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

Lea M. Versbach\textsuperscript{1*}, Viktor Linders\textsuperscript{1}, Robert Klöfkorn\textsuperscript{1}, Philipp Birken\textsuperscript{1}

\textsuperscript{1}Centre for Mathematical Sciences, Numerical Analysis, Lund University, Lund, Sweden

*lea.miko.versbach@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.

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, resulting in a fully discrete numerical scheme \cite{59}. This approach has several advantages: Moving boundaries can be treated more easily \cite{74} and parallelization in time is made possible \cite{31}. 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 \cite{73}. 
Here, we consider the discontinuous Galerkin spectral element method (DG-SEM); see e.g. [8] for an overview and [48] for a detailed exposition. These methods have been very successful for spatial discretizations as they are unstructured, of high order and are very suitable for high performance computing [51]. Further, DG-SEM fits the so-called Summation-By-Parts Simultaneous-Approximation-Term (SBP-SAT) framework [16, 33], implying $L_2$ stability for linear problems. Further, in the last decade, work within this framework has resulted in the development of entropy stable (i.e. nonlinearly stable) discretizations of arbitrarily high order [27, 26].

Our motivation to consider DG-SEM in a space-time formulation is twofold: Firstly, with a specific choice of numerical fluxes, entropy stability can be extended to incorporate the temporal discretization for hyperbolic conservation laws [29], thereby resulting in a nonlinearly stable, fully discrete scheme. 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 [60].

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, 65]. 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 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 Appendix A contains instructions on how to install the code discussed in this paper.

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 [48] provides much background material on spatial DG-SEM as well as a guide to its implementation. An overview of entropy stable DG-SEM is given in [34] and full mathematical detail is provided in [15]. The theory builds upon the SBP-SAT framework, reviews of which are found in [25, 72].

For a broad background on implicit Runge-Kutta methods, see [37]. 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 [55]. An evaluation of fully implicit RK methods for use in computational fluid dynamics is given in [42], including discussions of how to solve the nonlinear algebraic systems.

Several nonlinear solvers with application to RK and DG methods have recently been presented in the literature. For Newton-type methods, see e.g. [64, 24] and the references therein. Solvers utilizing multigrid techniques are discussed and analyzed in [32, 75, 28].

The implementation of space-time methods with a focus on the challenge of 4D problems has been studied in [30], and the generation of different 4D space-time meshes have been presented in [6, 14].

2 Governing Equations and Simulation Software

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

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

for a vector valued function \( u: \Omega \times (0, T) \to \mathbb{R}^r \) 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) (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 (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 (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 [5], 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 [22, 19] and the tutorial [20].

The implementation of the Lobatto IIIC method (see [53]) has been done in ASSIMULO [2], which is also a Python package that can be readily used together with DUNE-FEM. ASSIMULO provides a high-level interface for a wide variety of classical and modern solvers of ordinary differential equations. e.g. SUNDIALS [40] and implicit Runge-Kutta solvers [36, 37]. The original codes, which are written in FORTRAN, C or Python, are wrapped into ASSIMULO keeping their original form, while the user only needs to interact with the Python interface, where the ODE and the initial condition of the problem need to be provided, as well as other additional information as e.g. the Jacobian, depending on the solver used. Existing solver options in ASSIMULO are for instance explicit and implicit Euler, Runge-Kutta34, RADAU5ODE, CVODE, IDA, ODASSL, LSODAR, GLIMDA.

Comments on how to install DUNE-FEM-DG and ASSIMULO are found in Appendix A.

3 Method of Lines DG-SEM

In this section we describe the method of lines (MOL) approach to discretizing (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 \( \Omega \) into elements \( E \) with \( \bigcup_{E \in \mathcal{T}_h} E = \Omega \), consider the piecewise polynomial space

\[
V_h = \{ \mathbf{v} \in L^2(\Omega, \mathbb{R}^r) : \mathbf{v}|_E \in [P_p(E)]^r, \ E \in \mathcal{T}_h \}, \ p \in \mathbb{N},
\]  

(2)

where \( P_p(E) \) is the space of polynomials whose degree do not exceed \( p \). We let \( \Gamma_i \) denote the set of intersections between all pairs of elements in \( \mathcal{T}_h \) and accordingly \( \Gamma \) the set of all
intersections including the boundary of $\Omega$. In DUNE, the following commands generate the tessellation $T_h$ and the space $V_h$:

```python
import dune.grid
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
# create DG space with Lagrange basis and Gauss-Lobatto interpolation points
V_h = dglagrangelobatto(T_h, order=p)
```

We seek an approximate solution $u_h \in V_h$ by discretizing the spatial operator $L(u)$ in (1). To this end we define for all test functions $\psi \in V_h$,

$$
\langle \psi, L_h(u_h) \rangle := \langle \psi, K_h(u_h) \rangle + \langle \psi, I_h(u_h) \rangle.
$$

Here, the element integrals are given by

$$
\langle \psi, K_h(u_h) \rangle := \sum_{E \in T_h} \int_E \left( \left( F_c(u_h) - F_v(u_h, \nabla u_h) \right) : \nabla \psi + S(u_h) \cdot \psi \right) dx,
$$

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
u = TrialFunction(V_h)
psi = TestFunction(V_h)
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

$$
\langle \psi, I_h(u_h) \rangle := \sum_{e \in T_h} \int_e \left( \left( F_c(u_h) - F_v(u_h, \nabla u_h) \right) : \nabla \psi \right) e + \left( F_v(u_h, \nabla u_h) \right) e : \left[ \psi \right] e dS
$$

$$
- \sum_{e \in T_h} \int_e \left( H_c(u_h) - H_v(u_h, \nabla u_h) \right) : \left[ \psi \right] e dS.
$$

This formulation arises from considering the weak form of the problem: Replace $u$ by $u_h$ in (1), multiply by the test function $\psi$ and integrate the spatial terms by parts. Here, $H_c$ and $H_v$ are suitable numerical fluxes, imposed at the element interface $e$. Further, $\left\{ u \right\}_e$ and $\left[ u \right]_e$ denote the average and jump of $u$ over $e$,

$$
\left\{ u \right\}_e := \frac{1}{2} (u_E + u_K) \quad \text{and} \quad \left[ u \right]_e := n_e \cdot (u_E - u_K)
$$

where $E$ and $K$ are neighboring elements over intersection $e$ and $n_e$ is outward pointing from element $E$.

The corresponding code reads:
To obtain the DG-SEM we follow [50, 49]. First, we restrict our focus to cuboid meshes and map each $E \in \mathcal{T}_h$ 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 $u_h(t) \in V_h$ takes the form

$$u_h(t,x) = \sum_i u_i(t) \psi_i(x),$$

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

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

The convective and viscous fluxes are approximated using the interpolation

$$F_h(t,x) \approx \sum_{i=1}^{\infty} F(u_i(t)) \psi_i(x),$$

where $F$ is either $F_c$ or $F_v$. A variety of implementations for $F_c$ and $F_v$ is provided by the dolfin-dg package (see [41]), which we use for the Euler equations.

Finally, the element and surface integrals in (4) and (5) 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.

The convective numerical flux $H_c$ can be any appropriate numerical flux known for standard finite volume methods. We use the local Lax-Friedrichs (Rusanov) flux function

$$H_c^{LLF}(u_h)|_e := \left\{ \frac{\lambda_e}{2} \frac{\|u_h\|^2}{\|u_h\|} \right\}_e + \lambda_e e \left\{ F_c(u_h) \right\}_e,$$  \hspace{1cm} (7)

where $\lambda_e$ is an estimate of the maximum wave speed on the interface $e$. Other options are implemented in DUNE-FEM-DG (cf. [18, 19]) as well.

A wide range of diffusion fluxes $H_v$ can be found in the literature (cf. [11] and 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 the other fluxes. For the Interior Penalty method the flux is chosen to be

\[ H_{IP}^{u}(u, \nabla u) = \| \nabla u \|_{e} - \frac{\eta}{h_{e}} \| F_{e}(u, \nabla u) \|_{e} \| u \|_{e} \] (8)

with \( \eta \) being the penalty parameter.

### 3.2 Temporal Discretization

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

\[ u'(t) = F(t, u(t)), \quad t \in (0, T], \quad u(0) = u_0. \] (9)

Here, \( F(t, u(t)) = M^{-1} \mathcal{L}_h(u_h(t)) \), where \( \mathcal{L}_h \) is defined in (3) and \( M \) is the (diagonal) global mass matrix of the DG-SEM discretization. The initial data \( u_0 \) for (9) is given by the projection of \( u_0 \) onto \( V_h \).

Any Runge-Kutta method can in principle be used to solve (9). 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

\[
\begin{array}{c|c}
 c & A \\
 \hline
 b^T \\
\end{array}
\]

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

\[ u = 1 \otimes u^n + \Delta t_n (A \otimes I_\xi) F^n, \] (10)

where the vector \( u^T = (u^1, \ldots, u^{N_T}) \) contains the \( N_T \) intermediate solution stages and \( F^n = (F(t_n + \Delta t_n c_1, u^1), \ldots, F(t_n + \Delta t_n c_{N_T}, u^{N_T}))^T \). Here, \( u^n \) denotes the RK solution in the previous time step. The new solution is given by

\[ u^{n+1} = u^n + \Delta t_n (b^T \otimes I_\xi) F^n. \] (11)

Herein we consider a particular family of implicit RK methods, namely Lobatto IIIC [43, 53]. These methods are A-, L- and B-stable and are thus suitable for stiff and nonlinear problems. The order of the \( N_T \)-stage Lobatto IIIC method is \( 2(N_T - 1) \) and the order of the individual stages is \( N_T - 1 \). 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 Appendix B.

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
```
4 Space-Time DG-SEM

We now consider DG-SEM applied to (1) with the time variable $t$ treated simply as an additional dimension. The result is space-time DG-SEM.

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

$$F_c = \begin{bmatrix} F_c \ u \end{bmatrix}, \quad F_v = \begin{bmatrix} F_v \ 0 \end{bmatrix},$$

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

$$\nabla \cdot \left( F_c(u) - F_v(u, \nabla u) \right) = S(u) \quad \text{in } \Omega,$$

(12)

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

$$V_h = \left\{ v \in L^2(\Omega, \mathbb{R}^r) : \nabla v \in \left[ P_p(E) \right]^r, \ E \in \mathcal{T}_h \right\}, \quad p \in \mathbb{N}.$$ (13)

Then the space-time DG-SEM discretization of (12) follows analogously to (4) and (5):

$$\langle \psi, L_h(u_h) \rangle := \langle \psi, K_h(u_h) \rangle + \langle \psi, I_h(u_h) \rangle,$$ (14)

with the element integrals

$$\langle \psi, K_h(u_h) \rangle := \sum_{E \in \mathcal{T}_h} \int_E \left( (F_c(u_h) - F_v(u_h, \nabla u_h)) : \nabla \psi + S(u_h) \cdot \psi \right) dx,$$ (15)

and the surface integrals

$$\langle \psi, I_h(u_h) \rangle := \sum_{e \in \Gamma_i} \int_e \left( \left[ F_c(u_h), \left[ u_h \right]_e \right]^T : \nabla \psi \right)_e + \left[ (F_v(u_h, \nabla u_h))_e : \left[ \psi \right]_e \right]_e dS$$

$$- \sum_{e \in \Gamma} \int_e (H_c(u_h) - H_v(u_h, \nabla u_h)) : \left[ \psi \right]_e dS.$$ (16)

Here, $\Gamma_i$ and $\Gamma$ have analogous meanings to their spatial counterparts $\Gamma_i$ and $\Gamma$. The numerical fluxes are given by

$$H_c = \begin{bmatrix} H_c \ u^* \end{bmatrix}, \quad H_v = \begin{bmatrix} H_v \ 0 \end{bmatrix},$$

where $u^*$ is a simple upwind flux in time.

In our considered framework, (14) can be implemented quite nicely by increasing the dimension and applying the above discussed modifications$^1$.

$^1$Note that for the 4D version ($3d + time$) a UFL patch (see Appendix C) was added to introduce the 4D reference elements to UFL code.
Remark 4.1. It is of practical interest to generalize the space $V_h$ so that the time dimension may be discretized by polynomials of a different order than the spatial dimensions. We will henceforth refer to the number of temporal nodes in each element as $N_\tau$ 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 $N_\tau$ to count the degrees of freedom within a time element, whether this pertains to the DG or RK interpretation.

After space-time discretization, the discrete solution $\mathbf{u}_h \in V_h$ takes the form $\mathbf{u}_h(t,x) = \sum_{i,n} u_{ni}(t) \psi_i(x)$. Here, the sum is taken over all tensor product LGL nodes in $d+1$ dimensions. The vector of coefficients is now given by

$$\mathbf{u} = (\mathbf{u}^1, \ldots, \mathbf{u}^{N_\tau})^T,$$

where $\mathbf{u}^i$ contains all the spatial unknowns in the $i$th time element.
The space-time discretization (14) can alternatively be derived by starting from (9) and discretizing in time with DG-SEM. Multiplying (9) by a test function $\psi(t)$ and integrating over the $n$th time element results in

$$
\int_{t_n}^{t_{n+1}} u_i \psi dt = \int_{t_n}^{t_{n+1}} F(t, u(t)) \psi dt.
$$

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

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

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

$$
u \approx \sum_{j=1}^{N_r} u^j \psi_j(\tau),
$$

$$
F \approx \sum_{j=1}^{N_r} F^j \psi_j(\tau),
$$

and the integrals by Gauss-Lobatto quadrature with nodes $\tau_j$ and weights $\omega_j$. Using the cardinal property of the Lagrange basis polynomials $\psi$, the resulting DG-SEM discretization becomes

$$
\delta_{iN_r} u^* - \delta_{i1} u^* - \sum_{j=1}^{N_r} \omega_j u^j \frac{d\psi_i}{d\tau} \bigg|_{\tau_j} = \frac{2}{\Delta t_n} \omega_i F^i, \quad i = 1, \ldots, N_r.
$$

(18)

Here, we have replaced the boundary terms with numerical fluxes $u^*$. With DG-SEM in time, the numerical flux $u^*$ is always chosen as the upwind flux

$$
u^* = (u^n, 0, \ldots, 0, u^{N_r})^T,
$$

(19)

where $u^n$ is the numerical solution from the previous time element. This choice leads to an entropy stable numerical scheme if the spatial terms are handled appropriately [29]. It also has the advantage of decoupling the temporal elements. Thus, (20) can be solved as a stand-alone nonlinear system on the $n$th time element.

Defining the boundary, mass and differentiation matrices

$$
B_\tau = \text{diag}([-1, 0, \ldots, 0, 1]) \in \mathbb{R}^{N_r \times N_r},
$$

$$
M_\tau = \text{diag}([\omega_1, \ldots, \omega_{N_r}]) \in \mathbb{R}^{N_r \times N_r},
$$

$$
(D_\tau)_{ji} = \frac{d\psi_i}{d\tau} \bigg|_{\tau_j} \in \mathbb{R}^{N_r \times N_r},
$$

we can write (18) in matrix form on each reference element as

$$
(B_\tau \otimes I_\xi) u^* - (D^T_\tau M_\tau \otimes I_\xi) u = \frac{\Delta t_n}{2} (M_\tau \otimes I_\xi) F(u),
$$

(20)

where $M_\tau$ is the local temporal mass matrix and $M_\tau D_\tau$ defines the corresponding stiffness matrix. Here, $F^T(u) = (F^T(t_n + \frac{\Delta t_n}{2} (1 + \tau_1), u^1), \ldots, F^T(t_n + \frac{\Delta t_n}{2} (1 + \tau_{N_r}), u^{N_r}))$, where $u$ is given by (17) and $\tau_k$ is the $k$th LGL node; see [33] for details. The operation $\otimes$ denotes the Kronecker product and $I_\xi$ is the identity matrix whose dimension is given by the number of spatial nodes.

We finish this section by remarking that while (14) describes the global space-time DG-SEM discretization, the alternative formulation (20) 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.

\[ u_{N_r}^n = (e_{N_r}^\top \otimes I_\xi)u, \]

where \( e_{N_r}^\top = (0, \ldots, 0, 1) \in \mathbb{R}^{N_r} \). We will also make use of the vector \( e_1^\top = (1, 0, \ldots, 0) \in \mathbb{R}^{N_r} \).

Following [10], we set out to show that the DG-SEM discretization (20) constitutes a so-called Summation-By-Parts (SBP) method [33], meaning that the following conditions are satisfied:

\[ M_\tau = M_\tau^\top > 0, \quad M_\tau D_\tau + (M_\tau D_\tau)^\top = B_\tau. \]

The SBP property (22) 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 [52, 70]. For an overview of these techniques, see [72, 25]. In recent years, their use as time stepping schemes has been explored [62] and connections to implicit Runge-Kutta methods have been discovered [10]. Here we summarize the steps showing that (20) can be reformulated as an implicit RK method applied to the system of ODEs (9).

We begin by using the SBP property (22) in the second term of (20) and then multiplying by \((M_\tau^{-1} \otimes I_\xi)\) to obtain the so-called strong form,

\[ (D_\tau \otimes I_\xi)u = (M_\tau^{-1} B_\tau \otimes I_\xi)(u - u^*) + \frac{\Delta t_n}{2} F. \]

Note that \( B_\tau = e_{N_r}^\top e_{N_r}^\top - e_1 e_1^\top \) and \((e_{N_r}^\top \otimes I_\xi)(u - u^*) = (u_{N_r}^n - u_{N_r}^n) = 0\). Using (19), the second term in (23) can therefore be expressed as

\[ (M_\tau^{-1} B_\tau \otimes I_\xi)(u - u^*) = -(M_\tau^{-1} \otimes I_\xi)[(e_1 e_1^\top \otimes I_\xi)u - (e_1 \otimes u^*)]. \]

Grouping together terms that multiply the solution \( u \), we rewrite (23) as

\[ ((D_\tau + M_\tau^{-1} e_1 e_1^\top) \otimes I_\xi)u = (M_\tau^{-1} e_1 \otimes u^n) + \frac{\Delta t_n}{2} F. \]

Next, we multiply (24) by \((D_\tau + M_\tau^{-1} e_1 e_1^\top) \otimes I_\xi)^{-1}\). Upon doing this, first note that

\[ (D_\tau + M_\tau^{-1} e_1 e_1^\top)^{-1} M_\tau^{-1} e_1 = 1 := (1, \ldots, 1) \in \mathbb{R}^{N_r}, \]

which follows from observing that \((D_\tau + M_\tau^{-1} e_1 e_1^\top)1 = M_\tau^{-1} e_1\) since \( D_\tau 1 = 0 \) by consistency. Thus, the following system arises:

\[ u = 1 \otimes u^n + \frac{\Delta t_n}{2} \left( (D_\tau + M_\tau^{-1} e_1 e_1^\top) \otimes I_\xi \right)^{-1} F \]

\[ = 1 \otimes u^n + \frac{\Delta t_n}{2} \left( (D_\tau + M_\tau^{-1} e_1 e_1^\top)^{-1} \otimes I_\xi \right) F. \]
The equation system (25) should be compared with the stage equations (10) that arose from the MOL discretization using implicit RK. We see that the temporal DG-SEM discretization defines an RK method with coefficient matrix \( A = \frac{1}{2}(D_\tau + M_{\tau}^{-1}e_1e_1^\top) \) and nodes \( c = (1 + \tau)/2 \), where \( \tau = (\tau_1, \ldots, \tau_{N_\tau})^\top \) is the vector of LGL nodes. Further, the vector \( u \), which in the DG-SEM context contains the interpolation coefficients \( u_1^n \), 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, \( u^{N_\tau} = (e_{N_\tau}^\top \otimes I_\xi)u \). To this end we observe that the SBP property (22) gives the relation

\[
1^\top M_{\tau}(D_\tau + M_{\tau}^{-1}e_1e_1^\top) = 1^\top (e_{N_\tau}e_{N_\tau}^\top - D_{\tau}^\top M_{\tau}) = e_{N_\tau}^\top,
\]

so that

\[
e_{N_\tau}^\top(D_\tau + M_{\tau}^{-1}e_1e_1^\top)^{-1} = 1^\top M_{\tau}.
\]

Consequently, multiplying (25) by \((e_{N_\tau}^\top \otimes I_\xi)u\) yields

\[
u^{N_\tau} = u^n + \Delta t_n\frac{1}{2}(1^\top M_{\tau} \otimes I_\xi)\hat{F}.
\]

Comparing (26) with the solution (11) of the implicit RK method, we see that the vector \( b \) in the Butcher tableau is related to the DG-SEM discretization by \( b^\top = 1^\top M_{\tau}/2 \), and that the RK solution is simply the \( N_\tau \)th component of the DG solution \( u \).

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

\[
A = \frac{1}{2}(D_\tau + M_{\tau}^{-1}e_1e_1^\top)^{-1}, \quad b = \frac{1}{2}M_{\tau}1, \quad c = \frac{1+\tau}{2}.
\]

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

\[
(B_\tau \otimes I_\xi)u^* - (D_{\tau}^\top M_{\tau} \otimes I_\xi)u = \frac{\Delta t_n}{2}(M_{\tau} \otimes I_\xi)\hat{F},
\]

(28a)

\[
u = 1 \otimes u^n + \Delta t_n(A \otimes I_\xi)\hat{F}.
\]

(28b)

These systems have the same solution \( u \) since we can transition from (28a) to (28b) in a series of algebraic steps. More precisely, the connection is made by rewriting (28a) in strong form, then multiplying by \((D_\tau + M_{\tau}^{-1}e_1e_1^\top \otimes I_\xi)^{-1}\).

Note that the latter step demands that \( D_\tau + M_{\tau}^{-1}e_1e_1^\top \) is invertible. This is the case if and only if \( D_\tau \) is null-space consistent, i.e. if \( \ker(D_\tau) = \text{span}(1) \) [55]. This is known to hold for all \( N_\tau \geq 1 \) [67, 54].

Finally, the Butcher tableau formed from (27) 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 [65]. 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 \( u^{N_\tau} = u^{n+1} \). The coefficients for the DG-SEM matrices with \( N_\tau \in \{2, 3, 4\} \) are listed in Appendix D.
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 (9). The (classical) notion of order is defined as follows:

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

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

holds, whenever problem (9) is sufficiently smooth.

The norm can be any vector norm and $u^{n+1} - u(t_{n+1})$ 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

\begin{align*}
B(p_B): \quad b^T c^{j-1} &= \frac{1}{j}, \quad j = 1, \ldots, p_B, \\
C(p_C): \quad A c^{j-1} &= c^j, \quad j = 1, \ldots, p_C, \\
D(p_D): \quad A^T \text{diag}(b) c^{j-1} &= \frac{1}{j} \text{diag}(b)(1 - c^j), \quad j = 1, \ldots, p_D,
\end{align*}

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

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

**Definition 5.3.** 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 [37, Chapter IV.5].

**Theorem 5.4.** The Lobatto IIIC method with $N_\tau$ 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 $u_h$, this norm can be computed via the quadrature rule exactly:

$$\|u_h\|_{L_2(\Omega)} = \sum_i (u^T_{h_i} M_{\xi_i} u_{h_i})^\frac{1}{2}.$$  \hspace{1cm} (29)

The sum is taken over all elements and $M_{\xi_i}$ 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 (1), the choice of numerical fluxes, and sometimes on whether $p$ is odd or even [38, 77].

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 $N_\tau$ [56]. 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 $D_\tau \tau^q = q \tau^{q-1}$. The exponentiation should be interpreted elementwise, and we take $\tau^0 = 1$ as a definition. For DG-SEM we have $q = N_\tau - 1$.

  Multiplying $C(p_C)$ by $A^{-1}$ as given in (27) and utilizing the fact that the first element in $c$ is zero, we see that the RK order condition $C(p_C)$ actually describes precisely the order of the operator $D_\tau$. 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 is the highest degree $m$ such that $(m + 1) \tau^m = 1 - (-1)^{m+1}$, i.e. for which $M_\tau$ exactly integrates polynomials. For DG-SEM, $1^T M_\tau$ is a row vector with the $N_\tau$ weights of the Gauss-Lobatto quadrature rule and we consequently have $m = 2 (N_\tau - 1)$.

  Using (27) we note that the condition $B(p_B)$ simply describes the order of the quadrature, although applied to $c$ rather than $\tau$. 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 $u^{N_\tau} \equiv u^{n+1}$ is $2(N_\tau - 1)$ [56], at least for smooth nonstiff problems. This superconvergence can be proven using the theory of dual consistent SBP methods [39]. Here it suffices to say that it is a consequence of the order of the quadrature and choosing the upwind numerical flux (19).

  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 (22) and the diagonality of $M_\tau$ it is shown in [10] that $C(p_C)$ is satisfied with $p_C = N_\tau - 1$, which is consistent with Theorem 5.4.

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 (9) satisfies a one-sided Lipschitz condition,

$$\langle u - v, F(t, u) - F(t, v) \rangle \leq \beta \| u - v \|^2,$$

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 \( \beta = 0 \) if \( N_r > 2 \); see [68] for details.

Regularity of the type (30) is not standard in the literature on spatial discretization using high order DG methods. Rather, estimates of the form \( \langle u, F(t, u) \rangle \leq 0 \) 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 [61].

If \( F(t, 0) = 0 \), 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 \( \langle \eta'(u), F(t, u) \rangle \leq 0 \). Here, \( \eta \) is some convex function of \( u \) referred to as an entropy [27, 26]. A convergence theory for implicit RK methods applied to entropy stable problems is desirable but currently appears to be missing from the literature.

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 inevitable face when deciding on which implementation to select.

Several multi-dimensional DG-SEM solvers exist, such as Nektar++ [45], Fluxo, Flexi and the latest iteration Trixi [66] and others. In particular, this approach is popular for weather and climate prediction and has been used e.g. in NUMA [57] and HOMAM [58]. 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 [35] 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 di-
mension \( d < 3 \), 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 neces-
sary. 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 in-
stance 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 (28b) while the STDG code solves (28a), or equivalently solves $L_h(u_h) = 0$ from (14). 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 straightforward 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 $u^{n+1}$ in a simple way since this requires the extraction of a specific subset of coefficients from the numerical solution vector $u$. 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 $u^{n+1} = (e_{N_r}^T \otimes I_ξ) u$ with $e_{N_r}^T = (0, \ldots, 0, 1)$, as was done in Section 5. However, this assumes that the ordering of the unknowns in $u$ is identical to the one used in that analysis. If not, $(e_{N_r}^T \otimes I_ξ)$ must be suitably permuted into some matrix $E_{N_r}$ 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 $\hat{b}$ to compute a second numerical solution $\hat{u}^{n+1}$ from (11) whose accuracy is one order lower than that of $u^{n+1}$. The difference $u^{n+1} - \hat{u}^{n+1}$ can be used to estimate the local error without the need to solve the nonlinear system (28b) more than once. A detailed strategy for estimating the error and choosing the time step based on the embedding technique is available in [37, Chapter IV.8] for the Radau IIA method, but can be easily adapted to Lobatto IIIC [53].

An STDG code that follows the steps outlined in Section 4 will not have an embedded method. However, if the matrix $E_{N_r}$ can be found that extracts $u^{n+1}$, then it is also possible to construct a matrix $\hat{E}_{N_r}$ that extracts $\hat{u}^{n+1}$ 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 [36, Chapter II.4]. This procedure requires solving the nonlinear system (28a) three times; once with a step size $2\Delta t$ and twice with a step size $\Delta t$. 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 [44, 17] 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 [76, 14], 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 [69].

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 (28a) and (28b) 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 [37], 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. [63, 64]. Likewise, nonlinear solvers designed for space-time DG and FEM discretizations have been developed [47, 71].

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, 64, 46] for recent developments. However, attention must be payed to the fact that the systems (28a) and (28b) have different algebraic properties and therefore likely will respond differently to preconditioners and nonlinear solvers.

For the nonlinear LoDG system (28b), the Jacobian is given by

\[ I - \Delta t_n (A \otimes I_\xi) J(F), \]  

(31)

where \( J(F) \) contains the Jacobian of the spatial discretization. The solver in [37], and many recent developments that build upon it, instead use the mathematically equivalent

\[ (\Delta t_n A)^{-1} \otimes I_\xi - J(F). \]  

(32)

For the nonlinear STDG system (28a), the Jacobian is given by

\[ \left( D^\tau M^\tau - e_{N_\tau} e_{N_\tau}^\tau \right) \otimes I_\xi + \frac{\Delta t_n}{2} \left( M^\tau \otimes I_\xi \right) J(F). \]  

(33)

Note from (27) that (32) arises by multiplying (31) by \( (D^\tau + M^{-1}_{N_\tau} e_1 e_1^\tau) \otimes I_\xi \). This formulation is therefore very closely related to the STDG Jacobian (33). In fact, they only differ by an application of the SBP property (22) 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 1 shows the sparsity patterns of the Jacobians using order 1, 2 and 3 in space and time. In each figure quadruplet, the Jacobian (33) of STDG is shown in the top left and the Jacobian (31) of LoDG in the top right. In the bottom right, the alternative formulation (32) 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 \( A^{-1} \) 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 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 $A^{-1}$ coincides with STDG as seen in the bottom left of each figure quadruplet.

These observations suggest that the LoDG system (28b) is more expensive to work with than the STDG system (28a), and that the $A^{-1}$ formulation (32) 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.

![Figure 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).](image)

7 Experiments

We now perform a series of numerical tests. We start with solving the linear test equation to validate the 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,

\[ u_t = -u, \quad t \in (0,1], \]

\[ u(0) = 4. \quad (34) \]

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 \( |u^{n+1} - u(1)| \) is shown for LoDG and STDG in Table 1. Here, \( N \) denotes the number of time steps/time elements and \( N_r \in \{2,3,4\} \). Recall from Section 5 that the order of LoDG, and correspondingly the superconvergence of STDG, is \( 2(N_r - 1) \). This is indeed what we observe in Table 1. With \( N_r = 3 \), the errors are approaching machine precision when \( N = 2^9 \), hence a drop in the convergence rate is seen in Table 1b. The STDG appears to be more sensitive in this respect than LoDG. The same thing happens when \( N_r = 4 \) and \( N = 2^5 \), as seen in Table 1c.

Table 1: EOC of pointwise error for LoDG and STDG applied to the test equation (34).

| \( N_r = 2 \) | \( N_r = 3 \) | \( N_r = 4 \) |
|---|---|---|
| \( 2^4 \) | 1.93 | 3.96 | 5.98 |
| \( 2^5 \) | 1.97 | 3.98 | 6.64 |
| \( 2^6 \) | 1.98 | 3.99 | 4.22 |
| \( 2^7 \) | 1.99 | 3.99 | 4.01 |
| \( 2^8 \) | 1.99 | 4.01 | 4.28 |
| \( 2^9 \) | 1.99 | 4.05 | 0.47 |

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 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 \( N_r \). Table 2 shows that this indeed is observed. For \( N_r = 4 \) with \( N = 2^8 \) time elements the convergence rate drops due to very small errors, as seen in Table 2c. Again, STDG appears to be more sensitive to this phenomenon than LoDG.

7.2 Advection-Diffusion

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

\[ \partial_t u + b \cdot \nabla u - \varepsilon \Delta u = 0 \quad \text{in} \ (\Omega \times (0,T]), \ \Omega \in \mathbb{R}^2, \]

\[ u(0) = u_0 \quad \text{in} \ \Omega. \quad (35) \]
Table 2: EOC of $L_2$ error for LoDG and STDG applied to the test equation (34).

|   | $N = 2$  | $N = 3$  | $N = 4$  |
|---|----------|----------|----------|
|   | Lobatto | DG-SEM  | Lobatto | DG-SEM  | 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     |

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

$$u(t, x) = \frac{0.004}{0.004 + 4\varepsilon t} \exp \left( -\frac{x_q^2 + y_q^2}{0.004 + 4\varepsilon t} \right),$$

$$x_q = x_0 \cos(4t) + y_0 \sin(4t) + 0.25,$$

$$y_q = -x_0 \sin(4t) + y_0 \cos(4t).$$

Here, $x_0 = x - 0.5$, $y_0 = y - 0.5$, $b = [-4y_0, 4x_0]$, $\varepsilon = 0.001$, and $(t, x) \in [0, 1] \times [0, 1]^2$. The initial condition is given by $u(0, x)$ 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 $N_t \in \{2, 3, 4\}$ are shown in Figure 2. 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 $N_t = 2$ the problem is significantly under-resolved, leading to a smeared solution. As $N_t$ 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 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 behaviour of 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. The results indicate a convergence rate higher than $N_t$, although not quite as high as $2(N_t - 1)$.

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 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 3. Here, the exact solution over the whole space-time domain is also shown for reference.
(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 2: Numerical solution of a rotating pulse subject to the advection-diffusion equation (35) using LoDG (top) and STDG (bottom).

| $N_\tau$ | $N$ | LoDG | STDG | LoDG | STDG | LoDG | STDG |
|----------|-----|------|------|------|------|------|------|
| 2        | $2^2$ | 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 |

Table 3: Errors and EOC for the advection-diffusion problem (35).

(a) Error

(b) EOC

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:

\[
\begin{align*}
\frac{\partial}{\partial t} \mathbf{u} + \nabla \cdot F_c(\mathbf{u}) &= 0 \quad \text{in } (\Omega \times (0, T]), \quad \Omega \subset \mathbb{R}^d, \quad d \in \{1, 2, 3\}, \\
\mathbf{u}(0) &= \mathbf{u}_0 \quad \text{in } \Omega,
\end{align*}
\]

(36)

where the vector of the conservative variables has the form

\[
\mathbf{u} = \begin{pmatrix} \rho \\ \rho \mathbf{v} \\ \varepsilon \end{pmatrix}, \quad \rho \mathbf{v} = (\rho v_1, \ldots, \rho v_d)^T, \quad \varepsilon = \rho \mathcal{E},
\]

(37)

augmented with suitable boundary conditions (which are discussed in detail in [8]). Here, \(\rho\) denotes the density of the fluid, \(\mathbf{v}\) the velocity, \(\varepsilon\) the internal energy, and \(\mathcal{E}\) the total energy. The convective flux function \(F_c(\mathbf{u}) := (f_1(\mathbf{u}), \ldots, f_d(\mathbf{u}))\) has for \(i = 1, \ldots, d\) the
with vortex strength $S$

\[
\begin{align*}
\text{Figure 4. Here, a uniform mesh on the space-time domain } x & \text{ with }
p
\text{the simulation. Again, the problem is significantly under-resolved with }
(37) \text{ we obtain the three flux functions}
\end{align*}
\]

where $\delta_{ij}$ is the Kronecker delta. For example, choosing $d = 3$ and directly using $u$

We use periodic boundary conditions and a smooth bubble advection initial condition

\[
\begin{align*}
\frac{\partial v_1}{\partial t} + \nabla \cdot \mathbf{u} & = f_1(v_1), \\
\frac{\partial v_2}{\partial t} + \nabla \cdot \mathbf{u} & = f_2(v_2), \\
\frac{\partial v_3}{\partial t} + \nabla \cdot \mathbf{u} & = f_3(v_3),
\end{align*}
\]

We first consider the two-dimensional Euler equations with periodic boundary conditions

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

\[
v_1 = 1 - S \rho \frac{\exp(f)}{2\pi}, \quad v_2 = S \rho \frac{\exp(f)}{2\pi}, \quad v_3 = \rho \gamma - 1 + 0.5 \frac{v_1^2 + v_2^2}{\rho}, \quad P = \frac{\rho \gamma}{\gamma M^2},
\]

with vortex strength $S = 5$, Mach number $M = 0.5$, $\gamma = 1.4$ and $f = 1 - x_3 - x_1$.

The numerical solutions obtained by the two codes with $N_x \in \{2, 3\}$ are shown in Figure 4. 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 $N_x = 2$, leading to a smeared solution. As $N_x$ 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.

To show the potential of our STDG code we present a 3D Euler test case with $N_x = 3$. 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 $\hat{x} = 16 \sum_{i=1}^d (x_i - 0.25 - tv_i)^2$. 

25
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 [3, 4]. 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 5. 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 [30], but to the best of our knowledge this is the first 4D DG-SEM implementation available publicly.

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 Appendix A 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 available 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

There exist different ways to install DUNE and ASSIMULO. Here, we only describe the simplest and most straightforward way to install both, DUNE and ASSIMULO, which is to use a conda environment. Then the installation is done in the following way:

```bash
conda create -n duneproject # create a new conda environment
conda activate duneproject # activate the conda environment
conda install -c conda-forge assimulo # install assimulo
conda install -c conda-forge scipy # install scipy
pip install -U dune-fem-dg # install dune using pip, no conda package yet
git clone https://bitbucket.org/nate-sime/dolfin_dg.git _dg.git # install dolfin
```

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

Appendix B

The Butcher tableaus for the $N_T$-stage Lobatto IIIC methods with $N_T = 2, 3, 4$ can be seen in Table 4.

Appendix C

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

```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)
```
Table 4: Butcher Tableaus for Lobatto IIIC methods

(a) $N_\tau = 2$

(b) $N_\tau = 3$

(c) $N_\tau = 4$
Appendix D

For $N = 2$ we get

$$B = \begin{pmatrix}
-1 & 0 \\ 0 & 1
\end{pmatrix}, \quad M = \begin{pmatrix}
1 & 0 \\ 0 & 1
\end{pmatrix}, \quad D = \begin{pmatrix}
-\frac{1}{2} & \frac{1}{2} \\ -\frac{1}{2} & \frac{1}{2}
\end{pmatrix}. \quad (38)$$

For $N = 3$ we get

$$B = \begin{pmatrix}
-1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1
\end{pmatrix}, \quad M = \begin{pmatrix}
\frac{1}{6} & 0 & 0 \\ 0 & \frac{1}{3} & 0 \\ 0 & 0 & \frac{1}{2}
\end{pmatrix}, \quad D = \begin{pmatrix}
-\frac{3}{2} & 2 & -\frac{1}{2} \\ -\frac{1}{2} & 0 & \frac{1}{2} \\ -\frac{1}{2} & \frac{1}{2} & \frac{1}{2}
\end{pmatrix}. \quad (39)$$

For $N = 4$ we get

$$B = \begin{pmatrix}
-1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1
\end{pmatrix}, \quad M = \begin{pmatrix}
\frac{1}{6} & 0 & 0 & 0 \\ 0 & \frac{5}{6} & 0 & 0 \\ 0 & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & \frac{1}{6}
\end{pmatrix}, \quad D = \begin{pmatrix}
-3 & \frac{5+\sqrt{5}}{4} & \frac{5-\sqrt{5}}{4} & \frac{1}{4} \\ -\frac{1}{4} & 0 & \frac{1+\sqrt{5}}{2} & \frac{1}{4} \\ -\frac{1+\sqrt{5}}{4} & -\frac{\sqrt{5}}{2} & 0 & \frac{1}{4} \\ -\frac{1}{2} & \frac{5\sqrt{5}-5}{4} & -\frac{5-\sqrt{5}}{4} & 3
\end{pmatrix}. \quad (40)$$

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. Simulat.*, 116(0):26 – 43, 2015.

[3] S. Balay, S. Abhyankar, M. F. Adams, S. Benson, J. Brown, P. Brune, K. Buschelman, E. 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.
[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] P. Bastian, M. Blatt, M. Dedner, N.-A. Dreier, R. Engwer, Ch. Fritze, C. Gräser, C. 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.

[6] M. Behr. Simplex space–time meshes in finite element simulations. Int. J. Numer. Meth. Fl., 57(9):1421–1434, 2008.

[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, Boca Raton, London, New York, 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. Haines, D. L. Darmofal, and M. C. Galbraith. Four-dimensional anisotropic mesh adaptation. Comput. Aided Design, 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. Computat. 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. *Comm. App. Math. Comp. 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)*, 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, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[24] 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.

[25] D. C. D. R. 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.

[26] 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.

[27] 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.

[28] 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.

[29] L. Friedrich, G. Schnücke, A. R. Winters, D. C. R. 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.

[30] 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.
[31] 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, Cham, 2015. Springer.

[32] 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.

[33] 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.

[34] 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.

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

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

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

[38] J. S. Hesthaven and T. Warburton. *Nodal discontinuous Galerkin methods: algorithms, analysis, and applications*. Springer-Verlag New York, 2008.

[39] 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.

[40] 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 Transactions on Mathematical Software (TOMS)*, 31(3):363–396, 2005.

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

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

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

[44] 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. *Computat. Geosci.*, 22(1):107–123, 2018.

[45] G. Karniadakis and S. Sherwin. *Spectral/hp Element Methods for Computational Fluid Dynamics*. Oxford University Press, Oxford, 2013.
[46] 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.

[47] C. M. Klaaij, M. H. van Raalte, H. van der Ven, and J. J. 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.

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

[49] D. A. Kopriva and G. Gassner. On the quadrature and weak form choices in collocation type discontinuous galerkin spectral element methods. J. Sci. Comput., 44:136–155, 2010.

[50] 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.

[51] N. Krais, A. Beck, T. Bolemann, H. Frank, D. Flad, G. 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.

[52] 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, London, 1974.

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

[54] V. Linders. On an eigenvalue property of Summation-By-Parts operators. arXiv preprint arXiv:2201.01193, 2022.

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

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

[57] 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.

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

[59] M. Neumüller. Space-Time Methods, volume 20 of Monograph Series TU Graz: Computation in Engineering and Science. TU Graz, 2013.
[60] J. Nievergelt. Parallel methods for integrating ordinary differential equations. *Commun. ACM*, 7(12):731–733, 1964.

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

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

[63] 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.

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

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

[66] H. Ranocha, M. Schlottke-Lakemper, A. R. Winters, E. Faulhaber, J. Chan, and G. Gassner. Adaptive numerical simulations with Trixi.jl: A case study of Julia for scientific computing, 08 2021.

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

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

[69] 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.

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

[71] 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.

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

[73] 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.

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

[75] L. M. Versbach, P. Birken, V. Linders, and G. Gassner. Local Fourier Analysis of a Space-Time Multigrid Method for DG-SEM for the Linear Advection Equation. *arXiv preprint arXiv:2112.03115*, 2021.
[76] M. Yano and D. L. Darmofal. An optimization-based framework for anisotropic simplex mesh adaptation. *J. Comput. Phys.*, 231(22):7626–7649, 2012.

[77] 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.