Theoretical and Practical Aspects of Space-Time DG-SEM Implementations

Lea Miko Versbach, Viktor Linders,
Robert Klöfkorn & Philipp Birken

Theoretical and Practical Aspects of Space-Time DG-SEM Implementations

Lea Miko Versbach [1]
Viktor Linders [2]
Robert Klöfkorn [3]
Philipp Birken [4]

[1] Centre for Mathematical Sciences, Numerical Analysis, Lund University, Lund, Sweden
E-mail address: lea_miko.versbach@math.lu.se
[2] Centre for Mathematical Sciences, Numerical Analysis, Lund University, Lund, Sweden
E-mail address: viktor.linders@math.lu.se
[3] Centre for Mathematical Sciences, Numerical Analysis, Lund University, Lund, Sweden
E-mail address: robertk@math.lu.se
[4] Centre for Mathematical Sciences, Numerical Analysis, Lund University, Lund, Sweden
E-mail address: philipp.birken@math.lu.se.

Abstract. We discuss two approaches for the formulation and implementation of space-time discontinuous Galerkin spectral element methods (DG-SEM). In one, time is treated as an additional coordinate direction and a Galerkin procedure is applied to the entire problem. In the other, the method of lines is used with DG-SEM in space and the fully implicit Runge–Kutta method Lobatto IIIC in time. The two approaches are mathematically equivalent in the sense that they lead to the same discrete solution. However, in practice they differ in several important respects, including the terminology used to describe them, the structure of the resulting software, and the interaction with nonlinear solvers. Challenges and merits of the two approaches are discussed with the goal of providing the practitioner with sufficient consideration to choose which path to follow. Additionally, implementations of the two methods are provided as a starting point for further development. Numerical experiments validate the theoretical accuracy of these codes and demonstrate their utility, even for 4D problems.

2020 Mathematics Subject Classification. 65M60, 65M99.

Keywords. Space-time, Discontinuous Galerkin, DG-SEM.

1. Introduction

Typically, partial differential equations are numerically treated with a method of lines ansatz; the spatial directions are discretized first, leaving the time variable continuous. The resulting system of ordinary differential equations is then solved using a numerical method for initial value problems.

An alternative ansatz is to treat the time dimension simply as another coordinate direction, and discretize the whole space-time problem simultaneously. This approach has several advantages: Moving boundaries can be treated more easily [89] and parallelization in time is made possible [33]. However, it also imposes new challenges since the temporal direction is special and needs to follow a causality principle: The solution at a given time is affected and determined only by the solution at earlier times, never the other way around. An overview of space-time computations in practical engineering applications during the last 25 years can be found in [88].

Here, we consider the discontinuous Galerkin spectral element method (DG-SEM); see e.g. [8] for an overview and [58] for a detailed exposition. This method has been very successful for spatial discretizations as it is unstructured, of high order and is very suitable for high performance computing [61].

Further, DG-SEM fits the so-called Summation-By-Parts Simultaneous-Approximation-Term (SBP-SAT) framework [16, 35], implying L_2 stability for linear problems. Further, in the last decade, work within this framework has resulted in the development of entropy stable discretizations of arbitrarily high order [27, 28]. Such methods are provably stable for nonlinear time-dependent partial differential equations adjoined with a convex entropy, e.g. the compressible Euler and Navier–Stokes equations. Under the assumption of nonnegative density and pressure, no additional mechanisms such as filters or artificial dissipation are required for stability (although there may be other reasons for incorporating such techniques in a solver). This makes DG-SEM a very attractive choice for applications in computational fluid dynamics (CFD).

Our motivation to consider DG-SEM in a space-time formulation is twofold: Firstly, entropy stability can be extended to incorporate the temporal discretization [31], thereby resulting in a nonlinearly stable, fully discrete scheme. This stability propery would generally be lost if instead a generic time marching scheme were to be used. Secondly, the formulation naturally allows for perfectly scaling parallelization in time, with a speedup equal to the number of discretization points within a time element. There are other approaches for parallelization in time that allow for much larger speedups, but need an initial factor of additional processors before giving any speedup at all [74].

There is a strong connection between DG discretizations in time and fully implicit Runge–Kutta (RK) methods: DG-SEM in time using an upwind numerical flux is equivalent to the Lobatto IIIC family of RK methods, in the sense that the two methods give the same numerical solution [10, 69, 80]. This observation lends itself to two very different strategies for implementing DG-SEM in space and time. We can either use the method of lines with DG-SEM in space and Lobatto IIIC in time, or we can use space-time DG-SEM.

While mathematically equivalent, there are important differences between these two approaches:

- DG and RK methods have been developed largely independently. Hence, there is a language barrier between these communities, where different terminology is used, e.g. when it comes to the various concepts of order.

- The two approaches leads to different algebraic systems of linear or nonlinear equation. If solved exactly, these systems have the same solutions. However, in practice these solutions must be approximated, typically using iterative solvers. The interplay between iterative methods and the algebraic systems will in general be different, thus the two methods yield unequal numerical solutions.

- Implementing the two approaches lead to very different software structure, in particular if we wish to reuse existing software. This implies that various numerical tools and techniques may be more readily accessible in one implementation than the other, depending on whether the DG or the RK approach is chosen.

In this paper, we discuss these differences in detail so that practitioners can make an educated choice about which path to follow. Further, we present a code base for the two approaches that may be used as a basis for further development of the methods. In particular, we make use of the open source softwares Dune, the Distributed and Unified Numerics Environment, which is a modular toolbox for solving partial differential equations (PDEs) with grid-based methods [21]. We also make use of Assimulo [2], a solver package for initial value problems.

This paper is organized as follows: Following a brief literature review below, our target equation and choice of software is introduced in Section 2. In Section 3 we introduce the method of lines approach using DG-SEM with Lobatto IIIC for time stepping. In Section 4 the space-time DG-SEM is described. Throughout, code snippets are included to illustrate the details of the implementations. Theoretical aspects of the two approaches are discussed in Section 5. In particular, we demonstrate the

mathematical equivalence of DG-SEM in time and Lobatto IIIC methods, and relate the terminology employed by the DG and RK communities. Practical aspects of the respective implementations are the subject of Section 6. Here we compare algorithmic and implementation specific requirements and merits of the two approaches. In Section 7 we present numerical experiments that validate some of our discussion points before we finish the article with some concluding remarks in Section 8.

The material in this paper is part of the thesis [96]. For full disclaimer, see Author Contributions.

1.1. **Further reading**

For the practitioner who wishes to delve deeper into various aspects of the topics discussed in this paper, we here suggest a few places to start for further reading.

The book [58] provides a background on spatial DG-SEM as well as a guide to its implementation. An overview of entropy stable DG-SEM is given in [36] and full mathematical detail is provided in [15]. The theory builds upon the SBP-SAT framework, reviews of which are found in [24, 87].

For a broad background on implicit Runge–Kutta methods, see [40]. An overview of the properties of DG-SEM and other SBP-SAT methods for time integration viewed from the Runge–Kutta perspective is given in [67]; see also [76]. An evaluation of fully implicit RK methods for use in computational fluid dynamics is given in [48], including discussions of how to solve the nonlinear algebraic systems.

An overview of space-time finite element (FE) methods (with particular focus on polynomial-based methods such as DG) is provided in [32, Section 1]. See the references therein for a broad perspective on unstructured meshes, choices of basis functions, interpolation and integration.

There is an extensive literature on applications of space-time FE/DG methods for particular classes of partial differential equations. For application to hyperbolic problems, see e.g. [26, 31, 47], advection-diffusion problems [56, 86], the compressible Euler equations [92, 94], wave equations [32, 57, 93], the Schrödinger equation [52], the Navier–Stokes equations [55, 56, 73] and fractional PDEs [63].

Several nonlinear solvers with application to implicit RK and DG methods have recently been presented in the literature. For Newton-type methods, see e.g. [25, 79] and the references therein. Solvers utilizing multigrid techniques are discussed and analyzed in [29, 34, 97].

Preconditioners for space-time solvers have been studied for the advection, Euler, and Navier–Stokes equations [79], compressible Navier–Stokes equations [25, 29, 98], different parabolic PDEs [51, 70], convection-diffusion problems in particular [30, 38, 95], elliptic problems [41, 42], and linearized versions of the compressible Euler equations [90, 91].

2. **Governing Equations and Simulation Software**

We consider a general class of time dependent nonlinear advection-diffusion-reaction problems

$$\partial_t u = \mathcal{L}(u) := -\nabla \cdot \big(F_c(u) - F_v(u, \nabla u)\big) + S(u) \quad \text{in } \Omega \times (0, T), \tag{2.1}$$

for a vector valued function $u \colon \Omega \times (0, T) \to \mathbb{R}^r$ with $r \in \mathbb{N}^+$ components. Here, $\Omega \subset \mathbb{R}^d$, $d = 1, 2, 3$. Suitable initial and boundary conditions are assumed to be available. F_c and F_v describe the convective and viscous fluxes respectively, and S is a source term. We allow for the possibility that any of the coefficients in the partial differential equation (PDE) (2.1) depend explicitly on the spatial variable x and on time t, but to simplify the presentation we suppress this dependency in our notation.

For the discretization of (2.1) we consider two approaches: The first is a method of lines approach, in which the spatial differential operator is discretized using a DG-SEM approximation, yielding a system of ordinary differential equations (ODEs). This system is then solved using a time stepping scheme. In particular, we consider the Lobatto IIIC family of implicit Runge–Kutta methods.

The second approach is to apply the DG-SEM methodology to the entire equation (2.1), thereby obtaining a fully implicit DG space-time discretization.

In the following we will include code snippets to clarify the overall structure of the mathematical formulations at hand and to illustrate how the two approaches can be implemented in an existing code base. We utilize Dune [6], which is a free and open source software framework for the grid-based numerical solution of PDEs. Dune provides one of the most flexible and comprehensive grid interfaces available, allowing n-dimensional grids, which we will use in this paper. Additionally, state-of-the-art features such as parallelization, grid adaptivity and load balancing, and moving grids are supported. From the variety of Dune modules available we will make use of the Python based front-end for Dune-Fem [22] and Dune-Fem-DG [19], which is able to handle weak forms of PDEs described in the Unified Form Language (UFL) [1]. As shown in the next section, the description of weak forms with UFL is straight forward and easy to use. Internally, PDEs described in UFL are translated into C++ code just-in-time, to ensure that the resulting simulation code is performant. For a more detailed description we refer to [19, 22] and the tutorial [20].

The implementation of the Lobatto IIIC method (see [64]) has been done in Assimulo [2], a Python package that can be readily used together with Dune-Fem. Assimulo provides a high-level interface for a wide variety of classic and modern solvers of ordinary differential equations. e.g. SUNDIALS [45] and implicit Runge–Kutta solvers [39, 40]. The original codes, which are written in FORTRAN, C or Python, are wrapped into Assimulo keeping their original form. The user only needs to interact with the Python interface to provide the ODE and the initial condition, and possible additional information such as the Jacobian if needed by the solver. Solver options in Assimulo include explicit and implicit Euler, Runge–Kutta34, RADAU5ODE, CVODE, IDA, ODASSL, LSODAR and GLIMDA.

The appendix "Installation of Dune and Assimulo" contains instructions on how to install Dune-Fem-DG and Assimulo.

3. Method of Lines DG-SEM

In this section we describe the method of lines (MOL) approach to discretizing (2.1). A generic DG method is first presented, followed by the specifications needed to obtain the DG-SEM. Finally, the Lobatto IIIC time stepping method is specified.

3.1. DG-SEM in Space

Given a tessellation $\mathcal{T}_h$ of the computational domain Ω into open elements E with $\bigcup_{E \in \mathcal{T}_h} E \subset \Omega$, consider the piecewise polynomial space

$$V_h = \{\boldsymbol{v} \in L^2(\Omega, \mathbb{R}^r) : \boldsymbol{v}|_E \in [\mathcal{P}_p(E)]^r, \ E \in \mathcal{T}_h\}, \quad p \in \mathbb{N}, \tag{3.1}$$

where $\mathcal{P}_p(E)$ is the space of polynomials whose degree do not exceed p. We let Γ_i denote the set of intersections between all pairs of elements in $\mathcal{T}_h$ and accordingly Γ the set of all intersections including the boundary of Ω. In Dune, the following commands generate the tessellation $\mathcal{T}_h$ and the space V_h:

```
d = 2 # 1,2,3
from dune.grid import cartesianDomain, structuredGrid as leafGrid
# create grid that tessellates [0,1]^d with 10 elements in each coordinate direction
T_h = leafGrid(cartesianDomain([0]*d, [1]*d, [10]*d))


from dune.fem.space import dglagrangelobatto
p = 3 # polynomial degree
# create DG space with Lagrange basis and Gauss-Lobatto interpolation points
V_h = dglagrangelobatto( T_h, order=p )
```

We seek an approximate solution $\boldsymbol{u}_h \in V_h$ by discretizing the spatial operator $\mathcal{L}(u)$ in (2.1). To this end we define for all test functions $\boldsymbol{\psi} \in V_h$,

$$\langle \boldsymbol{\psi}, \mathcal{L}_h(\boldsymbol{u}_h) \rangle := \langle \boldsymbol{\psi}, K_h(\boldsymbol{u}_h) \rangle + \langle \boldsymbol{\psi}, I_h(\boldsymbol{u}_h) \rangle. \tag{3.2}$$

Here, the element integrals are given by

$$\langle \boldsymbol{\psi}, K_h(\boldsymbol{u}_h) \rangle := \sum_{E \in \mathcal{T}_h} \int_E \left((F_{\mathrm{c}}(\boldsymbol{u}_h) - F_{\mathrm{v}}(\boldsymbol{u}_h, \nabla \boldsymbol{u}_h)) : \nabla \boldsymbol{\psi} + S(\boldsymbol{u}_h) \cdot \boldsymbol{\psi} \right) \mathrm{d}x, \tag{3.3}$$

where : denotes the inner product of two second order tensors. In the code this looks as follows:

```python
from ufl import TrialFunction, TestFunction, inner, grad, dx
# trial and test function
u   = TrialFunction(V_h)
psi = TestFunction(V_h)
# element integral from equation (3.3)
K_h = inner(F_c(u) - F_v(u)*grad(u)), grad(psi)) * dx \   # fluxes
    + inner(S(u), psi) * dx                               # source term
```

The surface integrals are given by

$$\langle \boldsymbol{\psi}, I_h(\boldsymbol{u}_h) \rangle := \sum_{e \in \Gamma_i} \int_e \left(\{\!\!\{ F_{\mathrm{v}}(\boldsymbol{u}_h, [\![\boldsymbol{u}_h]\!]_e)^T : \nabla \boldsymbol{\psi} \}\!\!\}_e + \{\!\!\{ F_{\mathrm{v}}(\boldsymbol{u}_h, \nabla \boldsymbol{u}_h) \}\!\!\}_e : [\![\boldsymbol{\psi}]\!]_e \right) \mathrm{d}S$$

$$- \sum_{e \in \Gamma} \int_e \left(H_{\mathrm{c}}(\boldsymbol{u}_h) - H_{\mathrm{v}}(\boldsymbol{u}_h, \nabla \boldsymbol{u}_h) \right) : [\![\boldsymbol{\psi}]\!]_e \, \mathrm{d}S. \tag{3.4}$$

Here, $\{\!\!\{\boldsymbol{u}_h\}\!\!\}_e$ and $[\![\boldsymbol{u}_h]\!]_e$ denote the average and jump of $\boldsymbol{u}_h$ over interface e between neighbouring elements E and K. Specifically,

$$\{\!\!\{\boldsymbol{u}_h\}\!\!\}_e := \frac{1}{2}(\boldsymbol{u}_E + \boldsymbol{u}_K) \quad \text{and} \quad [\![\boldsymbol{u}_h]\!]_e := \neq: (\boldsymbol{u}_E - \boldsymbol{u}_K), \tag{3.5}$$

where $\neq$ is the outward pointing normal vector from the point of view of element E. This formulation arises from considering the weak form of the problem: Replace u by $\boldsymbol{u}_h$ in (2.1), multiply by the test function $\boldsymbol{\psi}$ and integrate the spatial terms by parts. Here, H_{c} and H_{v} are suitable numerical fluxes imposed at e, approximating F_{c} and F_{v} respectively. Details of the derivation are found in [3].

The convective numerical flux H_{c} can be any flux known for standard finite volume methods. Among other options available in Dune-Fem-DG [18, 19], we use the local Lax–Friedrichs (Rusanov) flux

$$H_{\mathrm{c}}^{\mathrm{LLF}}(\boldsymbol{u}_h)|_e := \{\!\!\{ F_{\mathrm{c}}(\boldsymbol{u}_h) \}\!\!\}_e + \frac{\lambda_e}{2} [\![\boldsymbol{u}_h]\!]_e \tag{3.6}$$

where λ_e is an estimate of the maximum wave speed on the interface e.

A wide range of diffusion fluxes H_{v} can be found in the literature; see [11] and the references therein. However, only fluxes from the Interior Penalty family can currently be described in UFL due to the missing description and implementation in UFL of lifting terms needed for other fluxes. For the Interior Penalty method the flux is chosen to be

$$H_{\mathrm{v}}^{\mathrm{IP}}(u, \nabla u) = \{\!\!\{\nabla u\}\!\!\}_e - \frac{\eta}{h_e} \{\!\!\{ F_{\mathrm{v}}(u, \nabla u) \}\!\!\}_e [\![u]\!]_e \tag{3.7}$$

with η being the penalty parameter. The corresponding code reads:

```python
from ufl import FacetNormal, FacetArea, CellVolume, avg, jump, dS, ds
# normal and mesh width
n = FacetNormal(V_h)
h_e = avg( CellVolume(V_h) ) / FacetArea(V_h)
# surface integral from equation (3.4)
```

```
I_h = inner(jump(H_c(u), jump(psi)) * dS \   # interior skeleton for convective part
    + H_cb(u)*psi*ds \                        # domain boundary for convective part
    - inner(jump(F_v(u),n),avg(grad(psi)))   * dS \   # symmetry term
    - inner(avg(F_v(u)*grad(u)),jump(psi,n)) * dS \   # consistency term
    + eta/h_e*inner(jump(u, avg(F_v(u))*n),jump(psi,n)) * dS   # penalty term
```

To obtain the DG-SEM we follow [59, 60]. First, we restrict our focus to cuboid meshes and map each element $E \in \mathcal{T}_h$ to a reference element using an affine mapping. In the Dune implementation, the reference element is $[0, 1]^d$. This is due to a generic construction of reference elements of different shapes in arbitrary dimensions in Dune; see [23] for details.

In each spatial dimension, a set of $p + 1$ Legendre–Gauss–Lobatto (LGL) nodes are introduced and a corresponding set of Lagrange basis polynomials are defined. The discrete solution $\boldsymbol{u}_h(t) \in V_h$ takes the form

$$\boldsymbol{u}_h(t, x) = \sum_i u_i(t) \boldsymbol{\psi}_i(x),$$

where the sum is taken over all tensor product LGL nodes in d dimensions and $\boldsymbol{\psi}_i(x)$ is constructed as the product of Lagrange basis polynomials along each dimension. This is achieved through the command

```
# create discrete function given a discrete space
u_h = V_h.function(name="u_h")
```

The convective flux F_c and the viscous flux F_v are similarly approximated in the Lagrange basis. A variety of implementations for F_c and F_v is provided by the dolfin_dg package (see [46]), which we use for the Euler equations.

Finally, the element and surface integrals in (3.3) and (3.4) are approximated using Gauss–Lobatto quadrature rules. The collocation of the quadrature with the LGL nodes results in a diagonal positive definite local mass matrix. The choice of a cuboid mesh and a tensor product formulation of the basis functions ensures that the global mass matrix remains diagonal positive definite and is consequently trivially invertible.

3.2. Temporal Discretization

After spatial discretization, we obtain a system of ODEs for the coefficient functions $\boldsymbol{u}(t) = (u_1(t), u_2(t), \dots)^\top$, which reads

$$\boldsymbol{u}'(t) = \boldsymbol{F}(t, \boldsymbol{u}(t)), \quad t \in (0, T), \quad \boldsymbol{u}(0) = \boldsymbol{u}_0. \tag{3.8}$$

Here, $\boldsymbol{F}(t, \boldsymbol{u}(t)) = \boldsymbol{M}^{-1}\mathcal{L}_h(\boldsymbol{u}_h(t))$, where $\mathcal{L}_h$ is defined in (3.2) and $\boldsymbol{M}$ is the (diagonal) global mass matrix of the DG-SEM discretization. The initial data $\boldsymbol{u}_0$ for (3.8) is given by the projection of u_0 onto V_h.

Any Runge–Kutta method can in principle be used to solve (3.8). Explicit methods are easy to implement but suffer from severe time step restrictions for stiff systems.

Consider instead an implicit RK method with Butcher tableau

$$\begin{array}{c|c} \boldsymbol{c} & \boldsymbol{A} \\ \hline & \boldsymbol{b}^\top \end{array}$$

The *stage equations* of the RK method take the form

$$\underline{\boldsymbol{u}} = \boldsymbol{1} \otimes \boldsymbol{u}^n + \Delta t_n (\boldsymbol{A} \otimes \boldsymbol{I}_\xi)\underline{\boldsymbol{F}}, \tag{3.9}$$

where the vector $\underline{\boldsymbol{u}}^\top = (\boldsymbol{u}^1, \ldots, \boldsymbol{u}^{N_\tau})$ contains the N_τ intermediate solution stages and $\underline{\boldsymbol{F}}^\top = (\boldsymbol{F}(t_n + \Delta t_n c_1, \boldsymbol{u}^1), \ldots, \boldsymbol{F}(t_n + \Delta t_n c_{N_\tau}, \boldsymbol{u}^{N_\tau}))^\top$. Here, $\boldsymbol{u}^n$ denotes the RK solution in the previous time step and $\Delta t_n = t_{n+1} - t_n$ is the time step. The new solution is given by

$$\boldsymbol{u}^{n+1} = \boldsymbol{u}^n + \Delta t_n (\boldsymbol{b}^\top \otimes \boldsymbol{I}_\xi) \underline{\boldsymbol{F}}. \tag{3.10}$$

Herein we consider a particular family of implicit RK methods, namely Lobatto IIIC [49, 64]. These methods are A-, L- and B-stable and are thus suitable for stiff and nonlinear problems. The order of the N_τ-stage Lobatto IIIC method is $2(N_\tau - 1)$ and the order of the individual stages is $N_\tau - 1$. However, this choice of method is also motivated by its equivalence to a space-time DG-SEM formulation, which is described in the next section. The Butcher tableaus for the 2-, 3- and 4-stage Lobatto IIIC methods are found in the appendix "Butcher tableaus".

The following code is an example how to use the Lobatto IIIC solvers in ASSIMULO:

```python
# import solver form assimulo
import assimulo.ode as aode
import assimulo.solvers as aso
# import Lobatto IIIC solvers
from Lobatto_IIIC_2s import Lobatto2ODE
from Lobatto_IIIC_3s import Lobatto3ODE
from Lobatto_IIIC_4s import Lobatto4ODE

# set up explicit problem, user-defined rhs
prob = aode.Explicit_Problem(rhs, y0, t0)
# user-defined Jacobian
prob.jac = jacobian
# choose solver
solver = Lobatto2ODE(prob)
# run solver until endTime
t, y = solver.simulate(endTime)
```

4. Space-Time DG-SEM

We now consider DG-SEM applied to (2.1) with the time variable t treated simply as an additional dimension. The result is space-time DG-SEM.

Defining the gradient $\underline{\nabla} := \left(\nabla, \frac{\partial}{\partial t}\right)$ and the new convective and viscous fluxes

$$\underline{F}_{\mathrm{c}} = \begin{bmatrix} F_{\mathrm{c}} & u \end{bmatrix}, \quad \underline{F}_{\mathrm{v}} = \begin{bmatrix} F_{\mathrm{v}} & 0 \end{bmatrix},$$

we rewrite (2.1) as a $d+1$-dimensional problem on the space-time domain $\underline{\Omega} := \Omega \times (0, T) \subset \mathbb{R}^{d+1}$ as

$$\underline{\nabla} \cdot \left(\underline{F}_{\mathrm{c}}(u) - \underline{F}_{\mathrm{v}}(u, \nabla u)\right) = S(u) \quad \text{in } \underline{\Omega}. \tag{4.1}$$

Given a tessellation $\underline{\mathcal{T}_h}$ of $\underline{\Omega}$ we introduce the piecewise polynomial space

$$\underline{V_h} = \{\underline{\boldsymbol{v}} \in L^2(\underline{\Omega}, \mathbb{R}^r) : \underline{\boldsymbol{v}}|_E \in [\mathcal{P}_p(E)]^r, \ E \in \underline{\mathcal{T}_h}\}, \quad p \in \mathbb{N}. \tag{4.2}$$

Then the space-time DG-SEM discretization of (4.1) follows analogously to (3.3) and (3.4):

$$\langle \underline{\boldsymbol{\psi}}, \underline{\mathcal{L}}_h(\underline{\boldsymbol{u}}_h) \rangle := \langle \underline{\boldsymbol{\psi}}, \underline{K}_h(\underline{\boldsymbol{u}}_h) \rangle + \langle \underline{\boldsymbol{\psi}}, \underline{I}_h(\underline{\boldsymbol{u}}_h) \rangle, \tag{4.3}$$

with the element integrals

$$\langle \underline{\boldsymbol{\psi}}, \underline{K}_h(\underline{\boldsymbol{u}}_h) \rangle := \sum_{E \in \underline{\mathcal{T}_h}} \int_E \left((\underline{F}_{\mathrm{c}}(\underline{\boldsymbol{u}}_h) - \underline{F}_{\mathrm{v}}(\underline{\boldsymbol{u}}_h, \underline{\nabla}\underline{\boldsymbol{u}}_h)) : \nabla\underline{\boldsymbol{\psi}} + S(\underline{\boldsymbol{u}}_h) \cdot \underline{\boldsymbol{\psi}}\right) \mathrm{d}x, \tag{4.4}$$

67

and the surface integrals

$$\langle \underline{\psi}, \underline{I}_h(\underline{\boldsymbol{u}}_h) \rangle := \sum_{e \in \underline{\Gamma}_i} \int_e \left(\{\!\!\{ \underline{F}_{\mathrm{v}}(\underline{\boldsymbol{u}}_h, [\![\underline{\boldsymbol{u}}_h]\!]_e)^\top \}\!\!\}_e : \nabla \underline{\psi} \}\!\!\}_e + \{\!\!\{ \underline{F}_{\mathrm{v}}(\underline{\boldsymbol{u}}_h, \nabla \underline{\boldsymbol{u}}_h) \}\!\!\}_e : [\![\underline{\psi}]\!]_e \right) \mathrm{d}\underline{S}$$

$$- \sum_{e \in \underline{\Gamma}} \int_e \left(\underline{H}_{\mathrm{c}}(\underline{\boldsymbol{u}}_h) - \underline{H}_{\mathrm{v}}(\underline{\boldsymbol{u}}_h, \nabla \underline{\boldsymbol{u}}_h) \right) : [\![\underline{\psi}]\!]_e \, \mathrm{d}S. \quad (4.5)$$

Here, $\underline{\Gamma}_i$ and $\underline{\Gamma}$ have analogous meanings to their spatial counterparts Γ_i and Γ. The numerical fluxes are given by

$$\underline{H}_{\mathrm{c}} = \begin{bmatrix} H_{\mathrm{c}} & u^* \end{bmatrix}, \quad \underline{H}_{\mathrm{v}} = \begin{bmatrix} H_{\mathrm{v}} & 0 \end{bmatrix},$$

where u^* is a simple upwind flux in time. The reason for using the upwind flux is that it is the only choice that *decouples* the temporal elements into time slabs. Effectively, the space-time problem on $\Omega \times (0, T)$ is replaced by a sequence of considerably smaller problems, each defined on $\Omega \times (t_n, t_{n+1})$ for $n = 0, 1, \ldots$

In our framework, (4.3) can be implemented quite nicely by increasing the dimension and applying the above discussed modifications[1].

```
d = 2 # 1,2,3 is the spatial dimension
from dune.grid import cartesianDomain, structuredGrid as leafGrid
t_end, timeSteps = 1.0, 10
dt = t_end / timeSteps
# create grid that tessellates [0,1]^d x [0,Δt] with 10 elements in space and 1 element in time
T_h = leafGrid(cartesianDomain([0]*d + [0], [1]*d + [dt], [10]*d + [1])) # create a space-time grid
p = 3 # polynomial degree
# create DG space with Lagrange basis and Gauss-Lobatto interpolation points
V_h = dglagrangelobatto( T_h, order=p )

def appendTime( F, u ):
  return ufl.as_tensor([ *[[F[k,i] if i<d else u[k] for i in range(d+1)] for k in range(len(u))] ])

def F_c( u ):
  from molspacediscr import F_c # import F_c used in MOL discretization
  F_spc = F_c(u) # compute spatial fluxes
  # append time derivative as last column
  return appendTime( F_spc, u )

def F_v( u ):
  from molspacediscr import F_v # import F_v used in MOL discretization
  F_spc = F_v(u) # compute spatial fluxes
  # append column of zeros since there is no diffusion in time
  return appendTime( F_spc, [0.]*len(u) )

# trial and test function
u   = TrialFunction(V_h)
psi = TestFunction(V_h)
# element integral from equation (4.4)
K_h = inner(F_c(u) - F_v(u)*grad(u)), grad(psi)) * dx \   # fluxes
    + inner(S(u), psi) * dx                               # source term

# normal and mesh width
n   = FacetNormal(V_h)
h_e = avg( CellVolume(V_h) ) / FacetArea(V_h)
# penalty parameter for Symmetric Interior Penalty scheme
eta = Constant( 10*V_h.order**2 if V_h.order > 0 else 1, "penalty" )
# surface integral from equation (4.5)
```

---

[1]Note that in 4D (i.e. 3D + time) a UFL patch has been added to introduce the 4D reference elements to UFL code. See the appendix "UFL patch".

```
I_h  =  inner(jump(H_c(u), jump(psi)) * dS \    # interior skeleton for convective part
     +  H_cb(u)*psi*ds \                          # domain boundary for convective part
     -  inner(jump(F_v(u),n),avg(grad(psi)))    * dS \    # symmetry term
     -  inner(avg(F_v(u)*grad(u)),jump(psi,n)) * dS \     # consistency term
     +  eta/h_e*inner(jump(u, avg(F_v(u))*n),jump(psi,n)) * dS   # penalty term
```

Remark 4.1. It is of practical interest to generalize the space $\underline{V_h}$ so that the time dimension may be discretized by polynomials of a different degree than the spatial dimensions. We will henceforth refer to the number of temporal nodes in each element as N_τ so that the polynomial degree in time is $N_\tau - 1$. This notation contrasts standard DG terminology, where nodes are typically indexed from 0 to p. Additionally, note that this is the same notation used for the number of stages of the Lobatto IIIC method in Section 3. Stages are typically indexed from 1 to s. However, to minimize the use of notation and to make the connection between the two viewpoints clearer, we write N_τ to count the degrees of freedom within a time element, whether this pertains to the DG or RK interpretation.

The discrete solution $\underline{u}_h \in \underline{V_h}$ takes the form $\underline{u}_h(t,x) = \sum_{i,n} u_i^n \boldsymbol{\psi}_i(x) \psi_n(t)$. Here, the sum is taken over all tensor product LGL nodes in $d+1$ dimensions. The vector of coefficients is now given by

$$\underline{u} = (\boldsymbol{u}^1, \ldots, \boldsymbol{u}^{N_\tau})^\top, \tag{4.6}$$

where $\boldsymbol{u}^i$ contains all the spatial unknowns in the ith time element.

The space-time discretization (4.3) can alternatively be derived by starting from (3.8) and discretizing in time with DG-SEM. Multiplying (3.8) by a test function $\psi(t)$ and integrating over the nth time element results in

$$\int_{t_n}^{t_{n+1}} \boldsymbol{u}_t \psi \mathrm{d}t = \int_{t_n}^{t_{n+1}} \boldsymbol{F}(t, \boldsymbol{u}(t)) \psi(t) \mathrm{d}t.$$

We transform this equation to the reference element $[-1, 1]$ using the mapping $t = t_n + \frac{\Delta t_n}{2}(1 + \tau)$, where $\Delta t_n = t_{n+1} - t_n$. After integration by parts the resulting equation reads

$$[\boldsymbol{u}\psi]_{-1}^1 - \int_{-1}^1 \boldsymbol{u}\psi_\tau \mathrm{d}\tau = \frac{2}{\Delta t_n} \int_{-1}^1 \boldsymbol{F}(\tau, \boldsymbol{u}(\tau)) \psi(\tau) \mathrm{d}\tau.$$

We now follow the steps of DG-SEM, i.e. approximating $\boldsymbol{u}$ and $\boldsymbol{F}$ by interpolants

$$\boldsymbol{u} \approx \sum_{j=1}^{N_\tau} \boldsymbol{u}^j \psi_j(\tau), \quad \boldsymbol{F} \approx \sum_{j=1}^{N_\tau} \boldsymbol{F}^j \psi_j(\tau),$$

and the integrals by Gauss–Lobatto quadrature with nodes τ_j and weights ω_j. Using the cardinal property of the Lagrange basis polynomials ψ, the resulting DG-SEM discretization becomes

$$\delta_{iN_\tau}\boldsymbol{u}^* - \delta_{i1}\boldsymbol{u}^* - \sum_{j=1}^{N_\tau} \omega_j \boldsymbol{u}^j \left. \frac{\mathrm{d}\psi_i}{\mathrm{d}\tau} \right|_{\tau_j} = \frac{2}{\Delta t_n} \omega_i \boldsymbol{F}^i, \quad i = 1, \ldots, N_\tau. \tag{4.7}$$

Here, we have replaced the boundary terms with numerical fluxes $\boldsymbol{u}^*$. As previously, an upwind flux is chosen, which has the advantage of decoupling the temporal elements. Thus, (4.8) can be solved as a stand-alone nonlinear system on the nth time element.

Defining the boundary, mass and differentiation matrices

$$\boldsymbol{B}_\tau = \mathrm{diag}([-1, 0, \ldots, 0, 1]) \in \mathbb{R}^{N_\tau \times N_\tau},$$

$$\boldsymbol{M}_\tau = \mathrm{diag}([\omega_1, \ldots, \omega_{N_\tau}]) \in \mathbb{R}^{N_\tau \times N_\tau},$$

$$(\boldsymbol{D}_\tau)_{ji} = \left. \frac{\mathrm{d}\psi_i}{\mathrm{d}\tau} \right|_{\tau_j} \in \mathbb{R}^{N_\tau \times N_\tau},$$

69

we can write (4.7) in matrix form on each reference element as

$$(\boldsymbol{B}_\tau \otimes \boldsymbol{I}_\xi)\,\underline{\boldsymbol{u}}^* - \left(\boldsymbol{D}_\tau^\top \boldsymbol{M}_\tau \otimes \boldsymbol{I}_\xi\right)\underline{\boldsymbol{u}} = \frac{\Delta t_n}{2}(\boldsymbol{M}_\tau \otimes \boldsymbol{I}_\xi)\underline{\boldsymbol{F}}(\underline{\boldsymbol{u}}). \tag{4.8}$$

Here, $\underline{\boldsymbol{u}}^* = (\boldsymbol{u}^n, \boldsymbol{0}, \ldots, \boldsymbol{0}, \boldsymbol{u}^{N_\tau})^\top$ where $\boldsymbol{u}^n$ is the numerical solution from the previous time element. Further, $\boldsymbol{M}_\tau$ is the local temporal mass matrix and $\boldsymbol{M}_\tau \boldsymbol{D}_\tau$ defines the corresponding stiffness matrix. Here, $\underline{\boldsymbol{F}}^\top(\underline{\boldsymbol{u}}) = (\boldsymbol{F}^\top(t_n + \frac{\Delta t_n}{2}(1 + \tau_1), \boldsymbol{u}^1), \ldots, \boldsymbol{F}^\top(t_n + \frac{\Delta t_n}{2}(1 + \tau_{N_\tau}), \boldsymbol{u}^{N_\tau}))$, where $\underline{\boldsymbol{u}}$ is given by (4.6) and τ_k is the kth LGL node; see [35] for details. The operation $\otimes$ denotes the Kronecker product and $\boldsymbol{I}_\xi$ is the identity matrix whose dimension is given by the number of spatial nodes.

We finish this section by remarking that while (4.3) describes the global space-time DG-SEM discretization, the alternative formulation (4.8) pertains to a single time element.

5. Theoretical Aspects of Space-Time DG-SEM

In this section we discuss important properties of the space-time DG-SEM, in particular the equivalence of the temporal discretization and the Lobatto IIIC family of Runge–Kutta methods. To make the connection between DG-SEM and Runge–Kutta methods clear, we consider the solution at the final point in the time element, i.e.

$$\boldsymbol{u}^{N_\tau} \equiv (\boldsymbol{e}_{N_\tau}^\top \otimes \boldsymbol{I}_\xi)\underline{\boldsymbol{u}}, \tag{5.1}$$

where $\boldsymbol{e}_{N_\tau}^\top = (0, \ldots, 0, 1) \in \mathbb{R}^{N_\tau}$. We will also make use of the vector $\boldsymbol{e}_1^\top = (1, 0, \ldots, 0) \in \mathbb{R}^{N_\tau}$. Following [10], we set out to show that $\boldsymbol{u}^{N_\tau} = \boldsymbol{u}^{n+1}$, where $\boldsymbol{u}^{n+1}$ is the numerical solution arising from the Lobatto IIIC method in (3.10).

The DG-SEM discretization (4.8) constitutes a so called *Summation-By-Parts* (SBP) method [35], meaning that the following conditions are satisfied:

$$\boldsymbol{M}_\tau = \boldsymbol{M}_\tau^\top > \boldsymbol{0}, \quad \boldsymbol{M}_\tau \boldsymbol{D}_\tau + (\boldsymbol{M}_\tau \boldsymbol{D}_\tau)^\top = \boldsymbol{B}_\tau. \tag{5.2}$$

The SBP property (5.2) is at the heart of the connection of DG-SEM in time to implicit Runge–Kutta methods.

5.1. DG-SEM and Lobatto IIIC

SBP methods were historically developed to be used as spatial discretizations [62, 85]. For an overview of these techniques, see [24, 87]. In recent years, their use as time stepping schemes has been explored [77] and connections to implicit Runge–Kutta methods have been discovered [10, 69]. Here we summarize the steps showing that (4.8) can be reformulated as an implicit RK method applied to the system of ODEs (3.8).

We begin by using the SBP property (5.2) in the second term of (4.8) and then multiplying by $(\boldsymbol{M}_\tau^{-1} \otimes \boldsymbol{I}_\xi)$ to obtain the so called *strong form*,

$$(\boldsymbol{D}_\tau \otimes \boldsymbol{I}_\xi)\underline{\boldsymbol{u}} = (\boldsymbol{M}_\tau^{-1}\boldsymbol{B}_\tau \otimes \boldsymbol{I}_\xi)(\underline{\boldsymbol{u}} - \underline{\boldsymbol{u}}^*) + \frac{\Delta t_n}{2}\underline{\boldsymbol{F}}. \tag{5.3}$$

Note that $\boldsymbol{B}_\tau = \boldsymbol{e}_{N_\tau}\boldsymbol{e}_{N_\tau}^\top - \boldsymbol{e}_1\boldsymbol{e}_1^\top$ and $(\boldsymbol{e}_{N_\tau}^\top \otimes \boldsymbol{I}_\xi)(\underline{\boldsymbol{u}} - \underline{\boldsymbol{u}}^*) = (\boldsymbol{u}^{N_\tau} - \boldsymbol{u}^{N_\tau}) = \boldsymbol{0}$. Using (5.1), the second term in (5.3) can therefore be expressed as

$$(\boldsymbol{M}_\tau^{-1}\boldsymbol{B}_\tau \otimes \boldsymbol{I}_\xi)(\underline{\boldsymbol{u}} - \underline{\boldsymbol{u}}^*) = -(\boldsymbol{M}_\tau^{-1} \otimes \boldsymbol{I}_\xi)[(\boldsymbol{e}_1\boldsymbol{e}_1^\top \otimes \boldsymbol{I}_\xi)\underline{\boldsymbol{u}} - (\boldsymbol{e}_1 \otimes \boldsymbol{u}^n)].$$

Grouping together terms that multiply the solution $\underline{\boldsymbol{u}}$, we rewrite (5.3) as

$$((\boldsymbol{D}_\tau + \boldsymbol{M}_\tau^{-1}\boldsymbol{e}_1\boldsymbol{e}_1^\top) \otimes \boldsymbol{I}_\xi)\underline{\boldsymbol{u}} = (\boldsymbol{M}_\tau^{-1}\boldsymbol{e}_1 \otimes \boldsymbol{u}^n) + \frac{\Delta t_n}{2}\underline{\boldsymbol{F}}. \tag{5.4}$$

Next, we multiply (5.4) by $((\boldsymbol{D}_\tau + \boldsymbol{M}_\tau^{-1}\boldsymbol{e}_1\boldsymbol{e}_1^\top) \otimes \boldsymbol{I}_\xi)^{-1}$. Upon doing this, first note that

$$(\boldsymbol{D}_\tau + \boldsymbol{M}_\tau^{-1}\boldsymbol{e}_1\boldsymbol{e}_1^\top)^{-1}\boldsymbol{M}_\tau^{-1}\boldsymbol{e}_1 = \mathbf{1} := (1, \ldots, 1)^\top \in \mathbb{R}^{N_\tau},$$

which follows from observing that $(\boldsymbol{D}_\tau + \boldsymbol{M}_\tau^{-1}\boldsymbol{e}_1\boldsymbol{e}_1^\top)\mathbf{1} = \boldsymbol{M}_\tau^{-1}\boldsymbol{e}_1$ since $\boldsymbol{D}_\tau\mathbf{1} = \mathbf{0}$ by consistency. Thus, the following system arises:

$$\begin{aligned}
\underline{\boldsymbol{u}} &= \mathbf{1} \otimes \boldsymbol{u}^n + \Delta t_n \frac{1}{2}((\boldsymbol{D}_\tau + \boldsymbol{M}_\tau^{-1}\boldsymbol{e}_1\boldsymbol{e}_1^\top) \otimes \boldsymbol{I}_\xi)^{-1}\underline{\boldsymbol{F}} \\
&= \mathbf{1} \otimes \boldsymbol{u}^n + \Delta t_n \frac{1}{2}((\boldsymbol{D}_\tau + \boldsymbol{M}_\tau^{-1}\boldsymbol{e}_1\boldsymbol{e}_1^\top)^{-1} \otimes \boldsymbol{I}_\xi)\underline{\boldsymbol{F}}.
\end{aligned} \tag{5.5}$$

The equation system (5.5) should be compared with the stage equations (3.9) that arose from the MOL discretization using implicit RK. We see that the temporal DG-SEM discretization defines an RK method with coefficient matrix $\boldsymbol{A} = \frac{1}{2}(\boldsymbol{D}_\tau + \boldsymbol{M}_\tau^{-1}\boldsymbol{e}_1\boldsymbol{e}_1^\top)^{-1}$ and nodes $\boldsymbol{c} = (\mathbf{1} + \boldsymbol{\tau})/2$, where $\boldsymbol{\tau} = (\tau_1, \ldots, \tau_{N_\tau})^\top$ is the vector of LGL nodes. Further, the vector $\underline{\boldsymbol{u}}$, which in the DG-SEM context contains the interpolation coefficients u_i^n, has adopted the role of the stage vectors of the RK method.

To complete the transition from DG-SEM to RK, we compute the numerical solution at the final time node, $\boldsymbol{u}^{N_\tau} = (\boldsymbol{e}_{N_\tau}^\top \otimes \boldsymbol{I}_\xi)\underline{\boldsymbol{u}}$. To this end we observe that the SBP property (5.2) gives the relation

$$\mathbf{1}^\top \boldsymbol{M}_\tau(\boldsymbol{D}_\tau + \boldsymbol{M}_\tau^{-1}\boldsymbol{e}_1\boldsymbol{e}_1^\top) = \mathbf{1}^\top(\boldsymbol{e}_{N_\tau}\boldsymbol{e}_{N_\tau}^\top - \boldsymbol{D}_\tau^\top \boldsymbol{M}_\tau) = \boldsymbol{e}_{N_\tau}^\top,$$

so that

$$\boldsymbol{e}_{N_\tau}^\top(\boldsymbol{D}_\tau + \boldsymbol{M}_\tau^{-1}\boldsymbol{e}_1\boldsymbol{e}_1^\top)^{-1} = \mathbf{1}^\top \boldsymbol{M}_\tau.$$

Consequently, multiplying (5.5) by $(\boldsymbol{e}_{N_\tau}^\top \otimes \boldsymbol{I}_\xi)$ yields

$$\boldsymbol{u}^{N_\tau} = \boldsymbol{u}^n + \Delta t_n \frac{1}{2}(\mathbf{1}^\top \boldsymbol{M}_\tau \otimes \boldsymbol{I}_\xi)\underline{\boldsymbol{F}}. \tag{5.6}$$

Comparing (5.6) with the solution (3.10) of the implicit RK method, we see that the vector $\boldsymbol{b}$ in the Butcher tableau is related to the DG-SEM discretization by $\boldsymbol{b}^\top = \mathbf{1}^\top \boldsymbol{M}_\tau/2$, and that the RK solution is simply the N_τth component of the DG solution $\underline{\boldsymbol{u}}$.

To summarize, a DG-SEM time discretization is equivalent to an implicit RK method whose Butcher tableau is defined in terms of the DG method as

$$\boldsymbol{A} = \frac{1}{2}(\boldsymbol{D}_\tau + \boldsymbol{M}_\tau^{-1}\boldsymbol{e}_1\boldsymbol{e}_1^\top)^{-1}, \quad \boldsymbol{b} = \frac{1}{2}\boldsymbol{M}_\tau\mathbf{1}, \quad \boldsymbol{c} = \frac{\mathbf{1} + \boldsymbol{\tau}}{2}. \tag{5.7}$$

The two methods yield two different nonlinear systems; for DG-SEM and RK they are respectively given by

$$(\boldsymbol{B}_\tau \otimes \boldsymbol{I}_\xi)\underline{\boldsymbol{u}}^* - \left(\boldsymbol{D}_\tau^\top \boldsymbol{M}_\tau \otimes \boldsymbol{I}_\xi\right)\underline{\boldsymbol{u}} = \frac{\Delta t_n}{2}(\boldsymbol{M}_\tau \otimes \boldsymbol{I}_\xi)\underline{\boldsymbol{F}}, \tag{5.8a}$$

$$\underline{\boldsymbol{u}} = \mathbf{1} \otimes \boldsymbol{u}^n + \Delta t_n(\boldsymbol{A} \otimes \boldsymbol{I}_\xi)\underline{\boldsymbol{F}}. \tag{5.8b}$$

These systems have the same solution $\underline{\boldsymbol{u}}$ since we can transition from (5.8a) to (5.8b) in a series of algebraic steps. More precisely, the connection is made by rewriting (5.8a) in strong form, then multiplying by $(\boldsymbol{D}_\tau + \boldsymbol{M}_\tau^{-1}\boldsymbol{e}_1\boldsymbol{e}_1^\top \otimes \boldsymbol{I}_\xi)^{-1}$. Note that the last step demands that $\boldsymbol{D}_\tau + \boldsymbol{M}_\tau^{-1}\boldsymbol{e}_1\boldsymbol{e}_1^\top$ is invertible. This is the case if and only if $\boldsymbol{D}_\tau$ is *null-space consistent*, i.e. if $\ker(\boldsymbol{D}_\tau) = \mathrm{span}(\mathbf{1})$ [67]. This is known to hold for all $N_\tau > 1$ [65, 82].

Finally, the Butcher tableau formed from (5.7) coincides with that of the Lobatto IIIC family of implicit Runge–Kutta methods. This follows from the use of LGL nodes and quadrature weights, together with a set of accuracy conditions satisfied by the two formulations [80]. We will detail these in the next section. The derivation above therefore shows that DG-SEM in time and the Lobatto IIIC methods are mathematically equivalent, and that we in fact have $\boldsymbol{u}^{N_\tau} = \boldsymbol{u}^{n+1}$. The coefficients for the DG-SEM matrices with $N_\tau \in \{2, 3, 4\}$ are listed in the appendix "DG-SEM operators".

5.2. **Comparison of terminology**

While DG-SEM in time and Lobatto IIIC are algebraically equivalent methods, they have been developed in different research communities and disparities have consequently arisen in terms of terminology. This pertains in particular to the notions of order and stability.

Beginning with RK methods, we take as our starting point the system of ODEs (3.8). The (classical) notion of order is defined as follows:

Definition 5.1. A Runge–Kutta method is of *order p* if

$$\|\boldsymbol{u}^{n+1} - \boldsymbol{u}(t_{n+1})\| = \mathcal{O}(\Delta t^p), \quad \Delta t \to 0$$

holds, whenever problem (3.8) is sufficiently smooth.

The norm can be any vector norm and $\boldsymbol{u}^{n+1} - \boldsymbol{u}(t_{n+1})$ is called the global error. The classical order of RK methods is determined by certain order conditions. To make the connection with DG-SEM as clear as possible, we present here a set of simplified conditions that are sufficient for the method to be of order p [12]:

Theorem 5.2. *Suppose that an implicit Runge–Kutta method satisfies the conditions*

$B(p_B)$**:** $\boldsymbol{b}^\top \boldsymbol{c}^{j-1} = \frac{1}{j}, \quad j = 1, \ldots, p_B,$

$C(p_C)$**:** $\boldsymbol{A}\boldsymbol{c}^{j-1} = \frac{\boldsymbol{c}^j}{j}, \quad j = 1, \ldots, p_C,$

$D(p_D)$**:** $\boldsymbol{A}^\top \operatorname{diag}(\boldsymbol{b})\boldsymbol{c}^{j-1} = \frac{1}{j} \operatorname{diag}(\boldsymbol{b})(\mathbf{1} - \boldsymbol{c}^j), \quad j = 1, \ldots, p_D,$

where $p_B \leq 2(p_C + 1)$ and $p_B \leq p_C + p_D + 1$. Then the method is of order $p = p_B$.

Remark 5.3. Here and elsewhere, $\boldsymbol{c}^j$ should be understood as an elementwise exponentiation. That is, the elements of $\boldsymbol{c}^j$ are the jth powers of the elements of $\boldsymbol{c}$.

The conditions $C(p_C)$ play a particularly important role in the context of stiff problems and have its own moniker:

Definition 5.4. A Runge–Kutta method that satisfies the order conditions $C(p_C)$ is said to have *stage order p_C*.

The stage order of the RK method describes the accuracy with which the intermediate stages are approximated. We will delve into the meanings of the conditions B, C and D shortly. However, first we summarize the various order concepts for Lobatto IIIC; see [40, Chapter IV.5].

Theorem 5.5. *The Lobatto IIIC method with N_τ stages satisfies $B(2N_\tau - 2)$, $C(N_\tau - 1)$ and $D(N_\tau - 1)$. Consequently it has stage order $N_\tau - 1$ and is of order $2N_\tau - 2$.*

We momentarily leave the RK viewpoint and focus on DG methods. DG-SEM was developed for spatial discretizations of time-dependent PDEs. Errors are measured in an L_2 norm over a spatial domain. For a discrete solution $\boldsymbol{u}_h$, this norm can be computed via the quadrature rule exactly:

$$\|\boldsymbol{u}_h\|_{L_2(\Omega)} = \sum_k \left(\boldsymbol{u}_{h_k}^\top \boldsymbol{M}_{\xi_k} \boldsymbol{u}_{h_k} \right)^{\frac{1}{2}}. \tag{5.9}$$

The sum is taken over all elements and $\boldsymbol{M}_{\xi_k}$ is the local spatial mass matrix on element i. Assuming vanishingly small errors from the time discretization, the order of convergence measured in this norm is typically $p + 1$ or $p + \frac{1}{2}$, depending on the nature of problem (2.1), the choice of numerical fluxes, and sometimes on whether p is odd or even [43, 100].

Conversely, when using DG-SEM as a time integration method, one works in the space $L_2((0,T))$. With an upwind flux in time, for sufficiently smooth and nonstiff problems, the order of convergence in this norm is N_τ [68]. This order is much smaller than the one of the Lobatto IIIC method, which requires some discussion.

There are several other order concepts in the DG literature. Here we follow [10] and relate these to the corresponding concepts in the RK framework.

- The *order of the operator* is the highest degree q for which $\boldsymbol{D}_\tau \boldsymbol{\tau}^q = q\boldsymbol{\tau}^{q-1}$. The exponentiation should be interpreted elementwise, and we take $\boldsymbol{\tau}^0 = \mathbf{1}$ as a definition. For DG-SEM we have $q = N_\tau - 1$.

  Multiplying $C(p_C)$ by $\boldsymbol{A}^{-1}$ as given in (5.7) and utilizing the fact that the first element in $\boldsymbol{c}$ is zero, we see that the RK order condition $C(p_C)$ actually describes precisely the order of the operator $\boldsymbol{D}_\tau$. A transformation of the reference element to $[0,1]$ is necessary in this step. In other words, the order of the operator is a concept identical to the stage order of the corresponding Lobatto IIIC method.

- The *order of the norm/quadrature/mass matrix* [66] is the highest degree m such that
  $$(m+1)\mathbf{1}^\top \boldsymbol{M}_\tau \boldsymbol{\tau}^m = 1 - (-1)^{m+1},$$
  i.e. for which $\boldsymbol{M}_\tau$ exactly integrates polynomials. For DG-SEM, $\mathbf{1}^\top \boldsymbol{M}_\tau$ is a row vector with the N_τ weights of the Gauss–Lobatto quadrature rule and we consequently have $m = 2(N_\tau - 1)$.

  Using (5.7) we note that the condition $B(p_B)$ simply describes the order of the quadrature, although applied to $\boldsymbol{c}$ rather than $\boldsymbol{\tau}$. Again, this amounts to a transformation from $\tau \in [-1, 1]$ to $[0, 1]$.

- Pertinently, it turns out that the order of accuracy of the final component $\boldsymbol{u}^{N_\tau} \equiv \boldsymbol{u}^{n+1}$ is $2(N_\tau - 1)$ [68], at least for smooth nonstiff problems. This *superconvergence* can be proven using the theory of dual consistent SBP methods [44]. Here it suffices to say that it is a consequence of the order of the quadrature and choosing the upwind numerical flux in time.

  The superconvergence result pertaining to DG-SEM corresponds to the classical order of Lobatto IIIC as introduced in Definition 5.1. Note that this is a consequence of considering the pointwise error in time rather than $\|\cdot\|_{L_2([0,T])}$.

To the best of our knowledge, conditions $D(p_D)$ have no clear interpretation in the language of DG. Nevertheless, using the SBP property (5.2) and the fact that $\boldsymbol{M}_\tau$ is diagonal it is shown in [10] that $C(p_C)$ is satisfied with $p_C = N_\tau - 1$, which is consistent with Theorem 5.5.

The convergence theory for RK methods relies on certain regularity properties of the problem being solved. In particular, it is assumed that the right-hand side of the system of ODEs (3.8) satisfies a one-sided Lipschitz condition,
$$\langle \boldsymbol{u} - \boldsymbol{v}, \boldsymbol{F}(\cdot, \boldsymbol{u}) - \boldsymbol{F}(\cdot, \boldsymbol{v}) \rangle \leq \beta \|\boldsymbol{u} - \boldsymbol{v}\|^2, \tag{5.10}$$
where $\langle \cdot, \cdot \rangle$ denotes some inner product and $\|\cdot\|$ the corresponding norm. If $\beta \leq 0$, the problem is *contractive*. DG-SEM in time, and hence Lobatto IIIC, is stable for contractive problems, i.e. they are B-stable methods. Convergence for contractive problems is correspondingly known as B-convergence. B-convergence can be shown for Lobatto IIIC if $\beta < 0$, but in general not if $\beta = 0$ and $N_\tau > 2$; see [83] for details.

Regularity of the type (5.10) is not standard in the literature on spatial discretization using high order DG methods. Rather, estimates of the form $\langle \boldsymbol{u}, \boldsymbol{F}(\cdot, \boldsymbol{u}) \rangle \leq 0$ are common. Such discretizations are referred to as *semi-bounded*, or in Runge–Kutta parlance, as *monotonic*. They arise particularly for

discretizations of linear, homogeneous hyperbolic or parabolic problems, but also e.g. for the velocity components of the incompressible Navier–Stokes equations [75].

If $\boldsymbol{F}(\cdot, \boldsymbol{0}) = \boldsymbol{0}$, then semi-boundedness is a special case of contractivity and results on B-stability and B-convergence apply. However, for e.g. the equations of compressible flow, semi-boundedness must typically be replaced by *entropy stability*, i.e. regularity of the form $\langle \eta'(\boldsymbol{u}), \boldsymbol{F}(\cdot, \boldsymbol{u}) \rangle \leq 0$. Here, η is some convex function of $\boldsymbol{u}$ referred to as an entropy [27, 28]. A convergence theory for implicit RK methods applied to entropy stable problems is desirable but currently appears to be missing from the literature.

Finally, for a space-time DG-SEM method, no general theoretical results on convergence order have been obtained yet as it depends on an interplay between spatial and temporal errors in ways that depend on the specifics of the problem and numerical setup. We will test the convergence rate on the compressible Euler equations in section 7.

6. **Practical Aspects of Space-Time DG-SEM**

In this section we discuss two archetypal implementations: On the one hand, the method of lines approach with Lobatto IIIC as discussed in Section 3 and on the other hand the space-time DG approach, as discussed in Section 4. We will refer to these as LoDG and STDG, respectively.

It is of course possible to produce a code that uses elements from both the LoDG and STDG formulation and thereby falls somewhere in between these approaches. However, here we adopt the point of view of a user who seeks to use an already available code base rather than producing a brand new solver.

Even though STDG and LoDG are mathematically equivalent methods, their respective implementations differ in several key aspects, each with particular requirements and accompanying merits. Here we will outline several such differences, and the choices a user will inevitably face when deciding on which implementation to select.

Several multi-dimensional DG-SEM solvers exist, such as Nektar++ [53], Fluxo, Flexi and the latest iteration Trixi [81], and others. In particular, this approach is popular for weather and climate prediction and has been used e.g. in NUMA [71] and HOMAM [72]. Thus, in the following discussion we assume that the user has access to a multi-dimensional DG solver for spatial discretization.

Our work here is based on the DUNE-FEM framework, hence the challenges outlined below are flavoured by this choice. Depending on the software framework at hand, a user may find that one approach is easier to implement than the other.

6.1. **STDG**

As described in Section 4, the defining feature of STDG is the treatment of the d-dimensional time-dependent problem as a $d+1$-dimensional stationary problem.

Requirements. The problem description in the code must be extended to a $d+1$-dimensional stationary PDE, which requires the software to be able to handle such problems. In particular, this includes the assembly of mass and stiffness matrices as well as having access to appropriate solvers for the resulting nonlinear system.

This new stationary PDE requires the use of different numerical fluxes in space and time; the temporal direction follows a causality principle, enforced by the upwind flux, which may not be the best choice for the spatial directions.

It is desirable to be able to choose different orders of accuracy in space and time, which then needs to be made possible in the DG code. In DUNE this is implemented for certain DG spaces [37] but not yet available for the Lagrange basis used in this work.

The numerical solution should be accessible at specific time points in order for the user to visualize intermediate results and the final solution. For $d = 3$, this includes extracting 3-dimensional slices from 4-dimensional data sets. Related to this issue is the problem of 4-dimensional mesh generation. An example of how to handle this for the STDG ansatz in Dune-Fem is found in Section 4.

Merits. If the requirements above are met, then existing software can be reused to solve the problem, which implies full control over the code. Moreover, only one code is needed. This code closely follows the mathematical derivation of the space-time method and may therefore be more intuitive than alternatives.

Due to the relatively simple adaption of an existing code for spatial problems of dimension $d < 3$, the STDG approach is fast for preliminary testing. An existing DG code is most likely optimized for computational resources and might even allow for parallelization in time by solving for several time steps at once.

In summary, this technique allows re-usability and full control over the code.

6.2. **LoDG**

The defining feature of LoDG is the method of lines approach outlined in Section 3. In this DG-SEM solver, each time step is solved individually.

Requirements. The (spatial) DG-SEM code needs to be coupled with an ODE solver with an implementation of a Lobatto IIIC method, most likely coming from another code. Difficulties may arise from the particular requirements of the two codes, such as interfaces for time and space adaptivity, parallelization etc.

The ODE solver might require input in a specific format not native to the DG code. Further, an efficient solution procedure may require information from the DG-SEM solver not directly available, such as the Jacobian of the spatial discretization.

An example of a Lobatto IIIC solver implemented in Assimulo [2] is found at the end of Section 3.

Merits. No adaption of the DG code with respect to the PDE or its dimension is necessary. There are no additional difficulties arising in the treatment of 3-dimensional problems, and the resulting solution can be visualized in a straightforward way.

Most existing ODE solvers are optimized and equipped with several options, for instance adaptive time stepping. Intermediate results are easy to access and the order of accuracy in space and time can be chosen independently.

In summary, this technique provides flexibility and allows reuse of existing simulation workflows.

6.3. **Algorithmic Aspects**

In the following we suppose that we have overcome the most important challenges of the two approaches presented in the previous subsections. Thus we now have access to

(a) a code that generates a $d + 1$-dimensional space-time DG-SEM discretization by following the steps in Section 4 (STDG).

(b) a code that generates a d-dimensional spatial DG-SEM discretization by following the steps in Section 3, and a code for time marching using Lobatto IIIC (LoDG).

In each time step, the LoDG code (approximately) solves (5.8b) while the STDG code solves (5.8a), or equivalently solves $\underline{\mathcal{L}}_h(\underline{\boldsymbol{u}}_h) = 0$ from (4.3). We now discuss the impact of choosing LoDG or STDG on a variety of algorithmic aspects.

Accessing time steps and stages. Accessing the numerical solution at a particular time is straight-forward in most ODE solvers. The solution is computed either by aligning the step sizes with the target times or through accurate interpolation.

While it is possible to implement such techniques with STDG, they are unlikely to be available in a pre-existing DG code. Further, the code will return the numerical solution at all points in one (or several) time elements simultaneously. In fact, it is not obvious that the STDG code will be able to return $\boldsymbol{u}^{n+1}$ in a simple way since this requires the extraction of a specific subset of coefficients from the numerical solution vector $\underline{\boldsymbol{u}}$. Yet, this may be necessary e.g. for visualization, to use adaptive time stepping, or in case the solution needs to be filtered or otherwise modified between time steps. The solution can in principle be constructed using $\boldsymbol{u}^{n+1} = (\boldsymbol{e}_{N_\tau}^\top \otimes \boldsymbol{I}_\xi)\underline{\boldsymbol{u}}$ with $\boldsymbol{e}_{N_\tau}^\top = (0, \dots, 0, 1)$, as was done in Section 5. However, this assumes that the ordering of the unknowns in $\underline{\boldsymbol{u}}$ is identical to the one used in that analysis. If not, $(\boldsymbol{e}_{N_\tau}^\top \otimes \boldsymbol{I}_\xi)$ must be suitably permuted into some matrix $\boldsymbol{E}_{N_\tau}$ before application. Finding the appropriate permutation matrix may be a nontrivial task, in particular in 4D.

On the other hand, with STDG we have access to all intermediate time stages by default, something that may be challenging with LoDG. This may be useful to compute L_2 errors of the numerical solution and has the additional advantage of allowing visualization of the solution away from the main time steps.

Adaptive time-stepping. Adaptive time-stepping for RK methods is standard in modern software and thus will be available in an implementation of LoDG. It requires a way of estimating the numerical error in the next time step. This information is used to adapt the time step to fit a predefined tolerance. Embedding techniques use a vector $\hat{\boldsymbol{b}}$ to compute a second numerical solution $\hat{\boldsymbol{u}}^{n+1}$ from (3.10) whose accuracy is one order lower than that of $\boldsymbol{u}^{n+1}$. The difference $\boldsymbol{u}^{n+1} - \hat{\boldsymbol{u}}^{n+1}$ can be used to estimate the local error without the need to solve the nonlinear system (5.8b) more than once. A detailed strategy for estimating the error and choosing the time step based on the embedding technique is available in [40, Chapter IV.8] for the Radau IIA method, but can be easily adapted to Lobatto IIIC [64].

An STDG code that follows the steps outlined in Section 4 will not have an embedded method. However, if the matrix $\boldsymbol{E}_{N_\tau}$ can be found that extracts $\boldsymbol{u}^{n+1}$, then it is also possible to construct a matrix $\hat{\boldsymbol{E}}_{N_\tau}$ that extracts $\hat{\boldsymbol{u}}^{n+1}$ such that an embedding technique can be used. This procedure may be more invasive than desirable.

Alternatively, the numerical error can be estimated using Richardson extrapolation [39, Chapter II.4]. This procedure requires solving the nonlinear system (5.8a) three times; once with a step size $2\Delta t$ and twice with a step size Δt. The difference between the two solutions yields an error estimate. Due to its expense, this approach hardly seems feasible for a 4D problem.

Adaptive Mesh Refinement (AMR). Since STDG uses a 4D mesh it is straightforward to set up a system that accounts for multiple temporal elements at once, which is not possible with LoDG. This introduces the possibility of using AMR in time in addition to space; see [17, 50] and the references therein. Like with Richardson extrapolation, using AMR forces us to solve the nonlinear system multiple times. Additionally, the system now consists of multiple coupled time steps. However, the additional cost may be offset by two factors: Firstly, we expect that the number of degrees of freedom necessary to achieve a given accuracy is significantly reduced by the AMR. Secondly, parallelism can be employed in the temporal direction.

Space-time AMR is not likely to be simple to set up with commercially available software. However, if the initial hurdles can be circumvented, then it is in principle possible to use completely unstructured space-time grids with h/p-refinement. The technique requires a generator for unstructured cuboid meshes in 4D (tesseracts) [14] and a way of estimating the numerical error in the final time. Such tools have been developed for 4D simplex meshes in [14, 99], but appear to be missing for other mesh types.

Shock capturing and limiting. The DG spatial discretizations used with RK time stepping are stable when applied to linear problems such as linear hyperbolic systems. However, for nonlinear problems spurious oscillations occur near strong shocks or steep gradients. In this case the DG method requires some extra stabilization unless a first order scheme ($p = 0$) is used that produces a monotonic structure in the shock region. For higher order schemes many approaches have been suggested to make this property available without introducing an excessive amount of numerical viscosity, which is a characteristic feature of first order schemes. Several approaches exist, including slope limiters, artificial diffusion (viscosity) techniques, and even a posteriori techniques and order reduction methods. A comprehensive literature list is presented in [84].

In Dune-Fem-DG [19], both limiter based approaches and artificial diffusion are available to stabilize a DG scheme. The slope limiter based approach implemented in Dune-Fem-DG is coupled with a troubled cell indicator which makes the overall scheme highly non-linear and therefore not suitable for implicit methods, since the selection of troubled cells could change between linear iterations and lead to divergence of the linear solver. On the other hand, artificial diffusion approaches require a discretization of a diffusion term. Stabilization diffusion coefficients only need to be re-computed every time step. A standard approach is available in Dune-Fem-DG. Thus, it is more suitable to apply artificial diffusion techniques for problems where strong shocks occur.

Nonlinear solvers and preconditioning. The solutions of the nonlinear systems (5.8a) and (5.8b) must be approximated, typically by iterative methods for large systems. There have been a multitude of suggestions on how to design such methods; early solvers for implicit Runge–Kutta methods based on modified Newton iterations were introduced in [13] and [7]. A more optimized algorithm is described in [40], and many later developments use this as a starting point. These can be considered black box solvers in the sense that they do not utilize information about the spatial terms in the solution process.

Methods designed specifically for spatial DG discretizations and implicit Runge–Kutta methods are found in e.g. [78, 79]. Likewise, nonlinear solvers designed for space-time DG and FEM discretizations have been developed [56, 86].

Unless the user is willing to make the (possibly considerable) effort to develop and/or implement a nonlinear solver specifically designed for LoDG or STDG, the natural recourse is to use a black box solver. Efficiency gains can possibly be made by introducing a preconditioner designed for DG discretizations; see e.g. [9, 54, 79] for recent developments. However, attention must be payed to the fact that the systems (5.8a) and (5.8b) have different algebraic properties and therefore likely will respond differently to preconditioners and nonlinear solvers.

For the nonlinear LoDG system (5.8b), the Jacobian is given by

$$\underline{\boldsymbol{I}} - \Delta t_n (\boldsymbol{A} \otimes \boldsymbol{I}_\xi) \mathcal{J}(\underline{\boldsymbol{F}}), \tag{6.1}$$

where $\mathcal{J}(\underline{\boldsymbol{F}})$ contains the Jacobian of the spatial discretization. The solver in [40], and many recent developments that build upon it, instead use the mathematically equivalent

$$(\Delta t_n \boldsymbol{A})^{-1} \otimes \boldsymbol{I}_\xi - \mathcal{J}(\underline{\boldsymbol{F}}). \tag{6.2}$$

For the nonlinear STDG system (5.8a), the Jacobian is given by

$$\left(\boldsymbol{D}_\tau^\top \boldsymbol{M}_\tau - \boldsymbol{e}_{N_\tau} \boldsymbol{e}_{N_\tau}^\top \right) \otimes \boldsymbol{I}_\xi + \frac{\Delta t_n}{2} (\boldsymbol{M}_\tau \otimes \boldsymbol{I}_\xi) \mathcal{J}(\underline{\boldsymbol{F}}). \tag{6.3}$$

Note from (5.7) that (6.2) arises by multiplying (6.1) by $(\boldsymbol{D}_\tau + \boldsymbol{M}_\tau^{-1} \boldsymbol{e}_1 \boldsymbol{e}_1^\top) \otimes \boldsymbol{I}_\xi$. This formulation is therefore very closely related to the STDG Jacobian (6.3). In fact, they only differ by an application of the SBP property (5.2) and a multiplication by the temporal mass matrix.

Consider the 1D linear advection equation $u_t + u_x = 0$ discretized using a single element in space and time. With STDG, the discretization is generated using Dune. With LoDG, the spatial terms are generated with Dune whereas the temporal terms are set up manually as in Section 5. Figure 6.1

shows the sparsity patterns of the Jacobians using order 1,2 and 3 in space and time. In each figure quadruplet, the Jacobian (6.3) of STDG is shown in the top left and the Jacobian (6.1) of LoDG in the top right. In the bottom right, the alternative formulation (6.2) is shown.

The first thing to note is that the straightforward LoDG formulation leads to a dense discretization whereas STDG is sparse. The LoDG formulation using $\boldsymbol{A}^{-1}$ is also sparse. It has the same number of nonzero elements as STDG, although their distribution is different. The explanation for this lies in the ordering of the unknowns. With LoDG, the node order is lexicographic in the temporal direction. However, the space-time element generated by Dune is as shown in the bottom right of Figure 6.1, here with $N_\tau = 4$. This ordering is the result of a generic construction of the reference elements, which is based on a recursion over the spatial dimension d starting at the 0-dimensional reference element, i.e. a point. This recursion also generates a natural ordering for the basis functions, starting with the basis functions located at points in an element and recursively down to the basis functions located inside the element. A detailed description of this construction is found in [23]. With a suitable permutation of the unknowns, the sparsity pattern of LoDG using $\boldsymbol{A}^{-1}$ coincides with STDG as seen in the bottom left of each figure quadruplet.

These observations suggest that the LoDG system (5.8b) is more expensive to work with than the STDG system (5.8a), and that the $\boldsymbol{A}^{-1}$ formulation (6.2) is a better choice. However, the interaction of particular preconditioners and solvers with these systems may also depend on the node ordering in ways that must be deduced through careful testing.

7. Experiments

We now perform a series of numerical tests. We start with solving the linear test equation to validate the temporal convergence rates of the two solvers. A two-dimensional advection-diffusion test case follows, with the purpose of highlighting slight differences in the numerical solutions and particular challenges with respect to visualizing the solutions. Finally, the two and three dimensional Euler equations of gas dynamics are solved to demonstrate that both codes are capable of handling nonlinear space-time dynamics in multiple dimensions.

As mentioned previously, the spatial parts of both LoDG and STDG are generated using Dune-Fem. The temporal part of LoDG is implemented in Assimulo whereas Dune-Fem is used for the entire space-time discretization in STDG.

7.1. Validation of Convergence Rates

To verify that the temporal discretizations converge as expected we perform a simple test on the linear test equation,

$$
\begin{aligned}
u_t &= -u, \quad t \in (0,1), \\
u(0) &= 4.
\end{aligned}
\tag{7.1}
$$

Python's sparse linear solver is used to solve any algebraic systems arising from the discretizations. The experimental order of convergence (EOC) of the pointwise error $|\boldsymbol{u}^{n+1} - u(1)|$ is shown for LoDG and STDG in Table 7.1. Here, N denotes the number of time steps/time elements and $N_\tau \in \{2,3,4\}$. Recall from Section 5 that the classical order of LoDG, and correspondingly the superconvergence of STDG, is $2(N_\tau - 1)$. This is indeed what we observe in Table 7.1. With $N_\tau = 3$, the errors are approaching machine precision when $N = 2^9$, hence a drop in the convergence rate is seen in Table 7.1b. The STDG appears to be more sensitive in this respect than LoDG. The same thing happens when $N_\tau = 4$ and $N = 2^5$, as seen in Table 7.1c.

We now repeat the experiment but measure the EOC via the L_2 norm $\|\cdot\|_{L_2[0,1]}$. This type of error measurement is straightforward to perform with the STDG code. However, the LoDG implementation
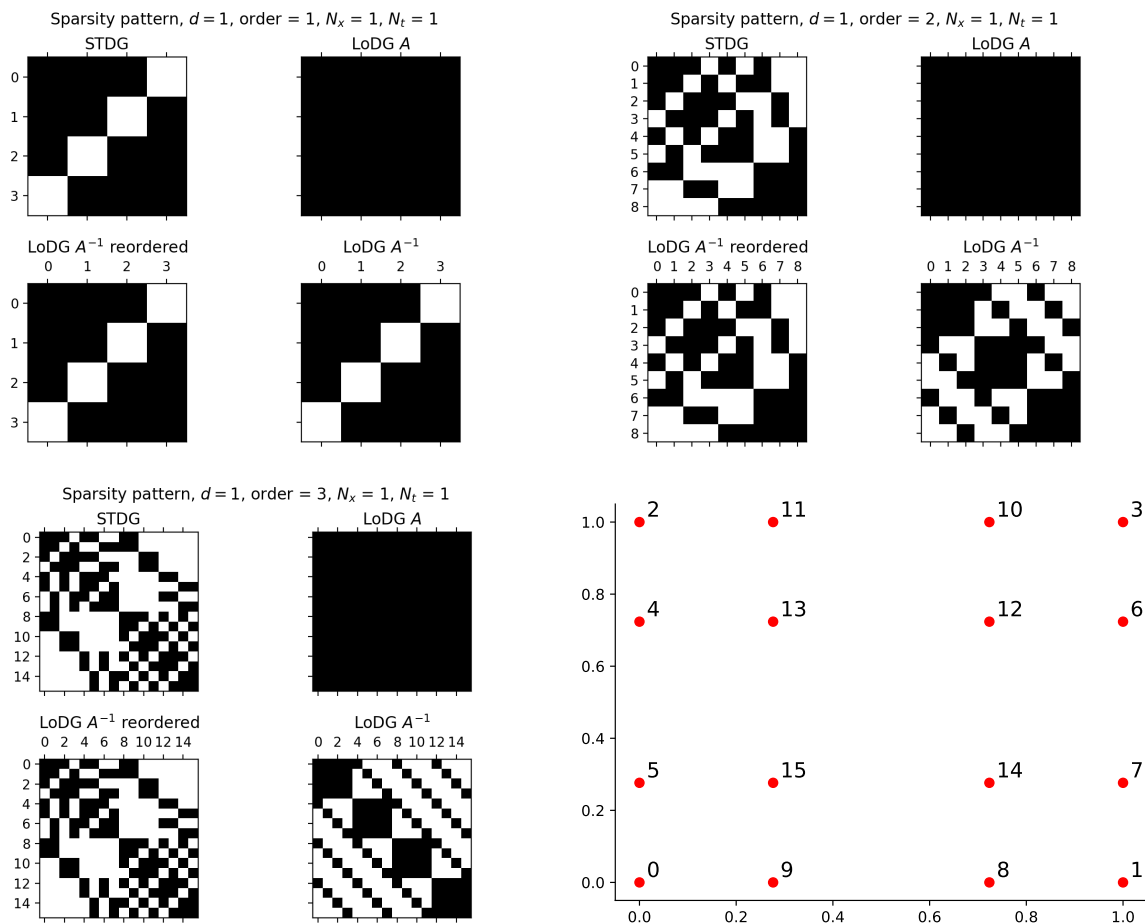
FIGURE 6.1. Sparsity patterns Jacobian of the advection problem for STDG and LoDG. Node order for one space-time element ($p = 3$) for STDG as generated by DUNE-FEM (bottom right).

TABLE 7.1. EOC of pointwise error for LoDG and STDG applied to the test equation (7.1).

(A) $N_\tau = 2$

N	Lobatto	DG-SEM
2^4	1.93	1.93
2^5	1.97	1.97
2^6	1.98	1.98
2^7	1.99	1.99
2^8	1.99	1.99
2^9	1.99	1.99

(B) $N_\tau = 3$

N	Lobatto	DG-SEM
2^4	3.96	3.96
2^5	3.98	3.98
2^6	3.99	3.99
2^7	3.99	4.01
2^8	3.99	4.28
2^9	4.05	0.47

(C) $N_\tau = 4$

N	Lobatto	DG-SEM
2^4	5.98	5.96
2^5	6.64	4.22

does not by default save the intermediate RK stages necessary to perform the computation of the L_2 error. We expect the EOC to be given by N_τ. Table 7.2 shows that this indeed is observed. For $N_\tau = 4$

with $N = 2^8$ time elements the convergence rate drops due to very small errors, as seen in Table 7.2c. Again, STDG appears to be more sensitive to this phenomenon than LoDG.

Table 7.2. EOC of L_2 error for LoDG and STDG applied to the test equation (7.1).

	(A) $N_\tau = 2$			(B) $N_\tau = 3$			(C) $N_\tau = 4$	
N	Lobatto	DG-SEM	N	Lobatto	DG-SEM	N	Lobatto	DG-SEM
2^4	1.96	1.96	2^4	2.98	2.98	2^4	3.99	3.99
2^5	1.98	1.98	2^5	2.99	2.99	2^5	4.0	4.0
2^6	1.99	1.99	2^6	2.99	2.99	2^6	4.0	4.0
2^7	2.0	2.0	2^7	3.0	3.0	2^7	4.0	4.0
2^8	2.0	2.0	2^8	3.0	3.0	2^8	4.0	3.6
2^9	2.0	2.0	2^9	3.0	3.0			

7.2. Advection-Diffusion

The next test case is the linear advection-diffusion problem in two dimensions;

$$\partial_t \boldsymbol{u} + \mathbf{b} \cdot \nabla \boldsymbol{u} - \varepsilon \Delta \boldsymbol{u} = 0 \qquad \text{in } (\Omega \times (0, T)), \ \Omega \subset \mathbb{R}^2,$$
$$\boldsymbol{u}(0) = \boldsymbol{u}_0 \qquad \text{in } \Omega. \tag{7.2}$$

We test both implementations for the rotating pulse problem with analytic solution

$$\boldsymbol{u}(t, \mathbf{x}) = \frac{0.004}{0.004 + 4\varepsilon t} \exp\left(-\frac{x_q^2 + y_q^2}{0.004 + 4\varepsilon t}\right),$$
$$x_q = x_0 \cos(4t) + y_0 \sin(4t) + 0.25,$$
$$y_q = -x_0 \sin(4t) + y_0 \cos(4t).$$

Here, $x_0 = x - 0.5$, $y_0 = y - 0.5$, $\mathbf{b} = [-4y_0, 4x_0]$, $\varepsilon = 0.001$, and $(t, \mathbf{x}) \in [0, 1] \times [0, 1]^2$. The initial condition is given by $\boldsymbol{u}(0, \mathbf{x})$ and we apply periodic boundary conditions in space. The linear systems arising from the discretizations are solved using built-in routines in Dune and Assimulo. Thus, despite the mathematical equivalence of LoDG and STDG, we do not expect the two codes to yield identical solutions.

The numerical solutions obtained by the two codes with $N_\tau \in \{2, 3, 4\}$ are shown in Figure 7.1. Here, a uniform mesh is used in space with $\Delta x = \Delta y = 0.04$. Time steps of uniform size $\Delta t = 0.1$ are used throughout the simulation. With $N_\tau = 2$ the problem is significantly under-resolved, leading to a smeared solution. As N_τ is increased, this phenomenon is reduced. To the eye, the numerical solutions using the two codes are barely distinguishable.

To get a more detailed comparison of the numerical results obtained by the two implementations we compare their spatial L_2 error in the final time point, $t = 1$. This time we vary the space-time grid with $\Delta x = \Delta y = \Delta t = 1/N$. The errors and the EOC are shown in Table 7.3. Notice that the L_2 errors are very similar, although not identical, testifying to the influence of the different solvers of the algebraic equations. Note also that the EOC is less clear than it was for the linear test equation. In this experiment we have refined space and time simultaneously, and therefore do not have a theoretical convergence result to rely on, as mentioned in section 5.2. The results indicate a convergence rate higher than N_τ, although not quite as high as $2(N_\tau - 1)$.

Finally, we highlight a feature of the STDG code that may be of use in certain situations. Since this code returns all points in a given time element (or equivalently, all intermediate RK stages in

(A) LoDG, $N_\tau = 2$ (B) LoDG, $N_\tau = 3$ (C) LoDG, $N_\tau = 4$

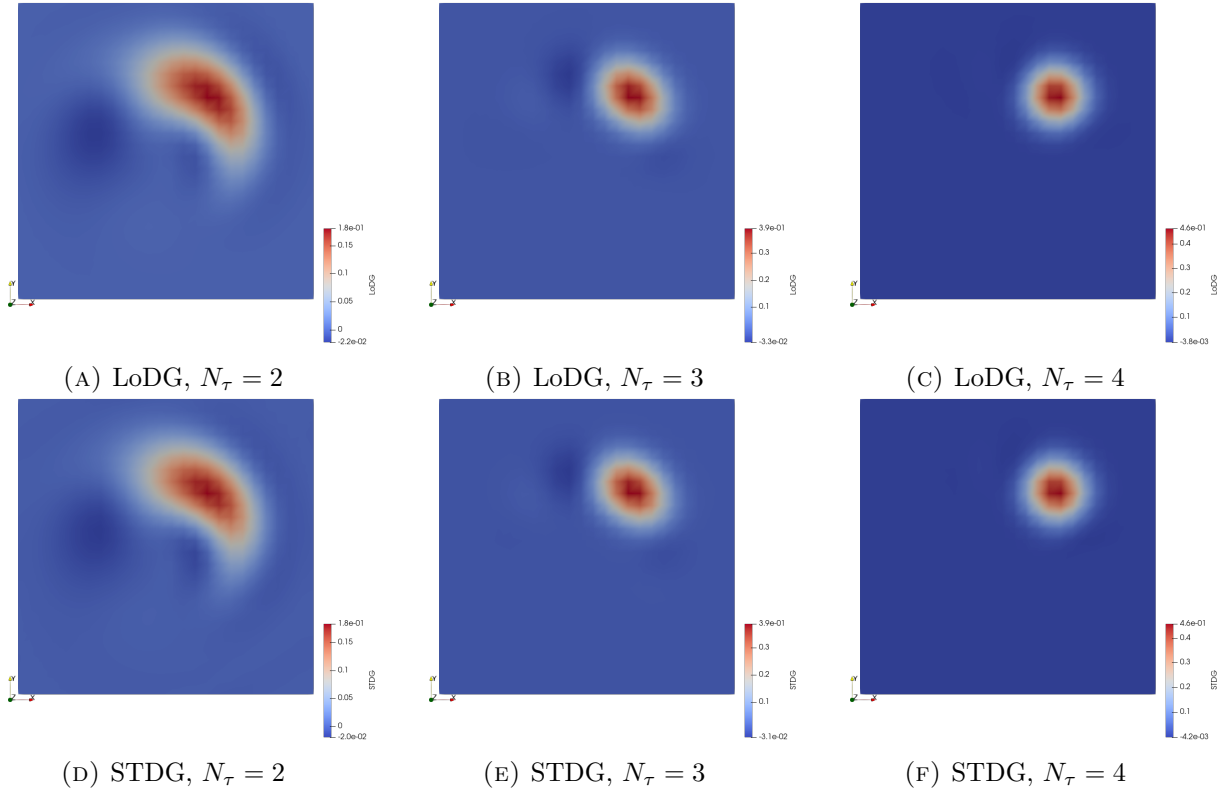(D) STDG, $N_\tau = 2$ (E) STDG, $N_\tau = 3$ (F) STDG, $N_\tau = 4$

FIGURE 7.1. Numerical solution of a rotating pulse subject to the advection-diffusion equation (7.2) using LoDG (top) and STDG (bottom).

each time step), these can be visualized using a 3D plotting software, thereby obtaining a space-time visualization of the solution. These stages are usually discarded by ODE solvers for efficiency reasons. The visualization is done for a single time step in Figure 7.2. Here, the exact solution over the whole space-time domain is also shown for reference.

7.3. Euler Equations

A prime example for evolution equations are the *Euler equations of gas dynamics*. They are derived from the conservation of mass, momentum, and energy of a compressible inviscid fluid. In Eulerian coordinates they have the form:

$$
\begin{aligned}
\partial_t \boldsymbol{u} + \nabla \cdot F_{\mathrm{c}}(\boldsymbol{u}) &= 0 &&\text{in } \Omega \times (0,T),\ \Omega \subset \mathbb{R}^d, \quad d \in \{1,2,3\}, \\
\boldsymbol{u}(0) &= \boldsymbol{u}_0 &&\text{in } \Omega,
\end{aligned}
\tag{7.3}
$$

where the vector of the conservative variables has the form

$$
\boldsymbol{u} = \begin{pmatrix} \rho \\ \rho\boldsymbol{v} \\ \varepsilon \end{pmatrix}, \quad \rho\boldsymbol{v} = (\rho v_1, \dots, \rho v_d)^T, \quad \varepsilon = \rho\mathcal{E},
\tag{7.4}
$$

augmented with suitable boundary conditions (which are discussed in detail in [8]). Here, ρ denotes the density of the fluid, $\boldsymbol{v}$ the velocity, ε the internal energy, and $\mathcal{E}$ the total energy.

TABLE 7.3. Errors and EOC for the advection-diffusion problem (7.2) when refining space and time simultaneously. The results indicate a convergence rate higher than N_τ, although not quite as high as $2(N_\tau - 1)$.

(A) Error

N	$N_\tau = 2$		$N_\tau = 3$		$N_\tau = 4$	
	LoDG	STDG	LoDG	STDG	LoDG	STDG
2^2	8.94E-2	7.28E-2	4.45E-2	4.37E-2	2.68E-2	2.69E-2
2^3	4.66E-2	4.46E-2	2.42E-2	2.41E-2	6.05E-3	6.04E-3
2^4	3.49E-2	3.39E-2	5.36E-3	5.38E-3	4.92E-4	4.93E-4
2^5	1.86E-2	1.84E-2	5.85E-4	5.94E-4	1.06E-5	9.88E-6

(B) EOC

N	$N_\tau = 2$		$N_\tau = 3$		$N_\tau = 4$	
	LoDG	STDG	LoDG	STDG	LoDG	STDG
2^3	0.9	0.7	0.9	0.9	2.1	2.2
2^4	0.4	0.4	2.2	2.2	3.6	3.6
2^5	0.9	0.9	3.2	3.2	5.5	5.6

The convective flux function $F_{\mathrm{c}}(\boldsymbol{u}) := (\boldsymbol{f}_1(\boldsymbol{u}), \dots, \boldsymbol{f}_d(\boldsymbol{u}))$ has for $i = 1, \dots, d$ the form

$$\boldsymbol{f}_i(\boldsymbol{u}) := \begin{pmatrix} \boldsymbol{u}_{i+1} \\ \boldsymbol{u}_{i+1}\boldsymbol{u}_2/\boldsymbol{u}_1 + \delta_{i,1}\, P(\boldsymbol{u}) \\ \vdots \\ \boldsymbol{u}_{i+1}\boldsymbol{u}_{d+1}/\boldsymbol{u}_1 + \delta_{i,d}\, P(\boldsymbol{u}) \\ (\boldsymbol{u}_{d+2} + P(\boldsymbol{u}))\, \boldsymbol{u}_{i+1}/\boldsymbol{u}_1 \end{pmatrix},$$

where $\delta_{i,j}$ is the Kronecker delta. For example, choosing $d = 3$ and directly using $\boldsymbol{u}$ from (7.4) we obtain the three flux functions

$$\boldsymbol{f}_1(\boldsymbol{u}) = \begin{pmatrix} \rho v_1 \\ \rho v_1^2 + P \\ \rho v_1 v_2 \\ \rho v_1 v_3 \\ (\varepsilon + P)v_1 \end{pmatrix}, \quad \boldsymbol{f}_2(\boldsymbol{u}) = \begin{pmatrix} \rho v_2 \\ \rho v_2 v_1 \\ \rho v_2^2 + P \\ \rho v_2 v_3 \\ (\varepsilon + P)v_2 \end{pmatrix}, \quad \boldsymbol{f}_3(\boldsymbol{u}) = \begin{pmatrix} \rho v_3 \\ \rho v_3 v_1 \\ \rho v_3 v_2 \\ \rho v_3^2 + P \\ (\varepsilon + P)v_3 \end{pmatrix}.$$

We first consider the two-dimensional Euler equations with periodic boundary conditions and vortex initial condition

$$\rho = \left(1 - S^2(\gamma - 1)M^2\frac{\exp(f)}{(8\pi^2)}\right)^{\frac{1}{\gamma-1}},$$

$$v_1 = 1 - Sx_2\frac{\exp\left(\frac{f}{2}\right)}{2\pi}, \qquad v_2 = Sx_1\frac{\exp\left(\frac{f}{2}\right)}{2\pi},$$

$$\varepsilon = \frac{P}{\gamma - 1} + 0.5\frac{v_1^2 + v_2^2}{\rho}, \qquad P = \frac{\rho^\gamma}{\gamma M^2},$$

with vortex strength $S = 5$, Mach number $M = 0.5$, $\gamma = 1.4$ and $f = 1 - x_1^2 - x_2^2$, where x_i denotes the ith coordinate axis.

(A) $N_\tau = 2$

(B) $N_\tau = 3$

(C) $N_\tau = 4$
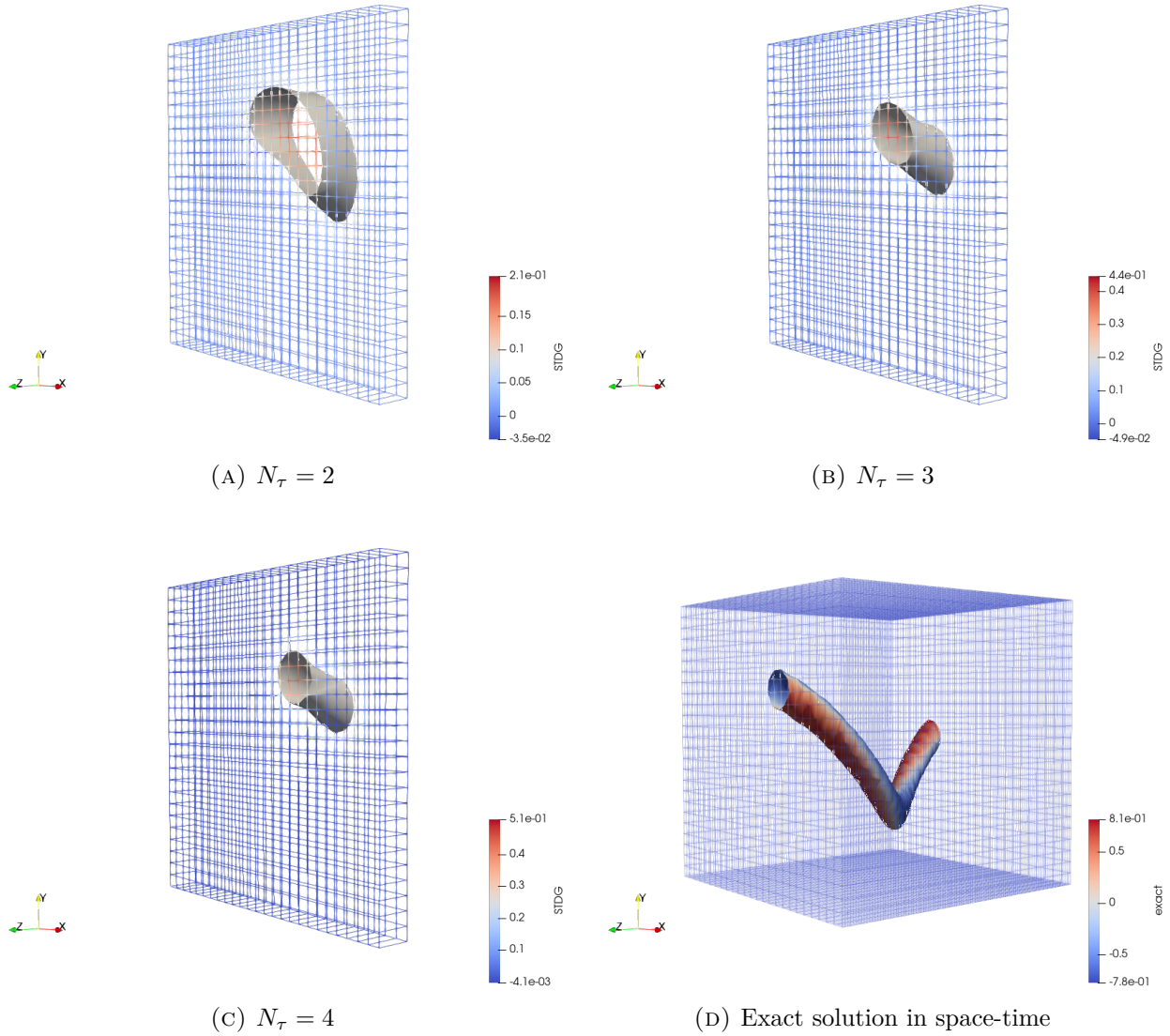
(D) Exact solution in space-time

FIGURE 7.2. 3D space-time visualization of the rotating pulse produced with the STDG code. In (A), (B) and (C), intermediate stages of a single time step are shown, while (D) shows the solution across the full time domain.

The numerical solutions obtained by the two codes with $N_\tau \in \{2, 3\}$ are shown in Figure 7.3. Here, a uniform mesh on the space-time domain $[-10, 10]^2 \times (0, 2.5]$ is used in space with $\Delta x = \Delta y = 0.04$. Time steps of uniform size $\Delta t = 0.01$ are used throughout the simulation. Again, the problem is significantly under-resolved with $N_\tau = 2$, leading to a smeared solution. As N_τ is increased, this phenomenon is reduced. Some differences can be seen in the numerical results for the two implementations. This is likely caused by the fact that two different solvers, inherent to ASSIMULO and DUNE respectively, are used for the nonlinear systems arising from the discretizations. Due to the differences between these solvers we can in general not expect identical numerical solutions despite the mathematical equivalence of the two algorithms.
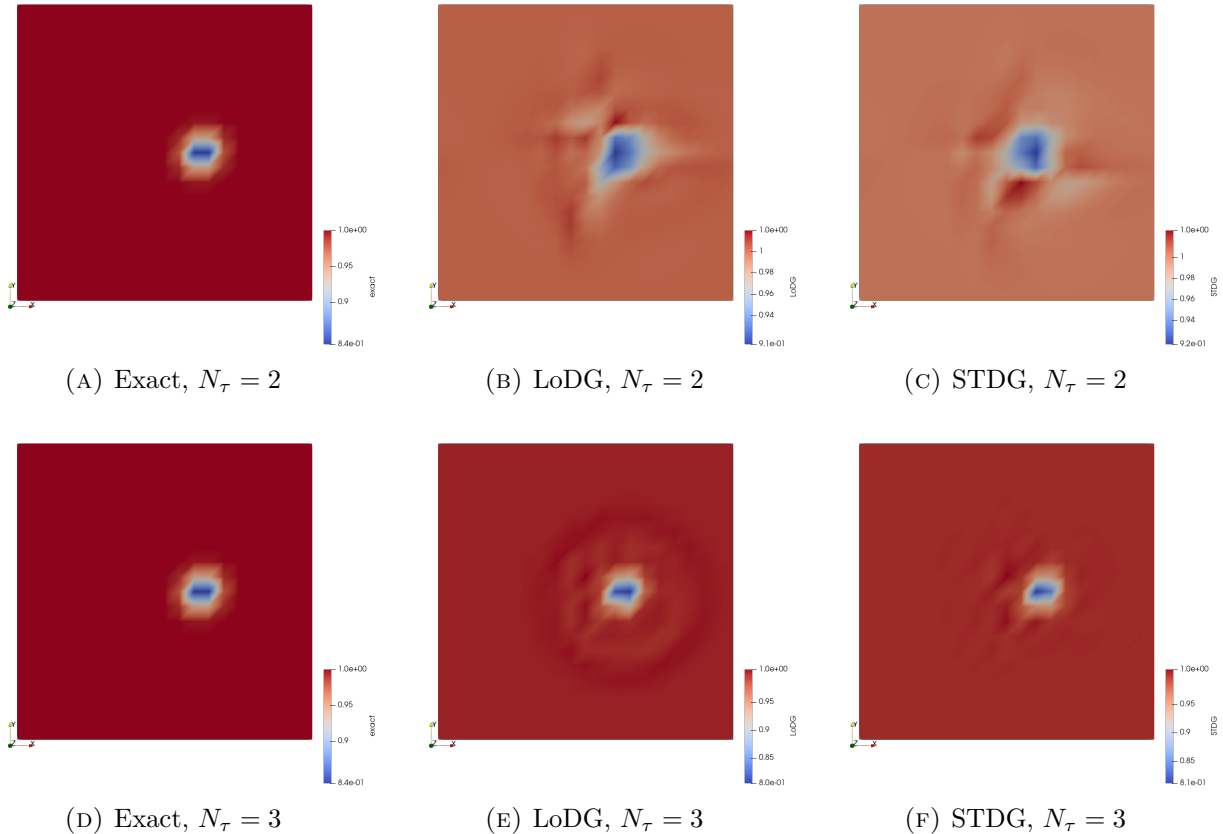
(A) Exact, $N_\tau = 2$

(B) LoDG, $N_\tau = 2$

(C) STDG, $N_\tau = 2$

(D) Exact, $N_\tau = 3$

(E) LoDG, $N_\tau = 3$

(F) STDG, $N_\tau = 3$

Figure 7.3. Numerical solution of ρ for a vortex problem subject to the 2D Euler equations (7.3) using LoDG and STDG.

To show the potential of our STDG code we present a 3D Euler test case with $N_\tau = 3$. We use periodic boundary conditions and a smooth bubble advection initial condition

$$\rho = \begin{cases} 0.5, & \hat{x} > 1.0, \\ 0.25(\cos(\hat{x}\pi) + 1)^2 + 0.5, & \hat{x} \leq 1.0, \end{cases}$$

$$v_1 = \cos\left(\frac{\pi}{5}\right), \quad v_2 = \sin\left(\frac{\pi}{5}\right), \quad v_3 = \sin\left(\frac{\pi}{5}\right)$$

$$\varepsilon = \frac{P}{\gamma - 1} + 0.5\rho(v_1^2 + v_2^2 + v_3^2),$$

with $P = 0.3$, $\gamma = 1.4$ and $\hat{x} = 16\sum_{i=1}^{d}(x_i - 0.25 - tv_i)^2$.

We consider the space-time domain $[0,1]^3 \times (0, 0.6]$ and use a uniform mesh in space with $\Delta x = \Delta y = \Delta z = 0.05$ and time steps of uniform size $\Delta t = 0.2$ and $\Delta t = 0.05$ throughout the simulation. The results have been computed on the LUNARC Aurora cluster at Lund University using 640 Intel Xeon E5-2650 v3 processors and a Newton-GRMES solver with SOR preconditioning based on the PETSc library [4, 5]. On average we observe 3 Newton iterations and about 25 linear iterations per timestep for $\Delta t = 0.05$ and about 50 linear iterations per timestep for $\Delta t = 0.2$. The initial condition and the final time element of the density can be seen in Figure 7.4. They show that some care must be taken for the numerical solution to be properly resolved in time, with a time step size $\Delta t = 0.2$ resulting in a smeared out solution with an L_2 error of 0.05 in the last time point. When decreasing the time step to $\Delta t = 0.05$ the numerical solution is more accurate with an L_2 error of 0.007. These results

show the potential of the Dune code even for 4D problems. Recently, 4D problems have been taken into consideration [32], but to the best of our knowledge this is the first 4D DG-SEM implementation available publicly.
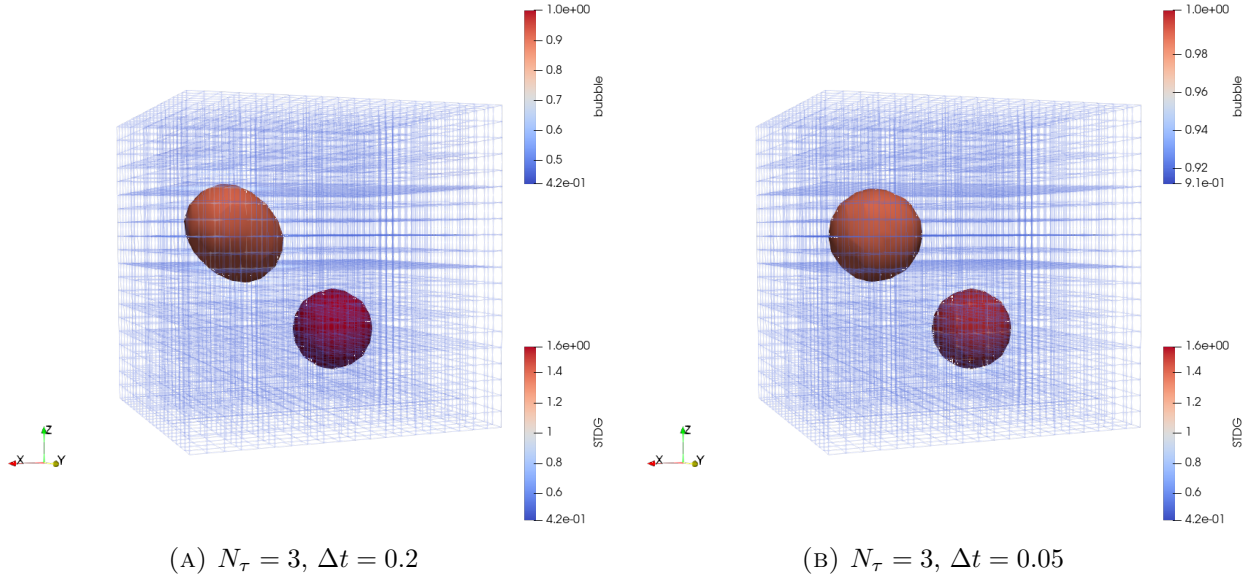

(A) $N_\tau = 3$, $\Delta t = 0.2$ (B) $N_\tau = 3$, $\Delta t = 0.05$

FIGURE 7.4. Numerical solution of ρ for a smooth bubble advection problem subject to the 3D Euler equations (7.3) using STDG. Initial condition (lower right bubble) and numerical solution (upper left bubble).

8. Conclusions

In this paper we have presented a comparison of the theoretical and practical aspects of two different space-time DG-SEM implementations. DG-SEM in time using an upwind numerical flux is equivalent to the Lobatto IIIC family of Runge–Kutta methods in the sense that they yield the same numerical solution when solved exactly. The two methods consequently have identical order, stability and convergence properties. However, since they emanate from different research communities, the terminology used to describe them are different. In this article we have made an effort to bridge this gap and to clarify the mathematical connections between the two methods.

Having two equivalent formulations of the same discretization raises the question of which path to take towards its implementation. Two strategies are immediately obvious: Either use the method of lines with DG-SEM in space and Lobatto IIIC in time (LoDG) or use a space-time DG-SEM approach (STDG). LoDG and STDG have been implemented using Dune-Fem and Assimulo and the codes are available as supplementary material to this paper; see the appendix Installation of Dune and Assimulo for details.

Despite the mathematical equivalence, there are important differences between the approaches. Not only are they described by different terminology in the literature. Additionally, the approaches lead to different systems of linear or nonlinear algebraic equations. When approximating their solutions using iterative methods, the solvers interact with these systems in different ways that are difficult to predict.

On the practical side, the two approaches lead to very different software structures that lend themselves to an assortment of opportunities and challenges. An overview of algorithmic capabilities has

been given, that will be useful in different simulation contexts. These include adaptive time-stepping, adaptive mesh refinement, shock capturing, preconditioning, and other computational techniques. Some of these are likely to be more readily available in one implementation than the other, especially when reusing pre-existing code. The choice of an appropriate implementation thus depends on the needs of the user as well as on the software already available.

Several general conclusions about the two implementations can be drawn: For an STDG-type implementation, the code must be able to handle stationary 4D problems. It is desirable that the temporal dimension can be discretized with a different order of accuracy than the spatial dimensions. Further, adaptive time-stepping is nontrivial. On the other hand, this approach opens the door to parallelization in the time domain in ways that are usually not available. For problems in fewer than three spatial dimensions, software packages are already available and likely highly optimized.

For an LoDG-type implementation, black-box time-stepping routines are readily availble that implement Lobatto IIIC. Adaptive time-stepping is thus no issue, and the solution can easily be obtained at any desired intermediate times. There is no extension to 4D necessary and the order of accuracy in time can be set independently of the spatial discretization. On the other hand, this approach requires the coupling of two different codes. Parallelization in time also becomes less flexible.

Appendix A. **Installation of Dune and Assimulo**

The simulations presented here are based on Dune version 2.8 and Assimulo version 3.2.9. There are different ways to install Dune and Assimulo. Here, we only describe the simplest and most straight forward way to install both, which is to use a Conda environment. Then the installation is done in the following way:

```
conda create -n duneproject # create a new Conda environment

conda activate duneproject # activate the Conda environment

conda install -c conda-forge assimulo # install Assimulo

pip install -U dune-fem-dg # install DUNE using pip, no Conda package yet

conda install -c conda-forge scipy # install scipy

git clone https://bitbucket.org/nate-sime/dolfin_dg.git # install dolfin_dg. No pip or Conda package yet
cd dolfin_dg
python3 setup.py install
```

The space-time DG-SEM code is available online at `https://gitlab.maths.lth.se/dune/spacetimelobattocode`.

Appendix B. **Butcher tableaus**

The Butcher tableaus for the N_τ-stage Lobatto IIIC methods with $N_\tau = 2, 3, 4$ are seen in Table B.1.

Appendix C. **UFL patch**

This patch adds 4D simplex and cuboid reference elements to UFL needed for the 3D+time simulations. This patch is implemented in Dune-Fem and will be discussed with the UFL community.

Table B.1. Butcher tableaus for Lobatto IIIC.

(A) $N_\tau = 2$

$$
\begin{array}{c|cc}
0 & \frac{1}{2} & -\frac{1}{2} \\
1 & \frac{1}{2} & \frac{1}{2} \\
\hline
 & \frac{1}{2} & \frac{1}{2}
\end{array}
$$

(B) $N_\tau = 3$

$$
\begin{array}{c|ccc}
0 & \frac{1}{6} & -\frac{1}{3} & \frac{1}{6} \\
\frac{1}{2} & \frac{1}{6} & \frac{5}{12} & -\frac{1}{12} \\
1 & \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \\
\hline
 & \frac{1}{6} & \frac{2}{3} & \frac{1}{6}
\end{array}
$$

(C) $N_\tau = 4$

$$
\begin{array}{c|cccc}
0 & \frac{1}{12} & -\frac{\sqrt{5}}{12} & \frac{\sqrt{5}}{12} & -\frac{1}{12} \\
\frac{1}{2} - \frac{\sqrt{5}}{10} & \frac{1}{12} & \frac{1}{4} & \frac{10-7\sqrt{5}}{60} & \frac{\sqrt{5}}{60} \\
\frac{1}{2} + \frac{\sqrt{5}}{10} & \frac{1}{12} & \frac{10+7\sqrt{5}}{60} & \frac{1}{4} & -\frac{\sqrt{5}}{60} \\
1 & \frac{1}{12} & \frac{5}{12} & \frac{5}{12} & \frac{1}{12} \\
\hline
 & \frac{1}{12} & \frac{5}{12} & \frac{5}{12} & \frac{1}{12}
\end{array}
$$

```python
# 4d patching of reference elements
def _patchufl4d():
    from ufl.sobolevspace import H1
    from ufl.finiteelement.elementlist import ufl_elements, any_cell, register_element
    from ufl.cell import num_cell_entities, cellname2facetname,
    from ufl.cell import _simplex_dim2cellname, _hypercube_dim2cellname

    # check if this has been added before
    if not 'pentatope' in ufl.cell.num_cell_entities:
        # 4d-simplex
        ufl.cell.num_cell_entities["pentatope"] = (5, 10, 10, 5, 1)
        # 4d-cube
        ufl.cell.num_cell_entities["tesseract"]  = (16, 32, 24, 8, 1)

        # recompute cell name to dimension mapping
        ufl.cell.cellname2dim = dict((k, len(v) - 1) for k, v \
          in ufl.cell.num_cell_entities.items())

        ufl.cell.cellname2facetname["pentatope"] = "tetrahedron"
        ufl.cell.cellname2facetname["tesseract"] = "hexahedron"

        ufl.cell._simplex_dim2cellname[4]   = "pentatope"
        ufl.cell._hypercube_dim2cellname[4] = "tesseract"

        # add types to element lists
        ufl.finiteelement.elementlist.simplices =\
            ufl.finiteelement.elementlist.simplices + ("pentatope",)
        ufl.finiteelement.elementlist.cubes = \
            ufl.finiteelement.elementlist.cubes + ("tesseract",)
        ufl.finiteelement.elementlist.any_cell =\
              ufl.finiteelement.elementlist.any_cell + ("pentatope", "tesseract", )

        # register Lagrange again with new element type list
        ufl_elements.pop("Lagrange")
        ufl_elements.pop("CG")
        register_element("Lagrange", "CG", 0, H1, "identity", (1, None), \
          ufl.finiteelement.elementlist.any_cell)
```

```python
# selecting a cell based on the dimension of the domain and or grid
def cell(dimDomainOrGrid):
    if isinstance(dimDomainOrGrid,ufl.Cell):
        return dimDomainOrGrid
    try:
        dimWorld = int(dimDomainOrGrid.dimWorld)
        dimDomain = int(dimDomainOrGrid.dimGrid)
    except:
        dimDomain = dimDomainOrGrid
```

```
    if isinstance(dimDomain, tuple):
        if len(dimDomain) != 2:
            raise Exception('dimDomain tuple must contain exactly two elements.')
        dimWorld = int(dimDomain[1])
        dimDomain = dimDomain[0]
    else:
        dimWorld = int(dimDomain)
if dimDomain == 1:
    return ufl.Cell("interval", dimWorld)
elif dimDomain == 2:
    return ufl.Cell("triangle", dimWorld)
elif dimDomain == 3:
    return ufl.Cell("tetrahedron", dimWorld)
elif dimDomain == 4:
    # add 4d cell types to ufl data structures
    _patchufl4d()
    return ufl.Cell("pentatope", dimWorld)
else:
    raise NotImplementedError('UFL cell not implemented for dimension '\
        + str(dimDomain) + '.')
```

Appendix D. **DG-SEM operators**

For $N_\tau = 2$,

$$\boldsymbol{B}_\tau = \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix}, \; \boldsymbol{M}_\tau = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \; \boldsymbol{D}_\tau = \begin{pmatrix} -\frac{1}{2} & \frac{1}{2} \\ -\frac{1}{2} & \frac{1}{2} \end{pmatrix}. \tag{D.1}$$

For $N_\tau = 3$,

$$\boldsymbol{B}_\tau = \begin{pmatrix} -1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \; \boldsymbol{M}_\tau = \begin{pmatrix} \frac{1}{3} & 0 & 0 \\ 0 & \frac{4}{3} & 0 \\ 0 & 0 & \frac{1}{3} \end{pmatrix}, \; \boldsymbol{D}_\tau = \begin{pmatrix} -\frac{3}{2} & 2 & -\frac{1}{2} \\ -\frac{1}{2} & 0 & \frac{1}{2} \\ \frac{1}{2} & -2 & \frac{3}{2} \end{pmatrix}. \tag{D.2}$$

For $N_\tau = 4$,

$$\boldsymbol{B}_\tau = \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \; \boldsymbol{M}_\tau = \begin{pmatrix} \frac{1}{6} & 0 & 0 & 0 \\ 0 & \frac{5}{6} & 0 & 0 \\ 0 & 0 & \frac{5}{6} & 0 \\ 0 & 0 & 0 & \frac{1}{6} \end{pmatrix}, \; \boldsymbol{D}_\tau = \begin{pmatrix} -3 & \frac{5+5\sqrt{5}}{4} & \frac{5-5\sqrt{5}}{4} & \frac{1}{2} \\ \frac{-1-\sqrt{5}}{4} & 0 & \frac{-\sqrt{5}}{2} & \frac{1-\sqrt{5}}{4} \\ \frac{-1+\sqrt{5}}{4} & \frac{-\sqrt{5}}{2} & 0 & \frac{1+\sqrt{5}}{4} \\ -\frac{1}{2} & \frac{5\sqrt{5}-5}{4} & \frac{-5-5\sqrt{5}}{4} & 3 \end{pmatrix}. \tag{D.3}$$

Author contributions

References

[1] M. S. Alnæs, A. Logg, K. B. Ølgaard, M. E. Rognes, and G. N. Wells. Unified Form Language: A Domain-Specific Language for Weak Formulations of Partial Differential Equations. *ACM Trans. Math. Softw.*, 40(2), 2014.

[2] C. Andersson, C. Führer, and J. Åkesson. Assimulo: A unified framework for ODE solvers. *Math. Comput. Simul.*, 116(0):26–43, 2015.

[3] D. N. Arnold, F. Brezzi, B. Cockburn, and L. D. Marini. Unified analysis of discontinuous Galerkin methods for elliptic problems. *SIAM J. Numer. Anal.*, 39(5):1749–1779, 2002.

[4] S. Balay, S. Abhyankar, M. F. Adams, S. Benson, J. Brown, P. Brune, K. Buschelman, E. M. Constantinescu, L. Dalcin, A. Dener, V. Eijkhout, W. D. Gropp, V. Hapla, T. Isaac, P. Jolivet, D. Karpeev, D. Kaushik, M. G. Knepley, F. Kong, S. Kruger, D. A. May, L. C. McInnes, R. T. Mills, L. Mitchell, T. Munson, J. E. Roman, K. Rupp, P. Sanan, J. Sarich, B. F. Smith, S. Zampini, H. Zhang, H. Zhang, and J. Zhang. PETSc Web page. `https://petsc.org/`, 2021.

[5] S. Balay, S. Abhyankar, M. F. Adams, S. Benson, J. Brown, P. Brune, K. Buschelman, E. M. Constantinescu, L. Dalcin, A. Dener, V. Eijkhout, W. D. Gropp, V. Hapla, T. Isaac, P. Jolivet, D. Karpeev, D. Kaushik, M. G. Knepley, F. Kong, S. Kruger, D. A. May, L. C. McInnes, R. T. Mills, L. Mitchell, T. Munson, J. E. Roman, K. Rupp, P. Sanan, J. Sarich, B. F. Smith, S. Zampini, H. Zhang, H. Zhang, and J. Zhang. PETSc/TAO Users Manual. Technical Report ANL-21/39 - Revision 3.16, Argonne National Laboratory, 2021.

[6] P. Bastian, M. Blatt, M. Dedner, N.-A. Dreier, R. Engwer, Ch. Fritze, C. Gräser, Ch. Grüninger, D. Kempf, R. Klöfkorn, M. Ohlberger, and O. Sander. The Dune framework: Basic concepts and recent developments. *Comput. Math. Appl.*, 81:75–112, 2021.

[7] T. A. Bickart. An efficient solution process for implicit Runge–Kutta methods. *SIAM J. Numer. Anal.*, 14(6):1022–1027, 1977.

[8] P. Birken. *Numerical Methods for Unsteady Compressible Flow Problems*. CRC Press, 2021.

[9] P. Birken, G. J. Gassner, and L. M. Versbach. Subcell finite volume multigrid preconditioning for high-order discontinuous Galerkin methods. *Int. J. Comput. Fluid Dyn.*, 33(9):353–361, 2019.

[10] P. D. Boom and D. W. Zingg. High-Order Implicit Time-Marching Methods Based on Generalized Summation-by-Parts Operators. *SIAM J. Sci. Comput.*, 37:A2682–A2709, 2015.

[11] S. Brdar, A. Dedner, and R. Klöfkorn. Compact and stable Discontinuous Galerkin methods for convection-diffusion problems. *SIAM J. Sci. Comput.*, 34(1):263–282, 2012.

[12] J. C. Butcher. Implicit Runge-Kutta Processes. *Math. Comput.*, 18(85):50–64, 1964.

[13] J. C. Butcher. On the implementation of implicit Runge-Kutta methods. *BIT*, 16(3):237–240, 1976.

[14] P. C. Caplan, R. Haimes, D. L. Darmofal, and M. C. Galbraith. Four-dimensional anisotropic mesh adaptation. *Comput.-Aided Des.*, 129:102915, 2020.

[15] 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 J. Sci. Comput.*, 36(5):B835–B867, 2014.

[16] M. H. Carpenter and D. Gottlieb. Spectral methods on arbitrary grids. *J. Comput. Phys.*, 129(1):74–86, 1996.

[17] Z. Chen, H. Steeb, and S. Diebels. A space-time discontinuous Galerkin method applied to single-phase flow in porous media. *Comput. Geosci.*, 12(4):525–539, 2008.

[18] A. Dedner, S. Girke, R. Klöfkorn, and T. Malkmus. The DUNE-FEM-DG module. *Archive of Numerical Software*, 5(1), 2017.

[19] A. Dedner and R. Klöfkorn. Extendible and Efficient Python Framework for Solving Evolution Equations with Stabilized Discontinuous Galerkin Method. *Commun. Appl. Math. Comput. Sci.*, 2021.

[20] A. Dedner and R. Klöfkorn. The DUNE-FEM tutorial, 2022. `https://dune-project.org/sphinx/content/sphinx/dune-fem/`.

[21] A. Dedner, R. Klöfkorn, and M. Nolte. Python Bindings for the DUNE-FEM Module. Zenodo (Mar 2020), `https://doi.org/10.5281/zenodo.3706994`, 2020.

[22] A. Dedner, R. Klöfkorn, M. Nolte, and M. Ohlberger. A Generic Interface for Parallel and Adaptive Scientific Computing: Abstraction Principles and the DUNE-FEM Module. *Computing*, 90(3–4):165–196, 2010.

[23] A. Dedner and M. Nolte. Construction of Local Finite Element Spaces Using the Generic Reference Elements. In A. Dedner, B. Flemisch, and R. Klöfkorn, editors, *Advances in DUNE*, pages 3–16. Springer, 2012.

[24] D. C. Del Rey Fernández, J. E. Hicken, and D. W. Zingg. Review of summation-by-parts operators with simultaneous approximation terms for the numerical solution of partial differential equations. *Comput. Fluids*, 95:171–196, 2014.

[25] L. T. Diosady and S. M. Murman. Tensor-product preconditioners for higher-order space–time discontinuous Galerkin methods. *J. Comput. Phys.*, 330:296–318, 2017.

[26] W. Dörfler, S. Findeisen, and C. Wieners. Space-time discontinuous Galerkin discretizations for linear first-order hyperbolic evolution systems. *Comput. Methods Appl. Math.*, 16(3):409–428, 2016.

[27] T. C. Fisher and M. H. Carpenter. High-order entropy stable finite difference schemes for nonlinear conservation laws: Finite domains. *J. Comput. Phys.*, 252:518–557, 2013.

[28] T. C. Fisher, M. H. Carpenter, J. Nordström, N. K. Yamaleev, and C. Swanson. Discretely conservative finite-difference formulations for nonlinear conservation laws in split form: Theory and boundary conditions. *J. Comput. Phys.*, 234:353–375, 2013.

[29] M. Franciolini and S. M. Murman. Multigrid preconditioning for a space-time spectral-element discontinuous-Galerkin solver. In *AIAA Scitech 2020 Forum*, page 1314, 2020.

[30] S. Friedhoff, S. MacLachlan, and C. Borgers. Local Fourier analysis of space-time relaxation and multigrid schemes. *SIAM J. Sci. Comput.*, 35(5):S250–S276, 2013.

[31] L. Friedrich, G. Schnücke, A. R. Winters, D. C. Del Rey Fernández, G. J. Gassner, and M. H. Carpenter. Entropy Stable Space–Time Discontinuous Galerkin Schemes with Summation-by-Parts Property for Hyperbolic Conservation Laws. *J. Sci. Comput.*, 80(1):175–222, 2019.

[32] C. V. Frontin, G. S. Walters, F. D. Witherden, C. W. Lee, D. M. Williams, and D. L. Darmofal. Foundations of space-time finite element methods: Polytopes, interpolation, and integration. *Appl. Numer. Math.*, 166:92–113, 2021.

[33] M. J. Gander. 50 Years of Time Parallel Time Integration. In T. Carraro, M. Geiger, S. Körkel, and R. Rannacher, editors, *Multiple Shooting and Time Domain Decomposition Methods*, pages 69–113. Springer, 2015.

[34] M. J. Gander and M. Neumüller. Analysis of a new space-time parallel multigrid algorithm for parabolic problems. *SIAM J. Sci. Comput.*, 38(4):A2173–A2208, 2016.

[35] G. J. Gassner. A skew-symmetric discontinuous Galerkin spectral element discretization and its relation to SBP-SAT finite difference methods. *SIAM J. Sci. Comput.*, 35(3):A1233–A1253, 2013.

[36] G. J. Gassner and A. R. Winters. A novel robust strategy for discontinuous Galerkin methods in computational fluid mechanics: Why? When? What? Where? *Front. Phys.*, page 612, 2021.

[37] Ch. Gersbacher. *Higher-order discontinuous finite element methods and dynamic model adaptation for hyperbolic systems of conservation laws.* Phd thesis, University of Freiburg, 2017.

[38] J. Gopalakrishnan and G. Kanschat. A multilevel discontinuous Galerkin method. *Numer. Math.*, 95(3):527–550, 2003.

[39] E. Hairer, S. P. Nørsett, and G. Wanner. *Solving Ordinary Differential Equations I*. Springer, 2009.

[40] E. Hairer and G. Wanner. *Solving Ordinary Differential Equations II*, volume 14 of *Springer Series in Computational Mathematics*. Springer, 2010.

[41] P. W. Hemker, W. Hoffmann, and M. H. Van Raalte. Two-level Fourier Analysis of a Multigrid Approach for Discontinuous Galerkin Discretization. *SIAM J. Sci. Comput.*, 25(3):1018–1041, 2003.

[42] P. W. Hemker, W. Hoffmann, and M. H. Van Raalte. Fourier two-level analysis for discontinuous Galerkin discretization with linear elements. *Numer. Linear Algebra Appl.*, 11(5-6):473–491, 2004.

[43] J. S. Hesthaven and T. Warburton. *Nodal discontinuous Galerkin methods: algorithms, analysis, and applications.* Springer, 2008.

[44] J. E. Hicken and D. W. Zingg. Superconvergent functional estimates from summation-by-parts finite-difference discretizations. *SIAM J. Sci. Comput.*, 33(2):893–922, 2011.

[45] A. C. Hindmarsh, P. N. Brown, K. E. Grant, S. L. Lee, R. Serban, D. E. Shumaker, and C. S. Woodward. Sundials: Suite of nonlinear and differential/algebraic equation solvers. *ACM Trans. Math. Softw.*, 31(3):363–396, 2005.

[46] P. Houston and N. Sime. Automatic Symbolic Computation for Discontinuous Galerkin Finite Element Methods. *SIAM J. Sci. Comput.*, 40(3):C327–C357, 2018.

[47] G. M. Hulbert and T. Hughes, Jr. Space-time finite element methods for second-order hyperbolic equations. *Comput. Methods Appl. Math.*, 84(3):327–348, 1990.

[48] A. Jameson. Evaluation of fully implicit Runge Kutta schemes for unsteady flow calculations. *J. Sci. Comput.*, 73(2-3):819–852, 2017.

[49] L. O. Jay. Lobatto Methods. In B. Engquist, editor, *Encyclopedia of Applied and Computational Mathematics*, pages 817–826. Springer, 2015.

[50] S. Jayasinghe, D. L. Darmofal, N. K. Burgess, M. C. Galbraith, and S. R. Allmaras. A space-time adaptive method for reservoir flows: formulation and one-dimensional application. *Comput. Geosci.*, 22(1):107–123, 2018.

[51] X. Jiao, X. Wang, and O. Chen. Optimal and Low-Memory Near-Optimal Preconditioning of Fully Implicit Runge–Kutta Schemes for Parabolic PDEs. *SIAM J. Sci. Comput.*, 43(5):A3527–A3551, 2021.

[52] O. Karakashian and C. Makridakis. A space-time finite element method for the nonlinear Schrödinger equation: the discontinuous Galerkin method. *Math. Comput.*, 67(222):479–499, 1998.

[53] G. Karniadakis and S. Sherwin. *Spectral/hp Element Methods for Computational Fluid Dynamics.* Oxford University Press, 2013.

[54] J. Kasimir, L. M. Versbach, P. Birken, G. J. Gassner, and R. Klöfkorn. An Finite Volume Based Multigrid Preconditioner for DG-SEM for Convection-Diffusion. In *14th WCCM-ECCOMAS Congress 2020*, volume 600, 2021.

[55] C. M. Klaij, J. J. W. van der Vegt, and H. van der Ven. Space–time discontinuous Galerkin method for the compressible Navier–Stokes equations. *J. Comput. Phys.*, 217(2):589–611, 2006.

[56] C. M. Klaij, M. H. van Raalte, H. van der Ven, and J. J. W. van der Vegt. h-Multigrid for space-time discontinuous Galerkin discretizations of the compressible Navier-Stokes equations. *J. Comput. Phys.*, 227(2):1024–1045, 2007.

[57] U. Köcher and M. Bause. Variational space–time methods for the wave equation. *J. Sci. Comput.*, 61(2):424–453, 2014.

[58] D. A. Kopriva. *Implementing Spectral Methods for Partial Differential Equations.* Springer, 2009.

[59] D. A. Kopriva and G. J. Gassner. On the Quadrature and Weak Form Choices in Collocation Type Discontinuous Galerkin Spectral Element Methods. *J. Sci. Comput.*, 44:136–155, 2010.

[60] D. A. Kopriva, S. L. Woodruff, and M. Y. Hussaini. Computation of electromagnetic scattering with a non-conforming discontinuous spectral element method. *Int. J. Numer. Methods Eng.*, 53(1):105–122, 2002.

[61] N. Krais, A. Beck, T. Bolemann, H. Frank, D. Flad, G. J. Gassner, F. Hindenlang, M. Hoffmann, T. Kuhn, M. Sonntag, et al. Flexi: A high order discontinuous Galerkin framework for hyperbolic–parabolic conservation laws. *Comput. Math. Appl.*, 81:186–219, 2021.

[62] H.-O. Kreiss and G. Scherer. Finite Element and Finite Difference Methods for Hyperbolic Partial Differential Equations. In C. de Boor, editor, *Mathematical Aspects of Finite Elements in Partial Differential Equations*, pages 195–212. Academic Press Inc., 1974.

[63] J. Lai, F. Liu, V. V. Anh, and Q. Liu. A space-time finite element method for solving linear Riesz space fractional partial differential equations. *Numer. Algorithms*, pages 1–22, 2021.

[64] E. Lehsten. *Implementation of 3 stage Lobatto IIIC into Assimulo package.* Bachelor thesis, Lund University, 2021.

[65] V. Linders. On an eigenvalue property of Summation-By-Parts operators. *J. Sci. Comput.*, 93:82, 2022.

[66] V. Linders, T. Lundquist, and J. Nordström. On the order of accuracy of finite difference operators on diagonal norm based summation-by-parts form. *SIAM J. Numer. Anal.*, 56(2):1048–1063, 2018.

[67] V. Linders, J. Nordström, and S. H. Frankel. Properties of Runge-Kutta-Summation-By-Parts methods. *J. Comput. Phys.*, 419:109684, 2020.

[68] T. Lundquist and J. Nordström. The SBP–SAT technique for initial value problems. *J. Comput. Phys.*, 270:86–104, 2014.

[69] C. Makridakis and R. H. Nochetto. A posteriori error analysis for higher order dissipative methods for evolution problems. *Numer. Math.*, 104(4):489–514, 2006.

[70] M. Masud Rana, V. E. Howle, K. Long, A. Meek, and W. Milestone. A New Block Preconditioner for Implicit Runge–Kutta Methods for Parabolic PDE Problems. *SIAM J. Sci. Comput.*, 43(5):S475–S495, 2021.

[71] A. Müller, M. A. Kopera, S. Marras, L. C. Wilcox, T. Isaac, and F. X. Giraldo. Strong scaling for numerical weather prediction at petascale with the atmospheric model NUMA. *Int. J. High Perform. Comput. Appl.*, 33(2):411–426, 2019.

[72] R. D. Nair, L. Bao, M. D. Toy, and R. Klöfkorn. A High-Order Multiscale Global Atmospheric Model. *AIAA AVIATION Forum*, 2016.

[73] M. Neumüller. *Space-Time Methods*, volume 20 of *Monograph Series TU Graz: Computation in Engineering and Science.* TU Graz, 2013.

[74] J. Nievergelt. Parallel methods for integrating ordinary differential equations. *Commun. ACM*, 7(12):731–733, 1964.

[75] J. Nordström and C. La Cognata. Energy stable boundary conditions for the nonlinear incompressible Navier–Stokes equations. *Math. Comput.*, 88(316):665–690, 2019.

[76] J. Nordström and V. Linders. Well-posed and stable transmission problems. *J. Comput. Phys.*, 364:95–110, 2018.

[77] J. Nordström and T. Lundquist. Summation-by-parts in time. *J. Comput. Phys.*, 251:487–499, 2013.

[78] W. Pazner and P.-O. Persson. Stage-parallel fully implicit Runge–Kutta solvers for discontinuous Galerkin fluid simulations. *J. Comput. Phys.*, 335:700–717, 2017.

[79] W. Pazner and P.-O. Persson. Approximate tensor-product preconditioners for very high order discontinuous Galerkin methods. *J. Comput. Phys.*, 354:344–369, 2018.

[80] H. Ranocha. Some notes on summation by parts time integration methods. *Results Appl. Math.*, 1:100004, 2019.

[81] H. Ranocha, M. Schlottke-Lakemper, A. R. Winters, E. Faulhaber, J. Chan, and G. J. Gassner. Adaptive numerical simulations with Trixi.jl: A case study of Julia for scientific computing. *Proceedings of the JuliaCon Conferences*, 1(1):77, 2022.

[82] A. A. Ruggiu and J. Nordström. On pseudo-spectral time discretizations in summation-by-parts form. *J. Comput. Phys.*, 360:192–201, 2018.

[83] J. Schneid. B-convergence of Lobatto IIIC formulas. *Numer. Math.*, 51(2):229–235, 1987.

[84] C.-W. Shu. High order WENO and DG methods for time-dependent convection-dominated PDEs: A brief survey of several recent developments. *J. Comput. Phys.*, 316:598–613, 2016.

[85] B. Strand. Summation by parts for finite difference approximations for d/dx. *J. Comput. Phys.*, 110(1):47–67, 1994.

[86] J. J. Sudirham, J. J. W. van der Vegt, and R. M. J. van Damme. Space-time discontinuous Galerkin method for advection-diffusion problems on time-dependent domains. *Appl. Numer. Math.*, 56(12):1491–1518, 2006.

[87] M. Svärd and J. Nordström. Review of summation-by-parts schemes for initial–boundary-value problems. *J. Comput. Phys.*, 268:17–38, 2014.

[88] T. E. Tezduyar and K. Takizawa. Space–time computations in practical engineering applications: A summary of the 25-year history. *Comput. Mech.*, 63(4):747–753, 2019.

[89] J. J. W. van der Vegt. *Space-time discontinuous Galerkin finite element methods*, pages 1–37. Von Karman Institute for Fluid Dynamics, 2006.

[90] J. J. W. van der Vegt and S. Rhebergen. hp-multigrid as smoother algorithm for higher order discontinuous Galerkin discretizations of advection dominated flows: Part I. Multilevel analysis. *J. Comput. Phys.*, 231(22):7537–7563, 2012.

[91] J. J. W. van der Vegt and S. Rhebergen. hp-Multigrid as Smoother algorithm for higher order discontinuous Galerkin discretizations of advection dominated flows. Part II: Optimization of the Runge–Kutta smoother. *J. Comput. Phys.*, 231:7564–7583, 2012.

[92] J. J. W. van der Vegt and H. van der Ven. Space-time discontinuous Galerkin finite element method with dynamic grid motion for inviscid compressible flows. I. General formulation. *J. Comput. Phys.*, 182(2):546–585, 2002.

[93] J. J. W. van der Vegt and Y. Xu. Space–time discontinuous Galerkin method for nonlinear water waves. *J. Comput. Phys.*, 224(1):17–39, 2007.

[94] H. Van der Ven and J. J. W. van der Vegt. Space–time discontinuous Galerkin finite element method with dynamic grid motion for inviscid compressible flows: II. Efficient flux quadrature. *Comput. Methods Appl. Math.*, 191(41-42):4747–4780, 2002.

[95] M. H. Van Raalte and P. W. Hemker. Two-level multigrid analysis for the convection–diffusion equation discretized by a discontinuous Galerkin method. *Numer. Linear Algebra Appl.*, 12(5-6):563–584, 2005.

[96] L. M. Versbach. *Efficient Solvers for Space-Time Discontinuous Galerkin Spectral Element Methods*. PhD thesis, Mathematics (Faculty of Sciences), Lund University, 2022.

[97] L. M. Versbach, P. Birken, V. Linders, and G. J. Gassner. Local Fourier Analysis of a Space-Time Multigrid Method for DG-SEM for the Linear Advection Equation. https://arxiv.org/abs/2112.03115, 2021.

[98] Y. Xia, X. Liu, H. Luo, and R. Nourgaliev. A third-order implicit discontinuous Galerkin method based on a Hermite WENO reconstruction for time-accurate solution of the compressible Navier–Stokes equations. *Int. J. Numer. Methods Fluids*, 79(8):416–435, 2015.

[99] M. Yano and D. L. Darmofal. An optimization-based framework for anisotropic simplex mesh adaptation. *J. Comput. Phys.*, 231(22):7626–7649, 2012.

[100] Q. Zhang and C.-W. Shu. Error estimates to smooth solutions of Runge–Kutta discontinuous Galerkin method for symmetrizable systems of conservation laws. *SIAM J. Numer. Anal.*, 44(4):1703–1720, 2006.