boldsymbol{A}(\boldsymbol{u}^j)\boldsymbol{v} \approx \frac{\boldsymbol{f}(\boldsymbol{u}^j + \epsilon\boldsymbol{v}) - \boldsymbol{f}(\boldsymbol{u}^j)}{\epsilon}. \tag{3.87}$$

Because the finite-difference method only requires a means of evaluating the right-hand side function $\boldsymbol{f}(\boldsymbol{u})$ with and without the added gradient, its implementation costs are relatively low and straightforward. The drawback of this method is that the result is inexact. Furthermore, as stated in Subsection 3.4.4,

the value of ϵ chosen is important to prevent the round-off error from polluting the result while also ensuring that ϵ is taken small enough not to introduce large truncation errors from the finite-difference approximation. Several formulations for ϵ are shown in [5]. In this work ϵ as proposed by [107] is used

$$\epsilon = \frac{\sqrt{10^{-16}}}{\|v\|}. \tag{3.88}$$

This formulation, as stated by [107], has been proven to produce good results for simulations using double precision and is simple compared to other methods proposed in [5].

Forward algorithmic differentiation

The second method considered is the use of algorithmic differentiation (AD) in a forward manner. In contrast to the finite-difference approximation, AD outputs an exact derivative of any input function f. The working principle of AD is the application of the chain rule of differentiation of every operator evaluation inside f. The application of $f(u)$ is nothing more than a sequence of mathematical operations (addition, subtraction, multiplication and other elementary functions) of which the derivative can be easily calculated [108].

For instance, considering that f is composed of N operators L^N, we can write out the evaluation of f as a sequence of operators, each resulting in a temporary result w

$$f(x) = L^N \circ L^{N-1} \circ \ldots \circ L^1 = L^N(w_N) = L^N(L^{N-1}(\ldots(L^1(x)))). \tag{3.89}$$

Applying the chain rule to evaluate $f^{'}(x)$ we get

$$\frac{\partial f(x)}{\partial x} = \frac{\partial L^N(w_N)}{\partial w_N} \frac{\partial L^{N-1}(w_{N-1})}{\partial\, w_{N-1}} \cdots \frac{\partial L^1(x)}{\partial x}. \tag{3.90}$$

The derivative of these elementary operations can be easily defined. For instance, for $f(x) = \sin(x)$ the derivative operation is defined as $f^{'}(x) = \cos(x)$. To carry this derivative information throughout the numerical sequences, every real number has to carry an additional derivative number, which is used to evaluate the derivative of the operation. With this additional "number" attached to every real number, derivatives can be propagated through the numerical sequences. Denoting the additional number with the prefix Δ we can write the AD variant of a multiplication operator $f(x_1, x_2) = x_1 x_2$

$$f(x_1, x_2) = x_1 x_2, \tag{3.91}$$
$$f'(x_1, x_2) = x_1 \Delta x_2 + \Delta x_1 x_2. \tag{3.92}$$

By defining all derivative forms of the elementary operators, and enabling the code structure to allow the usage of the "AD" number types, an input to a function f can be seeded with derivative information by placing it in the secondary derivative number of the input, and be able to evaluate $f^{'}(x)$ by reading the derivative numbers of the output. As a final note, the usage of this AD variable to store the derivative is closely related to the complex step method, where the imaginary number is used as this storage point. More details on the complex-step method, which is not considered in this project, and its relation to forward AD are described in [109].

4

1D Time-implicit DGSEM Solver

This chapter will cover the setup and numerical experiments performed on the 1D time-implicit DGSEM solver. This solver was developed in order to better understand the workings and structure of a DGSEM time-implicit solver, as well as to have a light solver that could be used to perform small numerical experiments that would otherwise be more complex in the TRACE solver. Firstly, in Section 4.1, an overview of the 1D solver structure and features is given, and the two test cases used to perform the numerical experiments are described. After this, in Section 4.2, the correct working of the 1D solver is verified by a spatial and temporal convergence rate test. In Section 4.3, the testing on the 1D solver continues with an analysis of the inexact Newton method and a Jacobian sparsity pattern analysis. Following this, in Section 4.4, the matrix-free approach applied to the 1D solver is analysed by a comparison of the matrix-based GMRES method. Here, preconditioning methods are also explored. Finally, in Section 4.5, a small conclusion regarding the results of the analysis on the 1D solver is given.

4.1. 1D Solver Setup

In this section, an overview of the features of the 1D solver is given. Next to this, the test cases that are used in the numerical experiments on the 1D solver are described.

4.1.1. Structure and Features of the 1D Solver

The 1D solver was written in C++. For handling matrices and matrix-vector products, the Eigen library was used[1] [110]. The solver is written with an object-oriented approach, and it was chosen not to implement any parallel capabilities. All test cases in this chapter were run on a single processor. As an input, the solver must be given an appropriate `input.dat` ASCII file containing the node locations and initial solution at each node. Next to this, a `cntl.json` control file must be given, which specifies the specifics of the simulation (e.g. equation type to be solved and quadrature nodes used).

For the nonlinear solver, the inexact Newton method as described in Subsection 3.4.1 is used. Both the fixed and adaptive forcing terms can be used. The solver is able to use the matrix-based and matrix-free restarted GMRES algorithms that were manually implemented in the solver, as well as the matrix-based GMRES solver provided by the Eigen library. The matrix-free GMRES method uses the finite-difference differentiation method as described in Subsection 3.4.4. For the matrix-based GMRES methods, this same finite-difference approximation is used to explicitly construct the Jacobian matrix. Usage of colouring algorithms such as described by [111, 112] would be very suitable for matrix generation using the finite-difference method. However, due to implementation complexities, this was not considered. Instead, the Jacobian matrix is constructed by finite-difference evaluations for every degree of freedom, filling the Jacobian matrix column by column. This is most certainly an inefficient approach to the matrix construction problem. Nonetheless, the main goal of the 1D solver was to understand the usage of the DGSEM method paired with a matrix-free linear solver.

The solver was applied to solve the linear advection equation as well as the viscous Burgers equation.

[1] `https://eigen.tuxfamily.org`

More information on these equations is in the following subsection. For the viscous terms, both the BR1 and BR2 viscous approximation schemes were applied. The solver is capable of using either LG- or LGL-DGSEM. The LG quadrature nodes were only used in the Jacobian sparsity pattern analysis described in Subsection 4.3.2. In all other cases, the LGL-DGSEM method was used. For the viscous flux terms, a central flux method is used. For the advective flux terms, an upwind flux method, as well as a Lax-Friedrich flux scheme, were implemented. For boundary conditions, Dirichlet and periodic boundary conditions were implemented. These were the boundary conditions required to perform the test cases of interest. Due to time limitations, no stabilising split-form formulations were implemented for the 1D solver. The solver did show signs of instability, but these instabilities were resolved by reducing the element sizes (i.e. increasing the DOF). This was possible because the 1D solver is lightweight, the test cases considered were not expensive, and simulation times in general were short. Many aspects of the code written, specifically the parts related to the spatial discretisation, were inspired by the example algorithm snippets in [113], as well as help from DGSEM experts in the TRACE development team.

4.1.2. Test Cases and Setup of the 1D Solver

For the analysis and verification of the 1D solver, the following test cases have been used.

Sine wave test case

The sine wave test case consists of a sine wave travelling to the right and periodic boundary conditions. This test case was made for the linear advection equation. The linear advection equation in 1D is written as

$$\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x} = 0. \tag{4.1}$$

The equation is trivial, and so are the analytical solutions. This made the test case perfect for debugging and testing purposes of the 1D solver. For the sine wave, a domain of $[0, 1]$ is used and the analytical solution is defined as

$$f(x, t) = \sin(2\pi(x + ct)). \tag{4.2}$$

Where the advection speed $c = 1$ is used. The test case was started at $t = 0$ and advanced through time, where it could be compared with the analytical solution. A numerical domain of $[0, 1]$ was used with periodic boundary conditions. For this test case, the upwind flux scheme is used, as the wave propagation is constant. A simulation time of $t = 1.0$ is used, resulting in a full advection of the sine wave over the domain. Thereby making the initial condition equal to the analytical solution of the test case.

Travelling Burgers wave test case

To solve diffusive terms with the solver and apply it to a nonlinear equation, the 1D Burgers equation was implemented in the solver and used in the numerical experiments. The one-dimensional viscous Burgers equation is defined as

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} - \mu\frac{\partial^2 u}{\partial x^2} = 0. \tag{4.3}$$

In the Burgers equation, the nonlinearity comes from the hyperbolic $u\frac{\partial u}{\partial x}$ term. The Burgers equation represents a simple 1D equation that mimics the general properties of the Navier-Stokes equations. For experiments using the Burgers equation, it was decided to use a test case with a moving wave described by

$$\lim_{x\to-\infty} f(x) = 2, \quad \lim_{x\to\infty} f(x) = 0, \tag{4.4}$$

$$f(x, t) = 1 - \tanh\frac{x - t}{2\mu}. \tag{4.5}$$

This solution is, in fact, a steady solution for the Burgers equation in a frame of reference moving with the wave, as shown by [114]. In this case, the frame of reference is fixed, and the wave moves towards

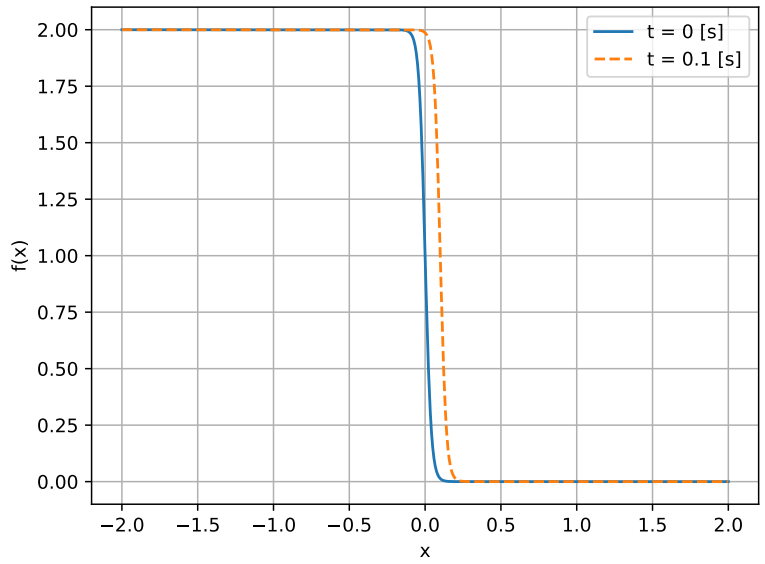


Figure 4.1: Analytical solution to the travelling Burgers wave test case at $t = 0.0$ and $t = 0.1$.

the right side of the boundary over time. Here μ acts as a parameter to determine the steepness of the wave, with $\mu = 0$ resulting in a discontinuity. For this test case, $\mu = 1/50$. Furthermore, the domain is chosen to be $[-2, 2]$ and Dirichlet boundary conditions are used. For the convective numerical flux, the Lax-Friedrichs flux scheme is used. This leads to the initial value problem with the following boundary conditions and initial solution

$$u(-2.0) = 2.0, \tag{4.6}$$
$$u(2.0) = 0.0, \tag{4.7}$$
$$u(x, 0) = 1.0 - \tanh\left(\frac{x}{2} \cdot 50\right). \tag{4.8}$$

This will be referred to as the travelling Burgers wave test case. Due to the finite numerical domain, the exact boundary conditions shown in (4.4) cannot be enforced. Due to this, the solution will deviate from the analytical solution shown in (4.5) if the travelling wave gets too close to the end points of the domain. Therefore, this test case was only simulated up until $t = 0.1$. The analytical solutions at $t = 0$ and 0.1 are shown in Figure 4.1. As shown, due to the short simulation time, the imposed Dirichlet boundary conditions at the end of the finite domain are expected not to influence the numerical solution, and the analytical solution can be used as a reference solution.

Time-integration schemes for the 1D solver

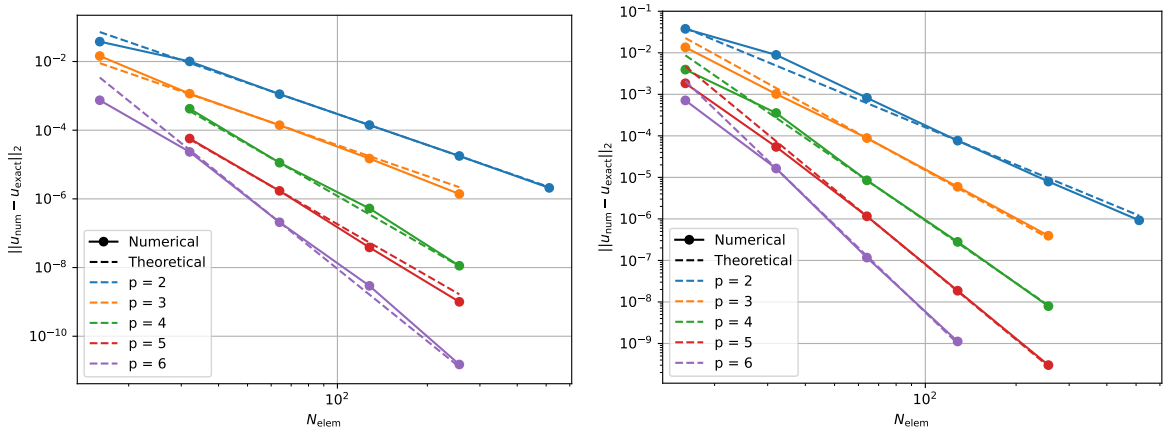
The implicit time-integration schemes implemented are the backwards Euler (BE) scheme, the SDIRK2-A method from [115], and the third-order four-stage (ESDIRK3) method as described in [116]. The ESDIRK2-A is denoted with "-A" because of the usage of a second, different SDIRK2 scheme for the TRACE solver. Although the solver was developed to investigate implicit time-integration schemes, ERK time-integration schemes were also implemented for comparison reasons. The explicit time-integration scheme implemented is "the" fourth-order Runge-Kutta scheme (ERK4) originally proposed by Kutta [117]. The Butcher tableaus for these schemes are listed in Appendix A.

4.2. Verification of the 1D Solver

This section will cover the numerical experiments performed to verify the proper working of the 1D implicit DGSEM solver. For this, a spatial and temporal convergence test has been performed.

4.2.1. Spatial Convergence Test

The spatial convergence of the 1D solver was verified using the travelling Burgers wave test case. Using this test case, simulations were performed with varying polynomial orders. The convergence



(a) BR1 **(b)** BR2

Figure 4.2: 1D Solver LGL-DGSEM spatial convergence rates for BR1 and BR2 viscous schemes on the travelling Burgers wave test case.

Table 4.1: Spatial convergence rates of LGL-DGSEM 1D solver with BR1 and BR2 viscous approximation schemes.

Scheme		$p = 2$	$p = 3$	$p = 4$	$p = 5$	$p = 6$
BR1	theor. rate	3.000	3.000	5.000	5.000	7.000
	meas. rate	3.040	3.328	4.983	5.367	6.880
BR2	theor. rate	3.000	4.000	5.000	6.000	7.000
	meas. rate	3.187	3.907	5.032	5.953	6.919

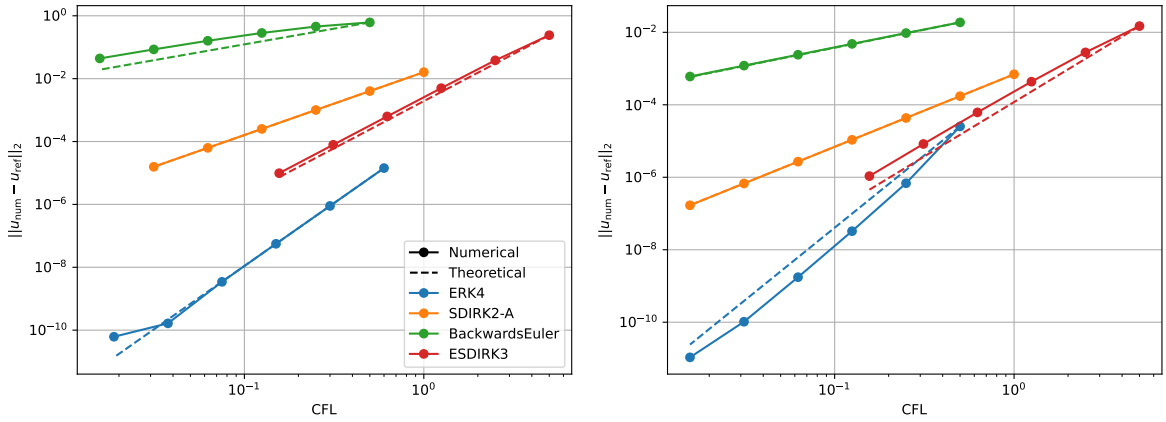
rate for each polynomial order was tested by increasing the number of elements, thereby refining the computational grid. In order to minimise temporal error, the ERK4 scheme with a CFL number of 0.01 was used. The L2 norm of the error between the numerical simulation and the analytical solution at $t = 0.1$ was measured by interpolating the numerical DGSEM solution onto a common grid with the analytical solution. This prevents numerical artefacts coming from the discontinuity between elements from polluting the error measurements.

This test case was done with both the BR1 and BR2 viscous schemes on the LGL-DGSEM. The results are shown in Figure 4.2 and the measured convergence rates are summarised in Table 4.1. For the computed convergence rates shown in this table, only the errors from the three finest grid resolutions for each polynomial degree are considered, as these correspond to the asymptotic range where the Burgers wave is well-resolved and the spatial convergence rate can be measured. Analysing the convergence rates, it can be seen that for the BR2 scheme, the measured convergence rate is $p + 1$ where p is the polynomial order. For the BR1 scheme, the measured convergence rate is $p + 1$ for even-order polynomials, but for odd-order polynomials, this is not obtained. Rather, the measured convergence rate is approximately p. This is in line with the expected behaviour of the BR1 and BR2 scheme as explained in Subsection 3.2.4.

Considering the measured convergence rates listed in Table 4.1, it can be observed that not the exact theoretical convergence rates are measured. Specifically for $p = 3$ and $p = 5$ on the BR1 scheme, the measured convergence rates are higher than expected. It is noted that the greater deviations in the convergence rates for the BR1 scheme could be caused by the instability of the BR1 method when no split-form stabilisation technique is applied, as shown by Gassner et al. [93]. Nonetheless, the general behaviour of the convergence rates shows the expected odd-even convergence behaviour.

4.2.2. Temporal Convergence Test

For the temporal convergence test, the linear advection sine wave test case and the travelling Burgers wave test case were used. For both the sine wave and travelling Burgers wave test cases, the numerical domain was discretised into 64 elements of polynomial order 2. Because the spatial error for this configuration dominates the temporal error, a numerical reference solution produced on the same



(a) Linear advection sine wave test case.

(b) Travelling Burgers wave test case.

Figure 4.3: Temporal convergence rates for varying time discretisation schemes with LGL-DGSEM.

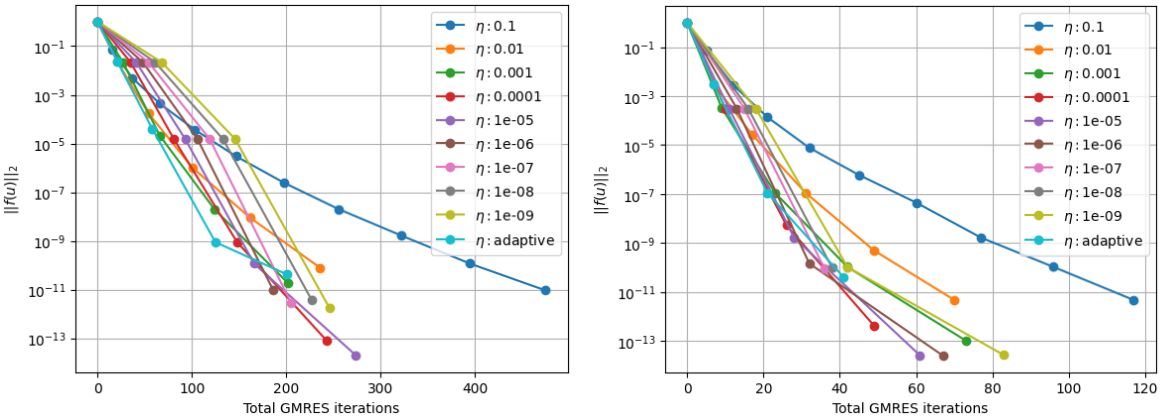
numerical grid with the ERK4 time-integration scheme at CFL = 0.01 was used rather than the analytical solution. Again, the L2 norm of the error was measured by interpolating both solutions onto a common grid. The convergence rates for the various time discretisation schemes implemented in the 1D solver are shown in Figure 4.3. As can be seen, the expected temporal convergence rates for the various implicit and explicit time discretisation schemes are obtained. Thus, verifying the temporal convergence of the 1D solver.

4.3. Numerical Experiments on the 1D Solver

With the temporal and spatial convergence of the solver verified, the 1D solver was further experimented with. Namely, an analysis on the inexact Newton method is given in Subsection 4.3.1. After this, in Subsection 4.3.2, the Jacobian sparsity patterns for the 1D solver on a simplified grid are analysed for several different configurations of the DGSEM spatial discretisation.

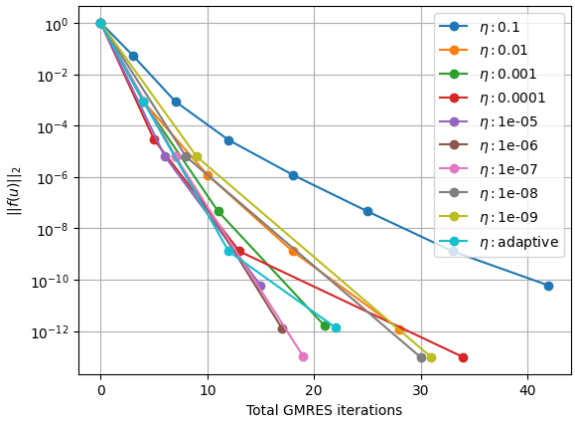
4.3.1. Inexact Newton Method with Adaptive Forcing Term

In this experiment, the effect of the forcing term η in the inexact Newton method as explained in Subsection 3.4.1 was explored. For this, the travelling Burgers test case was used. The test case was simulated for varying CFL numbers and varying fixed forcing terms η as well as the adaptive forcing term method described in Subsection 3.4.1. CFL numbers of 0.1, 1.0 and 10.0 were considered. For the forcing terms a range of η = [0.1, 1e-9] was considered as well as the adaptive forcing term. The simulation was performed for one outer solve of the inexact Newton method. Figure 4.4 shows the total GMRES iterations over the Newton residual $\|\boldsymbol{f}(\boldsymbol{u})\|$ that decreases towards the specified tolerance of 1e-10. Note that the symbols indicate the Newton steps, thus indicating the number of GMRES iterations per Newton step. From the results shown in Figure 4.4, the following observations were made. Firstly, for high CFL numbers such as 10.0 the tight η cases showed clear over-solving behaviour in the early Newton steps. This is shown in Figure 4.4a as with decreasing η, the amount of GMRES iterations increases, but the Newton residual stays constant for the first two Newton steps. The over-solving is reduced with decreasing CFL number, agreeing with the previous arguments made in Subsection 3.4.1. The η = 0.1 case for CFL of 10.0 is observed to show better convergence than the tighter tolerances in the first Newton steps, but worse convergence in later Newton steps. This indicates that the forcing term is required to become smaller as the Newton residual converges. This is confirmed in the results with the adaptive forcing term, which show a low amount of total GMRES iterations over the CFL range considered, as opposed to the fixed η terms. A second observation is that the adaptive forcing term is able to terminate the linear solve when the required Newton tolerance of 1e-10 is reached. This indicates that the additional modification to the forcing term to prevent over-solving in the final Newton steps also works as intended.



(a) CFL = 10.0

(b) CFL = 1.0



(c) CFL = 0.1

Figure 4.4: Newton residual $\|f(u^j)\|$ over total GMRES iterations for varying CFL numbers.

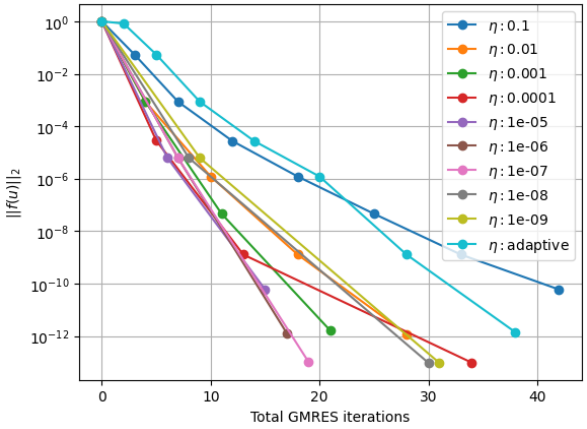


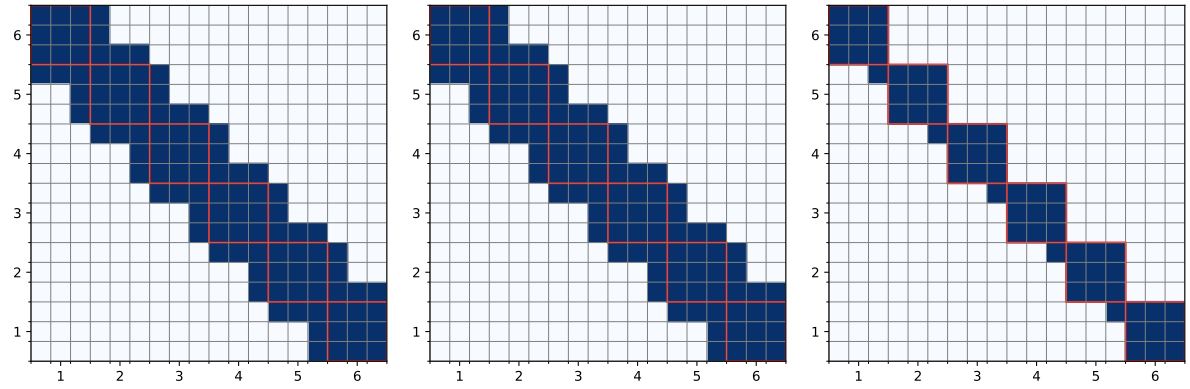
Figure 4.5: Newton residual $\|f(u^j)\|$ over total GMRES iterations for CFL = 0.1. The initial forcing term of the adaptive method has been set to $\eta_0 = 0.99$.

Discussion The numerical experiment of the inexact Newton method showed that the adaptive forcing term worked as desired and expected from the literature. However, one disadvantage of the adaptive forcing term method is that an initial forcing term η_0, as explained in Equation 3.71, must be manually specified. The disadvantage of having a fixed η_0 is that if it is not chosen properly, it will cost additional Newton steps in order for the method to adjust and reach an optimal η. If the total amount of Newton steps required is low, it can cause the adaptive method to be less efficient than the fixed methods. For the test case considered and other simulations done on the 1D solver in this study, an η_0 of 0.01 was considered. In Figure 4.5, the same Newton residual norm over GMRES iterations graph is shown for CFL of 0.1. For this case, η_0 was set to 0.99 for the adaptive forcing term method. What is observed is that in the initial Newton steps, the forcing term is high. Because the forcing term is only updated over the Newton steps, the result is that the adaptive forcing term takes a large number of GMRES iterations compared to the other fixed forcing terms. For the 1D test cases considered in this study, the initial forcing term of $\eta_0 = 0.01$ produced good results and thus was chosen. Nonetheless, this observed disadvantage of the adaptive forcing term method must be understood when using the method.

4.3.2. Jacobian Matrix Sparsity Pattern analysis

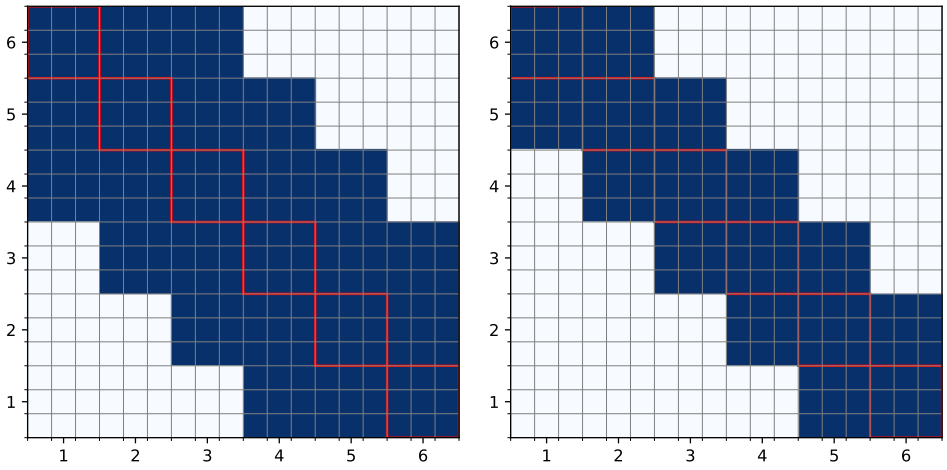
In the derivation of the DGSEM spatial discretisation in Subsection 3.2.4, it was derived that the usage of the BR1 method for the viscous terms would lead to a widening of the numerical stencil. Therefore, a preference of the BR2 scheme over the BR1 scheme for implicit solving has been found in literature due to the more compact stencil of the BR2 method [56, 23]. However, here it is often assumed that LG-DGSEM is used with Legendre-Gauss quadrature points. In this analysis, the Jacobian sparsity pattern will be analysed to see the effect of the BR1 and BR2 schemes on the numerical stencil of the DGSEM operator for both the LGL-DGSEM and LG-DGSEM.

The Jacobian matrix was constructed using the matrix-based GMRES method. The Jacobian matrix sparsity pattern for the BR1 and BR2 viscous schemes using the LGL nodes is shown in Figure 4.6. These Jacobian matrices correspond to a DG operator with 6 elements of polynomial order 2. This thus leads to six elements with three DOF per element. As can be seen in Figure 4.6, a clear structure of the block-diagonals can be observed in the Jacobian matrix. This is due to the local operating nature of the DG operator, overlaying the DOF per element indicated with the red outlines this structure is even better visible. Firstly, Figure 4.6c shows the sparsity pattern for the inviscid Burgers equation, using an upwind flux. Here the coupling of the upwind flux for the numerical advective flux function is clearly observed through the off-block-diagonal nonzero elements. Enabling the viscous terms for the viscous Burgers equation results in the sparsity patterns shown in Figure 4.6a and Figure 4.6b for the BR1 and BR2 schemes, respectively. Here, a clear change in the sparsity pattern caused by the viscous operator is observed. Firstly, every interior point now is coupled to the neighbour edge point because of the corrected gradient being used in the volume integral. The coupling of the edge points with the neighbouring interior points is caused by the diffusive numerical flux function using the gradient from



(a) LGL-BR1 **(b)** LGL-BR2 **(c)** Inviscid Burgers equation (upwind flux is used)

Figure 4.6: Jacobian matrix sparsity pattern for BR1 and BR2 viscous schemes with LGL nodes and inviscid Burgers equation. For six elements of polynomial order two.



(a) LG-BR1 **(b)** LG-BR2

Figure 4.7: Jacobian matrix sparsity pattern for BR1 and BR2 viscous schemes with LG nodes and inviscid Burgers equation. For six elements of polynomial order two.

both sides of the interfaces.

As can also be seen, there is no change in the sparsity pattern for the BR1 and BR2 schemes when using the LGL nodes. This confirms the results shown by [96], as also explained in Subsection 3.2.4. However, when using the LG nodes, a difference in Jacobian sparsity pattern can be observed between the BR1 and BR2 schemes, as is shown in Figure 4.7. What is observed is that the flux evaluations on the neighbouring elements are now dependent on all inner nodes of the neighbour, as can be seen by the widened stencil. This, in turn, also causes the predicted and observed difference in Jacobian sparsity pattern between the BR1 and BR2 methods when the LG nodes are used.

Lastly, the Jacobian matrix was also analysed for the case of a BR2 viscous scheme with LGL nodes and the ψ parameter as explained in Subsection 3.2.4 set to 1. With $\psi = 1$ it was found that the Jacobian matrix of the BR1 and BR2 method match exactly if LGL nodes are used, this aligns with Theorem four from [97].

4.4. The Matrix-free Approach

In this section, the matrix-free GMRES approach applied to the 1D solver is studied. As explained in Section 4.1, the matrix-free GMRES was implemented using a finite-difference approximation of the

Jacobian operator as described in Subsection 3.4.4. In this section, firstly, preconditioning methods for the matrix-based and matrix-free methods in the 1D solver will be described. After this, the performance of the matrix-based and matrix-free methods is analysed in terms of memory consumption and runtime. For all simulations considered in this section, the LGL-DGSEM method with a BR2 viscous scheme is used.

4.4.1. Preconditioning Approaches for the 1D Solver

The preconditioning method is a crucial component of solving large, sparse linear systems with the GMRES algorithm and therefore has to be chosen carefully. On the one hand, the preconditioner must approximate the Jacobian as well as possible. On the other hand, it must not become too expensive to construct and invert. Therefore, a balance must be found.

Matrix-based preconditioning approaches

For the matrix-based method, the following preconditioning methods have been considered.

The first matrix-based preconditioner considered was by performing an Incomplete Lower Upper (ILU) factorisation on the full Jacobian matrix constructed with no fill-in, commonly referred to as ILU(0). The ILU(0) is a common simple preconditioner that can be applied when the matrix is known. For the ILU(0) factorisation, the IncompleteLUT[2] class from the Eigen library was used. Where the fill factor has been set to one in order to prevent any additional fill-in.

The second preconditioner used was an element block-Jacobi (EBJ) preconditioner. This preconditioner follows from the observation that the Jacobian matrix sparsity pattern, analysed in Subsection 4.3.2, naturally follows a block diagonal pattern. Specifically for the 1D DGSEM operator, this block diagonal is dense. The idea of the EBJ preconditioner is to invert only the block diagonal elements corresponding to each element's DOF (indicated by the red outlines in Figure 4.6) independently. Because the diagonal blocks in the 1D method are dense, LU decompositions with the Eigen PartialPivLU[3] class are used. For higher-dimensional block-Jacobian matrices, ILU0 or higher fill-in ILU methods could be considered. With this, the inversions of the small diagonal block Jacobians are cheaper than the ILU0 inversion on the entire Jacobian matrix. A disadvantage of this method is that the off-diagonal elements are not included in the preconditioner. This results in a less accurate preconditioning matrix.

Matrix-free preconditioning approach

Preconditioning for matrix-free GMRES is challenging. This is because the matrix-free method aims to avoid constructing and storing expensive Jacobian matrices, but preconditioning methods often make use of an explicitly stored Jacobian, as already explained in Subsection 2.3.3.

For the 1D solver, it was decided to implement a matrix-free element block-Jacobi (MF-EBJ) preconditioner. This preconditioner works as follows. For every element, a local DGSEM operator is used to construct the diagonal block Jacobian for that element. The computation is done with a finite-difference approximation for every DOF within the element, just as the global explicit matrix of the MB-GMRES method is constructed. These block diagonal Jacobian matrices are then inverted using an LU decomposition just as is done for the MB-EBJ preconditioner. The usage of an MF-EBJ preconditioner is inspired by the work of Vangelatos [56], who applied a similar technique with an analytical block-Jacobi preconditioner for the DGSEM.

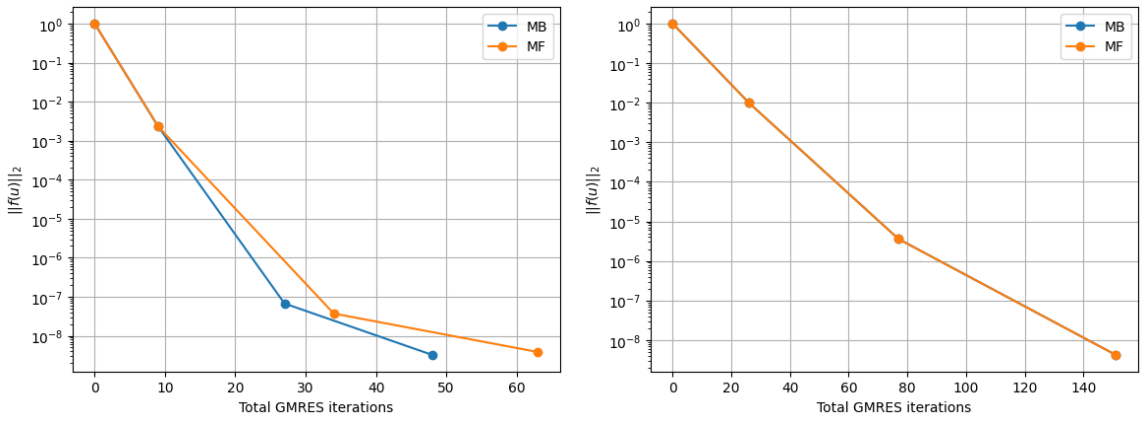
4.4.2. Comparison Matrix-based and Matrix-free Implicit DGSEM

In this section, the MB-GMRES and MF-GMRES methods are compared against each other in terms of convergence, simulation time and memory consumption. For the tests performed in this section, the travelling Burgers wave test case described in Subsection 4.1.2 is used.

Performance matrix-based GMRES and matrix-free GMRES on the 1D solver

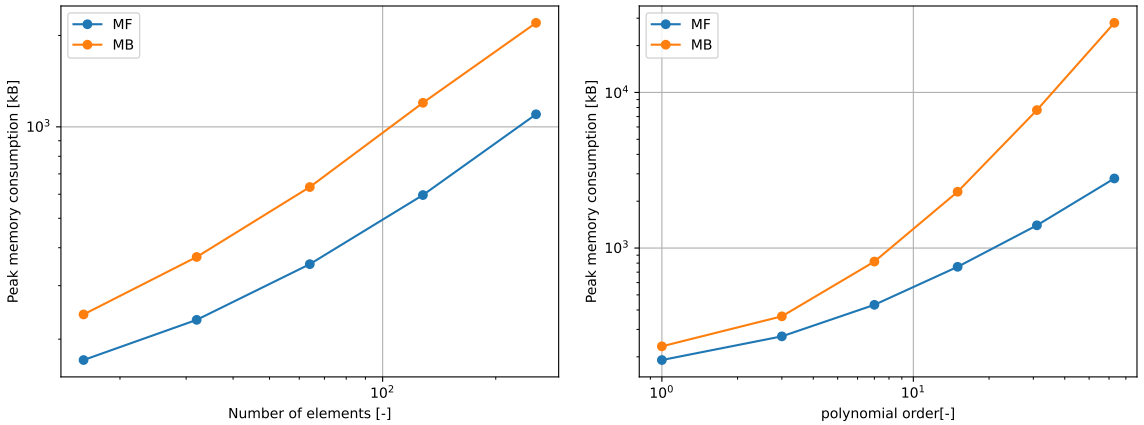
Firstly, the performance of the MB-GMRES and MF-GRMES algorithms with no preconditioner was analysed. For this test case, the LGL-DGSEM was used with 64 elements of polynomial order 5. A relative tolerance in the Newton solver of 1.0e-8 was used. In Figure 4.8, the convergence of the Newton residual over the total GMRES iterations is shown for one full Newton solve. The observed behaviour

[2] https://www.eigen.tuxfamily.org/dox/classEigen_1_1IncompleteLUT.html
[3] https://www.eigen.tuxfamily.org/dox/classEigen_1_1PartialPivLU.html



(a) CFL = 1.0 **(b)** CFL = 10.0

Figure 4.8: Convergence behaviour MF- and MB-GMRES for varying CFL number over Newton residual.



(a) Varying element count, polynomial order fixed at 5. **(b)** Varying polynomial order, element count fixed at 64.

Figure 4.9: Peak memory consumption of MB- and MF-GMRES solver over varying element count and polynomial order.

is that the MF-GMRES algorithm does not always produce the same convergence as the MB method. For this test case it is observed that for low CFL numbers, the MF method tends to have a higher total GMRES iteration count than the MB method. Whereas for higher CFL numbers, the MB and MF appear to match exactly in terms of Newton tolerance and GMRES iterations.

A possible reason for the difference in convergence behaviour between the MF and MB method, despite both using FD approximations to form the (either explicit or in matrix-vector product form) Jacobian, is that with the matrix-free approach, the pollution error is not constant with every GMRES iteration. Although in the matrix-based method the Jacobian is also constructed with the FD method and contains FD pollution errors, they are constant over the GMRES linear solve. For the matrix-free method, this is not the case. Hence, giving a reason for the difference in convergence behaviour between the two methods.

The peak memory consumption of the MB-GMRES and MF-GMRES methods was also analysed and compared. For the measurement of the memory consumption the Heaptrack[4] program was used. With this program, the peak memory consumption for an increasing number of elements and an increasing polynomial order was measured for the MB and MF methods. For the varying element count, a fixed polynomial order of 5 was used. For the varying polynomial order, a fixed element count of 64 was used. The results of these measurements are shown in Figure 4.9. From the memory consumption measurements, a clear pattern was observed. Namely, as expected, the MB method used more memory

[4] `https://github.com/KDE/heaptrack`

for all considered cases than the MF method. When analysing the memory usage of the MB method, it also showed that this increased memory was mostly caused by the (sparse) storage of the Jacobian matrix. Specifically, what was observed is that the two methods exhibit the same growth rate for an increasing number of elements. Although the MB-GMRES method approximately uses double the memory compared to the MF-GMRES method for all element counts considered. Whereas for an increase in polynomial order, there is a clear difference in growth rate between the two methods. The MB-GMRES method exhibits quadratic growth, whereas the MF-GMRES method exhibits linear growth. This is because the number of nonzero elements in the Jacobian matrix approximately grows as $(p+1)^2 N_{\text{elem}}$. This results in the explicit storage of the Jacobian matrix causing a quadratic growth rate of the memory consumption with an increase in polynomial order.

The non-preconditioned MB- and MF-GMRES methods were also simulated for CFL = 10.0 and a total simulation time of 1.0 seconds. The measured CPU runtimes are reported in Table 4.2 and Table 4.3 for change in element count and change in polynomial order, respectively.

Table 4.2: Total simulation CPU time for travelling Burgers wave test case over varying number of elements, CFL = 10.0. Polynomial order is fixed at 5.

	CPU time [ms]		
N_{elem}	16	32	64
MB	493	2125	15477
MF	191	680	2348

Table 4.3: Total simulation CPU time for travelling Burgers wave test case over varying polynomial order, CFL = 10.0. The number of elements is fixed at 64.

	CPU time [ms]		
p	1	2	4
MB	116	861	4548
MF	44	236	893

From the measured run times, it was observed that both methods, as expected, showed an increase in runtime with increasing element count or polynomial order. The simulation run time for the MB method increases at a higher rate than the MF method, agreeing with the observed behaviour for the memory consumption.

Discussion This analysis showed that the MB-GMRES was significantly slower than the MF-GMRES. The poor run-time performance of the MB-GMRES method can likely be attributed in part to the inefficient manner in which the Jacobian matrix is constructed in the MB-GMRES method. Namely, as said before, the Jacobian matrix is constructed with the finite-difference operator for every DOF independently, thereby filling the columns of the Jacobian matrix one by one. In other words, every DOF is perturbed individually, and the linearised gradient is computed with the finite-difference approximation.

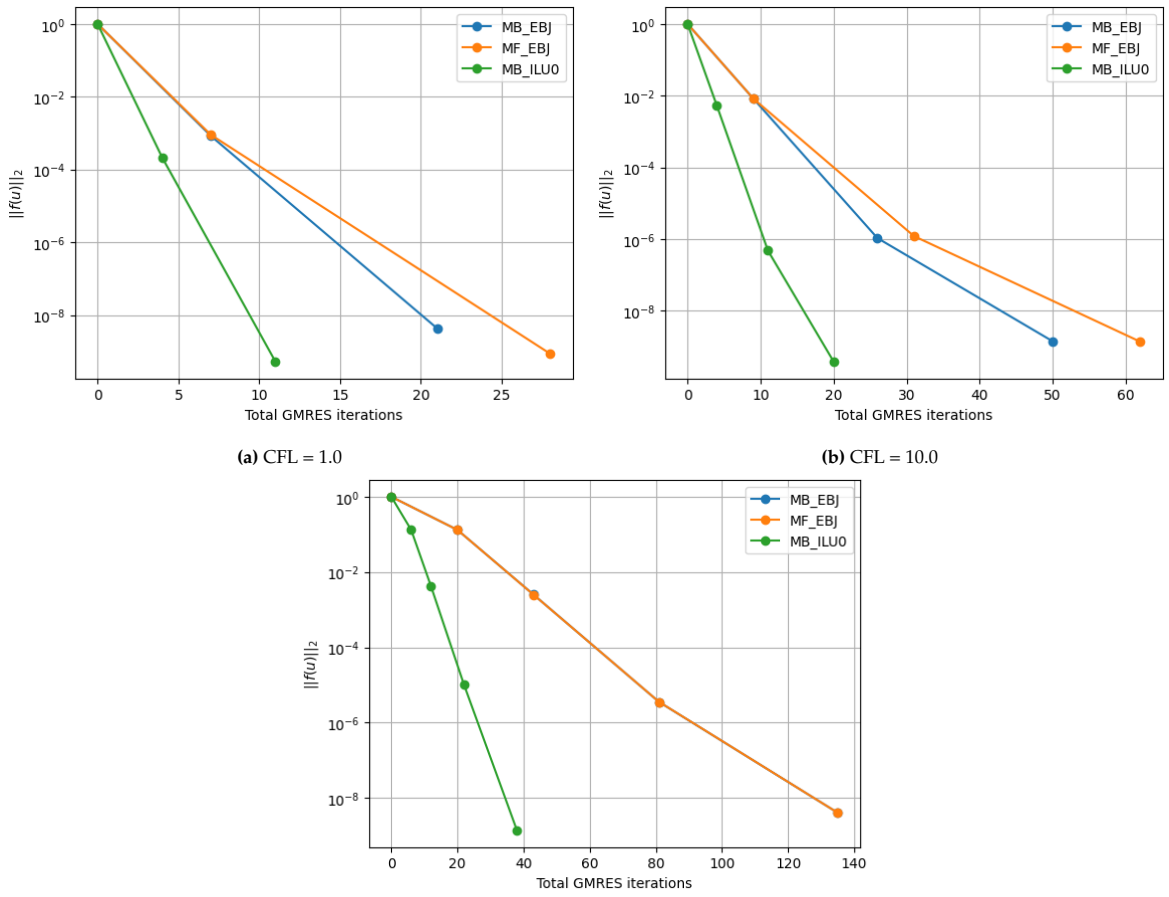
There are several ways this approach could be approved. Firstly, the amount of evaluations required to construct the Jacobian matrix with the FD operator could be reduced by making use of a colouring algorithm on the sparsity pattern of the Jacobian as described in [112]. This would preserve the advantage of using the complete Jacobian operator for the left-hand-side matrix. Secondly, the Jacobian matrix could be constructed more efficiently by deriving it analytically and filling in the matrix in code. This method has the lowest construction time, but finding the complete analytical Jacobian matrix is often impractical or impossible, and simplifying assumptions have to be applied, which makes the Jacobian inexact and possibly miss certain features of the spatial operator. This, in turn, may reduce the convergence of the GMRES linear solver.

In this analysis, the inefficient manner in which the MB-GMRES is implemented is acknowledged. Nonetheless, the observed memory consumption of the method and scaling provided insight into the comparison of the two methods.

Performance preconditioners

With the performance of the non-preconditioned MB- and MF-GMRES method established. The study was continued to analyse the effect of the preconditioners considered on the simulation run time, memory usage and GMRES iterations.

First of all, the convergence of the Newton method over the total GMRES iterations for a full Newton solve is shown for varying CFL numbers in Figure 4.10. First of all, for CFL = 1.0 and 10.0 the convergence and total amount of GMRES iterations can be compared to the non-preconditioned case shown in Figure 4.8.



(a) CFL = 1.0

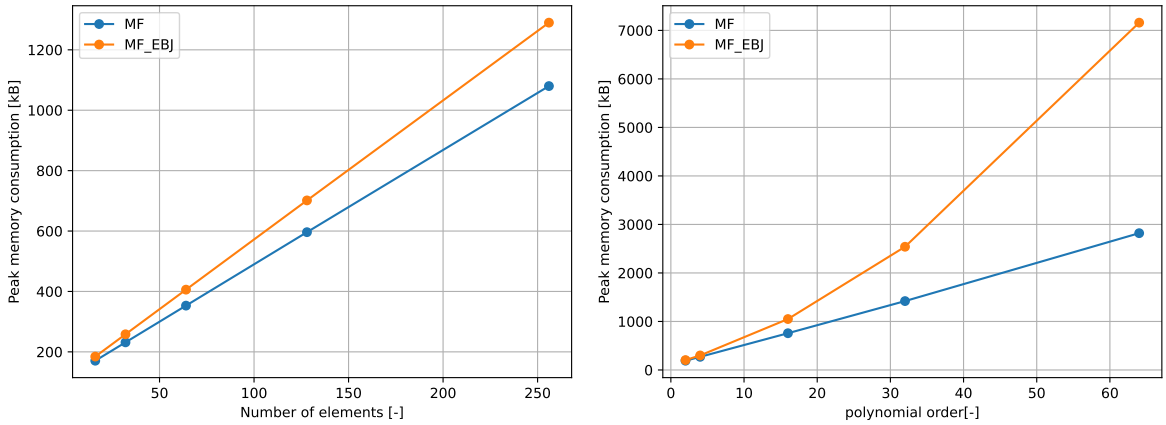
(b) CFL = 10.0

(c) CFL = 50.0

Figure 4.10: Convergence behaviour MF- and MB-GMRES for varying CFL number over Newton residual.

Here, it was observed that the preconditioners worked as intended and lowered the total number of GMRES iterations. Comparing the ILU0 and MB-EBJ preconditioners for the MB-GMRES method, the ILU0 preconditioner provides more effective preconditioning for both CFL numbers. This agrees with the observed characteristic of the EBJ preconditioner that it did not include off-block-diagonal elements, and thus provides a worse approximation of the inverse of the Jacobian matrix. Comparing the MF-EBJ and MB-EBJ preconditioners, it is observed that again the MF-GMRES method (with preconditioner now) exhibits larger GMRES iterations than the MB-GMRES method. Increasing the CFL number to 50.0, shown in Figure 4.10c, the two methods again reproduce the same GMRES and Newton iterations. Overall, it can be said that the effect of the MF-EBJ and MB-EBJ preconditioners is similar, and that the ILU0 preconditioner for the MB-GMRES method produces fewer GMRES iterations than the MB-EBJ method.

As for the non-preconditioned test case, full simulations with the various preconditioners were performed, and CPU times and memory consumption were measured. These results are shown in Table 4.4. Here it was observed that although the preconditioners reduced the GMRES iteration count, they did not reduce the simulation runtime. This indicates that for this test case considered, the cost of performing the preconditioning did not outweigh the reduction in GMRES iterations. More discussion on this observation will be done at the end of this section. Additionally, it was found that the preconditioning methods increased the peak memory consumption, as shown in Table 4.4. Specifically, the ILU0 preconditioner significantly increases the memory usage compared to the MB-EBJ preconditioner and MB-GMRES with no preconditioner. For the MF-GMRES method, it was observed that the preconditioner significantly increased the runtime for this test case. Although the memory usage did not increase significantly.



(a) Varying element count, polynomial order fixed at 5.

(b) Varying polynomial order, element count fixed at 64.

Figure 4.11: Peak memory consumption of MF-GMRES with no preconditioner and with MF-EBJ preconditioner over varying element count and polynomial order.

Table 4.4: Total simulation run times for travelling Burgers wave test case for various preconditioners. CFL = 10.0. A computational grid of 64 elements of polynomial order 5 is used.

	MB	MF	MB_EBJ	ILU0	MF_EBJ
CPU time [ms]	15477	2530	15485	15010	5298
peak mem. consumption [kB]	520.1	262.6	546.3	733.2	295.6

In order to analyse the effect of the MF-EBJ preconditioner on the memory usage over varying amounts of elements, a memory usage analysis was done and the result is shown in Figure 4.11. Here, it was observed that the cost of storing the local element block Jacobians was not significantly large as the memory consumption did not increase towards levels observed for the MB-GMRES method, which were analysed in Figure 4.9.

Preconditioner freezing In order to attempt to reduce the cost of preconditioning, a freezing of the preconditioners over the Newton solves was considered. Analysing the simulation run time and memory consumption for this frozen preconditioner test case, the results are shown in Table 4.5.

Table 4.5: Total simulation run times for travelling Burgers wave test case for various frozen preconditioners. CFL = 10.0. A computational grid of 64 elements of polynomial order 5 is used.

	MB_EBJ	ILU0	MF_EBJ
CPU time [ms]	15851	15636	2698
peak mem. consumption [kB]	546.3	733.2	295.6

It was observed that, specifically for the MF-GMRES method, freezing the preconditioner resulted in a significant reduction of the CPU runtime. This, however, was not the case for the MB-GMRES method. The difference is understood to be caused by the MB-GMRES method only requiring access to the already computed Jacobian in order to create the block diagonal Jacobian inverses. Hence the hard work has already been done in computing the Jacobian, which is done in every Newton step. For the MF-GMRES method, however, the MF-EBJ preconditioner needs to create the diagonal blocks from scratch.

Discussion

The comparisons of the preconditioners in this section were performed on the travelling Burgers wave test case. This test case was used to be consistent with the other numerical experiments. However, as already foreshadowed by the non-preconditioned methods converging and working well, the test case was found to not necessarily be a representative test case of the more challenging systems to be

solved in CFD. Namely, preconditioning for GMRES is required due to the system being solved being ill-conditioned. This is because the solution is smooth, the grid is evenly spaced, and over time, the solution only travels towards the right side. Printing out the Jacobian of the system for CFL = 10.0, $N = 64$ and $p = 5$ and computing the condition number of the Jacobian matrix using the Numpy linear algebra library[5] showed that the condition number was approximately 60 (defined with 2-norm). This showed that the test case was most probably too well-conditioned to test the power of the preconditioning methods. Nonetheless, the preconditioning methods did reduce the total GMRES iteration count, and the study was able to showcase the trade-off between the different preconditioners. As well as the effectiveness of the matrix-free preconditioning method in terms of CPU time and memory usage.

4.5. Conclusions 1D Solver

In this chapter, the implementation and analysis of a 1D implicit DGSEM solver was described. The 1D solver structure was first explained. After this, the code was verified for spatial and temporal convergence. Here, the expected convergence rates were obtained. The inexact Newton method was also analysed and showed that the adaptive forcing term was able to produce low GMRES iterations and CPU times for varying CFL numbers.

After this, the matrix-free approach was explored for the 1D solver. This started with the exploration of preconditioning approaches for both the MB-GMRES and MF-GMRES methods. For the MF-GMRES method, an element block-Jacobi (EBJ) preconditioner was constructed that made use of a local DGSEM operator.

Comparison of the performance of the 1D solver code for the MB- and MF-GMRES methods showed that the MB-GMRES method had significantly larger memory consumption compared to MF-GMRES, resulting in higher CPU run times for the simulations considered. Although it can be argued that the implementation of the MB-GMRES method was inefficient and that performance improvements are possible, this makes the comparison somewhat unfair. The preconditioning methods considered for the MB- and MF-GMRES methods were also analysed for performance. Here, it was found that although preconditioning lowered the GMRES iterations, the CPU run times did not decrease significantly, for both MB- and MF-GMRES. The culprit for this is most likely that the travelling Burgers wave test case considered is too well-conditioned and does not represent the tougher cases faced in industrial CFD applications.

[5] `https://numpy.org/doc/stable/reference/generated/numpy.linalg.cond.html`

5

Matrix-free Implicit Time-integration Scheme for DGSEM in TRACE

In the previous chapters, implicit time integration with the use of the matrix-free GMRES method was explored in theory and practice with the stand-alone 1D solver. In this chapter, the implementation of a prototype matrix-free implicit time-integration scheme for the DGSEM solver in the CFD code TRACE (Turbomachinery Research Aerodynamic Computational Environment) developed at DLR's Institute of Propulsion Technology is covered. TRACE is a CFD solver developed specifically for internal flow simulations of turbomachinery components, with ongoing development on the solver by DLR, industrial partners and universities for more than three decades. The solver is written in C/C++ and is capable of using a wide range of numerical methods to solve problems related to turbomachinery engineering. The implementation of this prototype solver allowed for the usage of readily available features in the TRACE code. The specific choices made on the implementation of the method in TRACE are reported in Section 5.1. Following this, the test cases used to validate and experiment with the solver are covered in Section 5.2. After this, the numerical experiments performed with these test cases are reported in Section 5.3. Finally, a discussion on the implementation of the solver in TRACE is given in Section 5.4.

5.1. Implementation Matrix-free DGSEM in TRACE

In this section, an overview of the implementation process of the matrix-free solver in TRACE is given. The overview serves to show how the TRACE code was used and modified to produce a working solver. Next to this, the main features and design choices of the implicit solver are presented and discussed.

5.1.1. Implementation Overview

The TRACE solver is an extensive code base that supports the usage of a wide range of numerical methods. For implicit schemes, such as (U)RANS or harmonic balance (HB), linear solving schemes such as LU-SGS or symmetric successive over-relaxation (SSOR) are commonly used paired with a pseudo-time stepping Newton method. These schemes have been preferred over GMRES as the Jacobian matrix is analytically built and is an approximation to the real Jacobian operator, as contributions from boundary conditions are neglected, and other simplifications are made. Nonetheless, GMRES is also available as a linear solver in TRACE, but as a matrix-based solver. Furthermore, for the DGSEM, only a matrix-based prototype implicit solver that worked with the inviscid compressible Euler equations was present. For this project, the goal was to implement an implicit solving scheme for the LGL-DGSEM using a matrix-free GMRES method. Usage of such a method applied to the DGSEM was not done in TRACE before. Fortunately, the sparse linear system solver (Spliss) developed by DLR [118] offered matrix-free operator capabilities and had also been applied to replace the legacy linear solvers in TRACE. Which meant that code and experience with connecting TRACE to Spliss were already present in the TRACE community. Therefore, the Spliss linear solver library was chosen.

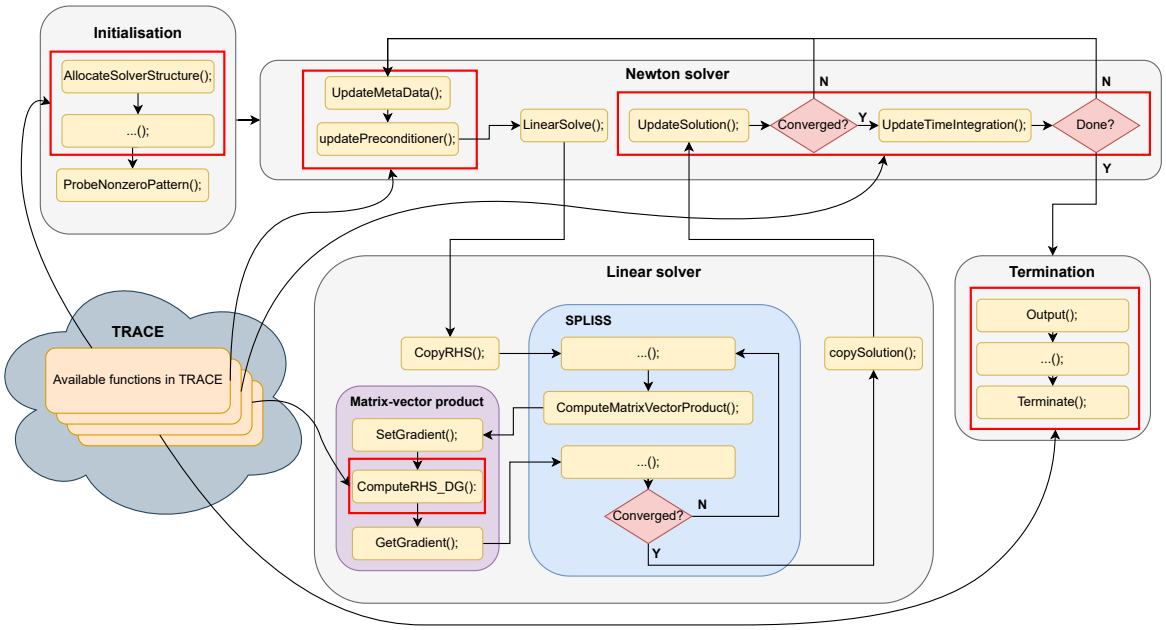


Figure 5.1: Overview of implementation of matrix-free GMRES solver logic in TRACE.

The usage of matrix-free GMRES meant that the linear solver must be passed a function that can evaluate the matrix-vector product of the Jacobian A and the "search" vector v. In order to do this in a matrix-free manner, using either the FD or AD differentiation techniques described in Subsection 3.4.4, an operator to evaluate the right-hand side operator $f(u)$ must be specified. For implicit time integration, the existing code in TRACE is based on a stack of functions that perform all operations necessary to construct the right- and left-hand sides of the system, which are then solved by the linear solver, before further function calls in the stack update the solution. This stack of functions is then iteratively called to perform the Newton solve. However, the sequence in which function calls are performed is not straightforward due to the code being optimised for parallel performance. This causes the different evaluations required (e.g. volume integrals for DG) to be organised in such a manner that, for instance, non-blocking communications can be utilised. Furthermore, this code structure also contains many functionalities which are not of interest for the prototype solver (e.g. combustion modelling, aerolasticity, mesh deformation).

As the solver was aimed to be a prototype, it was decided not to add the matrix-free solver to the existing solver logic. But instead, to use the existing functions in a new Newton solver loop. This allowed for functions not of interest in the project to be ignored, reducing implementation complexity and sources of possible bugs. A broad overview of how this Newton solver algorithm works, and how the existing TRACE functions were utilised, is shown in Figure 5.1. It must be noted that the functions displayed only serve as an example, and do not directly display all functions inside the solver. As the figure shows, the stack of available functions in TRACE was reorganised to fit into the matrix-free linear solver structure. Things such as memory allocation remained the same. Similarly, steps in the Newton solver, such as updating metadata, were also unchanged. Prior to the linear solve, the preconditioner is also updated if necessary, and these functions also had to be added. Contrary to matrix-based linear solvers, the linear solver must be passed an operator that can evaluate the matrix-vector product in a matrix-free manner. This operator is denoted with the matrix-vector product box and requires the specification of an operator that evaluates the right-hand side of the system. This right-hand side operator was constructed by piecing together the corresponding existing functions. These functions were also modified to ensure that they did not change any other variables. The representative function is named `computeRHS_DG()`. In order to transfer data to and from Spliss, functions were developed that copied the right-hand side to the linear solver ($f(u^j)$ in Algorithm 2 shown in Subsection 3.4.2) and similarly also copied the solution of the linear solve from Spliss to TRACE. Functions to control the Newton solver and time integration from TRACE were also directly used in this solver, and so were the functions regarding the termination of the TRACE solver.

Although the overview of the implementation of the matrix-free solver is rather broad, the following key aspects are noted. Instead of adding to the existing solver logic, pieces of the TRACE solver were used in a separate Newton loop for prototyping purposes. The Spliss linear solver library is used for the matrix-free GMRES solving. The largest implementation efforts required were the restructuring of the TRACE solver functions to fit into the Newton solver loop, as well as coupling Spliss with TRACE for the DGSEM method.

5.1.2. Solver Method Overview

Here, the choice and features of the nonlinear and linear solving methods will be described.

The first point of attention is the configuration of the spatial DGSEM discretisation. In this project, the implicit solver was only applied to the LGL-DGSEM paired with the Kennedy-Gruber split-form stabilisation [41] method with a BR2 method for the viscous terms. Although the LG-DGSEM method is also implemented in TRACE, it cannot be used together with split-form methods. Furthermore, other split-form schemes are also available in TRACE, but the implicit solver was not tested on these split-form schemes due to time limitations. Furthermore, in this project, LES without any sub-grid-scale models is considered, meaning that the only source of dissipation is from the spatial and time discretisations. This is commonly referred to as implicit LES, as the sub-grid-scale model is implicitly included in the equations by the numerical discretisation [119].

For the nonlinear solver, the inexact Newton method is used. In contrast to the inexact Newton method in the 1D solver, in this case, only fixed Newton tolerances are used rather than the adaptive forcing term that was explored for the 1D solver. The choice not to include the adaptive forcing term was made to limit the size of the project. Because the existing Newton solvers in TRACE are all pseudo-transient Newton solvers, the nonlinear solver is also capable of pseudo-time stepping. However, for all test cases considered in this project this was not used, and the pseudo-time stepping was turned off.

As explained, the Spliss library [118] was used to perform the linear solve. The tolerance for the linear solve is imposed by the inexact Newton method. The linear solver used is a matrix-free restarted GMRES method. For the matrix-vector product operators, both the finite-difference (FD) and (forward) automatic differentiation (AD) approaches were implemented. For the FD differentiation, the same choice of perturbation parameter ϵ as shown in (3.88) was chosen, and the overall implementation followed similarly to what was done for the 1D solver.

Forward automatic differentiation in TRACE The decision to implement both the FD and AD methods was based on the research goals set at the start of the thesis. For the AD method, the CodiPack [84] code differentiation package was used. It must be stated that the previous development of AD in TRACE allowed for an easy implementation of a forward AD solver for this project. More details regarding this implementation can be read in [82, 81]. Without this implementation of AD in TRACE, usage of the method would not have been possible in this project due to its complexity. In essence, AD is applied to TRACE and Spliss by using operator overloading, meaning that data types are overloaded with their AD counterparts (e.g. using `codi::RealForward` instead of `double`) and arithmetic operators such as $+, -, /$ are overloaded to apply the derivative propagation as was explained in Subsection 3.4.4. For usage of the forward AD method, the gradients originating from the Spliss solver (i.e. the v matrix in the Av matrix-vector product) were inserted with use of the `.SetGradient()` command and similarly the matrix-vector product was obtained by using the `.GetGradient()` command after the right-hand side evaluation was performed. This resulted in a clear and straightforward implementation of the forward AD matrix-vector product operator.

Preconditioning For the preconditioning method of the GMRES solver, the ILU0 method was used. The Spliss solver had the ILU(0) preconditioning feature readily available, and the only requirement was that the preconditioning matrix was fed to the linear solver. In order to construct the preconditioning matrix, the matrix-free solver probes a Jacobian nonzero pattern of the process local domain(s). Meaning that contributions from other processors are not considered, this was chosen to avoid parallel communication during the construction of the preconditioner as a first step in the implementation process. As also said in the previous analysis of the Jacobian nonzero pattern, this pattern represents a dependency graph of the operator over the DOF. With this pattern, DOFs that are independent of each other can be identified.

This leads to grouping of DOF into different *colour* groups. All DOF in a colour group can be evaluated within the same function evaluation without affecting other DOF in the same group. The Spliss solver, in turn, is able to use this colouring to efficiently evaluate or update the preconditioning matrix, which it then performs an ILU(0) inversion on. In this case, a partial distance-2 colouring is computed using a greedy algorithm. More information on this method and Jacobian matrix colouring in general can be found in [111, 112]. A very important note is that the implementation of this colouring algorithm was already present in TRACE. All credits regarding this work are attributed to Dr. Jan Backhaus. The Spliss solver also allowed other preconditioning methods to be used. Specifically, ILU(p) preconditioning was also made available, which used the same Jacobian matrix generator as described before. Next to this, an inner GMRES preconditioning method was also readily available in the Spliss solver library and was therefore made available. For the test cases considered in this project, however, only the ILU(0) preconditioning method was considered. This was chosen because the literature review performed suggested that an ILU(0) preconditioner would be the most performant preconditioner out of the three methods available at that point in time, and due to the limited time available.

5.2. Setup and Test Cases TRACE Solver

In this section, the test cases and setup used for the numerical experiments performed on the implicit DGSEM solver are described.

5.2.1. Isentropic Vortex Advection Test Case

The first test case considered is the advection of an isentropic vortex. This is a 2D test case using the Euler equations, and enabled an initial verification on the solving scheme before including viscous terms into the system of equations. The test case involves the advection of a vortex through a rectangular domain with periodic boundary conditions in all directions. This test case, therefore, tests the ability of the solver to correctly advect vortical flow structures.

The initial conditions are created with perturbations based on a Gaussian function

$$\Omega = \beta e^{f} \tag{5.1}$$

$$f = -\frac{1}{2\sigma^2}\left[\left(\frac{x}{R}\right)^2 + \left(\frac{y}{R}\right)^2\right]. \tag{5.2}$$

This perturbation was then used to construct an initial solution field defined by the functions shown in (5.3) to (5.10), that correspond to an isentropic vortex moving with a free-stream Mach number in the x-direction. The parameters chosen are summarised in Table 5.1. These parameters are commonly used for high-order workshops, such as [3]. Only the size of the vortex relative to the domain was increased to reduce computational costs. A rectangular domain of 0.1×0.1 [m] was used. In [120], a detailed survey of the different variants of this problem is described. A visualisation of the initial pressure field is shown in Figure 5.2.

$$\delta T = -\frac{\gamma - 1}{2}\Omega^2 \tag{5.3}$$

$$T = T_\infty(1 + \delta T) \tag{5.4}$$

$$\rho = \rho_\infty\,(1 + \delta T)^{\frac{1}{\gamma - 1}} \tag{5.5}$$

$$\delta u = \frac{-y}{R}\Omega \tag{5.6}$$

$$\delta v = \frac{x}{R}\Omega \tag{5.7}$$

$$u = (M_\infty + \delta u)\,a_\infty \tag{5.8}$$

$$v = (\delta v)\,a_\infty \tag{5.9}$$

$$p = \frac{p_\infty}{\gamma}\,(1 + \delta T)^{\frac{\gamma}{\gamma - 1}} \tag{5.10}$$

Table 5.1: Parameters chosen for isentropic vortex advection test case.

α	0[°]
M_∞	0.5 [-]
p_∞	100 [kPa]
T_∞	300 [K]
R_{gas}	287.15 $[\frac{\text{J}}{\text{kg·K}}]$
R	0.01
σ	1
β	0.2

For the spatial discretisation in this test case, only one configuration was used. Namely, a two-

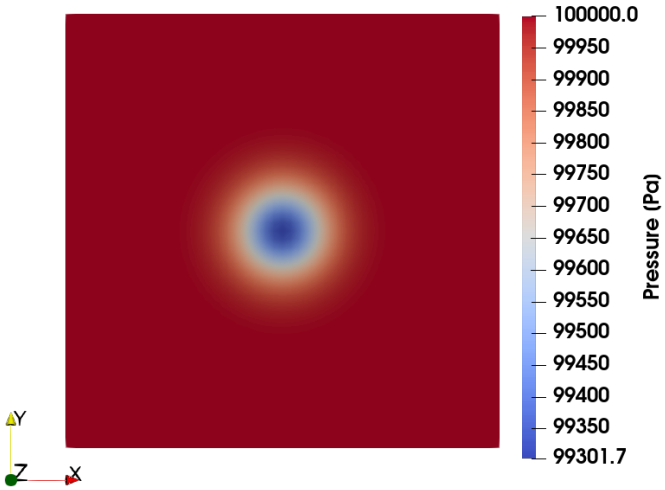


Figure 5.2: Visualisation of initial pressure field for isentropic vortex advection test case.

dimensional mesh consisting of 12×12 elements of polynomial order 6, resulting in a total of 84×84 DOF[1].

5.2.2. Laminar and Turbulent Channel Flow Test Case

In order to verify the working of the implicit solver using the Navier-Stokes equations, a channel flow simulation was used. For this channel flow, two configurations at different friction Reynolds numbers were used. The friction Reynolds number is defined as

$$Re_\tau = \frac{\bar{\rho}\tilde{u}_\tau}{\bar{\mu}}. \tag{5.11}$$

Where $\bar{\rho}$ and $\bar{\mu}$ are the bulk density and molecular viscosity, respectively. $\tilde{u}_\tau$ Is the Favre-averaged friction velocity as described in (5.17). Periodic boundary conditions are applied in the stream- (x) and span-wise (z) directions. In the wall-normal direction no-slip wall boundary conditions are used. The first test case considered was a laminar channel flow test case with $Re_\tau = 50$, and secondly, a turbulent channel flow test case with $Re_\tau = 395$ was used.

The computational domain is defined as $2\pi\delta \times 2\delta \times \pi\delta$ with $\delta = 0.006$ [m]. The Mach number was chosen to be M = 0.1 [-]. In the wall normal direction, the element sizes were stretched with relation to the distance of the wall using

$$y_i = \frac{1}{a}\tanh\left(\left(\frac{2i}{n} - 1\right)\operatorname{arctanh}(a)\right), \qquad i = 0, 1, ..., N, \tag{5.12}$$

with $a = 0.995$ being used. Equal element spacing was used in the other directions. For this test case, three computational domains were used. Firstly, the LCF_50 laminar channel flow test case at $Re_\tau = 50$ used a mesh of $8 \times 8 \times 8$ elements of polynomial order 3, resulting in 32^3 DOF. For the turbulent channel flow, two different spatial discretisations were used. Firstly, the TCF_395 case, which consisted of $16 \times 16 \times 16$ elements of polynomial order 3. And secondly, the TCF_395-fine case that consisted of $24 \times 24 \times 24$ elements of polynomial order 3. The properties of the three spatial discretisations are summarised in Table 5.2.

The simulations were initialised by imposing a 1D solution obtained from a steady RANS simulation onto the 3D initial flow field. The RANS simulation was performed using the FV solver in TRACE. For the laminar test case, this RANS simulation had no turbulence model. For the turbulent test case, a

[1]TRACE is a three-dimensional solver, so a mesh of $12 \times 12 \times 1$ elements results in DGSEM degrees of freedom (DOFs) still being present in the third dimension. However, because only one element is used in this direction, TRACE effectively behaves as a two-dimensional solver.

Table 5.2: Computational domains considered for the channel flow test case.

Name	Re_τ	p	N_{elem}	DOF
LCF_50	50.0	3	8^3	32^3
TCF_395	395.0	3	16^3	64^3
TCF_395-fine	395.0	3	24^3	96^3

RANS simulation with the Menter SST k-ω [121] turbulence model, as implemented in [122], was used. Next to this, for the turbulent test case, the initial flow field obtained by RANS was further modified to add synthetically generated turbulent velocity fluctuations to the flow field, reducing the required initialisation of the simulations to produce fully developed turbulent flow. In order to drive the flow to the specified friction Reynolds number, a constant body force source term in the streamwise direction is added to the momentum equation.

For the laminar test case of $Re_\tau = 50$, an analytical velocity profile could be derived as a reference solution. Namely, following (7.10) and (7.13) from [123], the streamwise velocity profile over the channel height, $U(y)$, of a laminar channel flow can be described using

$$U(y) = \frac{\tau_w \delta}{2\rho\nu} \frac{y}{\delta} \left(2 - \frac{y}{\delta}\right). \tag{5.13}$$

Where δ is the half-channel height, ρ is the bulk density, ν is the kinematic viscosity, and τ_w is the shear wall stress, which is determined by the pressure gradient enforced

$$-\frac{\mathrm{d}p_x}{\mathrm{d}x} = \frac{\tau_w}{\delta}. \tag{5.14}$$

Writing (5.13) in term of the friction Reynolds number $Re_\tau = 2u_\tau\delta/\nu$, friction velocity $U^+ = U/u_\tau$ and wall units $y^+ = yu_\tau/\nu$ the following is obtained

$$U^+(y^+) = y^+ - \frac{(y^+)^2}{2Re_\tau}. \tag{5.15}$$

5.2.3. Time-integration Schemes for the TRACE Solver

The time-integration schemes that were present in the TRACE solver were used. Namely, for the implicit solver, the first-order backward Euler (BE), the second-order two-stage SDIRK scheme (SDIRK2-B) from [124], the third-order three-stage SDIRK scheme (SDIRK3) from [125] and the fourth-order five-stage ESDIRK scheme (ESDIRK4) from [126] have been used. The first-order, second-order and third-order implicit schemes are solely used for the temporal convergence order test in Subsection 5.3.1. Only the ESDIRK4 scheme is considered for the LCF_50 and TCF_395 test cases. This is because of the observed efficiency of the fourth-order scheme for implicit time integration in literature as discussed in Section 2.2. Although the fifth-order eight-stage ARK5(4)8L[2]SA-ESDIRK scheme developed by [127] was also available in TRACE, the increase of stage numbers from five in the fourth-order scheme to eight in the fifth-order scheme was expected to reduce the numerical efficiency of the method, and therefore it was not used.

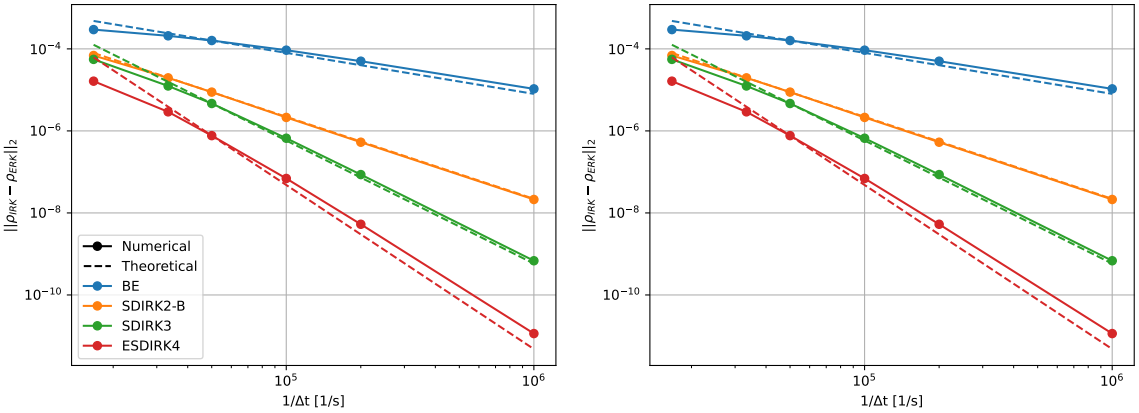
For the explicit solver, the third-order three-stage ERK scheme (ERK3) from [128] and "the" fourth-order Runge-Kutta scheme (ERK4) [117] were used. The ERK4 scheme was used for the LCF_50 test case in order to converge to the steady solution faster, although it is expected that the ERK3 scheme also would have provided the same result. For the unsteady TCF_395 test cases, the ERK3 scheme is used. This explicit time-integration scheme is the standard for unsteady LES with DGSEM in TRACE. The time-integration schemes are summarised in Table 5.3. The complete Butcher tableaus of these schemes are listed in Appendix A.

5.3. Numerical Experiments on Implicit DGSEM Solver in TRACE

In this section, the numerical experiments performed with the previously defined test cases are reported. In these experiments, the working of the implicit DGSEM solver is tested by comparison with experimental results and with results from the existing explicit DGSEM solver in TRACE.

Table 5.3: Time-integration schemes used in the TRACE solver.

Name	Type	Order	Stages
BE	Implicit	1	1
SDIRK2-B [124]	Implicit	2	2
SDIRK3 [125]	Implicit	3	3
ESDIRK4 [126]	Implicit	4	5
ERK3 [128]	Explicit	3	3
ERK4 [117]	Explicit	4	4



(a) FD **(b)** AD

Figure 5.3: Temporal convergence rates of backwards Euler (BE), SDIRK2-B, SDIRK3, and ESDIRK4 as specified in Appendix A for FD implicit solver (left) and AD implicit solver (right).

5.3.1. Temporal Convergence Rate Test

The first numerical experiment performed was to validate the temporal convergence rates of the implicit solver applied to existing SDIRK and ESDIRK integration schemes in TRACE. To perform this validation, the isentropic vortex advection test case described in Subsection 5.2.1 is used.

Although other usages of the isentropic vortex advection considered multiple vortex advections through the domain [120], for this test case the vortex was only propagated for 10% of a full advection through the domain. In order to neglect spatial discretisation errors, a solution obtained with the explicit DGSEM solver in TRACE using the ERK4 scheme was used as a reference solution. A fine time-step size was chosen for this explicit solution in order to ensure low temporal discretisation error. Next to this, the relatively fine spatial discretisation also ensured that the spatial discretisation error would not dominate the measured errors.

The results are shown in Figure 5.3. Both the implicit solver with AD- and FD-based differentiation show the same convergence results. As can be seen, the expected rates of convergence for the different temporal schemes are reached. This experiment verified the correct working of the solver with existing time-integration schemes in TRACE.

5.3.2. Performance Analysis FD and AD Matrix-free Differentiation Methods

With initial verification of the solver performed, the isentropic vortex advection test case described in Subsection 5.2.1 was again used to analyse the performance differences between the AD and FD differentiation methods.

The first point of interest was to evaluate the performance of the two methods when they both performed the same number of numerical operations. The performance of the solvers was measured by simulating over a fixed number of time steps, with a fixed number of Newton iterations per time step, as well as a fixed number of GMRES iterations per Newton solve. This test case was then performed using 3, 6, 12, and 24 CPUs to also analyse the scaling of both methods. For these tests, the preconditioning matrix was evaluated once at the start and frozen over the entire simulation. The tests were repeated,

Table 5.4: Performance evaluation of FD and AD differentiation methods applied to the 2D inviscid vortex advection test case. Both methods performed the exact same number of Newton steps and GMRES iterations.

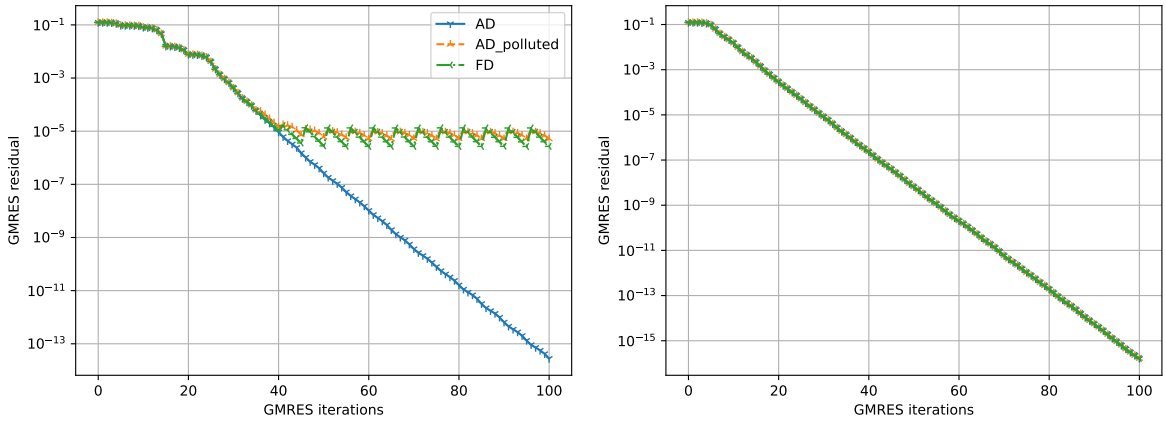
CPU	wall time [s]		scaling		peak mem. per CPU [MB]		$t_{wall,AD}/t_{wall,FD}$
Method	AD	FD	AD	FD	AD	FD	-
3	351.6	764.3	1.00	1.00	443.4	315.1	2.17
6	188.9	424.1	1.80	1.86	235.2	173.2	2.25
12	109.5	233.3	3.28	3.21	140.0	106.5	2.13
24	72.7	167.5	4.56	4.84	96.7	77.2	2.30

and average wall-clock times and peak memory consumption per CPU were measured. The results are shown in Table 5.4. The results showed that, on average, for this test case, the AD method was ≈ 2.2 times slower than the FD method. This aligns with the expected factor of approximately ≈ 2.0 that is based on AD variables requiring twice the amount of numerical evaluations due to the operator overloading. Therefore, if the same amount of numerical operations are performed, this factor should be expected. Looking at the peak memory consumption, it was found that the AD method consumes more memory per CPU on average, with the ratio between the AD and FD method being largest for 3 cores and smallest for 24 cores. The decrease in memory consumption relative to the FD method with increasing core count is understood to be caused by the metadata of the solver being of significant size compared to matrices and solution vectors. With increasing core count, every processor has to store a smaller part of the domain. Whereas the metadata for the solver probably does not decrease in size. This causes the ratio of memory consumption between the FD and AD methods to come closer to each other. Again, such observations are made with a fixed number of GMRES iterations between both methods. For example, an increase of the Krylov subspace could mean greater memory consumption for either of the methods in a practical use case.

Accuracy FD and AD differentiation methods A known disadvantage of the FD method is that the derivative obtained is a first-order approximation and contains truncation errors and possible round-off errors. The question remains, however, if this error is significant enough to affect the linear solver to an extent that more precise differentiation methods, such as forward AD, are necessary. This is a very broad question that also depends on the test case considered. Here, a small analysis regarding the effect of the pollution error for the FD method compared to the exact AD method is described.

While experimenting with the solver, it was observed that the FD method exhibited stagnation when the restarted GMRES was used. As an example, the isentropic vortex advection test case was again used. Two different GMRES configurations were used. One with a maximum Krylov subspace of $m = 5$ and a maximum number of GMRES iterations of 100 such that the method was forced to restart every 5 GMRES iterations. The other GMRES method imposed a maximum Krylov subspace of $m = 100$ with a maximum number of GMRES iterations of 100, such that no GMRES restarts were enforced. In order to analyse the effect of the FD error on the linear solve, the FD and AD differentiation methods were used. Next to this, a modified AD method that added a random pollution error that was of a similar size to the expected FD error was used. The results are shown in Figure 5.4. The modified AD method is denoted with AD_polluted. Comparing the FD and AD methods, the stagnation that was described before can clearly be seen for the restarted GMRES shown in Figure 5.4a. The verification that this stagnation is caused by the truncation error of the FD method is confirmed by the behaviour of the AD_polluted method, which shows a similar stagnation behaviour as the FD method. Furthermore, the residual value range at which the stagnation occurred for the AD_polluted method could be controlled by increasing and decreasing the random noise added to the AD gradient. Interestingly, the observed stagnation is not observed for non-restarted GMRES as shown in Figure 5.4b. Lastly, it is observed that for the AD method, the reduced Krylov subspace used by the restarted GMRES influences the convergence of the method over the 100 linear iterations considered when compared to the non-restarted GMRES. This is the expected behaviour of the method, as the increased Krylov subspace increases the accuracy of the GMRES approximation.

The observed stagnation of the restarted GMRES method is explained by the following hypothesis. When building the Krylov subspace, the GMRES method evaluates the linear residual based on the projected residual resulting from the Givens rotation (y_{j+1} in Algorithm 2 shown in Subsection 3.4.2).



(a) Restarted GMRES with $m = 5$ and maximum GMRES iterations of 100.

(b) GMRES with $m = 100$ and maximum GMRES iterations of 100.

Figure 5.4: GMRES residuals over a single linear solve for restarted GMRES (left) and GMRES with no restart(right) for FD, AD differentiation methods.

The GMRES algorithm assumes that firstly, the Jacobian A is a linear operator and secondly, that A does not change. By using the FD method, both of these assumptions are not strictly adhered to. This means that the projected residual using the Givens rotation is not equal to the true residual

$$\|\beta \boldsymbol{v}_1 - \boldsymbol{h}_m y_m\| \neq \| - f(\boldsymbol{u}^j) - A\Delta \boldsymbol{u}^j_{r+1}\|. \tag{5.16}$$

Indeed, the operator fed to the GMRES $\tilde{A}$ is an approximation to the actual operator A in the case of the finite-difference method. The Krylov subspace being built uses $\tilde{A}$ and thus also converges to the solution for the system of $\tilde{A}\boldsymbol{x} = \boldsymbol{b}$. Interestingly, although the matrix-vector product is not consistent because the added truncation error is random, the projected residual still converges to machine precision in the case of non-restarted GMRES. In [129], the convergence of GMRES with an inexact operator is proven. It is thus understood that the method can converge to the solution until the exactness of the operator has been reached. Hence, when the restarted GMRES re-evaluates the residual and the solution is close enough that the round-off error starts to significantly pollute the Jacobian evaluations, the GMRES method stagnates. It is therefore understood that the solutions of the non-restarted GMRES with the FD (and also AD_polluted) method are also only correct up until the observed truncation error for the restarted GMRES. To further support the given hypothesis, when increasing the GMRES iterations from 100 to 200, the AD method was also observed to hit a stagnation when the drop in linear residual reached machine precision, as shown in Figure 5.5. This shows that the AD method also has a random round-off error at machine precision, which causes the projected residual to be unequal to the real residual. The accuracy analysis clearly shows the disadvantage of the FD method. However, the question remains whether this flaw actually influences the practical usage of the method. This mostly depends on how tight the linear solves are required to be to achieve the most efficient Newton solve, which varies with the conditions of the system being solved, as was shown in Subsection 4.3.1. For unsteady implicit time integration of turbulent flows, it is known that often loose linear solves are sufficient. In this work, the linear solver is used inside a nonlinear solver, which means that the worst outcome is that the Newton method converges non-optimally. However, if linearised time-integration schemes such as the Rosenbrock methods are used, the stagnation of the GMRES method could mean a worsening of the solution [130].

5.3.3. DNS of Laminar Channel Flow at $Re_\tau = 50$

With the temporal convergence rates and working of the solver validated for the Euler equations, the next step was to validate the method for the compressible Navier-Stokes equations. The first numerical experiment performed to do this was to apply the solver to a laminar channel flow. More specifically, the LCF_50 test case described in Subsection 5.2.2. For this low friction Reynolds number, the channel flow is laminar and steady in time [131]. As explained before, the DGSEM simulations were initialised with a 1D solution obtained from a steady RANS simulation converged to machine precision. With

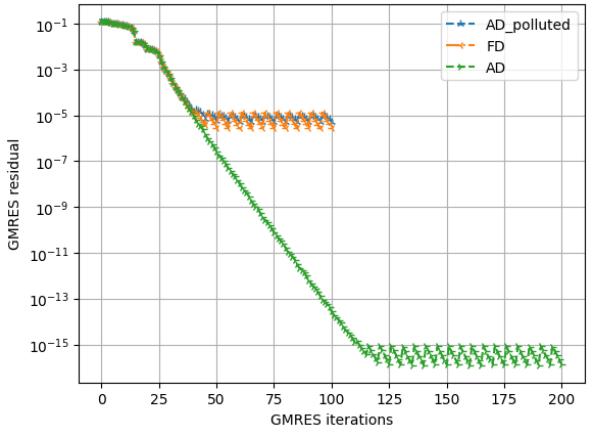


Figure 5.5: GMRES residuals over a single linear solver for restarted GMRES. AD is given 200 iterations to showcase stagnation at machine precision.

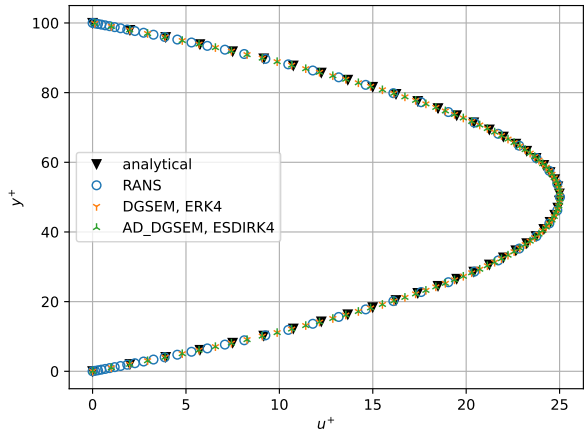


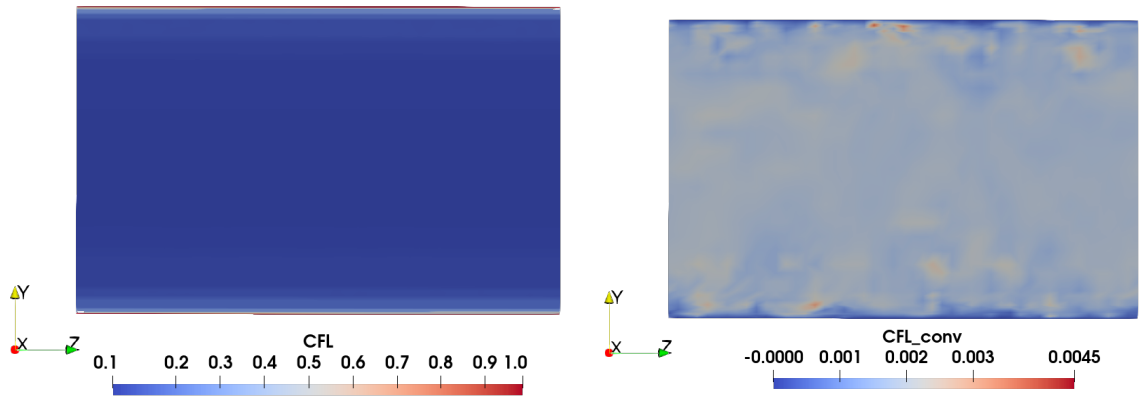
Figure 5.6: Mean friction velocity profile over complete channel height for $Re_\tau = 50$ followed from an analytical solution shown in (5.15), a 1D FV RANS simulation and 3D unsteady DGSEM simulations using explicit and implicit time-integration schemes.

this, the expected outcome of the unsteady DGSEM solutions is that the flow field remains steady over time, as the input flow was already a fully developed laminar steady flow. Nonetheless, a setup more commonly used for turbulent channel flows (3D, unsteady simulation) was opted for to validate the working of the unsteady solver. For the implicit solver, the ESDIRK4 time-integration scheme was used with a maximum CFL number of approximately 1000. For the explicit time-integration scheme, the ERK4 time-integration scheme was used, and a maximum CFL number of approximately 1.0 was used.

The friction velocity profiles of the analytical solution, RANS solution, and the solutions obtained by the DGSEM solver with time-explicit and time-implicit time-integration methods are shown in Figure 5.6. For the unsteady DGSEM simulations, the field considered was an instantaneous flow field at the end of the simulation. As can be seen, the outputs of both the existing time-explicit as well as the time-implicit scheme for the DGSEM solver agree well with the analytically obtained solution. This test case validated that the time-implicit solver produced the same results as the time-explicit solver for a steady test case using the compressible Navier-Stokes equations.

5.3.4. LES of Turbulent Channel Flow at $Re_\tau = 395$

With the basic working of the implicit solver verified for the Navier-Stokes equations, the next step was to apply the implicit scheme to an unsteady turbulent channel flow with friction Reynolds number $Re_\tau = 395$. The goal of this test case was to further validate the working of the implicit DGSEM solver for an unsteady LES of the channel flow. Next to this, it was also aimed at understanding the characteristics of the implicit DGSEM solver applied to such an unsteady test case. Namely, to analyse the effect of the



(a) Total CFL number

(b) Convective CFL number

Figure 5.7: Visualisation of the instantaneous CFL number and convective CFL number as reported by TRACE of the turbulent channel flow with the explicit time-scheme configuration.

increased time-step size on the accuracy of the solution. Although one of the major driving points for the usage of implicit schemes in unsteady simulations using DGSEM is to overcome the overly restrictive constraints for explicit schemes, the limit of the time-step size that can be taken while still resolving turbulent structures is not well defined. Moreover, this test case also allowed for the identification of suitable nonlinear and linear solver settings.

As explained in Subsection 5.2.2, the initial flow field was an FV RANS solution with superimposed synthetically generated turbulence. Nonetheless, the simulations had to be run for a sufficient amount of physical time in order to properly generate fully turbulent channel flow, and also long enough to gather statistical data of this flow. Therefore, the simulations were run for 20 eddy turnover times ($= tu_\tau/\delta$) to develop a fully turbulent flow. Following this, the simulation was continued for another 20 eddy turnover times to record the flow field and reduce statistical errors. For the first tests, the TCF_395 mesh was used as described in Table 5.2.

For the explicit DGSEM case, the ERK3 time-integration scheme is used, and the time-step size is driven by the maximum CFL number in the domain, being approximately CFL = 1.0. A visualisation of the CFL number for the explicit configuration over a slice of the domain is presented in Figure 5.7a. What can be seen from this representation is that for a large portion of the domain, the CFL number is much lower than the maximum CFL number found inside the domain. Only close to the walls the maximum is reached. Looking at the convective CFL number on the same slice in Figure 5.7b, it can be seen that the convective CFL number does not reach its maximum near the walls as the CFL number does. This is due to the lower flow velocities near the wall. Furthermore, due to the low Mach number chosen in the simulation, it can also be noticed that the convective CFL number is much lower (maximum of 4.5e-3) compared to the total CFL number. This is due to the CFL number taking into account the speed of sound as the solver is compressible. The low Mach number of the test case increases this difference as acoustic pressure waves move at smaller timescales than the convective waves. As stated before, for the implicit solver, the ESDIRK4 time-integration scheme is selected. In order to evaluate the effect of the time-step size, CFL numbers of 100.0 and 200.0 were considered for this test case. This corresponds to a maximum convective CFL number of approximately 0.45 and 0.9, respectively. For the nonlinear Newton solver, a relative tolerance of $\sigma = 10^{-2}$ was used. Although lower Newton relative tolerances ($\sigma = 10^{-3}$) were also considered, they did not increase the accuracy of the solution but did increase run times. For the linear solver, a relative drop of $\eta = 0.1$ was used.

As a relatively coarse linear solve was allowed, it was decided to use the FD differentiation method paired with a non-restarted GMRES. This choice was further motivated by the performance of the FD method compared to the AD method shown in Subsection 5.3.2. For the implicit schemes, the preconditioner was forced to update whenever it detected that the linear solver was not able to successfully reduce the residual to the specified limit. However, it was found that the preconditioner was never required to be updated during the simulations, resulting in a fully frozen preconditioner over the entire simulation.

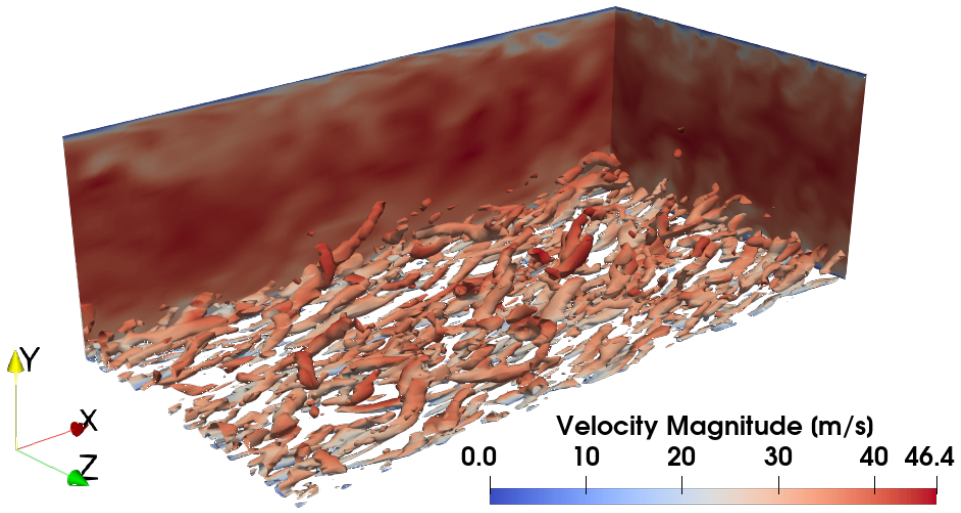


Figure 5.8: Q-criterion iso-contours of 4.0e7 coloured with velocity magnitude of an instantaneous flow field of the turbulent channel flow at $Re_\tau = 395$. Specific flow field generated from time-implicit simulation using ESDIRK4 [132] time-integration scheme with CFL number of 100.0.

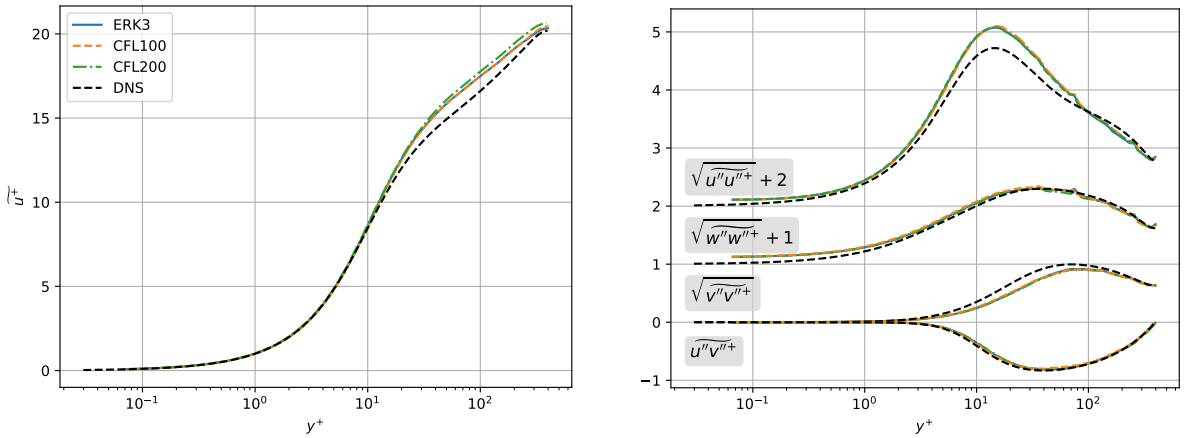
It is hypothesised that the frozen preconditioner remained effective due to the initial solution with synthetically generated turbulence imposed on the RANS solution being close to the real turbulent flow, meaning that the global characteristics of the Jacobian operator remained constant over the entire simulation and were captured in the initial construction of the preconditioner. Full freezing of the preconditioner was also found to be effective by Vangelatos for a similar turbulent channel flow test case [56]. A visualisation of the Q-criterion for an instantaneous flow field of the turbulent channel flow is shown in Figure 5.8.

In order to compare the solutions of the different test case configurations, the Favre-averaged mean friction velocity profile as well as the Reynolds stress profiles are compared to DNS reference data generated by [133]. Favre averaging is used due to the flow being compressible. The normalised friction velocity $\widetilde{u}^+$ and Reynolds stresses $\widetilde{u''u''}$ are defined as

$$y^+ = \frac{\bar{\rho}\widetilde{u}_\tau}{\bar{\mu}}, \quad \widetilde{u}^+ = \frac{\widetilde{u}}{\widetilde{u}_\tau}, \quad \widetilde{u_i'' u_j''} = \frac{\widetilde{u_i'' u_j''}}{\widetilde{u}_\tau^2}, \quad \widetilde{u}_\tau = \sqrt{\frac{\widetilde{\tau}_w}{\bar{\rho}}}, \quad \widetilde{\tau}_w = \bar{\mu}\frac{\partial \widetilde{u}_x}{\partial y}\bigg|_{y=0}. \tag{5.17}$$

The results are presented in Figure 5.9. Firstly, a general observation is made that the numerical results deviate from the DNS reference data but are in agreement with each other. Therefore, this deviation is understood to be related to the spatial accuracy and not the temporal resolution. The mean velocity profiles shown in Figure 5.9a indicate that the explicit and implicit LES cases agree with the DNS reference in the viscous sublayer ($y^+ \leq 5$) and start to deviate within the buffer layer ($5 \leq y^+ \leq 30$). Comparing the explicit case with the CFL100 and CFL200 implicit cases, it is observed that the CFL100 matches the explicit case more closely. The higher mean velocity in the outer layer ($y+ \geq 50$) could indicate that the flow is more laminar due to the greater dissipation provided by the larger time-step size. Looking at the mean Reynolds stress profiles in Figure 5.9b, the ERK3, CFL100 and CFL200 are observed to be in greater agreement with each other. What is observed, however, are irregularities in the solution resulting in a non-smooth behaviour across all three cases for the $\sqrt{\widetilde{u''u''}^+}$ and $\sqrt{\widetilde{w''w''}^+}$ profiles when compared to the reference data. This behaviour is understood to be caused by the presence of aliasing errors due to the flow being underresolved. Nevertheless, the irregularities are present across both the explicit and implicit cases, indicating that the issue is not caused by the time-integration scheme.

Influence of mesh refinement In order to further investigate the aliasing errors observed, a similar set of test cases with ERK3 at CFL 1.0, ESDIRK4 at CFL 100.0 and CFL 200.0 was simulated on the TCF_395-fine mesh with 96^3 DOF as shown in Table 5.2. The mesh resolutions normalised over the wall



(a) Mean friction velocity profile $\widetilde{u}^+$.

(b) Mean Reynolds stress profiles.

Figure 5.9: Mean streamwise friction velocity and Reynolds stress profiles for turbulent channel flow at $Re_\tau = 395$ for 64^3 DOF DGSEM simulation. DNS data from Iwamoto et al. [133].

Table 5.5: Mesh resolution normalised over wall unit for LES mesh of turbulent channel flow test cases at $Re_\tau = 395$.

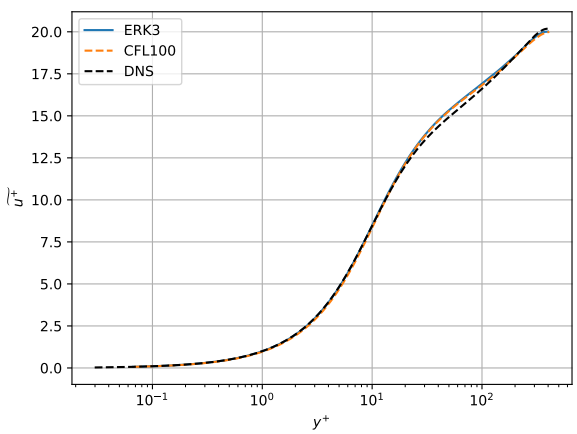
Name	Δx^+	Δy^+	Δz^+
TCF_395	51.527	0.731	25.764
TCF_395-fine	34.351	0.425	17.176

unit are presented in Table 5.5. This mesh resolution corresponds to the distance between equidistantly spaced points within a DG element, defined as

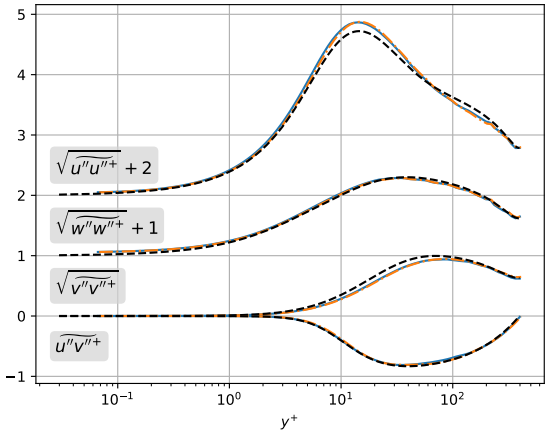
$$\Delta y = \frac{\Delta y_{\text{elem}}}{N}. \tag{5.18}$$

Although the inner-nodes of the DG element are not equidistantly spaced, as shown in Figure 3.1, the assumption of equidistant spacing to present mesh resolution is commonly made for DG methods [2]. Again, this test case was simulated for 40 eddy turnover times and statistical data were obtained from probes inside the channel. Although the CFL 200 case ran for nearly 90% of the specified simulation time, it eventually crashed at the end. Possibly, increasing the drop in nonlinear residual could prevent this configuration from crashing, but further analysis could not be performed due to time limitations. The velocity profiles produced for these test cases are shown in Figure 5.10. The first general observation that can be made is that the LES results now are in better agreement with the DNS reference data. This logically follows from the increased spatial resolution. Furthermore, the discontinuities in the mean Reynolds stress profiles observed for the TCF_395 case are not observed for this TCF_395-fine mesh, as can be seen in Figure 5.10b. Indicating that the TCF_395 mesh might be too coarse for the $Re_\tau = 395$ channel flow.

For further analysis, the one-dimensional energy spectrum of the streamwise velocity component was analysed for the explicit and implicit cases on the TCF_395 and TCF_395-fine meshes. In order to obtain the spectra, the probed streamwise velocity at a channel height of $y^+ \approx 70$ is used at multiple spanwise and streamwise locations in the channel. The probed velocity is then used to compute velocity fluctuations over time. These velocity fluctuations are then used to compute the power spectral density of the streamwise velocity component using Welch's method [134]. The spectra are presented in Figure 5.11. For the spectra of the TCF_395 mesh cases shown in Figure 5.11a, no difference in the energy spectra of the ERK3 and CFL100 and CFL200 simulations was observed. A possible explanation for why the previously observed difference in the mean streamwise velocity profile of the CFL200 and CFL100 cases is not found here is that the signal has high noise. What was observed, however, is that the energy corresponding to $f = 0$ [Hz] obtained from the Fourier transform was higher than that of the CFL100 and ERK3 cases. This observation is in line with the observed higher streamwise mean velocity observed in Figure 5.9a. The energy spectra for the different time-integration schemes on the

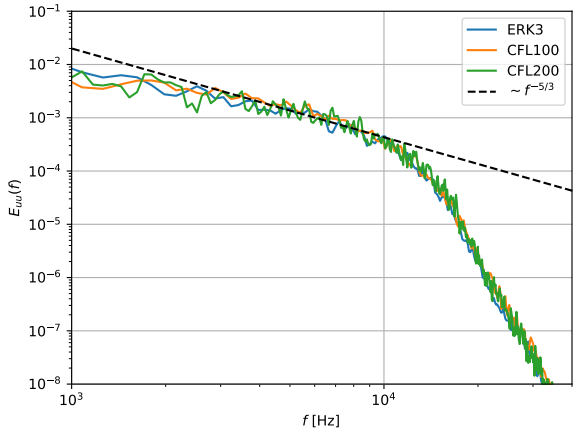


(a) Mean friction velocity profile $\widetilde{u}^+$.

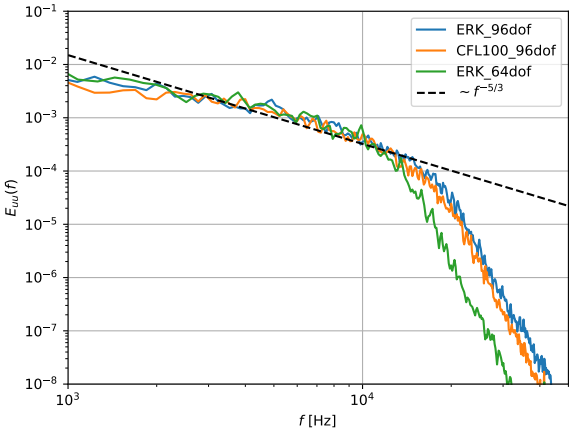


(b) Mean Reynolds stress profiles.

Figure 5.10: Mean streamwise friction velocity and Reynolds stress profiles for turbulent channel flow at $Re_\tau = 395$ for 96^3 DOF DGSEM simulation. DNS data from Iwamoto et al. [133].



(a) E_{uu} energy spectra for TCF_395 64^3 DOF mesh.

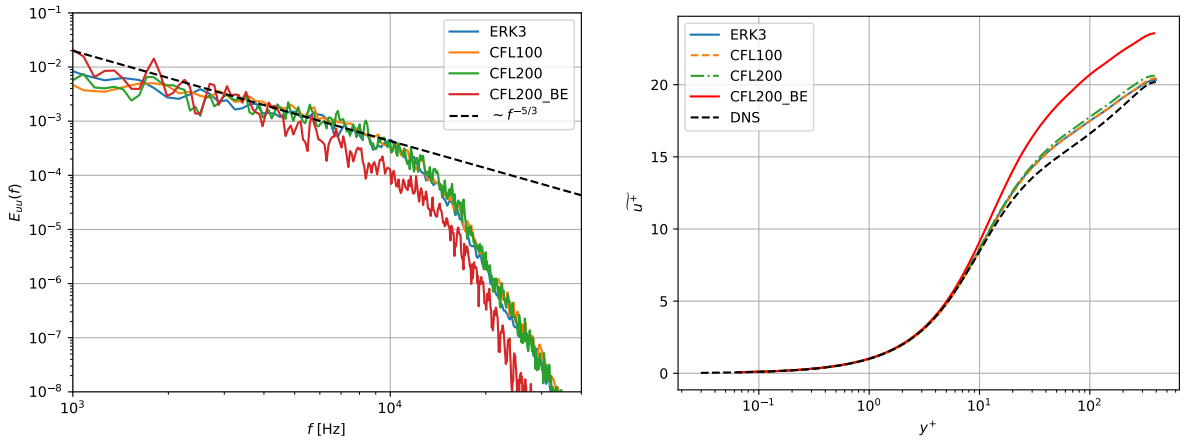


(b) E_{uu} energy spectra for TCF_395-fine 96^3 DOF, the ERK3 TCF_395 64^3 DOF mesh is added for reference.

Figure 5.11: One-dimensional energy spectrum of the streamwise velocity E_{uu} in x-direction at $y^+ \approx 70$ for different time-integration schemes and spatial resolution.

Table 5.6: Wall times of $Re_\tau = 395$ channel flow simulations of 40 eddy turnover times for simulations performed.

Temporal scheme	Diff. method	CFL	Mesh configuration	Cores	Wall time [hh:mm:ss]
ERK3	-	1.0	TCF_395	256	18:54:04
ESDIRK4	FD	100.0	TCF_395	256	13:49:07
ESDIRK4	FD	200.0	TCF_395	256	10:04:20
ERK3	-	1.0	TCF_395-fine	512	52:34:41
ESDIRK4	FD	100.0	TCF_395-fine	512	44:59:34



(a) One-dimensional energy spectrum of the streamwise velocity E_{uu} in x-direction at $y^+ \approx 70$.

(b) Mean friction velocity profile $\widetilde{u}^+$, DNS reference data generated by [133].

Figure 5.12: Comparison of ERK3, ESDIRK4 at CFL 100 and 200 against the BE time-integration scheme at CFL 200.

refined TCF_395-fine mesh are shown in Figure 5.11b. Here again, the ERK3 and CFL100 cases were observed to match well with each other. However, a clear difference is seen when comparing with the TCF_395 64^3 DOF ERK3 case, as also shown in this figure. Here, the effect of coarsening the mesh can clearly be seen, as high-frequency scales are dissipated more strongly. The clear effect of the spatial dissipation influencing only the small scales also shows the favourable dissipation-dispersion relations of the DGSEM.

The total wall times of the simulations considered in this experiment are reported in Table 5.6. Compared to the explicit solver, the implicit scheme produced lower wall times for both the CFL100 and CFL200 cases. This too is the case for the 96^3 CFL100 case. This result furthermore validates the proper working of the implicit solver in an HPC setting. However, it must be noted that this performance measurement of one specific test case cannot be directly assumed to be similar for different test cases. Nonetheless, the results of the implicit solver showcase that by increasing the time-step size beyond the explicit CFL limit, and using an efficient matrix-free GMRES solving scheme, the run times of turbulent simulations can be reduced.

Influence of temporal dissipation As a final experiment, it was decided to investigate the effect of temporal dissipation on the simulation of the turbulent flow. For this, the TCF_395 mesh paired with the BE scheme at a CFL number of 200 was used. This simulation was run for 40 eddy turnover times, just as the previous experiments. Although the simulation crashed at approximately 90% of the specified simulation time, enough statistical data was gathered to investigate the velocity profiles. The same relative linear tolerance of 10^{-1} and nonlinear tolerance of 10^{-2}, with a fully frozen ILU0 preconditioner, were used in the test case. The results are presented in Figure 5.12. Analysing the energy spectra shown in Figure 5.12a, it can be seen that the BE scheme introduces significant dissipation into the turbulent flow across all frequencies. This can be expected from the lower order of accuracy of the backward Euler method. This is further confirmed by looking at the mean friction velocity profile shown in Figure 5.12b. Here, it can be clearly seen that the velocity profile differs significantly from the ERK3 and CFL100 and CFL200 cases. The deviation can logically be explained by the increased dissipation, causing the flow to

become more laminar. This, in turn, results in the bulk velocity in the channel to increase, resulting in an equilibrium condition with the enforced pressure gradient.

Interestingly, the increased dissipation of the BE scheme appears to have an influence on all turbulent scales. In the context of SRS, an ideal time-integration scheme would have dispersion-dissipation characteristics that are similar to the DGSEM operator, in that they dissipate the high unresolved wave numbers. Then the time-step size would act as a cut-off filter just as the DGSEM spatial operator does in space, as is observed in Figure 5.11b. Clearly, the application of the BE scheme in this test case resulted in too much dissipation originating from the temporal integration. Due to time limitations, the specific dispersion-dissipation relations of the ESDIRK4 and BE scheme could not be analysed. However, it is understood that further research into suitable implicit time-integration schemes for high-order DG methods is required to fully leverage the strengths of the implicit solving method [135].

Discussion The usage of the implicit solver on the turbulent channel flow test case produced some interesting results. First of all, the test case showed the proper working of the solver in solving an unsteady turbulent flow. Compared to the explicit solver, the implicit solver produced very similar results, which further validated the implementation of the solver. Considering the run-time performance of the implicit solver compared to the explicit solver, the following notes regarding the test case considered must be kept in mind. Firstly, the turbulent channel flow considered was at a low Mach number. This resulted in the CFL condition for the explicit solver to be dominated by the speed of the pressure waves. Next to this, the turbulent channel flow did not exhibit significant pressure waves that affected the flow solution. Thus, it can be argued that the explicit method was artificially restricted and that such restrictions are less severe when higher Mach numbers are considered with pressure waves that do influence the flow field. Such test cases are definitely encountered in turbomachinery CFD. Next to this, the implicit solver was applied with an aggressive coarse nonlinear and linear solve and a full freezing of the Jacobian matrix was used. The usage of coarse nonlinear and linear solving is often used for unsteady implicit time integration. However, the full freezing of the preconditioner might cause issues in future test cases applied to the solver. If the preconditioner has to be updated within the solving at regular intervals, it will most definitely influence the performance of the solver negatively. All analyses in this project and the reviewed literature favour freezing of preconditioners if possible to achieve performance gains.

Another decision that could have influenced the performance of the implicit solver is the decision to use a fixed linear solver tolerance of 1.0e-1. A better alternative would be to use the adaptive forcing term explored in Subsection 4.3.1. This possibly could have reduced the total linear iterations required per Newton solve, and thus reduced run times. However, the usage of this method also brings uncertainty in that a suitable η_0 has to be applied. Therefore, in this work, it was left out and considered as a possible point of improvement on the method.

Lastly, for the turbulent channel flow test case CFL numbers of 1, 100 and 200 were considered. The ERK3 scheme served as an explicit reference, and a time step size corresponding to a maximum CFL number of 1 in the domain was chosen. With the conservative way the CFL number is estimated by TRACE (see Subsection 3.3.1), experience with the solver has shown that it sometimes is possible to increase the CFL number to approximately CFL ≤ 1.5 without the method crashing. Thus, the explicit reference could be further improved by attempting to increase the time-step size until the real limit is found. However, similarly for the implicit cases, the true time step size limit where the implicit scheme started to significantly influence the solution was not fully explored. For the 64^3 DOF case, a test at CFL 400 with the same solver configuration was attempted, but this crashed at about 80% of the simulation duration. Possibly, the Newton tolerance should have been tightened for this larger CFL number, or the Newton method should have been relaxed by using the pseudo-time stepping approach. Similarly, the 96^3 DOF CFL200 case that crashed at 90% could also be attempted again with adjusted solver settings. As the increase of the time step size is the main reason implicit schemes were investigated in the scope of this study, a more thorough analysis of the true limits of the temporal resolution would have been insightful.

5.4. Discussion on the Implementation of a Matrix-free Implicit DGSEM Solver in TRACE

In this chapter, the implementation and analysis of the implicit solver has been described. Here, some discussion regarding the choices made in the implementation and test cases used will be given.

The first point of discussion is that of the preconditioning methods used for the solver. As said before, the solver was implemented for the usage of ILU(p), ILU0 and inner GMRES preconditioning. However, only the ILU0 preconditioner was considered in the analysis of the method. From the literature, it was suspected that the ILU0 preconditioner would be more efficient than more accurate but also more expensive ILU(p) methods and therefore was not considered in the analysis. For the nested GMRES method, it must be noted that the flexible GMRES (FGMRES) algorithm must be used for the outer solve. This is because the preconditioner is nonlinear in this case, hence the requirement for the FGMRES method. The FGMRES method allows for a wider range of preconditioning methods but also has twice the memory costs of the standard GMRES method [136]. A look into the effect of different preconditioning strategies would have been a useful addition to the analysis of the solver. However, for the turbulent channel flow test case, the linear solver showed convergence to the specified relative drop of 10^{-1} within < 10 linear iterations and < 6 linear iterations for the CFL200 and CFL100 64^3 DOF case, respectively. Similarly, for the CFL100 96^3 DOF case, the GMRES converged in ≈ 6 linear iterations on average. The low GMRES iterations indicated that the frozen ILU0 preconditioner was sufficient. Nonetheless, the imposed 10^{-1} relative drop in linear residual can be considered very coarse. Thus, perhaps more effective preconditioning strategies are required if tighter linear solves are required or a more ill-conditioned system is to be solved.

Regarding the accuracy analysis on the FD and AD methods, it was found that the inexactness of the FD method caused the GMRES method to stagnate. The stagnation was not observed for non-restarted GMRES, but it is argued that this is because the true residual is never evaluated in non-restarted GMRES. Again, the coarse linear solve imposed in the turbulent channel flow test case was not influenced by this stagnation, which occurred at more tight solves. Nonetheless, if an adaptive forcing term is used with the inexact Newton method such tight solves might be required by the solver, which could result in FD not producing the desired result. Specifically, for non-restarted GMRES, this issue would not be reported if the given hypothesis is true. Therefore, this issue observed with the FD method should be investigated more thoroughly, as it might motivate the use of a different choice of ϵ, usage of the AD method, or even a limit on the relative drop the FD method is able to provide, at the cost of more linear iterations in the Newton method.

Another point of discussion is that in the comparison of the prototype implicit solver with the existing explicit solver, possibly performance improvements of the method can be realised by optimisation of the code. Indeed, the project was mostly focused on making the method work within TRACE and doing a subsequent analysis of its proper working. Although a majority of the functions used by the method from the TRACE code base are highly optimised, things like the communication with the Spliss solver should be analysed for performance. For instance, it is speculated that the process of copying inputs and outputs to and from the Spliss solver to TRACE could be optimised. Moreover, in this project, only periodic and Dirichlet boundary (non-slip) conditions were used. For real turbomachinery applications, more complex solver features such as e.g. sliding interfaces [137] are required to be implemented. Nevertheless, the results and performance shown in this project serve as a preliminary outlook on the usage of implicit solving methods to the DGSEM method.

6

Conclusion and Recommendations for Future Work

In this master's thesis project, the usage of a matrix-free implicit solving scheme for implicit time integration in scale-resolving simulations using the compressible Navier-Stokes equations with a high-order discontinuous Galerkin spectral element (DGSEM) spatial discretisation is researched. The motivation to use implicit solving schemes paired with a high-order (spatial accuracy of order three or higher) DGSEM spatial discretisation comes from the excessive time-step size constraint imposed on explicit time-integration schemes by the spatial operator. With implicit solving schemes, this constraint can be overcome, and the time step can be increased to what is required to temporally resolve the turbulent phenomena occurring in the flow. The main research objective in this thesis was to evaluate and understand matrix-free solving methods for implicit time integration with the DGSEM method applied to convection-diffusion equations, and to produce a prototype matrix-free implicit solver for the DGSEM solver in the CFD framework TRACE.

In the first part of this project, a one-dimensional DGSEM solver for the viscous Burgers equation written in C++ was developed from scratch. The development of this solver resulted in a fundamental understanding of the DGSEM, as well as the matrix-free GMRES linear solving algorithm combined with an inexact Newton method. For the 1D solver, only the finite-difference (FD) differentiation technique was considered to make the GMRES method matrix-free. The usage of the inexact Newton method showed that the most optimal linear solving tolerance is dependent on the time-step size, and that such an optimal solving tolerance can be automatically controlled by the use of an adaptive forcing term that adjusts the linear solver tolerance based on the nonlinear residual in the Newton solver. The Jacobian sparsity pattern analysis showed that the Bassi-Rebay 1 (BR1) viscous flux scheme does not suffer from an increased numerical stencil compared to the BR2 if the Legendre-Gauss-Lobatto (LGL) quadrature nodes are used. In contrast, when Legendre-Gauss (LG) quadrature nodes are used, the BR1 scheme does result in an increased numerical stencil. This aligns with the literature reviewed. Comparing the matrix-free and matrix-based GMRES algorithms developed in the 1D solver, it can be said that the matrix-based GMRES had a higher peak memory consumption and scaled more poorly with increasing degrees of freedom. Next to this, the matrix-based GMRES was also found to have slower run times than the matrix-free GMRES, although the implementation of the matrix-based GMRES could be considered quite inefficient. Lastly, the Burgers DGSEM solver was used to evaluate simple preconditioners for the matrix-based and matrix-free GMRES. Here, it was found that an element block-Jacobi (EBJ) preconditioner limited the memory consumption while still preconditioning the GMRES for both the matrix-based GMRES and matrix-free GMRES. The ILU0 preconditioner, however, still provided the least amount of linear iterations for the matrix-based GMRES, at the cost of being a more expensive preconditioner to compute.

In the second part of the project, the focus was shifted to developing a prototype matrix-free implicit solver for the DGSEM method in TRACE. The prototype solver makes use of an inexact Newton method

without an adaptive forcing term, paired with a matrix-free GMRES that can use either the FD or forward automatic differentiation (AD) methods. The comparison of the FD and AD differentiation techniques showed that the AD method is ≈ 2.1 times slower per numerical operation, but that the AD method is able to compute the matrix-vector product with accuracy up to floating point precision, whereas the FD method produces an approximation containing numerical noise coming from the truncation error and possible round-off error. The numerical noise from the FD method was observed to stagnate the restarted GMRES, but it is understood that for non-restarted GMRES, this stagnation is also present but not displayed because the true residual is never evaluated within the non-restarted GMRES. Thus, AD was found to be superior for performing tight linear solves. In practice, however, such tight linear solves have not been observed to be required for unsteady implicit time integration. This was also shown in the application of the prototype solver to the turbulent channel flow test case, where a relative linear residual drop of one order of magnitude performed well. Thus, for this test case, the FD method was chosen. The turbulent channel flow test case confirmed the proper working of the implicit solver in an academic test case. The solver paired with a fourth-order explicit first stage singly diagonally implicit Runge-Kutta (ESDIRK4) time-integration scheme produced similar statistical results as the time-explicit reference simulation, at time-step sizes 100 and 200 times larger than the explicit limit of the Courant-Friedrichs-Lewis condition (CFL) of CFL = 1.0. Comparing the total wall times of the simulations considered, the implicit solver was also observed to be faster than the explicit reference. However, the low Mach number of M = 0.1 and complete freezing of the ILU0 preconditioner most probably significantly increased the performance of the implicit solver compared to the explicit reference. The effect of the temporal dissipation error was clearly demonstrated when the first-order backwards Euler (BE) scheme with CFL = 200 was used, in which a clear laminarisation of the flow was observed. The temporal dissipation was shown to affect all turbulent scales, which is significantly different from the spatial dissipation of the DGSEM operator that tends to dissipate only high wave number scales.

6.1. Recommendations for Future Work

With the knowledge gained and developments made on implicit solving of the DGSEM solver in TRACE, the following recommendations for future work are given.

For the prototype solver, in this report, only an ILU0 preconditioner that is frozen for the entire simulation duration was used for the GMRES solver. Because the convergence of the GMRES method can be very sensitive to the usage of a suitable preconditioner, and because the ILU0 preconditioner still requires the full construction and storage of the Jacobian matrix, it is recommended to further investigate different preconditioning techniques that were investigated in the 1D solver and also ones that were observed to be used in literature. Examples of such preconditioners are the EBJ preconditioner and multigrid preconditioners.

Furthermore, in this prototype implementation only periodic and Dirichlet boundary conditions were considered. An unexplored area is the application of this solving method to more complex turbomachinery test cases. Where, for instance, sliding interfaces or non-local boundary conditions are considered. The proper working of the solver with such methods is important for the applicability of the method in industrial CFD cases, and therefore should be analysed in future work.

Regarding the solver settings of the implicit solver. Several recommendations are given. Firstly, when matrix-free GMRES is used with FD differentiation, the approximation error of the FD method results in a limit on the reduction of the linear residual that is not reported by the GMRES method itself. Therefore, it is recommended to find methods that estimate this limit and restrict the GMRES solver from performing linear residual reductions past what the gradient estimation is capable of. Furthermore, it is recommended to apply the adaptive forcing term explored in the 1D solver to the prototype solver, as this removes the need to specify a specific limit by the user and yields an optimal linear tolerance automatically. It is also recommended to analyse how the nonlinear solver tolerance influences the temporal accuracy of the solution and overall stability of the method.

In this work and in other literature reviewed, the use of the implicit solver is demonstrated by increasing the time-step size and showing that the accuracy of the solution is not affected. However, the true limit of the possible increase in time-step size is generally not discussed or investigated. It is expected

that this limit is a function of the time-step size, the order of accuracy of the scheme, the nonlinear solver tolerances set, and the test case investigated. Therefore, it is recommended to investigate the dissipation-dispersion relation of the time-integration schemes in more detail and also come up with methods to determine the maximum time-step size that can be used with an implicit time-integration scheme. This should allow the maximum possible leverage of the implicit solver to increase the performance of the method.

Lastly, the solver produced in this work is in a prototype state. The usage of the method requires knowledge of the code base and exists outside of the main solver logic in TRACE. For future use of the method, an integration of the prototype into the main TRACE solver is recommended.

Bibliography

[1] J. Tyacke, N. Vadlamani, W. Trojak, R. Watson, Y. Ma, and P. Tucker, "Turbomachinery simulation challenges and the future," *Progress in Aerospace Sciences*, vol. 110, p. 100554, 2019.

[2] M. Bergmann, *A split-form discontinuous Galerkin spectral element framework for scale-resolving simulations of turbomachinery flows*. Ph.d. dissertation, Ruhr-Universität Bochum, Universitätsbibliothek, 2024.

[3] Z. Wang, K. Fidkowski, R. Abgrall, F. Bassi, D. Caraeni, A. Cary, H. Deconinck, R. Hartmann, K. Hillewaert, H. Huynh, N. Kroll, G. May, P.-O. Persson, B. van Leer, and M. Visbal, "High-order CFD methods: current status and perspective," *International Journal for Numerical Methods in Fluids*, vol. 72, no. 8, pp. 811–845, 2013.

[4] A. Kanevsky, M. H. Carpenter, D. Gottlieb, and J. S. Hesthaven, "Application of implicit–explicit high order Runge–Kutta methods to discontinuous Galerkin schemes," *Journal of Computational Physics*, vol. 225, no. 2, pp. 1753–1781, 2007.

[5] D. Knoll and D. Keyes, "Jacobian-free Newton–Krylov methods: a survey of approaches and applications," *Journal of Computational Physics*, vol. 193, no. 2, pp. 357–397, 2004.

[6] J. H. Ferziger, M. Perić, and R. L. Street, *Introduction to Numerical Methods*, pp. 23–40. Cham: Springer International Publishing, 2020.

[7] Y.-T. Zhang and C.-W. Shu, "ENO and WENO schemes," in *Handbook of Numerical Methods for Hyperbolic Problems* (R. Abgrall and C.-W. Shu, eds.), vol. 17 of *Handbook of Numerical Analysis*, ch. 5, pp. 103–122, Elsevier, 2016.

[8] R. Bozorgpour and H. M. Darian, "Recent advancements in fluid flow simulation using the WENO scheme: A comprehensive review," *Journal of Nonlinear Mathematical Physics*, vol. 32, p. 14, Mar. 2025.

[9] C. Sheng, Q. Zhao, D. Zhong, and N. Ge, "A strategy to implement high-order weno schemes on unstructured grids," in *AIAA Aviation 2019 Forum*, jun 2019.

[10] S. Duczek, C. Willberg, and U. Gabbert, "Higher order finite element methods," in *Lamb-Wave Based Structural Health Monitoring in Polymer Composites* (R. Lammering, U. Gabbert, M. Sinapius, T. Schuster, and P. Wierach, eds.), pp. 117–159, Cham: Springer International Publishing, 2018.

[11] W. Gui and I. Babuška, "The h, p and h-p versions of the finite element method in 1 dimension," *Numerische Mathematik*, vol. 49, pp. 659–683, 577–612, Nov. 1986.

[12] B. Guo and I. Babuška, "The h-p version of the finite element method," *Computational Mechanics*, vol. 1, pp. 203–220, Sept. 1986.

[13] F. van de Vosse and P. Minev, *Spectral Element Methods: Theory and Applications*. EUT Report. W, Dept. of Mechanical Engineering, Eindhoven University of Technology, 1996.

[14] O. Zienkiewicz, R. Taylor, and P. Nithiarasu, "Introduction to the equations of fluid dynamics and the finite element approximation," in *The Finite Element Method for Fluid Dynamics (Seventh Edition)* (O. Zienkiewicz, R. Taylor, and P. Nithiarasu, eds.), ch. 1, pp. 1–29, Oxford: Butterworth-Heinemann, seventh edition ed., 2014.

[15] O. Zienkiewicz, R. Taylor, and P. Nithiarasu, "Convection-dominated problems: Finite element approximations to the convection-diffusion-reaction equation," in *The Finite Element Method for Fluid Dynamics (Seventh Edition)* (O. Zienkiewicz, R. Taylor, and P. Nithiarasu, eds.), ch. 2, pp. 31–85, Oxford: Butterworth-Heinemann, seventh edition ed., 2014.

[16] W. H. Reed and T. R. Hill, "Triangular mesh methods for the neutron transport equation," Los Alamos Scientific Lab., N.Mex. (USA), Oct. 1973.

[17] B. Cockburn and C.-W. Shu, "The Runge-Kutta local projection p1-discontinuous-Galerkin finite element method for scalar conservation laws," in *1st National Fluid Dynamics Conference, 1988*, 1988.

[18] B. Cockburn and C.-W. Shu, "TVB Runge-Kutta local projection discontinuous Galerkin finite element method for conservation laws II: General framework," *Mathematics of Computation*, vol. 52, no. 186, p. 411 – 435, 1989.

[19] B. Cockburn, S.-Y. Lin, and C.-W. Shu, "TVB Runge-Kutta local projection discontinuous Galerkin finite element method for conservation laws III: One-dimensional systems," *Journal of Computational Physics*, vol. 84, no. 1, p. 90 – 113, 1989.

[20] B. Cockburn, S. Hou, and C.-W. Shu, "The Runge-Kutta local projection discontinuous Galerkin finite element method for conservation laws IV: The multidimensional case," *Mathematics of Computation*, vol. 54, no. 190, p. 545 – 581, 1990.

[21] B. Cockburn and C.-W. Shu, "The Runge-Kutta discontinuous Galerkin method for conservation laws V: Multidimensional systems," *Journal of Computational Physics*, vol. 141, no. 2, p. 199 – 224, 1998.

[22] F. Bassi and S. Rebay, "A high-order accurate discontinuous finite element method for the numerical solution of the compressible Navier–Stokes equations," *Journal of Computational Physics*, vol. 131, no. 2, pp. 267–279, 1997.

[23] F. Bassi and S. Rebay, "A high order discontinuous Galerkin method for compressible turbulent flows," in *Discontinuous Galerkin Methods* (B. Cockburn, G. E. Karniadakis, and C.-W. Shu, eds.), (Berlin, Heidelberg), pp. 77–88, Springer Berlin Heidelberg, 2000.

[24] F. Q. Hu, M. Hussaini, and P. Rasetarinera, "An analysis of the discontinuous Galerkin method for wave propagation problems," *Journal of Computational Physics*, vol. 151, no. 2, pp. 921–946, 1999.

[25] L. Wei and A. Pollard, "Direct numerical simulation of compressible turbulent channel flows using the discontinuous Galerkin method," *Computers & Fluids*, vol. 47, no. 1, pp. 85–100, 2011.

[26] J.-B. Chapelier, M. de la Llave Plata, F. Renac, and E. Lamballais, "Evaluation of a high-order discontinuous Galerkin method for the DNS of turbulent flows," *Computers & Fluids*, vol. 95, pp. 210–226, 2014.

[27] A. Uranga, P.-O. Persson, M. Drela, and J. Peraire, "Implicit large eddy Simulation of transition to turbulence at low Reynolds numbers using a discontinuous Galerkin method," *International Journal for Numerical Methods in Engineering*, vol. 87, no. 1-5, pp. 232–261, 2011.

[28] N. Nguyen, P.-O. Persson, and J. Peraire, "RANS solutions using high order discontinuous Galerkin methods," in *45th AIAA Aerospace Sciences Meeting and Exhibit*, 2007.

[29] M. Wallraff, R. Hartmann, and T. Leicht, "Multigrid solver algorithms for DG methods and applications to aerodynamic flows," in *IDIHOM: Industrialization of High-Order Methods - A Top-Down Approach: Results of a Collaborative Research Project Funded by the European Union, 2010 - 2014* (N. Kroll, C. Hirsch, F. Bassi, C. Johnston, and K. Hillewaert, eds.), pp. 153–178, Cham: Springer International Publishing, 2015.

[30] F. Bassi, L. Botti, A. Colombo, A. Crivellini, C. De Bartolo, N. Franchina, A. Ghidoni, and S. Rebay, "Time integration in the discontinuous Galerkin code MIGALE - steady problems," in *IDIHOM: Industrialization of High-Order Methods - A Top-Down Approach: Results of a Collaborative Research Project Funded by the European Union, 2010 - 2014* (N. Kroll, C. Hirsch, F. Bassi, C. Johnston, and K. Hillewaert, eds.), pp. 179–204, Cham: Springer International Publishing, 2015.

[31] J. Shen, T. Tang, and L.-L. Wang, "Introduction," in *Spectral Methods: Algorithms, Analysis and Applications*, pp. 1–22, Berlin, Heidelberg: Springer Berlin Heidelberg, 2011.

[32] D. A. Kopriva, "Algorithms for non-periodic functions," in *Implementing Spectral Methods for Partial Differential Equations: Algorithms for Scientists and Engineers*, ch. 3, pp. 59–87, Dordrecht: Springer Netherlands, 2009.

[33] G. Gassner and D. A. Kopriva, "A comparison of the dispersion and dissipation errors of Gauss and Gauss–Lobatto discontinuous Galerkin spectral element methods," *SIAM Journal on Scientific Computing*, vol. 33, no. 5, pp. 2560–2579, 2011.

[34] D. A. Kopriva, "Survey of spectral approximations," in *Implementing Spectral Methods for Partial Differential Equations: Algorithms for Scientists and Engineers*, ch. 4, pp. 91–147, Dordrecht: Springer Netherlands, 2009.

[35] M. Atak, J. Larsson, and C.-D. Munz, "The multicore challenge: Petascale DNS of a spatially-developing supersonic turbulent boundary layer up to high Reynolds numbers using DGSEM," in *Sustained Simulation Performance 2015* (M. M. Resch, W. Bez, E. Focht, H. Kobayashi, J. Qi, and S. Roller, eds.), (Cham), pp. 171–183, Springer International Publishing, 2015.

[36] A. D. Beck, D. G. Flad, C. Tonhäuser, G. Gassner, and C.-D. Munz, "On the influence of polynomial de-aliasing on subgrid scale models," *Flow, Turbulence and Combustion*, vol. 97, pp. 475–511, Sept. 2016.

[37] G. J. Gassner, A. R. Winters, and D. A. Kopriva, "Split form nodal discontinuous Galerkin schemes with summation-by-parts property for the compressible Euler equations," *Journal of Computational Physics*, vol. 327, pp. 39–66, 2016.

[38] R. M. Kirby and G. E. Karniadakis, "De-aliasing on non-uniform grids: algorithms and applications," *Journal of Computational Physics*, vol. 191, no. 1, pp. 249–264, 2003.

[39] M. H. Carpenter, T. C. Fisher, E. J. Nielsen, and S. H. Frankel, "Entropy stable spectral collocation schemes for the Navier–Stokes equations: Discontinuous interfaces," *SIAM Journal on Scientific Computing*, vol. 36, no. 5, pp. B835–B867, 2014.

[40] T. C. Fisher and M. H. Carpenter, "High-order entropy stable finite difference schemes for nonlinear conservation laws: Finite domains," *Journal of Computational Physics*, vol. 252, pp. 518–557, 2013.

[41] C. A. Kennedy and A. Gruber, "Reduced aliasing formulations of the convective terms within the Navier–Stokes equations for a compressible fluid," *Journal of Computational Physics*, vol. 227, no. 3, pp. 1676–1700, 2008.

[42] S. Pirozzoli, "Numerical methods for high-speed flows," *Annual Review of Fluid Mechanics*, vol. 43, no. Volume 43, 2011, pp. 163–194, 2011.

[43] F. Hindenlang, G. J. Gassner, C. Altmann, A. Beck, M. Staudenmaier, and C.-D. Munz, "Explicit discontinuous Galerkin methods for unsteady problems," *Computers & Fluids*, vol. 61, pp. 86–93, 2012. "High Fidelity Flow Simulations" Onera Scientific Day.

[44] G. Gassner, M. Dumbser, F. Hindenlang, and C.-D. Munz, "Explicit one-step time discretizations for discontinuous Galerkin and finite volume schemes based on local predictors," *Journal of Computational Physics*, vol. 230, no. 11, pp. 4232–4247, 2011. Special issue High Order Methods for CFD Problems.

[45] A. R. Winters, D. A. Kopriva, G. J. Gassner, and F. Hindenlang, "Construction of modern robust nodal discontinuous Galerkin spectral element methods for the compressible Navier–Stokes equations," in *Efficient High-Order Discretizations for Computational Fluid Dynamics* (M. Kronbichler and P.-O. Persson, eds.), pp. 117–196, Cham: Springer International Publishing, 2021.

[46] L. Wang and D. J. Mavriplis, "Implicit solution of the unsteady Euler equations for high-order accurate discontinuous Galerkin discretizations," *Journal of Computational Physics*, vol. 225, no. 2, pp. 1994–2015, 2007.

[47] F. Bassi and S. Rebay, "GMRES discontinuous Galerkin solution of the compressible Navier-Stokes equations," in *Discontinuous Galerkin Methods* (B. Cockburn, G. E. Karniadakis, and C.-W. Shu, eds.), (Berlin, Heidelberg), pp. 197–208, Springer Berlin Heidelberg, 2000.

[48] P.-O. Persson and J. Peraire, "Newton-GMRES preconditioning for discontinuous Galerkin discretizations of the Navier–Stokes equations," *SIAM Journal on Scientific Computing*, vol. 30, no. 6, pp. 2709–2733, 2008.

[49] E. Hairer and G. Wanner, *Solving Ordinary Differential Equations II. Stiff and Differential-Algebraic Problems*, vol. 14. Springer Berlin, Heidelberg, Jan. 1996.

[50] F. Bassi, L. Botti, A. Colombo, A. Ghidoni, and F. Massa, "Linearly implicit Rosenbrock-type Runge–Kutta schemes applied to the Discontinuous Galerkin solution of compressible and incompressible unsteady flows," *Computers & Fluids*, vol. 118, pp. 305–320, 2015.

[51] F. Bassi, L. Botti, A. Colombo, A. Crivellini, A. Ghidoni, and F. Massa, "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*, vol. 55, pp. 367–379, 2016. Vortical Structures and Wall Turbulence.

[52] D. S. Blom, P. Birken, H. Bijl, F. Kessels, A. Meister, and A. H. van Zuijlen, "A comparison of Rosenbrock and ESDIRK methods combined with iterative solvers for unsteady compressible flows," *Advances in Computational Mathematics*, vol. 42, pp. 1401–1426, Dec. 2016.

[53] R. Hartmann, F. Bassi, I. Bosnyakov, L. Botti, A. Colombo, A. Crivellini, M. Franciolini, T. Leicht, E. Martin, F. Massa, F. Renac, A. Troshin, V. Vlasenko, M. Wallraff, and A. Wolkov, "Implicit methods," in *TILDA: Towards Industrial LES/DNS in Aeronautics: Paving the Way for Future Accurate CFD - Results of the H2020 Research Project TILDA, Funded by the European Union, 2015 -2018* (C. Hirsch, K. Hillewaert, R. Hartmann, V. Couaillier, J.-F. Boussuge, F. Chalot, S. Bosniakov, and W. Haase, eds.), pp. 11–59, Cham: Springer International Publishing, 2021.

[54] G. Noventa, F. Massa, S. Rebay, F. Bassi, and A. Ghidoni, "Robustness and efficiency of an implicit time-adaptive discontinuous Galerkin solver for unsteady flows," *Computers & Fluids*, vol. 204, p. 104529, 2020.

[55] T. Steihaug and A. Wolfbrandt, "An attempt to avoid exact Jacobian and nonlinear equations in the numerical solution of stiff differential equations," *Mathematics of Computation*, vol. 33, no. 146, pp. 521–534, 1979.

[56] S. Vangelatos, *On the efficiency of implicit discontinuous Galerkin spectral element methods for the unsteady compressible Navier-Stokes equations*. PhD thesis, Dissertation, Stuttgart, Universität Stuttgart, 2019, 2020.

[57] A. Beck, M. Gao, D. Kempf, P. Kopper, N. Krais, M. Kurz, J. Zeifang, and C.-D. Munz, "Increasing the flexibility of the high order discontinuous Galerkin framework FLEXI towards large scale industrial applications," in *High Performance Computing in Science and Engineering '20* (W. E. Nagel, D. H. Kröner, and M. M. Resch, eds.), (Cham), pp. 343–358, Springer International Publishing, 2021.

[58] U. M. Ascher, S. J. Ruuth, and R. J. Spiteri, "Implicit-explicit Runge-Kutta methods for time-dependent partial differential equations," *Applied Numerical Mathematics*, vol. 25, no. 2, pp. 151–167, 1997. Special Issue on Time Integration.

[59] P.-O. Persson, "High-order LES simulations using implicit-explicit Runge-Kutta schemes," in *49th AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition*, 2017.

[60] B. C. Vermeire and S. Nadarajah, "Adaptive IMEX schemes for high-order unstructured methods," *Journal of Computational Physics*, vol. 280, pp. 261–286, 2015.

[61] C. A. Pereira and B. C. Vermeire, "Hybridized implicit-explicit flux reconstruction methods for geometry-induced stiffness," *Journal of Computational Physics*, vol. 528, p. 113819, 2025.

[62] S. Pollock, L. G. Rebholz, X. Tu, and M. Xiao, "Analysis of the Picard-Newton iteration for the Navier-Stokes equations: global stability and quadratic convergence," *Journal of Scientific Computing*, vol. 104, no. 1, p. 25, 2025.

[63] X. Li, E. V. Hawkins, L. G. Rebholz, and D. Vargun, "Accelerating and enabling convergence of nonlinear solvers for Navier–Stokes equations by continuous data assimilation," *Computer Methods in Applied Mechanics and Engineering*, vol. 416, p. 116313, 2023.

[64] P. Birken, G. Gassner, M. Haas, and C.-D. Munz, "Preconditioning for modal discontinuous Galerkin methods for unsteady 3D Navier–Stokes equations," *Journal of Computational Physics*, vol. 240, pp. 20–35, 2013.

[65] H.-B. An, Z.-Y. Mo, and X.-P. Liu, "A choice of forcing terms in inexact Newton method," *Journal of Computational and Applied Mathematics*, vol. 200, no. 1, pp. 47–60, 2007.

[66] S. C. Eisenstat and H. F. Walker, "Choosing the forcing terms in an inexact Newton method," *SIAM Journal on Scientific Computing*, vol. 17, no. 1, pp. 16–32, 1996.

[67] C. T. Kelley and D. E. Keyes, "Convergence analysis of pseudo-transient continuation," *SIAM Journal on Numerical Analysis*, vol. 35, no. 2, pp. 508–523, 1998.

[68] P. Birken, "Solving nonlinear systems inside implicit time integration schemes for unsteady viscous flows," in *Recent Developments in the Numerics of Nonlinear Hyperbolic Conservation Laws: Lectures Presented at a Workshop at the Mathematical Research Institute Oberwolfach, Germany, Jan 15 – 21, 2012* (R. Ansorge, H. Bijl, A. Meister, and T. Sonar, eds.), pp. 57–71, Berlin, Heidelberg: Springer Berlin Heidelberg, 2013.

[69] A. Jameson and S. Yoon, "Lower-upper implicit schemes with multiple grids for the Euler equations," *AIAA Journal*, vol. 25, no. 7, pp. 929–935, 1987.

[70] R. F. Chen and Z. J. Wang, "Fast, block lower-upper symmetric Gauss-Seidel scheme for arbitrary grids," *AIAA Journal*, vol. 38, no. 12, pp. 2238–2245, 2000.

[71] F. Jia, Z. Wang, R. Bhaskaran, U. Paliath, and G. M. Laskowski, "Accuracy, efficiency and scalability of explicit and implicit FR/CPR schemes in large eddy simulation," *Computers & Fluids*, vol. 195, p. 104316, 2019.

[72] Z. Wang and S. Rahmani, "Implicit large eddy simulation of the NASA CRM high-lift configuration near stall," *Computers & Fluids*, vol. 220, p. 104887, 2021.

[73] S. K. Rahmani and Z. J. Wang, "Large eddy simulation of the Sandia axisymmetric transonic hump using a high-order method," in *AIAA Scitech 2022 Forum*, p. 1534, 2022.

[74] I. Menshov and P. Pavlukhin, "Highly scalable implementation of an implicit matrix-free solver for gas dynamics on GPU-accelerated clusters," *The Journal of Supercomputing*, vol. 73, pp. 631–638, Feb. 2017.

[75] Y. Saad and M. H. Schultz, "GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems," *SIAM Journal on Scientific and Statistical Computing*, vol. 7, no. 3, pp. 856–869, 1986.

[76] Y. Saad, *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, second ed., 2003.

[77] N. Cuong Nguyen, S. Terrana, and J. Peraire, "Large-eddy simulation of transonic buffet using matrix-free discontinuous Galerkin method," *AIAA Journal*, vol. 60, no. 5, pp. 3060–3077, 2022.

[78] A. M. Rueda-Ramírez, E. Ferrer, D. A. Kopriva, G. Rubio, and E. Valero, "A statically condensed discontinuous Galerkin spectral element method on Gauss-Lobatto nodes for the compressible Navier-Stokes equations," *Journal of Computational Physics*, vol. 426, p. 109953, 2021.

[79] E. Jourdan de Araujo Jorge Filho and Z. J. Wang, "A matrix-free GMRES algorithm on GPU clusters for implicit large eddy simulation," in *AIAA Scitech 2021 Forum*, p. 1837, 2021.

[80] P. Tranquilli, S. R. Glandon, A. Sarshar, and A. Sandu, "Analytical Jacobian-vector products for the matrix-free time integration of partial differential equations," *Journal of Computational and Applied Mathematics*, vol. 310, pp. 213–223, 2017. Numerical Algorithms for Scientific and Engineering Applications.

[81] M. Sagebaum, E. Özkaya, N. R. Gauger, J. Backhaus, C. Frey, S. Mann, and M. Nagel, "Efficient algorithmic differentiation techniques for turbomachinery design," in *18th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, no. AIAA-2, June 2018.

[82] M. Sagebaum, J. Blühdorn, N. Gauger, J. Backhaus, C. Frey, G. Geiser, F. Bayer, E. Kügeler, C. Battistoni, and M. Nagel, "On recent developments for efficient turbomachinery design using algorithmic differentiation," in *15th European Conference on Turbomachinery Fluid Dynamics and Thermodynamics 2023, ETC 2023*, April 2023.

[83] J. Backhaus, *Adjungierte Strömungssimulation und gradientenbasierte Ersatzmodelle in der Turbomaschinenauslegung*. PhD thesis, Ruhr universität Bochum, Oct. 2020.

[84] N. G. M. Sagebaum, T. Albring, "High-performance derivative computations using CoDiPack," *ACM Transactions on Mathematical Software (TOMS)*, vol. 45, no. 4, 2019.

[85] L. T. Diosady and D. L. Darmofal, "Preconditioning methods for discontinuous Galerkin solutions of the Navier–Stokes equations," *Journal of Computational Physics*, vol. 228, no. 11, pp. 3917–3935, 2009.

[86] L. M. Versbach, P. Birken, and G. J. Gassner, "Finite volume based multigrid preconditioners for discontinuous Galerkin methods," *PAMM*, vol. 18, no. 1, p. e201800203, 2018.

[87] P. Birken, G. Gassner, and L. Versbach, "Subcell finite volume multigrid preconditioning for high-order discontinuous Galerkin methods," *International Journal of Computational Fluid Dynamics*, vol. 33, no. 9, pp. 353–361, 2019.

[88] D. A. Kopriva, "Transformation methods from square to non-square geometries," in *Implementing Spectral Methods for Partial Differential Equations: Algorithms for Scientists and Engineers*, pp. 223–246, Dordrecht: Springer Netherlands, 2009.

[89] D. A. Kopriva, "Metric identities and the discontinuous spectral element method on curvilinear meshes," *Journal of Scientific Computing*, vol. 26, pp. 301–327, Mar. 2006.

[90] D. Bach, A. Rueda-Ramírez, D. A. Kopriva, and G. J. Gassner, "Mimetic metrics for the DGSEM," 2025. [Online]. Available: `https://arxiv.org/abs/2410.14502`.

[91] D. A. Kopriva, "Spectral approximation," in *Implementing Spectral Methods for Partial Differential Equations: Algorithms for Scientists and Engineers*, pp. 3–38, Dordrecht: Springer Netherlands, 2009.

[92] D. N. Arnold, F. Brezzi, B. Cockburn, and L. D. Marini, "Unified analysis of discontinuous Galerkin methods for elliptic problems," *SIAM Journal on Numerical Analysis*, vol. 39, no. 5, pp. 1749–1779, 2002.

[93] G. J. Gassner, A. R. Winters, F. J. Hindenlang, and D. A. Kopriva, "The BR1 scheme is stable for the compressible Navier–Stokes equations," *Journal of Scientific Computing*, vol. 77, pp. 154–200, Oct 2018.

[94] F. Brezzi, G. Manzini, D. Marini, P. Pietra, and A. Russo, "Discontinuous Galerkin approximations for elliptic problems," *Numerical Methods for Partial Differential Equations*, vol. 16, no. 4, pp. 365–378, 2000.

[95] S. Quaegebeur and S. Nadarajah, "Stability of energy stable flux reconstruction for the diffusion problem using the interior penalty and Bassi and Rebay II numerical fluxes for linear triangular elements," *Journal of Computational Physics*, vol. 380, pp. 88–118, 2019.

[96] J. Manzanero, A. M. Rueda-Ramírez, G. Rubio, and E. Ferrer, "The Bassi Rebay 1 scheme is a special case of the Symmetric Interior Penalty formulation for discontinuous Galerkin discretisations with Gauss–Lobatto points," *Journal of Computational Physics*, vol. 363, pp. 1–10, 2018.

[97] S. Ortleb, "A comparative Fourier analysis of discontinuous Galerkin schemes for advection–diffusion with respect to BR1, BR2, and local discontinuous Galerkin diffusion discretization," *Mathematical Methods in the Applied Sciences*, vol. 43, no. 13, pp. 7841–7863, 2020.

[98] G. J. Gassner, "A skew-symmetric discontinuous Galerkin spectral element discretization and its relation to SBP-SAT finite difference methods," *SIAM Journal on Scientific Computing*, vol. 35, no. 3, pp. A1233–A1253, 2013.

[99] A. Kværnø, S. Nørsett, and B. Owren, "Runge-Kutta research in Trondheim," *Applied Numerical Mathematics*, vol. 22, no. 1, pp. 263–277, 1996. Special Issue Celebrating the Centenary of Runge-Kutta Methods.

[100] R. Courant, K. Friedrichs, and H. Lewy, "On the partial difference equations of mathematical physics," *IBM journal of Research and Development*, vol. 11, no. 2, pp. 215–234, 1967.

[101] S. D. Fechter, *Compressible multi-phase simulation at extreme conditions using a discontinuous Galerkin scheme*. Phd thesis, University of Stuttgart, Stuttgart, Germany, July 2015. Available at `https://elib.uni-stuttgart.de/items/43d172c5-ba2a-49a1-b0ec-110df92f2541`.

[102] R. S. Dembo, S. C. Eisenstat, and T. Steihaug, "Inexact Newton methods," *SIAM Journal on Numerical Analysis*, vol. 19, no. 2, pp. 400–408, 1982.

[103] C. T. Kelley, *Iterative Methods for Linear and Nonlinear Equations*. Society for Industrial and Applied Mathematics, 1995.

[104] Y. Saad, "Krylov subspace methods, Part I," in *Iterative Methods for Sparse Linear Systems*, ch. 6, pp. 151–216, Society for Industrial and Applied Mathematics, 2nd ed., 2003.

[105] R. J. Hanson and D. R. Kincaid, "Notes on GMRES algorithm organization," tech. rep., Department of Computer Sciences, University of Texas at Austin, 2005.

[106] Q. Zou, "GMRES algorithms over 35 years," *Applied Mathematics and Computation*, vol. 445, p. 127869, May 2023.

[107] N. Qin, D. K. Ludlow, and S. T. Shaw, "A matrix-free preconditioned Newton/GMRES method for unsteady Navier–Stokes solutions," *International Journal for Numerical Methods in Fluids*, vol. 33, no. 2, pp. 223–248, 2000.

[108] A. H. Gebremedhin and A. Walther, "An introduction to algorithmic differentiation," *WIREs Data Mining and Knowledge Discovery*, vol. 10, no. 1, p. e1334, 2020.

[109] J. Martins, P. Sturdza, and J. Alonso, "The connection between the complex-step derivative approximation and algorithmic differentiation," in *39th Aerospace Sciences Meeting and Exhibit*, 2001.

[110] G. Guennebaud, B. Jacob, *et al.*, "Eigen v3." http://eigen.tuxfamily.org, 2010.

[111] A. H. Gebremedhin, F. Manne, and A. Pothen, "What color is your Jacobian? Graph coloring for computing derivatives," *SIAM Review*, vol. 47, no. 4, pp. 629–705, 2005.

[112] T. F. Coleman and J. J. Moré, "Estimation of sparse Jacobian matrices and graph coloring blems," *SIAM Journal on Numerical Analysis*, vol. 20, no. 1, pp. 187–209, 1983.

[113] D. A. Kopriva, "Spectral element methods," in *Implementing Spectral Methods for Partial Differential Equations: Algorithms for Scientists and Engineers*, ch. 8, pp. 293–354, Dordrecht: Springer Netherlands, 2009.

[114] J. D. Cole, "On a quasi-linear parabolic equation occurring in aerodynamics," *Quarterly of Applied Mathematics*, vol. 9, p. 225–236, Oct. 1951.

[115] K. Feng and M. Qin, "Symplectic Runge-Kutta methods," in *Symplectic Geometric Algorithms for Hamiltonian Systems*, pp. 277–364, Berlin, Heidelberg: Springer Berlin Heidelberg, 2010.

[116] C. A. Kennedy and M. H. Carpenter, "Diagonally implicit Runge-Kutta methods for ordinary differential equations. a review," Tech. Rep. NASA/TM–2016–219173, NASA, 2016.

[117] E. Hairer, S. Norsett, and G. Wanner, *Solving Ordinary Differential Equations I: Nonstiff Problems*, vol. 8. Springer Berlin, Heidelberg, Jan. 1993.

[118] O. Krzikalla, A. Rempke, A. Bleh, M. Wagner, and T. Gerhold, "Spliss: A sparse linear system solver for transparent integration of emerging HPC technologies into CFD solvers and applications," in *New Results in Numerical and Experimental Fluid Mechanics XIII* (A. Dillmann, G. Heller, E. Krämer, and C. Wagner, eds.), (Cham), pp. 635–645, Springer International Publishing, 2021.

[119] F. F. Grinstein, L. G. Margolin, and W. J. Rider, eds., *Implicit Large Eddy Simulation: Computing Turbulent Fluid Dynamics*. Cambridge: Cambridge University Press, 2007.

[120] S. C. Spiegel, H. Huynh, and J. R. DeBonis, "A survey of the isentropic Euler vortex problem using high-order methods," in *22nd AIAA computational fluid dynamics conference*, p. 2444, 2015.

[121] F. R. Menter, "Two-equation eddy-viscosity turbulence models for engineering applications," *AIAA Journal*, vol. 32, no. 8, pp. 1598–1605, 1994.

[122] F. R. Menter, M. Kuntz, R. Langtry, *et al.*, "Ten years of industrial experience with the SST turbulence model," *Turbulence, heat and mass transfer*, vol. 4, no. 1, pp. 625–632, 2003.

[123] S. B. Pope, *Turbulent Flows*, pp. 260–270. Cambridge University Press, 2000.

[124] R. Alexander, "Diagonally implicit Runge–Kutta methods for stiff ODE's," *SIAM Journal on Numerical Analysis*, vol. 14, no. 6, pp. 1006–1021, 1977.

[125] R. K. Alexander and J. J. Coyle, "Runge-Kutta methods and differential-algebraic systems," *SIAM Journal on Numerical Analysis*, vol. 27, no. 3, pp. 736–752, 1990.

[126] A. Kværnø, "Singly diagonally implicit Runge–Kutta methods with an explicit first stage," *BIT Numerical Mathematics*, vol. 44, no. 3, pp. 489–502, 2004.

[127] C. A. Kennedy and M. H. Carpenter, "Additive Runge–Kutta schemes for convection–diffusion–reaction equations," *Applied Numerical Mathematics*, vol. 44, no. 1, pp. 139–181, 2003.

[128] C.-W. Shu and S. Osher, "Efficient implementation of essentially non-oscillatory shock-capturing schemes," *Journal of Computational Physics*, vol. 77, no. 2, pp. 439–471, 1988.

[129] V. Simoncini and D. Szyld, "Theory of inexact Krylov subspace methods and applications to scientific computing," *SIAM Journal on Scientific Computing*, vol. 25, pp. 454–477, July 2006.

[130] L. Wang and M. Yu, "A comparative study of implicit Jacobian-free Rosenbrock-Wanner, ESDIRK and BDF methods for unsteady flow simulation with high-order flux reconstruction formulations," 2019.

[131] T. Tsukahara, Y. Seki, H. Kawamura, and D. Tochio, "DNS of turbulent channel flow at very low Reynolds numbers," in *Proceedings of the 4th International Symposium on Turbulence and Shear Flow Phenomena*, vol. 3, June 2005.

[132] M. Carpenter and C. Kennedy, "Fourth-order 2n-storage runge-kutta schemes," Technical Memorandum 109112, NASA, July 1994.

[133] K. Iwamoto, Y. Suzuki, and N. Kasagi, "Database of fully developed channel flow.," Tech. Rep. IRL-0201, University of Tokyo, Bunkyo-ku, Tokyo 113, June 2002. Internal report.

[134] P. Welch, "The use of fast Fourier transform for the estimation of power spectra: A method based on time averaging over short, modified periodograms," *IEEE Transactions on Audio and Electroacoustics*, vol. 15, no. 2, pp. 70–73, 1967.

[135] M. Briani, G. Puppo, and G. Visconti, "Dissipation-dispersion analysis of fully-discrete implicit discontinuous Galerkin methods and application to stiff hyperbolic problems," 2024.

[136] Y. Saad, "A flexible inner-outer preconditioned GMRES algorithm," *SIAM Journal on Scientific Computing*, vol. 14, no. 2, pp. 461–469, 1993.

[137] M. Bergmann, C. Morsbach, B. F. Klose, G. Ashcroft, and E. Kügeler, "A numerical test rig for turbomachinery flows based on large eddy simulations with a high-order discontinuous Galerkin scheme—part i: Sliding interfaces and unsteady row interactions," *Journal of Turbomachinery*, vol. 146, p. 021005, Nov. 2023.

A

Time-integration Schemes

Implicit time-integration schemes

Table A.1: Butcher tableau for the backward Euler scheme

1	1
	1

Table A.2: Butcher tableau for the SDIRK2-A second-order two-stage scheme [115].

$\frac{1}{4}$	$\frac{1}{4}$	0
$\frac{3}{4}$	$\frac{1}{2}$	$\frac{1}{4}$
	$\frac{1}{2}$	$\frac{1}{2}$

Table A.3: Butcher tableau for the SDIRK2-B second-order two-stage scheme ($\gamma = 1.0 - \frac{\sqrt{2}}{2}$) [116].

0.2928932188	0.2928932188	0
1.0000000000	0.7071067812	0.2928932188
	0.7071067812	0.2928932188

Table A.4: Butcher tableau for the ESDIRK3 third-order four-stage scheme ($\gamma = 0.4358665215$) [124].

0	0	0	0	0
0.871733043	0.4358665215	0.4358665215	0	0
0.6089666304	0.2648804871	-0.0917803783	0.4358665215	0
1.0000000000	0.1921013556	-0.6181218831	0.990154006	0.4358665215
	0.1921013556	-0.6181218831	0.990154006	0.4358665215

Table A.5: Butcher tableau for the SDIRK3 third-order three-stage scheme ($\gamma = 0.4358665215$) [125].

0.4358665215	0.4358665215	0	0
0.7179332608	0.2820667392	0.4358665215	0
1.0000000000	1.2084966492	-0.6443631707	0.4358665215
	1.2084966492	-0.6443631707	0.4358665215

Table A.6: Butcher tableau for the ESDIRK4 fourth-order five-stage scheme ($\gamma = 0.4358665215$) [126].

0	0	0	0	0	0
0.8717330430	0.4358665215	0.4358665215	0	0	0
0.6461752661	0.3186742960	-0.1083655514	0.4358665215	0	0
1.0000000000	0.1023994006	-0.3768784523	0.8386125302	0.4358665215	0
1.0000000000	0.1570248979	0.1173304414	0.6166780304	-0.3268998911	0.4358665215
	0.1570248979	0.1173304414	0.6166780304	-0.3268998911	0.4358665215

Explicit time-integration schemes

Table A.7: Butcher tableau for "the" fourth-order four-stage Runge-Kutta (ERK4) scheme [117].

0	0	0	0	0
$\frac{1}{2}$	$\frac{1}{2}$	0	0	0
$\frac{1}{2}$	0	$\frac{1}{2}$	0	0
1	0	0	1	0
	$\frac{1}{6}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{6}$

Table A.8: Butcher tableau for the third-order ERK3 third-order three-stage scheme used in TRACE [128].

0	0	0	0
1	1	0	0
$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{4}$	0
	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{2}{3}$