Untersuchung von Präkonditionierern für implizite Löser für das Local DG-Verfahren zur Lösung der Advektions-Diffusionsgleichung

Examination of preconditioners for implicit solvers for the Local DG method for solving the advection-diffusion equation

Masterarbeit

vorgelegt von **Niklas Böing**
am 06.01.2022

am Mathematischen Institut der
Universität zu Köln

Erstgutachter: Prof. Dr. Gregor Gassner

Acknowledgment

First of all, I would like to thank my thesis supervisor Prof. Dr. Gassner from the University of Cologne as well as my advisor Dr. Holke from the German Aerospace Center. Thank you for your valuable advice and consistent guidance throughout this thesis. I am deeply grateful for the support, patience, and encouragement I have received.
Furthermore, I would like to thank David Knapp for many helpful, inspiring, and enjoyable discussions.
A special thanks also goes to the members and colleagues from the High-Performance-Computing department of the German Aerospace Center for many motivating and valuable moments during the creation of this thesis.

Contents

1. Introduction

A large part of numerical simulation is the area of computational fluid dynamics (CFD) which covers many scientific applications from small to large scales ([A95], [AB17]).
Atmospheric modeling and simulations like in the filed of numerical weather prediction (NWP) also take a big part within CFD. The set of governing partial differential equations (PDEs) in NWP describes for example the change in the thermodynamics of the atmosphere, the motion and the temperature ([KMS20], [BTB15]).
In this thesis, we are focusing on the linear advection-diffusion equation, which describes the physical phenomena which transfers particles or other physical quantities due to the processes of advection and diffusion within a physical system.
A model application of these processes is the transport of (ash) particles or gases released after a volcanic eruption ([TCF11], [HHG15], [HRG16]).
The PDEs in NWP are mainly derived from several conservation laws like the conservation of mass or momentum.
The atmospherics variables, whose values and partial derivatives described by the PDEs, are considered at fixed (geographical) points and change over time. Generally, these variables are highly complex, which makes it unfeasible or even impossible to derive an analytical solution. Hence, an approximation with the help of numerical schemes is sought-after ([KMS20]).

Many common numerical methods employ a mesh or a grid which discretizes the physical or geographical domain into fixed points in space at which the underlying PDEs are evaluated in order to define an approximation of the solution. Popular choices for methods which rely on an underlying mesh are finite difference methods, finite volume methods, finite element methods, and discontinuous Galerkin methods ([HRX19], [L02], [CDG17]).
The mesh consists of elements, most commonly of triangles, quadrilaterals, tetrahedrons, and hexahedrons, whereas triangles and tetrahedrons are generally better suited to accurately model complex physical domains ([S15]). On the other hand, using quadrilaterals and hexahedrons enables us to exploit several simplifications due to their tensor product structure ([BL89]).

In contrast to an uniform mesh, an adaptive mesh enables us to focus on specific parts of the mesh (which are of interest, for example regions with a high computational error, shocks, or complex geometries) with a fine mesh, and simulate with a coarser mesh in regions where we are able to "afford" a coarser resolution without suffering a loss of accuracy of the simulation. Generally, adaptive meshes may reach the same accuracy of a simulation as a uniform mesh, with fewer elements than the uniform mesh ([GBG05]).

If dynamic simulations, with moving phenomena are considered, like simulating the dispersion of an ash cloud in the atmosphere after a volcanic eruption, the mesh may have to be (re-)adapted at every single time step in order to capture this local event of the eruption. Therefore, if a mesh, representing the whole earth's atmosphere, is considered and we want to simulate the ash cloud over a given time period, adaptive mesh refinement (AMR) suggests itself as an efficient approach to continuously capture this relatively local event with a high resolution near the cloud and with a coarser mesh in other areas "far away" from the event.
In contrast to uniform mesh refinement, in which all elements of the mesh are refined to the same "refinement level", which results in a structured mesh, AMR refines and coarsens certain parts of mesh based on a given refinement criterion, this may lead to unstructured and eventually even non-conforming meshes ([H18]).

However, even if the solver for the problem is capable of calculating on an adaptive mesh, it is not trivial to efficiently adapt the underlying mesh of the simulation frequently.
In order to incorporate adaptive meshes in the simulation, different approaches to AMR have been made, like block-structured AMR [BO84] and unstructured AMR [O98]. Another efficient approach is *tree-based adaptive mesh refinement*.
Tree-based mesh refinement utilizes a *coarse mesh* which adequately resembles the geometry of the underlying domain, on which we want to simulate a given problem.
Each of these coarse elements within the coarse mesh is considered as the *root element* of a *refinement tree*.
Subject to a predefined refinement rule, each of these elements is refined recursively. This defines a refinement hierarchy and a parent-child-relation of the elements, since the parent (coarser) element is replaced by its children (finer) elements which cover the same area of the mesh as their parent element ([H18]).
The coarse elements and, therefore, the refinement trees, may have different refinement rules, because they do not need to necessarily have the same geometrical shape, i.e. the coarse mesh may consist of hexahedrons and tetrahedrons.
We call a collection of these refinement trees a *forest*.

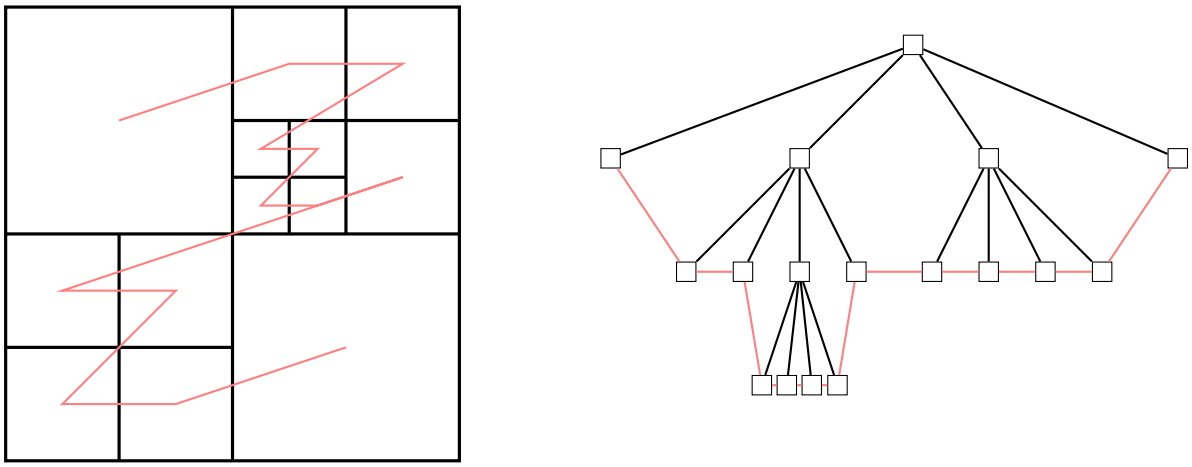


Figure 1.1: (Left:) A refined coarse mesh element; element ordering by SFC (red). (Right:) Corresponding refinement tree; SFC connects the (refined) elements which are currently present in the tree (they are called *leaves*).

The parallel adaptive mesh refinement library `t8code` ([t8code], [H18]) implements tree-based mesh refinement using space-filling curves (SFC). `t8code` extends `p4est` ([p4est]), which focuses on quadrilaterals in 2D and hexahedrons in 3D, by further geometrical element shapes and scales up to several $100,000$ MPI ranks while sustaining excellent parallel efficiency in terms of weak and strong scaling ([HKB21]).

The space-filling curve defines a linear order of the elements within a refinement tree by an indexing scheme.
It has to be distinguished between analytical and numerical space-filling curves. An analytical SFC generally represents a continuous mapping $sfc : I \mapsto \mathbb{R}^n$ from a compact set (often, a unit

interval) into an n-dimensional volume, whereas a numerical SFC maps from an finite index set, which resembles an analytical SFC discretely ([H18]).
Starting from a root element in a refinement tree, the numerical SFC corresponds to a discrete SFC on this element which is uniformly refined to a maximum refinement level.
Based on this SFC describing a (computational) maximum uniform refinement, the elements are ordered by a so called *SFC-index*.

All elements which are currently present in an adaptive mesh are called *leaves* of a refinement tree. Accordingly, due to `t8code`'s implementation, replacing a parent element (leaf) with its child elements changes the SFC-index only locally.
The implementation of space-filling curves provides an effective way to store and access the leaves of a refinement tree. Moreover, this enables us to easily partition the grid over several parallel processes.
A "cheap" computational way to partition the mesh is of great importance, especially in an AMR context. Because after an adaption step, the partitioned mesh may not be load-balanced across processes anymore, due to local refinement and coarsening in the mesh. But maintaining load-balance is important in order to achieve good parallel scaling behavior.
Repartitioning the mesh with the help of space-filling curves reduces this NP-hard problem to an approximation within linear runtime ([H18]). Therefore, if repartitioning steps have to be performed frequently, SFC partitioning is a common choice (e.g. [GZ99]).

`t8code` implements space-filling curves for lines, triangles, quadrilaterals, tetrahedrons, hexahedrons, prisms, and pyramids ([BH16], [K17], [K20]).
Therefore, hybrid meshes consisting of these element types are supported by `t8code`.
However, in this thesis we focus on quadrilaterals and hexahedrons as elements in order to make use of the tensor-product structure of these elements.

`t8dg` ([t8dg], [D21]) is a solver for the linear advection diffusion equation built on top of t8code and utilizing a local discontinuous Galerkin discretization in a matrix-free fashion.
It is designed to calculate on adaptive meshes produced and managed by `t8code`.

Discontinuous Galerkin methods (DG) enjoy a great popularity in a broad range of scientific applications ([GW21]), e.g. compressible flows, meteorology, geophysics and many more ([BHG18], [GR08],[GWK16], [B00], [SWR21], [KBB21]).
In the first place, Reed and Hill used a discontinuous Galerkin type discretization in order to model neutron transport ([RH73]).
In several papers Cockburn and Shu extended the discontinuous Galerkin approach and applied it to non-linear hyperbolic conservation laws and introduced the Runge Kutta discontinuous Galerkin methods (RKDG) ([CHS90], [CW91], [CS98], [CS981]).
DG methods connect ideas from finite element methods as well as from finite volume methods. The approximation space consists of piecewise polynomials on each grid cell and the method allows (and in fact, incorporates) discontinuities across cell interfaces.
Naturally, DG methods implement high order discretizations, while maintaining a minimal local stencil, because only data at the interface between neighboring elements needs to be exchanged. Due to this locality, DG methods are highly parallelizeable ([HGA12]) and the elements are weakly coupled via numerical fluxes at the element interfaces like in finite volume methods.

For the time being, `t8dg` is equipped with explicit time-stepping methods, which complete the

spatial discretization by the means of a discontinuous Galerkin ansatz, in order to perform the time integration and simulate linear advection diffusion problems.
Explicit time-stepping methods demand a high restriction on the size of the time steps. Generally, these methods require very small time steps in order to produce stable approximations during the simulation. The restriction of the size of the time step is enforced by the Courant-Friedrichs-Lewy condition (*CFL* number). In exchange, the application and computational complexity per time step is rather cheap.

On the other hand, we have implicit time-stepping methods which do not restrict the choice of the size of the time steps. But this relaxation of constraints comes with a cost. Their application per time step demands a solution to a (non-linear) system of equations, in some cases even the solution to several implicit systems of equations ([L06]).
If large simulations are considered, these systems and the computational complexity per time step becomes huge. However, implicit time stepping methods may have advantages over explicit ones, because of the omission of strong restrictions on the time steps and if stiff problems are considered.
Generally, the additional cost which comes along with an implicit discretization pays off if the enforced time step by the CFL for an explicit method is "relatively small" in comparison to an "interesting/meaningful" physical time step, but this highly depends on the underlying physical problem.
The implicit systems arising from these time stepping methods can be preconditioned in order to enhance their solvability as well as robustness of the solver.

In this thesis, we examine different preconditioning techniques in order to enhance iterative solvers which solve linear systems of equations, arising from implicit time-stepping methods which were applied to a semi-discretization of the linear advection diffusion equation.
Within this work, we extended `t8dg` by a set of implicit time-stepping methods and preconditioners.
We coupled `t8dg` with the "Portable, Extensible Toolkit for Scientific Computation" (PETSc) ([B21]) in order to use its implementations of Krylov subspace methods (i.e. GMRES) for solving the systems of equations resulting from our discretization of the linear advection diffusion equation.
Regarding the preconditioners, we focus especially on geometric multigrid preconditioners.
Multigrid methods ([BHM00], [BL11]) represent a class of fast iterative numerical algorithms in order to solve huge sparse linear systems of equations. These methods are characterized by their asymptotically optimal complexity and have been applied on adaptive meshes for a long time ([HKM06]). The parallel implementation of geometric multigrid methods in combination with adaptive tree-based mesh refinement, on so called *quadtree-* (2D) and *octtree-* (3D) grids, have been realized, e.g. in [TK19], [RVH13]. The concept of space-filling curves enables and simplifies the parallelization of adaptive mulitgrid methods, i.e. managing the coarse grids in parallel ([GZ99]).
A geometric multigrid preconditioner in a matrix-free fashion has been efficiently applied on adaptive meshes in parallel which even yield more robust results and better weak scaling than algebraic multigrid preconditioners for a Stokes problem [CH21].

Therefore, geometric multigrid preconditioners suggest themselves as an interesting and efficient choice as a preconditioner, which we are going to examine further throughout this thesis.
Additionally, we analyze the performance of block-preconditioners and compare their effect and

performance in regards to the CPU-time and the reduction of iterations which the iterative solver needs in order solve the implicit systems.

In chapter 2 we focus on the local Discontinuous Galerkin method and derive a spatial discretization of the linear advection diffusion equation.

After we obtained the spatial discretization, we discuss the time integration by the means of implicit time-stepping methods in chapter 3. These methods enable us to simulate our problem of interest within a given time interval.

In chapter 4 we present different iterative solvers, we are focusing especially on Krylov subspace methods. With the help of iterative solvers we are able to solve the linear system(s) of equations which results from our discretization and obtain an approximation of the solution at a single time step.

The implicit systems arising are often large and sparse. This may lead to large amount of iterations needed by the iterative solver in order to solve the linear system of equations sufficiently. In chapter 5 we are focusing on preconditioning techniques, which may reduce the computational complexity of the linear system and enhance the iterative solver.

Afterwards, we connect all components and perform numerical experiments in chapter 6. We examine the effect of different preconditioners in all three spatial dimensions.

Finally, we summarize the results and provide an outlook on further opportunities and investigations in chapter 7.

2. Discontinuous Galerkin Method for the Linear Advection-Diffusion Equation

In the following chapter we will derive a semi-discretization of the linear advection-diffusion equation by the means of the local discontinuous Galerkin method (LDG). This means, we are aiming only at a spatial discretization by the LDG method which uses discontinuous finite elements. The derivation substantially follows [CS98], [G20] and [D21].

The LDG method is a generalization of the Runge Kutta discontinuous Galerkin method (RKDG). The RKDG method is designed for non-linear hyperbolic systems of the form

$$u_t + \nabla f(u) = 0, \text{ in } \Omega \times (t_0, T). \tag{2.0.1}$$

The idea of the LDG method, in order to discretize the linear advection-diffusion equation 2.0.2, is to rewrite the system into a larger, degenerate first order system, which is discretized by the means of the discontinuous Galerkin method (DG) afterwards.

Therefore, let us first consider a formulation of the linear advection-diffusion equation. We suppose a physical domain $\Omega \subset \mathbb{R}^d$, a divergence-free flow velocity $\vec{\mathbf{c}} : \Omega \mapsto \mathbb{R}^d$, a scalar diffusion coefficient $\mathbf{d} \in \mathbb{R}^{\geq 0}$, and (time-dependent) source- and sink-terms $g : \Omega \times [t_0\,, T) \mapsto \mathbb{R}$ (with $T > t_0 \in \mathbb{R}^{\geq 0}$) as given information.

We want to solve

$$u_t + \nabla\left(\vec{\mathbf{c}}u - \mathbf{d}\nabla u\right) = g, \quad \text{in } \Omega \times (t_0, T) \tag{2.0.2}$$

with periodic boundary conditions and an initial condition $u(t_0) \equiv u_{init} : \Omega \mapsto \mathbb{R}$ on our physical domain.
For example, the solution $u := u(\vec{x}, t)$ may represent the temperature or the concentration of a specific chemical (depending on the underlying transfer processes) at the point $\vec{x} \in \Omega$ in space at the time $t_0 \leq t \in \mathbb{R}$.

We need an underlying mesh, which resembles the physical domain Ω for our discontinuous Galerkin approach. Let $Q_\ell \subset \Omega$ denote a subset of our physical domain. Q_ℓ will resemble one element domain of the initial underlying grid Ω^h.

Definition 2.1. *Ω^h describes the geometrical division of the physical domain Ω into $n^{elem} \in \mathbb{N}$ non-overlapping elements $Q_\ell \subseteq \Omega$, for $\ell = 1, \ldots, n^{elem}$, such that $\overline{\Omega} = \cup_{\ell=1}^{n^{elem}} Q_\ell$*

An 1D example grid Ω^h, in which the physical domain Ω is divided into five elements $Q_\ell = \left[x_{\ell-1/2}, x_{\ell+1/2}\right]$ with cell midpoints x_ℓ, for $\ell = 1, \ldots, 5$, is shown in figure 2.1.

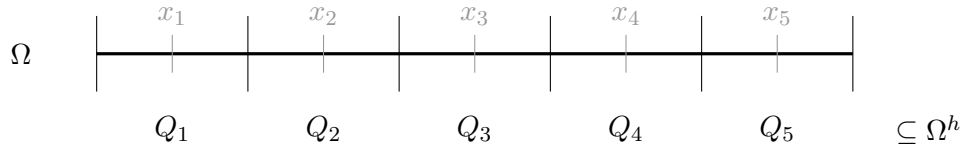


Figure 2.1: Division of the 1D physical domain Ω into the grid Ω^h, which consists of the elements Q_ℓ, $\ell = 1, \ldots, 5$ with cell midpoints x_ℓ.

In the next step we describe the discretization of the problem by the means of the discontinuous Galerkin approach, therefore, we will reformulate the problem in the style of the LDG method.

Regarding the d-dimensional case, we introduce new scalar variables $q_i = \varepsilon \frac{\partial}{\partial x_i} u$ with $\varepsilon = \sqrt{\mathbf{d}}$ for $i = 1, \ldots, d$.
With the help of these variables that represent the gradient of u, we can rewrite 2.0.2 to the following system 2.0.3.

$$u_t + \sum_{i=1}^{d} \frac{\partial}{\partial x_i} \underbrace{\left(\mathbf{c}_i u - \varepsilon q_i\right)}_{=: f_i(u,q)} = g \tag{2.0.3a}$$

$$q_i - \varepsilon \frac{\partial}{\partial x_i} u = 0 \tag{2.0.3b}$$

Similar to the finite element method, the starting point of the DG method is the variational formulation of the problem.
For that reason, we consider the test functions $\phi := \phi(\vec{x})$ and $\psi := \psi(\vec{x})$ from a trial space (which is defined later), for now, we assume that these test functions are arbitrarily differentiable. Let us consider one element of the mesh Ω^h, say $Q_\ell \left(\in \Omega^h\right)$. In order to obtain the variational formulation of our problem 2.0.3, we multiply the partial differential equation (PDE) with the

arbitrary test function ϕ (and the second equation of the system 2.0.3 with ψ) and integrate over the element domain of Q_ℓ.

$$\int_{Q_\ell} u_t \phi \, d\vec{x} + \sum_{i=1}^{d} \int_{Q_\ell} \frac{\partial}{\partial x_i} f_i(u,q) \, \phi \, \vec{x} \quad = \quad \int_{Q_\ell} g\phi \, d\vec{x} \tag{2.0.4a}$$

$$\int_{Q_\ell} q_i \psi \, d\vec{x} - \int_{Q_\ell} \varepsilon \frac{\partial}{\partial x_i} \psi \, d\vec{x} \quad = \quad 0, \; i = 1, \ldots, d \tag{2.0.4b}$$

Within the formulation 2.0.4 the divergence of $f(u,q)$ is required. This is considered as a rather strong formulation.
We want to receive a weaker formulation of the problem, which will be obtained if we formally apply a spatial integration by parts and, in a manner of speaking, "move the derivation over to the test functions".
The integration by parts introduces surface integrals and the weak formulation corresponds to

$$\int_{Q_\ell} u_t \phi \, d\vec{x} + \oint_{\partial Q_\ell} \vec{f}(u,q) \circ \vec{n} \, \phi \, ds - \sum_{i=1}^{d} \int_{Q_\ell} \frac{\partial}{\partial x_i} f_i(u,q) \, \nabla\phi \, d\vec{x} \quad = \quad \int_{Q_\ell} g\phi \, d\vec{x} \tag{2.0.5a}$$

$$\int_{Q_\ell} q_i \psi \, d\vec{x} - \oint_{\partial Q_\ell} (\varepsilon u) \, n_i \, \psi \, ds + \int_{Q_\ell} (\varepsilon u) \, \nabla\psi d\vec{x} \quad = \quad 0, \; i = 1, \ldots, d \tag{2.0.5b}$$

in which $\vec{n}$ denotes the outward unit normal (and n_i the i-th component of it) regarding the element Q_ℓ at $\vec{x} \in \partial Q_\ell$ the element boundary.

The base of the DG method is the weak formulation of the considered problem which is regarded element-wise. As described above in definition 2.1, we divide the physical domain Ω into non-overlapping elements Q_ℓ, for $\ell = 1, \ldots, n^{elem}$ and obtain our underlying mesh Ω^h. In order to approximate the solution on each element Q_ℓ, we have to define an approximation space, out of which the approximation $u^{Q_\ell}(\vec{x}, t)$ for each element is drawn.
We choose polynomials of degree p as the approximation on each element. The ansatz reads

$$u(\vec{x}, t)\big|_{Q_\ell} \approx u^{Q_\ell}(\vec{x}, t) := \sum_{j=1}^{N(p,d)} u_j^{Q_\ell}(t) \, \varphi_j^{Q_\ell}(\vec{x}) \tag{2.0.6}$$

in which $\left\{u_j^{Q_\ell}(t)\right\}_{j=1,\ldots,N}$ with $N := N(p,d)$ represents the unknown time-dependent coefficients (which are also referred to as the *degrees of freedom*). The approximation is defined as the linear combination of the degrees of freedom and the basis functions $\left\{\varphi_j^{Q_\ell}\right\}_{j=1,\ldots,N}$, which are in $\mathbb{P}_p(Q_\ell)$ (the space of polynomials of degree at most p over the domain of the element).
The amount of degrees of freedom on a element does not only relate to the degree of the basis functions of the approximation but also on the dimension and on the shape of the element (e.g. triangle, square, etc.).
Therefore, the approximation space corresponds to piece-wise polynomials. We define the approximation space as

$$V^h := \left\{ u \in L^2(\Omega) \; : \; u\big|_{Q_\ell} \in \mathbb{P}_k(Q_\ell) \;\; \forall \ell = 1, \ldots, n^{elem} \right\}. \tag{2.0.7}$$

In the same manner, we approximate the variable q and the source- and sink-terms g (2.0.10).

This choice guarantees a smooth approximation within each element. In contrast to the finite element method, DG methods generally result in discontinuities (or jumps) at the faces of adjacent elements.
An example of a discontinuity at the element interface of the approximations on adjacent elements is shown in figure 2.2.

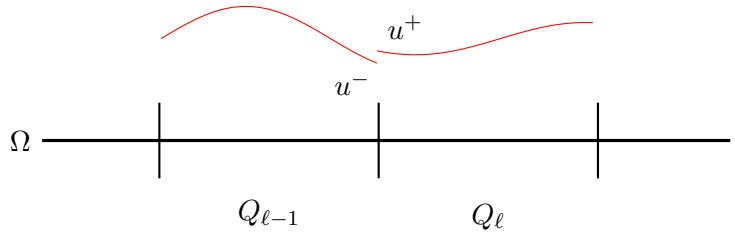


Figure 2.2: Discontinuity at the element interface between approximations on $Q_{\ell-1},\ Q_\ell \in \Omega^h$. u^- denotes the value of the approximation on $Q_{\ell-1}$ at the element interface, and u^+ the value of the approximation of Q_ℓ, respectively.

This is problematic, because of the discontinuities at the element interfaces, the surface integrals in our weak formulation 2.0.5 are not uniquely defined.
In order to remedy for this, we introduce a tool which is also used in finite volume methods - the numerical flux. The numerical flux only depends on the values at both sides of the discontinuity (at the element interface). An approximation at the interface of the non-linear flux is obtained if we regard these different values as initial values to a Riemann problem which we have to solve (approximately), for example with Godunov's scheme. Let f^* (and $\tilde{f}^*$) denote the approximation of the solution of the Riemann problem. The approximation of the non-linear flux at the element interface is then given by

$$\begin{aligned} \vec{f}(u,q)\circ\vec{n} \quad &= \quad (\vec{\mathbf{c}}u - \varepsilon\vec{q})\circ\vec{n}\,\big|_{\partial Q_\ell} \quad \approx \quad \left(f^*\left(u^-,u^+\right) - \tilde{f}^*\left(q^-,q^+\right)\right)\circ\vec{n} \\ &\qquad\qquad\qquad\qquad\quad =: \quad f^*\left(u^-,u^+,q^-,q^+\right)\circ\vec{n} \qquad (2.0.8a) \\ &\qquad (\varepsilon u)\,n_i\,\big|_{\partial Q_\ell} \quad \approx \quad \tilde{f}^*\left(u^-,u^+\right)n_i,\ i=1,\dots,d \qquad (2.0.8b) \end{aligned}$$

The numerical flux is defined at the interface of neighboring grid cells and only depends on the numerical approximation on both sides of the interface (e.g. u^- and u^+ in figure 2.2).
If conservation laws are considered, we should choose a *monotone flux* as the numerical flux ([C09]).

Definition 2.2. *A monotone flux $f^*\left(u^-,u^+\right)$ satisfies the conditions:*

- *It is consistent, $f^*(u,u) = \vec{f}(u)$.*
- *It is locally Lipschitz continuous in both arguments u^-, u^+.*

- *It is monotone, i.e.* $f^*\left(u^-, u^+\right)$ *is a function which is non-decreasing in the first variable and non-increasing in the second.*

A commonly used monotone flux is the *Lax-Friedrich flux*

$$f^*_{LF}\left(u^-, u^+\right) = \frac{1}{2}\left(f\left(u^-\right) + f\left(u^+\right)\right) - \frac{C}{2}\left(u^+ - u^-\right) \tag{2.0.9}$$

in which C denotes an upper bound on the global flow speed of the system (commonly, $C = \frac{\Delta x}{\Delta t}$). The implementation of the Lax-Friedrich flux is simple and the evaluation is quickly calculated. The flux is (rather strong) dissipative, which prevents oscillations near the discontinuities, but also "smears" the discontinuities rather strongly, hence, a finer grid is needed in order to achieve a given accuracy in comparison to Godunov's scheme ([G20]).
The literature provides many methods which may be used alternatively as the numerical flux, for a broard overview of approximate Riemann solvers see [T09].
Regarding the formulation 2.0.8, for the advective flux $f^*\left(u^-, u^+\right)$ the Lax-Friedrich flux represents a good choice.

For the diffusive fluxes ($\tilde{f}^*\left(q^-, q^+\right)$ and $\tilde{f}^*\left(u^-, u^+\right)$) may an upwinding principle, when choosing the fluxes, not be sufficient. An *alternating flux* represents an "attractive" choice for the diffusion fluxes ([C09]). For this approach, it is important to choose the diffusive flux values (for u and q) from different directions/sides at the interface, for example $\tilde{f}^*\left(q^-, q^+\right) = q^+$ and $\tilde{f}^*\left(u^-, u^+\right) = u^-$.
We form the approximation of the remaining variables in the same manner as in 2.0.6. For each element Q_ℓ, $\ell = 1, \ldots, n^{elem}$, we interpolate the variables onto the approximation space V^h, resulting in

$$q_i\left(u\left(\vec{x}, t\right)\right)\big|_{Q_\ell} \approx q_i^{Q_\ell}\left(\vec{x}, t\right) := \sum_{j=1}^{N} q_{i_j}^{Q_\ell}\left(t\right)\varphi_j^{Q_\ell}\left(\vec{x}\right),\ i = 1, \ldots, d \tag{2.0.10a}$$

$$g\left(\vec{x}, t\right)\big|_{Q_\ell} \approx g^{Q_\ell}\left(\vec{x}, t\right) := \sum_{j=1}^{N} g_j^{Q_\ell}\left(t\right)\varphi_j^{Q_\ell}\left(\vec{x}\right). \tag{2.0.10b}$$

This notation implies that each element needs its own basis functions $\left\{\varphi_j^{Q_\ell}\right\}_{j=1,\ldots,N}$ for $\ell = 1, \ldots, n^{elem}$ and in fact, it is true, because each element represents a different and unique part of the physical domain. However, if we define a reference element onto which each element of the grid Ω^h is mapped, the basis functions and their operations only have to be defined once for the reference element. Although, each element needs a mapping between its physical element domain and the reference element domain, this approach is advantageous, because it boosts the efficiency of the implementation and saves a lot of storage requirements.

Due to the Galerkin formulation, we choose the same polynomial basis functions of the ansatz (2.0.6) as test functions in our weak formulation (2.0.5). Therefore, it holds $\phi,\ \psi \in \left\{\varphi_j^{Q_\ell}\right\}_{j=1,\ldots,N}$.
So, we obtain N test functions and consequently N equations in order to calculate N polynomial coefficients for each element $Q_\ell \in \Omega^h$, for $\ell = 1, \ldots, N$, (for the approximations u^{Q_ℓ} and $q_i^{Q_\ell}$, respectively). Because of this, the approximation space equals the trial space V^h (2.0.7).

Incorporating the above formulations leads to the discretized system

$$\int_{Q_\ell} u_t^{Q_\ell}\varphi_j^{Q_\ell}\,d\vec{x} + \oint_{\partial Q_\ell} f^*\left(u^-,u^+,q^-,q^+\right)\vec{n}\varphi_j^{Q_\ell}\,ds$$
$$-\sum_{i=1}^{d}\int_{Q_\ell}\left(\mathbf{c}_i u^{Q_\ell}-\varepsilon q_i^{Q_\ell}\right)\frac{\partial}{\partial x_i}\varphi_j^{Q_\ell}\,dx \quad = \quad \int_{Q_\ell} g^{Q_\ell}\varphi_j\,d\vec{x} \tag{2.0.11a}$$

$$\int_{Q_\ell} q_i^{Q_\ell}\varphi_j^{Q_\ell}\,d\vec{x} - \oint_{\partial Q_\ell}\tilde{f}^*\left(u^-,u^+\right)n_i\varphi_j^{Q_\ell}\,ds$$
$$+\int_{Q_\ell}(\varepsilon u)\frac{\partial}{\partial x_i}\varphi_j^{Q_\ell}\,d\vec{x} \quad = \quad 0,\ i=1,\dots,d \tag{2.0.11b}$$

for $j=1,\dots,N$ and all elements Q_ℓ with $\ell=1,\dots,n^{elem}$.

If we rearrange the terms and write out the approximation of the variables in full, the equations comply with 2.0.12 (from now on, instead of using the subscript $\square_t$ as the time-derivative, it is represented by $\dot{\square}$).

$$\sum_{k=1}^{N}\dot{u}_k^{Q_\ell}\int_{Q_\ell}\varphi_k^{Q_\ell}\varphi_j^{Q_\ell}\,dx \quad = \quad -\oint_{\partial Q_\ell} f^*\left(u^-,u^+,q^-,q^+\right)\vec{n}\varphi_j^{Q_\ell}\,ds$$
$$+\sum_{i=1}^{d}\sum_{k=1}^{N}\left(c_i u_k^{Q_\ell}-\varepsilon q_{i_k}^{Q_\ell}\right)\int_{Q_\ell}\varphi_k^{Q_\ell}\frac{\partial}{\partial x_i}\varphi_j^{Q_\ell}\,dx+\sum_{k=1}^{N}g_k^{Q_\ell}\int_{Q_\ell}\varphi_k^{Q_\ell}\varphi_j^{Q_\ell}\,dx \tag{2.0.12a}$$

$$\sum_{k=1}^{N}q_{i_k}^{Q_\ell}\int_{Q_\ell}\varphi_k^{Q_\ell}\varphi_j^{Q_\ell}\,dx \quad = \quad +\oint_{\partial Q_\ell}\tilde{f}^*\left(u^-,u^+\right)n_i\varphi_j^{Q_\ell}\,ds$$
$$-\sum_{k=1}^{N}\left(\varepsilon u_k^{Q_\ell}\right)\int_{Q_\ell}\varphi_k^{Q_\ell}\frac{\partial}{\partial x_i}\varphi_j^{Q_\ell}\,dx,\quad i=1,\dots,d \tag{2.0.12b}$$

for $j=1,\dots,N$ and all elements Q_ℓ with $\ell=1,\dots,n^{elem}$.

In order to computationally solve for the unknown time dependent coefficients in our approximations, u^{Q_ℓ} and $q_i^{Q_\ell}$, for $i=1,\dots,d$, we formulate an element-wise matrix system of 2.0.12. Therefore, we are introducing the following definitions.

Definition 2.3. *Let Q_ℓ be an element of Ω^h equipped with its basis functions $\left\{\varphi_j^{Q_\ell}\right\}_{j=1,\dots,N}$. We define the **element mass matrix** $M^{Q_\ell}\in\mathbb{R}^{N\times N}$ by its entries as*

$$M_{i,j}^{Q_\ell}=\int_{Q_\ell}\varphi_i^{Q_\ell}\varphi_j^{Q_\ell}\,d\vec{x}. \tag{2.0.13}$$

Definition 2.4. *Furthermore, we define the **element stiffness matrix** $A^{Q_\ell}\in\mathbb{R}^{N\times N}$ for an element $Q_\ell\in\Omega^h$ and its local basis functions in the one dimensional case by*

$$A_{i,j}^{Q_\ell}=\int_{Q_\ell}\varphi_i^{Q_\ell}\frac{\partial}{\partial x}\varphi_j^{Q_\ell}\,dx. \tag{2.0.14}$$

If more than one dimension is considered, we obtain a ***directional stiffness matrix*** *for each dimension* $k = 1, \dots, d$

$$A_{i,j}^{k,Q_\ell} = \int_{Q_\ell} \varphi_i^{Q_\ell} \frac{\partial}{\partial x_k} \varphi_j^{Q_\ell} \, d\vec{x}. \tag{2.0.15}$$

Regarding the formulation 2.0.12, we split up the numerical flux within the surface integral into components of each face of the element $Q_\ell \in \Omega^h$. Therefore, we formulate the numerical flux as the sum over each face. Similarly, to the aforementioned approximation on the domain of the elements, we define the numerical flux as the linear combination of N_f face basis functions. For each face $f_i^{Q_\ell} \in \partial Q_\ell$, such that $\cup_i f_i^{Q_\ell} = \partial Q_\ell$, we receive

$$(\vec{\mathbf{c}}u - \varepsilon\vec{q}) \circ \vec{n}\,\big|_{f_i^{Q_\ell}} \quad \approx \quad f^*\left(u^-, u^+, q^-, q^+\right) \circ \vec{n} \quad \approx \quad \sum_{j=1}^{N_f} h_j^{f_i^{Q_\ell}} \, \varphi_j^{f_i^{Q_\ell}} \tag{2.0.16a}$$

$$(\varepsilon u)\, n_k\,\big|_{f_i^{Q_\ell}} \quad \approx \quad \tilde{f}^*\left(u^-, u^+\right) n_k \quad \approx \quad \sum_{j=1}^{N_f} \tilde{h}_{k,j}^{f_i^{Q_\ell}} \, \varphi_j^{f_i^{Q_\ell}}, \; k = 1, \dots, d \tag{2.0.16b}$$

By the means of this representation of the numerical flux, we can define additional matrices.

Definition 2.5. *The* ***face mass matrix*** $M^{f_i} \in \mathbb{R}^{N_f \times N_f}$ *of a face* $f_i := f_i^{Q_\ell}$ *of the element* Q_ℓ *with the face basis functions* $\left\{\varphi_j^{f_i}\right\}_{j=1,\dots,N_f}$ *is given by*

$$M_{k,j}^{f_i} = \int_{f_i} \varphi_k^{f_i} \varphi_j^{f_i} \, d\vec{x}. \tag{2.0.17}$$

In order to calculate the surface integrals which also depend on the test functions and, therefore, on the element basis functions $\left\{\varphi_j^{Q_\ell}\right\}_{j=1,\dots,N}$, we still need a connection between face and element basis functions.

Definition 2.6. *The* ***face interpolation matrix*** $\mathcal{R}^{f_i} \in \mathbb{R}^{N \times N_f}$ *restricts an element basis function* $\varphi_j^{Q_\ell}$ *onto a face* $f_i := f_i^{Q_\ell}$ *of the element* $Q_\ell \in \Omega^h$*. The coefficients of the linear combination of face basis functions, in order to represent an element basis function on this face, are collected in the* j^{th}*-row of* $\mathcal{R}^{f_i}$*. Therefore, the face interpolation matrix is defined by*

$$\varphi_j^{Q_\ell}\big|_{f_i} = \sum_{k=1}^{N_f} \mathcal{R}_{j,k}^{f_i} \, \varphi_k^{f_i}, \quad j = 1, \dots, N. \tag{2.0.18}$$

With the help of the aforementioned definitions of matrices, we are able to formulate a matrix system on each element in order to solve for the time-dependent coefficients of our approximations. We define these element-wise coefficients as a vector $u_h^{Q_\ell} = \left(u_1^{Q_\ell}, \dots, u_N^{Q_\ell}\right)^t$ and $q_{k,h}^{Q_\ell} = \left(q_{k,1}^{Q_\ell}, \dots, q_{k,N}^{Q_\ell}\right)^t$, for $k = 1, \dots, d$. Analogously, we represent the coefficients of the linear combination of the face basis functions in order to approximate the numerical flux as vectors $h^{f_i^{Q_\ell}} = \left(h_1^{f_i^{Q_\ell}}, \dots, h_{N_f}^{f_i^{Q_\ell}}\right)^t$, $\tilde{h}_k^{f_i^{Q_\ell}} = \left(\tilde{h}_{k,1}^{f_i^{Q_\ell}}, \dots, \tilde{h}_{k,N_f}^{f_i^{Q_\ell}}\right)^t$, as well as the coefficients of the interpolated source- and sink-terms $g_h^{Q_\ell} = \left(g_1^{Q_\ell}, \dots, g_N^{Q_\ell}\right)^t$.

Now, we receive the following system of size $N \times N$ on each element Q_ℓ, for $\ell = 1, \ldots, n^{elem}$

$$\dot{u}_h^{Q_\ell} = \left(M^{Q_\ell}\right)^{-1} \cdot \left(\sum_{k=1}^{d} A^{k,Q_\ell} \left(c_k u_h^{Q_\ell} - \varepsilon q_{k,h}^{Q_\ell} \right) - \sum_{f_i \in \partial Q_\ell} \left(\mathcal{R}^{f_i}\right)^t M^{f_i} h^{f_i} + M^{Q_\ell} g_h^{Q_\ell} \right) \tag{2.0.19a}$$

$$q_{k,h}^{Q_\ell} = \left(M^{Q_\ell}\right)^{-1} \cdot \left(A^{k,Q_\ell} \left(\varepsilon u_h^{Q_\ell} \right) + \sum_{f_i \in \partial Q_\ell} \left(\mathcal{R}^{f_i}\right)^t M^{f_i} \tilde{h}_k^{f_i} \right), \quad k = 1, \ldots, d \tag{2.0.19b}$$

If we collect the coefficients of all elements in global vectors, e.g. $U_h \in \mathbb{R}^{\left(N \cdot n^{elem}\right) \times \left(N \cdot n^{elem}\right)}$ with

$$U_h = \begin{pmatrix} u_1^{Q_\ell} \\ u_2^{Q_\ell} \\ \vdots \\ u_{n^{elem}}^{Q_\ell} \end{pmatrix}, \tag{2.0.20}$$

based on the element ordering within the grid Ω^h, we can formulate a global system for the time-dependent coefficients

$$\dot{U}_h = R\left(U_h\left(t\right), t\right) \tag{2.0.21a}$$

$$Q_{h,k} = \tilde{R}_k\left(U_h\left(t\right), t\right), \quad k = 1, \ldots, d \tag{2.0.21b}$$

in which $R\left(U_h\left(t\right), t\right)$ and $\tilde{R}_k\left(U\left(t\right), t\right)$ correspond to the global operators of 2.0.19.
Since we are considering a linear problem, the DG operators are also linear, and can be formulated in terms of a (global) matrix $A \in \mathbb{R}^{\left(N \cdot n^{elem}\right) \times \left(N \cdot n^{elem}\right)}$, e.g.

$$R\left(U_h\left(t\right), t\right) = A \cdot U_h\left(t\right). \tag{2.0.22}$$

We obtain a first order system, which is solved for the gradient within the linear advection-diffusion equation first, and used afterwards to calculate the approximation of the time derivative $\dot{U}_h$.

After we have obtained the underlying spatial discretization, we are able to apply a time-stepping scheme 3, in order to advance the approximation in time and simulate our problem in a time interval of interest $[t_0, T]$, starting from a given spatial initial function u_{init}, which describes the initial state of our problem at t_0.

But there are still some unanswered questions, which we need to address. For example, which and how many basis functions should be chosen or which numerical quadrature should be used.

As mentioned before, it is convenient to consider a reference element Q_{ref} (for each element geometry/shape within the grid), and for each element in the physical grid Ω^h a mapping from the reference element to the physical domain. An example of an one and two dimensional reference element is shown in figure 2.3.
This enables us to define the basis functions and matrices once for this reference element, instead

of defining different basis functions on each physical element $Q_\ell \in \Omega^h$, $\ell = 1, \dots n^{elem}$.

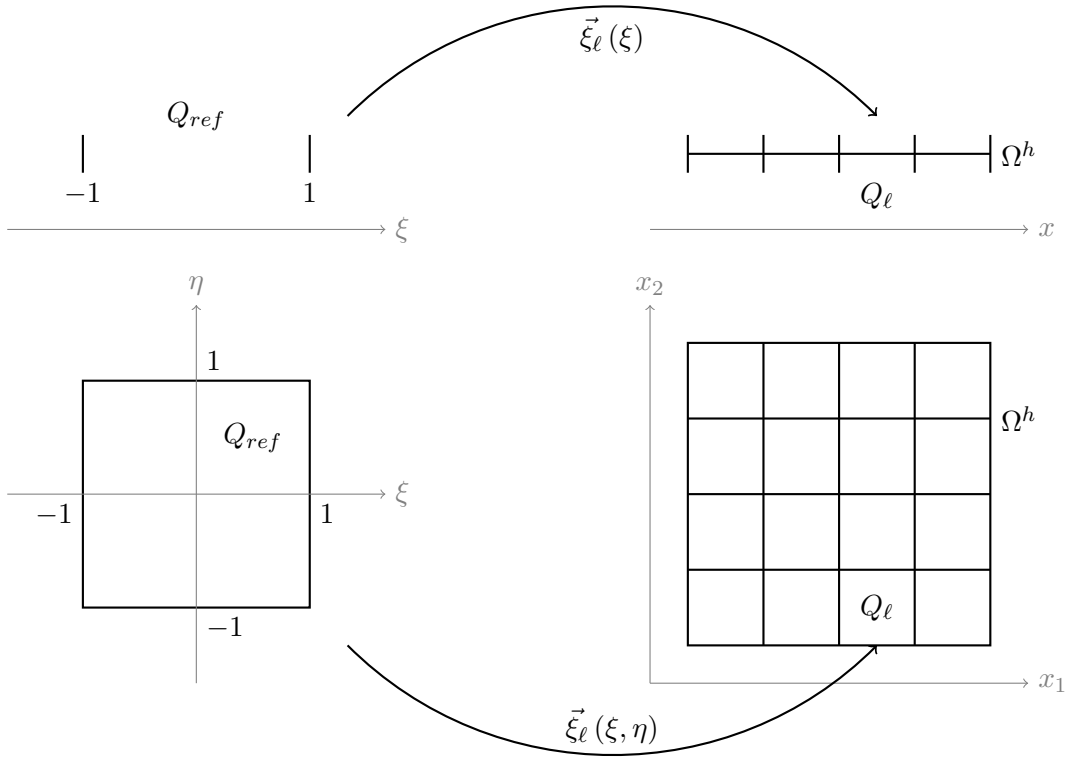


Figure 2.3: (Top): 1D reference element Q_{ref} on a reference coordinate system and the mapping $\vec{\xi}_\ell : Q_{ref} \mapsto Q_\ell$ from this reference element to the physical element $Q_\ell \in \Omega_h$.
(Bottom): Analogously, a 2D example of a reference element and its mapping onto the element $Q_\ell \in \Omega_h$

Another important aspect is the numerical quadrature, which is used to calculate the integrals numerically. Because we are calculating on a reference element, we have to transform the partial differential equations into the reference space first and then define the numerical quadrature as well as the basis functions of our approximation in the reference space.
For a detailed description of the application and the implementation of the transformation, we refer to ([G20], [D21]).

Numerical integration is based on a set of nodes $\{\xi_j\}_{j=0,\dots,N}$ within an (specific) interval and a set of weights $\{\omega_j\}_{j=0,\dots,N}$. In order to approximate an integral of an arbitrary function $p(\xi)$, we sum up

$$\int_{-1}^{1} p(\xi)\, d\xi \approx \sum_{j=0}^{N} \omega_j p(\xi_j)\,. \tag{2.0.23}$$

A convenient choice, which we will consider too, is the Legendre-Gauss-Lobatto quadrature (LGL). It is based on the Gauß-Lobatto points lying within the interval $[-1, 1]$.

Definition 2.7. *These nodes* $\{\xi_j\}_{j=0,\dots,N}$ *are calculated as the zeros of the first derivation of the* N^{th} *Legendre polynomial:* $P'_N(\xi)$.
The following three term recurrence holds for the Legendre polynomials,

$$P_{j+1}(\xi) = \frac{2j+1}{j+1}\xi P_j(\xi) - \frac{j}{j+1}P_{j-1}(\xi) \tag{2.0.24}$$

with $P_0(\xi) = 1$ *and* $P_1(\xi) = \xi$.
The derivation of the Legendre polynomials satisfies

$$(2j+1)P_j(\xi) = P'_{j+1}(\xi) - P'_{j-1}(\xi). \tag{2.0.25}$$

The corresponding weights $\{\omega_j\}_{j=0,\dots,N}$ *are defined by*

$$\omega_j = \frac{2}{j(j+1)P_N(\xi_j)^2}. \tag{2.0.26}$$

Lemma 2.8. *The LGL quadrature with* N *nodes computes polynomials of degree at most* $2N-3$ *exactly.*

In the figure 2.4 we see the distribution of seven $N = 7$ LGL nodes.

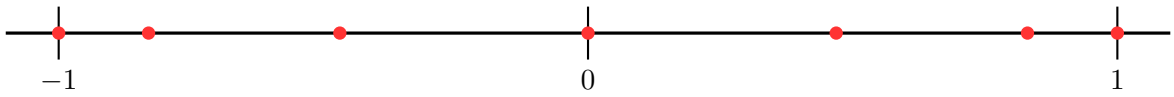


Figure 2.4: Distribution of seven LGL points (red) lying within the 1D interval $[-1, 1]$

If we consider more than one dimension, we are able to use a tensor product structure of the 1D LGL points.
As suggested in the figure 2.4, the LGL points are symmetrically distributed around the origin and become more dense towards the boundary of the interval $[-1, 1]$ and, especially, the boundary points are also included. This is useful, because this facilitates the computation of the numerical fluxes, which live on the boundaries of the elements. Otherwise, if the boundary points are not included, the approximation on the elements has be extrapolated to the boundary first, in order to calculate the numerical fluxes.

The way the basis functions $\left\{\varphi_j^{Q_\ell}\right\}_{j=1,\dots,N}$ are chosen is important. We want an easy to invert element mass matrix M^{Q_ℓ}, because the inverse of it is needed in order to extract the time dependent coefficients of the approximation, such that we can apply a time-stepping scheme.

There are mainly to ways to achieve this. Either we choose a *modal* or a *nodal* basis ansatz. If we choose Lagrange basis functions, we need a set of N *nodes* on our reference element $\{\xi_j\}_{j=1,\dots N}$ in order to define N Lagrange basis functions (hereafter, denoted as $l_i(\xi)$ for $i = 1, \dots, N$. Regarding the Lagrange basis functions and the underlying set of nodes which defines these functions, a Kronecker delta property is satisfied,

$$l_i(\xi_j) = \delta_{ij} = \begin{cases} 1, & \text{if } i = j \\ 0, & \text{if } i \neq j \end{cases}. \tag{2.0.27}$$

Therefore, the coefficients of the approximation (degrees of freedom) are immediately equal to the values of the approximation at these exact nodes.
This approach with Lagrange basis functions is called *nodal ansatz*.
If we use the LGL points as collocation points, thus, choosing these points as the interpolation nodes for the approximation and as integration nodes in our numerical quadrature, we obtain a diagonal element mass matrix M^{Q_ℓ}, due to *mass lumping*, which occurs because of the Kronecker delta property of the Lagrange basis functions.
We also note that the numerical integration during the computation of the element mass matrix is not exactly calculated by the LGL quadrature, because the integrand $l_i(\xi) \cdot l_j(\xi)$, consisting of the product of Lagrange polynomials, is a polynomial of degree $2N-2$ by construction (because of N interpolation nodes), but the LGL quadrature only calculates integrals of degree at most $2N-3$ (2.8) exactly (for N LGL points). However, this is reasonable considering the efficiency, which comes along by this choice.
Another approach in order to obtain an easily invertible mass matrix, is a *modal ansatz*, in which the basis functions are chosen to be mutually orthogonal and leading therefore, to a diagonal element mass matrix.

Now, after we discussed the semi-discretization, i.e. spatial discretization, of the linear advection-diffusion equation 2.0.2 which leads us to a linear system of equations, which we need to solve for the time-derivation of the coefficients of our approximation. We perform the time integration, starting from the initial value function u_{init} at time t_0, and simulate along the time axis until we reach the end time T. In order to advance the simulation, we calculate the next consecutive point in time with a time-stepping scheme.

3. Time-Stepping Methods

Let us consider an initial value problem

$$\dot{u} = f(t, u), \quad u(t_0) = u_{init}, \tag{3.0.28}$$

e.g. resulting from the semi-discretization of the linear advection-diffusion equation. Our goal is to find an approximate solution $u : [t_0, T] \mapsto \mathbb{R}^n$ in a time span of interest $t \in [t_0, T]$ (for $t_0 < T$, $n \in \mathbb{N}$). The right-hand-side $f(t, u)$ is a function $f : [t_0, T] \times \mathbb{R}^n \mapsto \mathbb{R}^n$ describing the (time) derivation of our wanted solution $u \in \mathbb{R}^n$, which corresponds to the degrees of freedom in our approximation (2.0.19) .

In order to obtain an approximate solution, we may apply an explicit or implicit time-stepping method, but even hybrid techniques are possible. Either of these methods has its advantages as well as its drawbacks.
Generally speaking, explicit time-stepping methods calculate the approximate solution (of the next time step) only based on the approximations of previous time steps. Therefore, they are rather inexpensive per time step regarding computational complexity and they are easier to implement than implicit methods. The main issue, however, is that the size of the time step, in order to advance the simulation further in time, is constrained by the *Courant-Friedirchs-Levy condition* (CFL) resulting in rather small time steps. If the CFL condition is rejected and a bigger time step is chosen nevertheless, the approximate solution might become numerical unstable, yielding to non-physical "solutions".

On the other hand, (certain) implicit time stepping methods overcome this obstacle, therefore, they are not demanding any constraints regarding the size of the time step. In exchange, calculating the approximate solution (of the next point in time) is much more expensive, because of the fact, that the approximation already relies on the future approximations.
Therefore, the approximation of the next time step is given implicitly. In order to obtain the solution, we have to solve a system of equations, in which these implicit methods result. Solving such a system, especially if it is large, is usually complex and costly.
Therefore, the use cases for these methods may vary. If for example a steady-state solution is sought-after, we may prefer large time steps in order to advance the simulation faster towards the steady-state, ergo an implicit time-stepping method may be superior to an explicit one. On the other hand, if the time interval of interest already has a high resolution (eventually because of other physical constraints), an explicit time-stepping method may be more suited.

3.1. Implicit Euler Method

The most basic implicit scheme of order one is the implicit Euler method (also referred to as the *backward Euler method*).
Let us consider a discretization of the time interval $[t_0, T]$ into discrete points in time t_k, $k = 0, \ldots, N$ with $t_{N+1} = T$ and a (not necessarily equidistant) time step of size $(\Delta t)_k = t_{k+1} - t_k$ between two consecutive points in time.

Let u_k denote the approximate values of $u(t_k)$, ergo, u_k corresponds to the current approximation of the solution at the time step t_k. We assume, that we do know the approximation u_k, which is given in the first place at time t_0 by the initial value function u_{init}.
In order to advance the approximation in time, we have to calculate the approximation u_{k+1} at the next point in time (t_{k+1}).

The implicit Euler method is described by the equation ([H09], p. 73)

$$u_{k+1} = u_k + (\Delta t)_k f(t_{k+1}, u_{k+1}). \tag{3.1.29}$$

We notice that the values u_{k+1} of the next point in time are given implicitly, because in the equation 3.1.29 the right-hand-side, describing u_{k+1}, depends on these exact same values of the next point in time within the evaluation of $f(t_{k+1}, u_{k+1})$, which are unknown to us.
So, in order to solve the equation and advance a time step, we rearrange the formulation and obtain

$$u_{k+1} - (\Delta t)_k f(t_{k+1}, u_{k+1}) = u_k. \tag{3.1.30}$$

If we consider a linear operator $f(t_{k+1}, u_{k+1})$, we may introduce a matrix $\tilde{A}_{t_{k+1}}$ which is capable of representing this operator.
In this case, we are able to define the values of the next point in time implicitly by

$$\left(\mathcal{I}_{n\times n} - (\Delta t)_k \tilde{A}_{t_{k+1}}\right) u_{k+1} = u_k. \tag{3.1.31}$$

This system of equations is described by the matrix $\left(\mathcal{I}_{n\times n} - (\Delta t)_k \tilde{A}_{t_{k+1}}\right)$ and we need to solve for the unknowns u_{k+1}.

Lemma 3.1. *The implicit Euler method is a first-order accurate and L-stable (3.3) method ([KC16], p. 69).*

Let us define the following linear stability concepts of DIRK-type methods (3.3).
The implicit Euler method also accounts for a DIRK-type method, as we will see, when we define DIRK methods.

Linear Stability Speaking of Stability of *DIRK*-type methods, a scalar test equation is considered. Namely,

$$\dot{u} = \lambda u, \quad u(t_0) = u_{init}. \tag{3.1.32}$$

Furthermore, $R((\Delta t)\lambda)$ denotes the rational function resulting from the application of a diagonal implicit Runge-Kutta method to the test equation 3.1.32, so that

$$u_{k+1} = R((\Delta t)\lambda)\, u_k. \tag{3.1.33}$$

This function $R((\Delta t)\lambda)$ is also referred to as the stability function regarding a specific *DIRK* method. Let $z = (\Delta t)\lambda$.
Important stability concepts of these methods are ([KC16], p. 17f)

Definition 3.2. *The method (3.2) is called* **A-*stable*** *if* $|R(z)| \leq 1$ *for* $Re(z) \leq 0$.

Definition 3.3. *If the method is A-stable and, additionally, if* $\lim_{z\to-\infty} R(z) = 0$ *holds, it is called* **L-*stable***.

These are favorable stability properties, which we should consider when choosing an implicit time stepping method, especially, if stiff problems are concerned.
A-stability ensures us that this method is stable, no matter how large we choose the time step, and it suppresses the development of oscillations. Beyond that, *L*-stability helps us reducing and damping local oscillations in the approximation fast (which may occur, although *A*-stability is satisfied).

3.2. Runge-Kutta Methods

Generally, an s-stage Runge-Kutta method for an initial value problem (3.0.28) is given by ([L06], p. 2110)

$$u_{k+1} = u_k + (\Delta t)_k \sum_{i=1}^{s} b_i f(t_k + c_i(\Delta t)_k, u_{k,i}), \tag{3.2.34}$$

in which $u_{k,i}$ describes the approximation of $u(t_k + c_i(\Delta t)_k)$, $i = 1, \ldots, s$.

A Runge-Kutta method is defined by its coefficients and these coefficients are conveniently displayed in a Butcher-Tableau 3.2.36. In order to obtain the approximations $u_{k,i}$, we have to (eventually) solve a set of implicit equations.

$$u_{k,i} = u_k + (\Delta t)_k \sum_{j=1}^{s} a_{ij} f(t_k + c_i(\Delta t)_k, u_{k,j}), \quad i = 1, \ldots, s. \tag{3.2.35}$$

The Butcher-Tableau of a Runge-Kutta method is portrayed as

$$\begin{array}{c|ccc} c_1 & a_{11} & \cdots & a_{1s} \\ \vdots & \vdots & \ddots & \vdots \\ c_s & a_{s1} & \cdots & a_{ss} \\ \hline & b_1 & \cdots & b_s \end{array} \tag{3.2.36}$$

3.3. DIRK Methods

A fully implicit Runge-Kutta method is described by a full, dense matrix according to the context of the Butcher-Tableau.
Therefore, in each step of the method we would require a solution of $n \cdot s$ implicit (potentially non-linear) equations - corresponding to a problem size n and a number of stages s within the Runge-Kutta method - in order to obtain the (approximate) solution u_{k+1} of the next point in time.

$DIRK$ is an acronym meaning **D**iagonal **I**mplicit **R**unge-**K**utta method. A $DIRK(stages, order)$ method is an implicit Runge-Kutta method, specified by a lower triangular matrix, regarding the context of a Butcher-Tableau.
Therefore, the iteration over s stages in order to obtain the (approximate) solution of the next point in time simplifies as follows

$$u_{k,i} = u_k + (\Delta t)_k \sum_{j=1}^{i} a_{ij} f\left(t_k + c_j \left(\Delta t\right)_k, u_{k,j}\right), \quad i = 1, \ldots, s. \tag{3.3.37}$$

$$u_{k+1} = u_k + (\Delta t)_k \sum_{i=1}^{s} b_i f\left(t_k + c_i \left(\Delta t\right)_k, u_{k,i}\right) \tag{3.3.38}$$

Because of the description by a lower triangular matrix, there is no need to solve a linear (or potentially non-linear) system of $n \cdot s$ equations per step. Instead, is suffices if we consecutively solve a set of n equations per stage.
This is due to the fact, that in stage $i \in [1, s]$ all former $u_{k,j}$ with $j = 1, \ldots, i-1$ were already calculated in prior stages.

Accordingly, a DIRK method requires the solution of s systems of problem size n per time step in order to obtain the (approximate) solution of the next consecutive point in time.
Furthermore, a DIRK method is characterized by providing equal values a_{ii} for $i = 1, \ldots, s$ on the main diagonal in the Butcher-Tableau.

The $DIRK(stages, order)$ methods which we consider in this work are $DIRK(2, 2)$, with 2 stages and of order 2, and on the other hand $DIRK(3, 3)$, with 3 stages and of order 3, respectively, as described in [A77].

The corresponding Butcher-Tableau of the $DIRK(2, 2)$ method ([A77], p. 1012) at hand equals

$$\begin{array}{c|cc} \alpha & \alpha & 0 \\ 1 & 1-\alpha & \alpha \\ \hline & 1-\alpha & \alpha \end{array} \tag{3.3.39}$$

in which $\alpha = 1 - \frac{1}{2} \cdot \sqrt{2}$.

The $DIRK(3, 3)$ method ([A77], p. 1012) is associated with the following Butcher-Tableau,

$$\begin{array}{c|ccc} \alpha & \alpha & 0 & \\ \tau_2 & \beta & \alpha & 0 \\ 1 & b_1 & b_2 & \alpha \\ \hline & b_1 & b_2 & \alpha \end{array} \tag{3.3.40}$$

with the respective coefficients ([A77], p. 1012) (mainly depending on α[1])

$$\begin{aligned} \alpha &\approx 0.43586652 \\ \tau_2 &= (1+\alpha)/2 \\ \beta &= \tau_2 - \alpha \\ b_1 &= -\left(6\alpha^2 - 16\alpha + 1\right)/4 \\ b_2 &= \left(6\alpha^2 - 20\alpha + 5\right)/4 \end{aligned} \tag{3.3.41}$$

As stated in [A77], we can formulate the following lemma.

Lemma 3.4. *The introduced methods DIRK(2,2) and DIRK(3,3) are A-stable as well as L-stable.*

3.4. ESDIRK Methods

The DIRK methods, which we consider, are also referred to as *SDIRK* methods (meaning "singly diagonal implicit Runge Kutta" methods).
Another class of implicit time-stepping methods is given by the *ESDIRK* methods.
They correspond to a SDIRK method, but before the SDIRK scheme is applied, an explicit stage is executed. Therefore, an ESDIRK scheme is similar to one of a SDIRK method (3.3.37), besides that the entry $a_{1,1} = 0$, within the Butcher-Tableau (which describes the ESDIRK method).
Because, as stated above, an explicit scheme does not need to solve a linear system within its stages, because it only relies on former approximations. For that reason, the diagonal entry of corresponding to the explicit stage vanishes.

DIRK-type methods may have a property which is called *stiffly-accurate*, this may be advantageous if stiff problems are considered. This property is established if the last row of the Butcher tableau describing an s-stage method $(a_{1,s}, \ldots, a_{s,s})$ is equal to $(b_1, \ldots, b_s)$ ([BBB16], p. 1404).

Examined by Kennedy and Carpenter [KC16], a high stage order within the method as well as L-stability and being stiffly-accurate lead to effective methods regarding stiff problems. But they stated, that the stage order of DIRK-type methods if they are constructed to be stiffly-accurate and L-stable, may not exceed two, which limits these methods if low error tolerances are required, but this limitation is often inconsequential.

SDIRK methods result in a stage order of one. By additionally performing an explicit first stage, ESDIRK methods may achieve a stage order of two. However, if a method of stage order greater than two is desired, fully implicit Runge Kutta schemes may be considered, but these are much more costly to implement ([KC16], p. 4).
A more detailed discussion on this topic is found in [KC16].

3.5. Formulation of the System of Equations

After the definition of the coefficients of the different implicit time-stepping methods, let us reconsider the equation 3.3.37

$$u_{k,i} = u_k + (\Delta t)_k \sum_{j=1}^{i} a_{ij} f\left(t_k + c_j (\Delta t)_k, u_{k,j}\right), \quad i = 1, \ldots, s. \tag{3.5.42}$$

[1] α equals the root of $x^3 - 3x^2 + \frac{3}{2}x - \frac{1}{6} = 0$ lying between $\left(\frac{1}{6}, \frac{1}{2}\right)$ ([A77], p. 1012)

Rearranging the terms leads us to the following system of equations within the i^{th} stage,

$$u_{k,i} - (\Delta t)_k \, a_{ii} f\left(t_k + c_j \, (\Delta t)_k \, , u_{k,i}\right) = u_k + (\Delta t)_k \sum_{j=1}^{i-1} a_{ij} f\left(t_k + c_j \, (\Delta t)_k \, , u_{k,j}\right). \tag{3.5.43}$$

If the operator $f\,(t, u)$ represents a linear operator, it may described by a matrix $\tilde{A}_t$. Therefore, we are able to display the equation in a matrix-wise fashion.

$$u_{k,i} - (\Delta t)_k \, a_{ii} \tilde{A}_{t_{k,i}} u_{k,i} = u_k + (\Delta t)_k \sum_{j=1}^{i-1} a_{ij} \tilde{A}_{t_{k,j}} u_{k,j}. \tag{3.5.44}$$

This yields a linear system of equations which has to be solved for $u_{k,i}$ during each stage $i = 1, \ldots, s$ of a DIRK method.

$$\underbrace{\left(\mathcal{I}_{n\times n} - (\Delta t)_k \, a_{ii} \tilde{A}_{t_{k,i}}\right)}_{=:A_i} u_{k,i} = \underbrace{u_k + (\Delta t)_k \sum_{j=1}^{i-1} a_{ij} \tilde{A}_{t_{k,j}} u_{k,j}}_{=:f_i} \tag{3.5.45}$$

in which $\mathcal{I}_{n\times n}$ corresponds to the identity matrix of size $n \times n$.
Except for $u_{k,i}$, the vector of unknowns, all values are already known from prior stages. Therefore, within a time step k, we obtain a linear system of the form

$$A_i u_{k,i} = f_i, \tag{3.5.46}$$

which we need to solve during each stage i of the DIRK method.

The application of one of these (implicit) methods to a partial differential equation leads to a linear system of equations (or even several systems) (3.1.31, 3.5.45), which we need to solve during each time step in order to obtain an approximation of the solution at the next point in time. These systems may be solved explicitly or could be approximately solved iteratively.

4. Iterative Solvers

Let us consider a linear system of equations, which may result from the application of an implicit time-stepping method, denoted by

$$A \cdot u = f, \tag{4.0.47}$$

with a coefficient matrix $A \in \mathbb{R}^{n\times n}$ and a right-hand-side $f \in \mathbb{R}^n$. We seek for an approximate solution $u_{approx} \in \mathbb{R}^n$, matching the exact solution $u_{exact} = A^{-1} \cdot f$ precisely, except for a small tolerance.
Computing the inverse matrix of A explicitly becomes unfeasible for large linear systems, therefore, an approximate solution to 4.0.47 is aimed at using iterative solvers.

4.1. Krylov Subspace Methods

A popular class of iterative solvers regarding (sparse) linear systems arising from the discretization of partial differential equations are the so called Krylov subspace methods ([PP20], p. 1).

As a type of projection methods, the attempt is to calculate approximations u_j for $j = 1, \ldots, n$ (of the solution of the linear system) lying within the affine subspace $u_0 + \mathcal{K}_j$, in which u_0 denotes an initial guess of the solution of the linear system and $\mathcal{K}_j$ represents a j-dimensional subspace of $\mathbb{R}^n$ ([BD17], p. 38).

Generally, j constraints must be imposed in order to obtain an approximation u_j. Therefore, the residual usually gets constrained to be orthogonal to j linearly independent vectors defining a second j-dimensional subspace of $\mathbb{R}^n$, namely, $\mathcal{L}$ - the so called *subspace of constraints* ([S03], p. 122).
Accordingly, within iteration $j \in [1, n]$ the approximation u_j of the solution is chosen so that the residual complies with

$$r_j = f - Au_j \perp \mathcal{L}_j. \tag{4.1.48}$$

This condition is also called *Petrov-Galerkin condition* ([BD17], p. 38).
Before we regard different Krylov subspace methods, let us first define a Krylov subspace ([H16], p. 179).

Definition 4.1. *The **Krylov subspace** concerning a given matrix $A \in \mathbb{R}^{n \times n}$ and a vector $v \in \mathbb{R}^n$ is defined by*

$$\mathcal{K}_j(A, v) := span\left\{v, Av, A^2v, \ldots, A^{j-1}v\right\}, \quad m \in \mathbb{N}, \tag{4.1.49}$$

with $\mathcal{K}_0(A, v) := \{0\}$.

Choosing $\mathcal{K}_j$ and $\mathcal{L}_j$ as Krylov subspaces results in Krylov subspace methods. Different choices of Krylov subspaces lead to different Krylov subspace methods.

Let us note some significant features of these methods ([BD17] p. 38).

Lemma 4.2. *Consecutive Krylov subspaces are nested, meaning:*

$$\mathcal{K}_j(A, v) \subseteq \mathcal{K}_{j+1}(A, v). \tag{4.1.50}$$

Furthermore, a Krylov subspace method will terminate, in terms of exact arithmetic, in at most n steps if the Petrov-Galerkin condition holds 4.1.48.

Within the class of Krylov subspace methods, it is distinguished between hermitian and non-hermitian problems. Solvers regarding hermitian problems are for example the CG-Method or MINRES. An option in the non-hermitian case is GMRES, an extension of MINRES, or methods like BiCGstab or QMR, which are based on bi-orthogonalization ([PP20], p. 4).
We will present the different Krylov subspace methods stated above in the course of this chapter. But firstly, we would like to focus on the core of these methods.

The core of a Krylov subspace method is a Krylov process which produces a basis of the corresponding Krylov subspace, in which the approximation is searched for. The scope of the application of these methods (e.g. applicable to hermitian or non-hermitian problems) is constrained by the underlying Krylov process.

For example, the core of the GMRES algorithm is the Arnoldi process, which produces an orthonormal basis of the Krylov subspace $\mathcal{K}_j$.

4.2. Arnoldi Process

The Arnoldi process was first described by ARNOLDI [A51] as an iterative solver for the matrix eigenvalue problem. The Arnoldi process is a type of the Gram-Schmidt process applied to a Krylov sequence, which spans a Krylov subspace.
In a classical Gram-Schmidt orthogonalization variant, we can define the algorithm as follows (4.1).

Algorithm 4.1 Arnoldi Process

Input: $A \in \mathbb{R}^{n \times n}$, initial vector $r \in \mathbb{R}^n$, grade m of basis $\mathbf{v}$

Output: orthonormal vectors $\{v_1, \ldots, v_m\}$ with $\text{span}\{v_1, \ldots, v_j\} = \mathcal{K}_j(A, r)$ for $j = 1, \ldots, m$

initialize $v_1 = r / \|r\|$
for $j = 1, \ldots, m$
  for $i = 1, \ldots, j$
    $h_{i,j} = \langle A v_j, v_i \rangle$
  end
  $\tilde{v}_{j+1} = A v_j - \sum_{k=1}^{j} h_{k,j} v_k$
  $h_{j+1,j} = \|\tilde{v}_{j+1}\|$
  if $(h_{j+1,j} == 0)$ ***then*** *stop*
  $v_{j+1} = \tilde{v}_{j+1} / h_{j+1,j}$
end

At each step $1 \leq j \leq m$, the previous basis vector v_j gets multiplied by the matrix A and is orthonormalized afterwards. The orthonormalization is performed against all other previous basis vectors.

More conventional than the implementation of the Arnoldi algorithm with the classical Gram-Schmidt process, is using the Modified-Gram-Schmidt (MGS) orthogonalization. Because of the occurrence of computational rounding errors, the vectors are not precisely orthogonal and this may lead to numerical instabilities ([LS12], p. 27). The MGS is more reliable and yields smaller errors while being mathematically equivalent to the classical method, in terms of exact arithmetic.
In the algorithm 4.2, we formulate the Arnoldi process in the Modified-Gram-Schmidt version ([A03], p. 271).

Due to the rounding errors, which still occur during the floating point operations, the orthogonality will continuously deteriorate. An advancement would be to perform a second orthogonalization step if the deterioration gets too severe. This additional step may be applied after the intermediate vectors $\tilde{v}_{j+1}$ are calculated ([S03], p. 162).

Let us remark the following lemma ([W07], p. 371)

Lemma 4.3. *The generated basis vectors $\{v_1, \ldots, v_j\}$ of the Arnoldi process are orthogonal if and only if the coefficients are chosen as $h_{i,j} = \langle A v_j, v_i \rangle$.*
The algorithm terminates if $h_{j+1,j} = 0$, which happens only if it produces an $A v_j \in \mathcal{K}_j(A, r)$. In this case $\mathcal{K}_j(A, r) = \text{span}\{v_1, \ldots, v_j\}$ is A-invariant.

Algorithm 4.2 Arnoldi Process - MGS

Input: $A \in \mathbb{R}^{n\times n}$, initial vector $r \in \mathbb{R}^n$, grade m of basis $\mathbf{v}$

Output: orthonormal vectors $\{v_1, \ldots, v_m\}$ with $\text{span}\{v_1, \ldots, v_j\} = \mathcal{K}_j(A, r)$ for $j = 1, \ldots, m$

initialize $v_1 = r/\|r\|$
for $j = 1, \ldots, m$
  $\tilde{v}_{j+1} = Av_j$
  for $i = 1, \ldots, j$
    $h_{i,j} = \langle \tilde{v}_{j+1}, v_i \rangle$
    $\tilde{v}_{j+1} = \tilde{v}_{j+1} - h_{i,j}v_i$
  end
  $h_{j+1,j} = \|\tilde{v}_{j+1}\|$
  if $(h_{j+1,j} == 0)$ ***then*** *stop*
  $v_{j+1} = \tilde{v}_{j+1}/h_{j+1,j}$
end

As carried out in [LS12] (p. 28f), we may consider a matrix notation, therefore, let the matrix $V_j \in \mathbb{R}^{n\times j}$ contain the orthonormal basis vectors $\{v_1, \ldots, v_j\}$ column-wise. And the matrix $H \in \mathbb{R}^{j\times j}$ represents the upper Hessenberg matrix accommodating the calculated entries $h_{i,j}$ as its non-zero entries.

$$H_{j,j} = \begin{pmatrix} h_{1,1} & h_{1,2} & \cdots & h_{1,j} \\ h_{2,1} & h_{2,2} & \cdots & h_{2,n} \\ & \ddots & \ddots & \vdots \\ & & h_{j,j-1} & h_{j,j} \end{pmatrix} \tag{4.2.51}$$

At the j^{th} step, $1 \leq j < m$, the calculation reads

$$AV_j = V_j H_{j,j} + h_{j+1,j} v_{j+1} \mathbf{e}_j, \tag{4.2.52}$$

with $\mathbf{e}_j \in \mathbb{R}^{1\times j}$, containing the only non-zero entry at $\mathbf{e}_j(1, j) = 1$. It follows that, due to the orthonormality, that the product $V_j^t V_j = \mathcal{I}_{n\times n}$ equals the identity matrix. Furthermore, $V_j^t v_{j+1}$ equals 0, otherwise $h_{j+1,j}$ would have been zero and v_{j+1} would have not been linearly independent from $\{v_1, \ldots, v_j\}$.

Let us assume that the algorithm terminates at the m^{th} iteration. Subsequently, $h_{m+1,m}$ becomes zero, and 4.2.52 results in

$$AV_m = V_m H_{m,m}. \tag{4.2.53}$$

$H_{m,m}$ represents the orthogonal restriction of A in the basis $\{v_1, \ldots, v_m\}$ of the *A-invariant* subspace $K_m(A, r)$. Multiplying both sides with V_m^t yields to

$$V_m^t A V_m = \underbrace{V_m^t V_m}_{\mathcal{I}_{n\times n}} H_{m,m}. \tag{4.2.54}$$

Let us consider the case, that A is a hermitian matrix. It follows, that $H_{m,m} = V_m^t A V_m$ has be hermitian too, therefore, $H_{m,m}$ must simplify to a tridiagonal matrix.

This reduces the recurrence within the Arnoldi process

$$\tilde{v}_{j+1} = Av_j - \sum_{k=1}^{j} h_{k,j} v_k \tag{4.2.55}$$

to a *three-term-recurrence*

$$\tilde{v}_{j+1} = Av_j - h_{j,j} v_j - h_{j-1,j} v_j. \tag{4.2.56}$$

The algorithm resulting from this property (i.e. in the hermitian case) is also called *(hermitian) Lanczos algorithm*.

4.3. Lanczos Biorthogonalization Algorithm

Another approach, instead of choosing the Arnoldi process, is the non-hermitian form of the Lanczos algorithm ([L50], [L52]), called *Lanczos Biorthogonalization* (it is also referred to as the *unsymmetric Lanczos process*).
The recurrence of the Arnoldi process only simplifies in the hermitian case to a short recurrence. In contrast to the Lanczos Biorthogonalization algorithm, which even produces short recurrences in the non-hermitian case by easing the orthogonality constraints of the basis vectors.
The basis of the Krylov subspace is built of a biorthonormal pair of sequences ([W07], p. 388f).

Definition 4.4. *Two sequences of vectors, $v_1, \ldots, v_m$ and $w_1, \ldots, w_m$, are called* ***biorthogonal*** *if*

$$\langle v_i, w_j \rangle = \begin{cases} 0, & \text{if } i \neq j, \\ d_{i,i}, & \text{if } i = j, \end{cases} \tag{4.3.57}$$

for some diagonal matrix $(d_{i,j})_{i,j=1,\ldots,m}$, or even ***biorthonormal*** *if*

$$\langle v_i, w_j \rangle = \begin{cases} 0, & \text{if } i \neq j, \\ 1, & \text{if } i = j. \end{cases} \tag{4.3.58}$$

The biorthonormal bases built by the unsymmetric Lanczos process span the following two Krylov subspaces ([S03], p. 229f)

$$\begin{aligned} \mathcal{K}_m(A, v_1) &= span\{v_1, Av_1, \ldots, A^{m-1}v_1\} \\ \mathcal{K}_m(A^t, w_1) &= span\left\{w_1, A^t v_1, \ldots, (A^t)^{m-1} w_1\right\} \end{aligned} \tag{4.3.59}$$

We can state the Lanczos Biorthogonalization algorithm ([W07], p. 391) as displayed in algorithm 4.3.

During the computation, this process may suffer from a continuous loss of biorthogonaltiy due to rounding errors. Therefore, in a similar way as in the Arnoldi process, a second re-biorthogonalization step may be performed within each iteration in order to keep the biorthonormality.
Like the Arnoldi process, the unsymmetric Lanczos process may be simplified to the (hermitian) Lanczos algorithm (4.2.56) if the matrix A is hermitian ([W07], p. 391f).

Algorithm 4.3 Lanczos Biorthogonalization

Input: $A \in \mathbb{R}^{n\times n}$, initial vector $v_1 \in \mathbb{R}^n$ (e.g. $v_1 = f - Au_0$), grade m of bases $\mathbf{v}, \mathbf{w}$

Output: approximation u_j

choose vectors v_1, w_1 *s.t.* $\langle v_1, w_1\rangle = 1$
set $\beta_1 = \delta_1 = 0$
set $v_0 = w_0 = (0,\dots,0)^t$
for $j = 1, 2, \dots, m$
  $\alpha_j = \langle Av_j, w_j\rangle$
  $\tilde{v}_{j+1} = Av_j - \alpha_j v_j - \beta_j v_{j-1}$
  $\tilde{w}_{j+1} = A^t w_j - \alpha_j w_j - \delta_j w_{j-1}$
  $\delta_{j+1} = \sqrt{|\langle \tilde{v}_{j+1}, \tilde{w}_{j+1}\rangle|}$
  if $(\delta_{j+1} = 0)$ ***then*** *stop*
  $\beta_{j+1} = \frac{\langle \tilde{v}_{j+1}, \tilde{w}_{j+1}\rangle}{\delta_{j+1}}$
  $w_{j+1} = \frac{\tilde{w}_{j+1}}{\beta_{j+1}}$
  $v_{j+1} = \frac{\tilde{v}_{j+1}}{\delta_{j+1}}$
end

Within the algorithm, we could choose the scaling factors δ_{j+1} and β_{j+1} in several ways, but the importance is to ensure that

$$\delta_{j+1}\beta_{j+1} = \langle \tilde{v}_{j+1}, \tilde{w}_{j+1}\rangle \tag{4.3.60}$$

in order to provide the biorthonormality of the two basis vectors v_{j+1} and w_{j+1}.

The recurrence that is applied to the bases can be represented by the tridiagonal matrix

$$T_{m\times m} = \begin{pmatrix} \alpha_1 & \beta_2 & & & \\ \delta_2 & \alpha_2 & \beta_3 & & \\ & \ddots & \ddots & \ddots & \\ & & \delta_{m-1} & \alpha_{m-1} & \beta_m \\ & & & \delta_m & \alpha_m \end{pmatrix}. \tag{4.3.61}$$

The algorithm breaks if $\delta_{j+1} = 0$. One case in which this may happen, is for example if the dot product of $\tilde{v}_{j+1}$ and $\tilde{w}_{j+1}$ equals 0. A satisfactory result for us would be if this is caused by either of these vectors being equal to the zero vector. Then either $Av_j \in span\{v_1,\dots,v_j\}$ resulting in $\tilde{v}_{j+1} = 0$ or $A^t w_j \in span\{w_1,\dots,w_j\}$ causing $\tilde{w}_{j+1}$ to be zero. This means that either $span\{v_1,\dots,v_j\}$ is A-invariant or if the second case applies ($A^t w_j \in span\{w_1,\dots,w_j\}$), that $span\{w_1,\dots,w_j\}$ is A^t-invariant.
Otherwise, the event in which δ_{j+1} becomes zero, although neither $\tilde{v}_{j+1} \neq (0,\dots,0)^t$ nor $\tilde{w}_{j+1} \neq (0,\dots,0)^t$, is stated as a *serious breakdown* by Saad.
In another case if $\tilde{v}_{j+1}$ and $\tilde{w}_{j+1}$ are nearly orthogonal, their dot product will be nearly zero, resulting in a near-breakdown scenario which eventually causes instabilities and substantial inaccuracy ([W07], p. 390).

Let us recall the following lemma if the algorithm terminates satisfactorily ([S03], p. 230f).

Lemma 4.5. *If the Lanczos Biorthgonalization algorithm terminates in step m (without a preliminary breakdown, e.g. $\delta_{j+1} = 0$, for $j < m$), the generated pair of sequences of vectors $v_1, \ldots, v_m$ and $w_1, \ldots, w_m$ is biorthogonal and either of them form a basis of the corresponding Krylov subspace*

$$span\{v_1, \ldots, v_m\} = \mathcal{K}_m(A, v_1) \tag{4.3.62a}$$

$$span\{w_1, \ldots, w_m\} = \mathcal{K}_m\left(A^t, w_1\right) \tag{4.3.62b}$$

Let us again consider the formulation in a matrix notation, in which $V \in \mathbb{R}^{n\times m}$ contains the basis vectors v_j, $j = 1, \ldots, m$, as its columns and, analogously, $W \in \mathbb{R}^{n\times m}$ contains the generated basis vectors of $\mathcal{K}_m(A^t, w_1)$.
Regarding the calculation within the algorithm, this matrix representation introduces the following relations:

$$AV_m = V_m T_{m\times m} + \delta_{m+1} v_{m+1} \mathbf{e}_m \tag{4.3.63a}$$

$$A^t W_m = W_m T_{m\times m}^t + \beta_{m+1} w_{m+1} \mathbf{e}_m \tag{4.3.63b}$$

in which $\mathbf{e}_m$ represents the m^{th} column of the identity matrix $\mathcal{I}_{m\times m}$.
If the algorithm terminates in step m, the scalar δ_{m+1} in 4.3.63 vanishes. If we consider the first of these equations, a multiplication with W_m^t yields to

$$W_m^t A V_m = W_m^t V_m T_{m\times m}. \tag{4.3.64}$$

Due to the biorthonormality of $\{v_i\}_{i=1,\ldots,m}$ and $\{w_i\}_{i=1,\ldots,m}$, the product $W_m^t V_m$ reduces to the identity matrix $\mathcal{I}_{m\times m}$, leading to the relation

$$W_m^t A V_m = T_{m\times m}. \tag{4.3.65}$$

As stated by Saad in [S03] (p. 231), these relations allow us the following interpretation.
$T_{m\times m}$ corresponds to the projection of A obtained from an oblique projection process onto $\mathcal{K}_m(A, v_1)$ which is orthogonal to $\mathcal{K}_m(A^t, w_1)$. This holds the other way around concerning $T_{m\times m}^t$.

In both processes, if A is a hermitian matrix, the Arnoldi process as well as the unsymmetric Lanczos process (Lanczos Biorthogonalization) may be reduced to the hermitian Lanczos algorithm. So, both processes are a generalization of the hermitian Lanczos algorithm to non-hermitian problems (4.1).

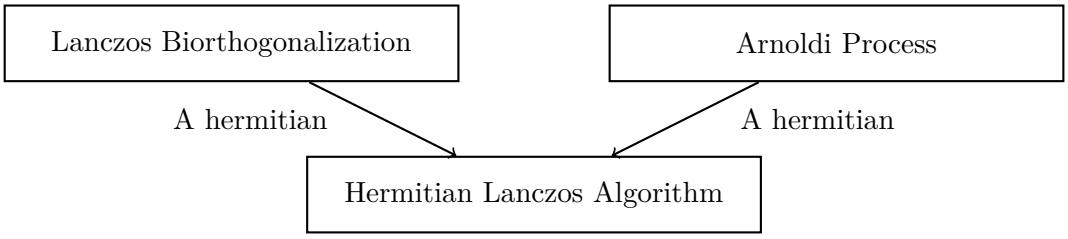


Figure 4.1: Reduction to hermitian Lanczos algorithm if A is hermitian

In conclusion, the Arnoldi process relies on orthogonal sequences during the generation of the basis of the Krylov subspace and results in long recurrences within in the calculation. In contrast

to the Lanczos Biorthogonalization which is formed of a three-term recurrence, while it produces a biorthogonal basis of the Krylov subspace. The biorthogonality refers to one basis which is built with a Krylov process on A supported by another basis which is built simultaneously with a Krylov process on A^t; and vice versa.
The Lanczos Biorthogonalization comes with more possible chances of breakdowns than the Arnoldi process. But in exchange, it requires less storage due to the short recurrences and has a constant computational complexity per iteration.

4.4. Overview: Different Krylov Subspace Methods

In this thesis, we focus mainly on GMRES methods (4.5). These methods are a popular and reliable choice regarding iterative solvers. Furthermore, due to the fact that the underlying linear systems, that we consider (2.0.19) are potentially non-symmetric.
A history of iterative solvers and particularly Krylov subspace methods can be found in [SV00].

In the following section, we are briefly presenting different Krylov subspace methods, which solve linear systems of the form $Au = f$ (4.0.47) iteratively (generally, assuming a non-singular matrix A).

CG-Method The **C**onjugate **G**radients method (CG) [HS52] proposed by Hestenes and Stiefel is an iterative algorithm for solving the linear system $Au = f$ of size n, in which $A \in \mathbb{R}^{n\times n}$ is essentially a symmetric positive definite matrix. In exact arithmetic, this Krylov subspace method terminates in at most n steps. It is a type of projection process, based on the hermitian Lanczos algorithm, with the relations

$$u_j \in u_0 + \mathcal{K}_j(A, r_0) \quad \& \quad r_j \perp K_j(A, r_0) \tag{4.4.66}$$

in which u_0 corresponds to an initial guess of the linear system and $r_0 = f - Au_0$ denotes the initial residual based on this guess ([LS12], p. 36).
The algorithm of the CG-Method as described by Hestenes and Stiefel [HS52] is displayed in algorithm 4.4.

Algorithm 4.4 CG-Method

Input: $A \in \mathbb{R}^{n\times n}$ symmetric positve definite, right-hand-side $f \in \mathbb{R}^n$,
initial guess $u_0 \in \mathbb{R}^n$

Output: approximation u_j

initialize $p_0 = r_0 = f - Au_0$
for $j = 0, 1, \ldots$ *until satisfied*
$\alpha_j = \frac{\|r_j\|^2}{\langle p_j, Ap_j\rangle}$
$u_{j+1} = u_j + \alpha_j p_j$
$r_{j+1} = r_j - \alpha_j Ap_j$
$\beta_j = \frac{\|r_{j+1}\|^2}{\|r_j\|^2}$
$p_{j+1} = r_{j+1} + \beta_j p_j$
end

The algorithm may be terminated in fewer than n iterations if a satisfying result is gained, for example, if a maximum iteration count is reached or the calculated residual r_{j+1} complies with a given tolerance.

Lemma 4.6. *The residuals $r_0, r_1, \ldots$ that we obtain are mutually orthogonal and the direction vectors $p_0, p_1, \ldots$ are mutually conjugate, meaning ([HS52], p. 411)*

$$\begin{aligned} \langle r_j, r_k \rangle &= 0 \\ \langle p_j, Ap_k \rangle &= 0 \end{aligned} \tag{4.4.67}$$

for $j \neq k$.

MINRES In case of indefinite symmetric problems the CG-Method may fail. The algorithm derived by Paige and Saunders [PS75] compensates for this possible failure - it is called **Min**imum **Res**idual method (MINRES).
The base of the MINRES algorithm is the hermitian Lanczos process forming an orthonormal basis of the Krylov subspace $\mathcal{K}_m(A, r_0)$, with the residual $r_0 = f - Au_0$ derived from the initial guess u_0 of the linear system.

As the name implies, we find an approximation u_j with the help of the iterative solver which satisfies

$$u_j = \underset{\tilde{u} \in u_0 + \mathcal{K}_j(A, r_0)}{\arg\min} \|f - A\tilde{u}\|. \tag{4.4.68}$$

Since the residual is minimized over the current Krylov subspace in each iteration and the current j-dimensional Krylov subspace $\mathcal{K}_j$ contains the preceding $(j-1)$-dimensional Krylov subspace $\mathcal{K}_{j-1}$ of the previous iteration step, the residual decreases (in terms of exact arithmetic) monotonically.

Both, the CG-Method and MINRES operate on symmetric matrices. MINRES minimizes the norm of the residual $\|r_j\|$, in contrast to the CG-Method which minimizes the residual regarding the energy norm $\|r_j\|_{A^{-1}}$ ([AG11], p. 198).
The generalization of MINRES to non-symmetric matrices is denoted by the class of GMRES 4.5 methods. The ability to work on non-symmetric matrices comes mainly by replacing the hermitian Lanczos algorithm within MINRES by the Arnoldi process. As stated in chapter 4.2, the Arnoldi process reduces to the hermitian Lanczos algorithm if symmetric matrices are concerned. This yields the reduction of GMRES to MINRES in the hermitian case.

BiCG Based on the Lanczos Biorthogonalization process (4.3), Fletscher [F76] derived in a "conjugate-gradient-like" fashion the **Bi-C**onjugate **G**radients algorithm (BiCG).
The BiCG method resembles the generalization of the CG-Method (for hermitian positive definite matrices) to the general case of non-hermitian (and indefinite) linear systems.
It is a projection process onto the Krylov subspace $\mathcal{K}_m(A, v_1) = span\{v_1, Av_1, \ldots, A^{m-1}v_1\}$, orthogonal to the subspace of constraints $\mathcal{L}_m(A^t, w_1) = span\left\{w_1, A^t w_1, \ldots, (A^t)^{m-1} w_1\right\}$.
In which v_1 is usually chosen as the normed initial residual of the linear system and w_1 is an arbitrary vector, such that $\langle v_1, w_1 \rangle \neq 0$ (often w_1 is chosen equal to v_1 or eventually as the normed initial residual of a linear system $A^t\overline{u} = \overline{f}$ if there is one to solve) [S03] (p. 235).

Algorithm 4.5 Bi-Conjugate Gradients

Input: $A \in \mathbb{R}^{n\times n}$, right-hand-side $f \in \mathbb{R}^n$, initial guess $u_0 \in \mathbb{R}^n$

Output: approximation $u_j \in \mathbb{R}^n$

initialize $p_0 = r_0 = f - Au_0$

choose $\overline{p}_0 = \overline{r}_0$, *s.t.* $\langle r_0, \overline{r}_0\rangle \neq 0$

for $j = 0, 1, \ldots$ *until satisfied*

$\alpha_j = \frac{\overline{r}_j^t r_j}{\overline{p}_j^t A p_j}$

$u_{j+1} = u_j + \alpha_j p_j$

$r_{j+1} = r_j - \alpha_j A p_j$

$\overline{r}_{j+1} = \overline{r}_j - \alpha_j A^t \overline{p}_j$

$\beta_j = \frac{\overline{r}_{j+1}^t r_{j+1}}{\overline{r}_j^t r_j}$

$p_{j+1} = r_{j+1} + \beta_j p_j$

$\overline{p}_{j+1} = \overline{r}_{j+1} + \beta_j \overline{p}_j$

end

In order to enforce the biorthogonality conditions

$$\overline{r}_{j+1}^t r_j = r_{j+1}^t \overline{r}_j = 0, \tag{4.4.69}$$

α_j is chosen accordingly within the algorithm.
In addition, β_j is chosen in order to ensure the biconjugacy condition

$$\overline{p}_{j+1}^t A p_j = p_{j+1}^t A^t \overline{p}_j = 0. \tag{4.4.70}$$

As stated and proofed in [F76] (p. 80f), we remark the following lemma.

Lemma 4.7. *Under the assumption that the algorithm does not break down preliminary, it holds for all* $j < \ell$

$$\begin{aligned} \overline{r}_\ell^t r_j = r_\ell^t \overline{r}_j &= 0 \\ \overline{p}_\ell^t A p_j = p_\ell^t A^T \overline{p}_j &= 0 \end{aligned} \tag{4.4.71}$$

The algorithm does not only solve the linear system $Au = f$ but also solves a "dual linear system" $A^t\overline{u} = \overline{f}$ implicitly. Therefore, in the case, that a kind of "dual linear system" is ought to be solved simultaneously, the initial vector $\overline{r}_0$ may be chosen as the initial residual $\overline{r}_0 = \overline{f} - A^t\overline{u}_{(0)}$. Analogously, the generated approximation of the second system may be updated by $\overline{u}_{j+1} = \overline{u}_j + \alpha_j\overline{p}_j$ ([S03], p. 234f).
Otherwise, if an approximation of a second system (described by A^t) is not of interest, the operations regarding the transpose of A might be omitted by choosing adaptions or variations of the biconjugate gradients algorithm, working without the transpose, e.g. CGS - Conjugate Gradients Squared method [S84].

BiCGstab One of the drawbacks of BiCG is the usage of the transposed matrix of A within the algorithm. Especially, if A is large, applying the transpose is rather costly if it is not needed explicitly. There are even some circumstances in which the transpose is usually

not available, for example in a matrix-free context in which the matrix is given implicitly as an application to a vector. But there are adaptations like BiCGstab which avoid these operations.

The transpose-free variant of BiCG, namely CGS, may lead to a substantial aggregation of rounding errors in cases of irregular convergence ([S03], p. 241). Therefore, this adaptation was developed even further by Van Der Vorst [V92] in order to remedy for this side effect.
The resulting algorithm obtained the name **Bi-C**onjugate **G**radient **stab**[ilized] method (BiCGstab) due to its favorable stability properties and similarity to the Conjugate Gradients Squared method.
We display the algorithm in a not-preconditioned fashion in algorithm 4.6 ([V92], p. 635f).

Algorithm 4.6 BiCGstab

Input: $A \in \mathbb{R}^{n \times n}$, right-hand side $f \in \mathbb{R}^n$, initial guess u_0

Output: approximation $u_j \in \mathbb{R}^n$ of the solution

compute *initial residual* $r_0 = f - Au_0$

choose *arbitrary vector* $\tilde{r}_0$*, s.t.* $\langle r_0, \tilde{r}_0 \rangle \neq 0$*, e.g.* $\tilde{r}_0 = r_0$

$\rho_0 = \alpha = \omega_0 = 1$

$v_0 = p_0 = (0, \ldots, 0)^t$

for $j = 1, 2, \ldots$

$\rho_j = \langle \tilde{r}_0, r_{j-1} \rangle$

$\beta = \frac{\rho_j}{\rho_{j-1}} \frac{\alpha}{\omega_{j-1}}$

$p_j = r_{j-1} + \beta \left(p_{j-1} - \omega_{j-1} v_{j-1} \right)$

$v_j = Ap_j$

$\alpha = \frac{\rho_j}{\langle \tilde{r}_0, v_j \rangle}$

$s = r_{j-1} - \alpha v_j$

$t = As$

$\omega_j = \frac{\langle t, s \rangle}{\langle t, t \rangle}$

$u_j = u_{j-1} + \alpha p_j + \omega_j s$

if $(u_j$ *is accurate enough)* ***then*** *quit*

$r_j = s - \omega_j t$

end

QMR The **Q**uasi-**M**inimal **R**esidual method (QMR) is also an iterative solver for non-hermitian linear systems, described by a non-singular matrix A. The algorithm was proposed by Freund and Nachtigal [FN91].

The iterates of the method are constructed, such that $u_j \in u_0 + \mathcal{K}_j(A, r_0)$. Herein, $u_{(0)}$ denotes an initial guess of the linear system $Au = f$ and r_0 corresponds to the initial residual.
In the case of a non-hermitian matrix A, QMR is not based on a projection process, unlike the other methods. It tries to combine the ideas of BiCG and GMRES in order to take on

the problem of breakdowns within the BiCG method ([LS12], p. 62).
The QMR method is also based on the unsymmetric Lanczos process (4.3) and addresses the possible breakdown scenarios and numerical instabilities, from which the BiCG method may suffer.
The iterates of the biconjugate gradient algorithm have a rather irregular convergence scheme with oscillations within in the norm of the residual, because they are not characterized by a minimization property, but rather by Galerkin-type condition. Furthermore, possible breakdowns (division by zero) may occur, but more likely are "near-breakdown-scenarios" which may lead to numerical instabilities in subsequent iterations.
BiCGstab has a good performance in several cases, but it is not secured against the latter issue. Concerning exact arithmetic, it will break down if the core (the unsymmetric Lanczos process) breaks down. To overcome this issue, QMR introduces a look-ahead strategy during the generation of the basis vectors of the Krylov subspace ([FN91], p. 316).
In addition, the iterates of the QMR method are defined by a quasi-minimization of the residual norm. In contrast to BiCG, this leads to a more smooth convergence curve.
Like the biconjugate gradient algorithm, QMR requires multiplications with the matrix A as well as with its transpose A^t. Freund developed a **T**rasnpose **F**ree **Q**uasi **M**inimal **R**esidual method (TFQMR) [F93]. It is derived from the CGS method which is similar to the biconjugate gradient algorithm, but without the need of applying the transposed matrix of A.
A quasi-minimization property can be introduced to the CGS algorithm by adapting just a few lines; a more detailed description can be found in [F93]. But the resulting method (TFQMR) can still break down, just like the standard CGS algorithm. To overcome this issue a look-ahead technique in order to skip iterations, in which breakdowns or near-breakdowns would occur, may be incorporated ([F93], p. 481).

Concluding, we portray the different Krylov subspace methods and their mathematical relations in the figure 4.4 (in style of [LS12], p. 64).

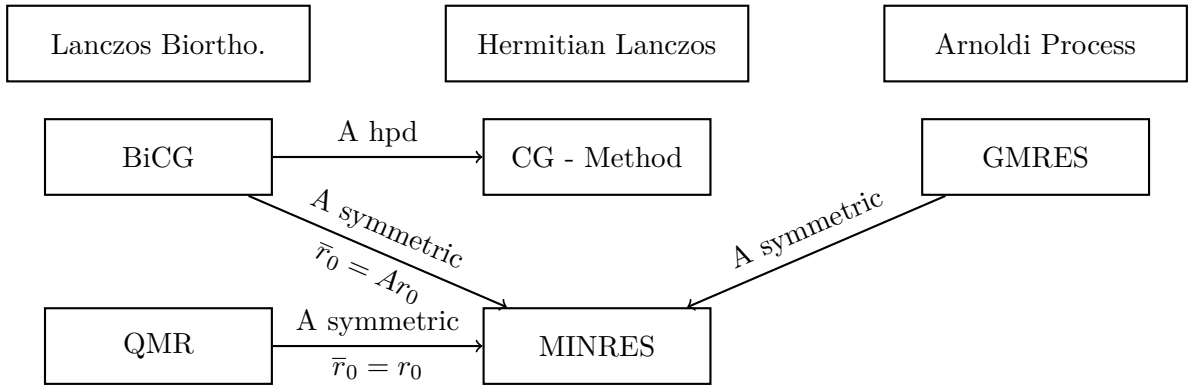


Figure 4.1: Mathematical equivalence of the Krylov subspace methods if the concerned problem matches the conditions on the connecting arrows.

By the means of preconditioning, these methods may eventually terminate/converge much faster. Therefore, though it comes with a computational cost, preconditioning is crucial in terms of efficiency (5).

4.5. Generalized Minimum Residual Method

A projection method choosing $\mathcal{K}_m(A, v_1)$, in which $v_1 = \frac{r_0}{\|r_0\|}$ with $r_0 = f - Au_0$, as the *search space* and $\mathcal{L}_m = A\mathcal{K}_m$ as the *subspace of constraints* is called **G**eneralized **M**inimal **Res**idual method (GMRES).

This Krylov subspace method was proposed by SAAD and SCHULTZ [SS86] and it is applicable to non-hermitian systems, due to being based on the Arnoldi process (4.2).
Accordingly, the approximation of the solution of 4.0.47 is extracted from the affine subspace $u_0 + \mathcal{K}_m(A, v_1)$, in which u_0 is an initial guess to the linear system ([S03], p. 171).

The solution u of the linear system $Au = f$ 4.0.47, which ought to be approximated iteratively with GMRES, is designed by the recurrence

$$u_j = u_0 + z_j, \quad j = 1, 2, \ldots. \tag{4.5.72}$$

The "adjustment"-vector z_j is drawn from the Krylov subspace $\mathcal{K}_j(A, v_1)$, so that the least squares problem

$$\min_{z_j \in \mathcal{K}_j} \| f - A\underbrace{(u_0 + z_j)}_{=u_j} \| \Leftrightarrow \min_{z \in \mathcal{K}_j} \|r_0 - Az_j\| \tag{4.5.73}$$

is solved ([SS86], p. 859).
Given that $z_j \in \mathcal{K}_j$, it can be represented as $z_j = V_j y_j$ by the generated basis of the Arnoldi process ($\mathcal{K}_j(A, v_1) = span\{v_1, \ldots, v_j\}$) multiplied by a "scaling" vector $y_j \in \mathbb{R}^j$.
Therefore, we reformulate the recurrence 4.5.72 to ¸

$$u_j = u_0 + V_j y_j, \quad j = 1, 2, \ldots. \tag{4.5.74}$$

Within Arnoldi's algorithm, the basis vectors get normalized, starting with the initial residual (of the linear system regarding the initial guess u_0) as the initial basis vector, it follows $v_1 = r_0 / \|r_0\|$.
Therefore, 4.5.73 transforms to

$$\min_{y_j \in \mathbb{R}^j} \| \|r_0\| v_1 - AV_j y_j \|. \tag{4.5.75}$$

Let us consider the equation 4.2.52 once more, it can be rewritten to

$$AV_j = V_{j+1} \tilde{H}_j \tag{4.5.76}$$

and denotes the setting of Arnoldi's algorithm after j steps.
Here, $\tilde{H}_j \in \mathbb{R}^{j+1 \times j}$ resembles the upper Hessenberg matrix H_j of the Arnoldi process (4.2.51) extended by an additional row, containing only zeros, except for the last entry $h_{j+1,j}$ $(= \|\tilde{v}_{j+1}\|)$.
$V_{j+1} \in \mathbb{R}^{n \times j+1}$ results from V_j, but received the generated $(j+1)^{\text{th}}$ basis vector v_{j+1} as an additional $(j+1)^{\text{th}}$ column.
Inserting 4.5.76 into the least squares problem 4.5.73 results in

$$\min_{y_j \in \mathbb{R}^j} \left\| \|r_0\| v_1 - V_{j+1} \tilde{H}_j y_j \right\|. \tag{4.5.77}$$

By refactoring, the minimization problem reformulates to

$$\min_{y_j \in \mathbb{R}^j} \left\| V_{j+1} \left(\|r_0\| \mathbf{e}_1 - \tilde{H}_j y_j \right) \right\|, \tag{4.5.78}$$

in which $\mathbf{e}_1$ equals the first column of the $(j+1) \times (j+1)$ identity matrix $\mathcal{I}_{j+1 \times j+1}$.
Due to the orthonormality of V_{j+1}, the norm simplifies to

$$\min_{y_j \in \mathbb{R}^j} \left\| \|r_0\| \mathbf{e}_1 - \tilde{H}_j y_j \right\|. \tag{4.5.79}$$

GMRES provides the approximation $u_j = u_0 + V_j y_j$ of the exact solution of the linear system $Au = f$, in which y_j minimizes 4.5.79, during the j^{th} step, $j = 1, 2, \ldots$ ([A03], p. 272).

We present the algorithm of GMRES 4.7 proposed by Saad and Schultz in its basic form ([SS86], p. 860) in algorithm 4.7.

Algorithm 4.7 GMRES

Input: $A \in \mathbb{R}^{n \times n}$, right-hand-side $f \in \mathbb{R}^n$, initial guess $u_0 \in \mathbb{R}^n$

Output: approximation $u_j \in \mathbb{R}^n$ of the solution

compute *initial residual* $r_0 = f - Au_{(0)}$
initialize $v_1 = r_0 / \|r_0\|$
for $j = 1, 2, \ldots$ *until satisfied*
  for $i = 1, \ldots, j$
    $h_{i,j} = \langle Av_j, v_i \rangle$
  end
  $\tilde{v}_{j+1} = Av_j - \sum_{i=1}^{j} h_{i,j} v_i$
  $h_{j+1,j} = \|\tilde{v}_{j+1}\|$
  $v_{j+1} = \tilde{v}_{j+1} / h_{j+1,j}$
end
$u_j = u_0 + V_j y_j$ *where* y_j *minimizes* 4.5.79

In order to calculate the approximation u_j within the algorithm, we have to solve the least squares problem 4.5.79.
This can be done by a QR-factorization of $\tilde{H}_j$ using plane rotations. But instead of recalculating the decomposition at every Arnoldi iteration (when a new basis vector is generated), it would be more favorable if we could update the factorization progressively at every step of the Arnoldi process. Because this allows us to obtain the residual norm of the approximation without calculating the approximation u_j explicitly, in order to decide whether the iteration has to be carried on further or not. And the vector y_j in 4.5.79 which updates the approximation at step j is only calculated if a given tolerance is satisfied, thus, saving some computational cost.
This procedure of the factorization is described in [SS86] (p. 860ff) and we will present it in the following section.

Let us define the rotation matrix which represents the rotation of the unit vectors $\mathbf{e}_j$ and $\mathbf{e}_{j+1}$

by a given angle θ_j, as

$$F_j = \begin{pmatrix} 1 & & & & & & & \\ & \ddots & & & & & & \\ & & 1 & & & & & \\ & & & \mathbf{c}_j & -\mathbf{s}_j & & & \\ & & & \mathbf{s}_j & \mathbf{c}_j & & & \\ & & & & & 1 & & \\ & & & & & & \ddots & \\ & & & & & & & 1 \end{pmatrix} \begin{matrix} \\ \\ \\ j^{\text{th}} \text{ row} \\ (j+1)^{\text{th}} \text{ row} \\ \\ \\ \\ \end{matrix} \tag{4.5.80}$$

in which $\mathbf{c}_j$ and $\mathbf{s}_j$ comply with $\cos(\theta_j)$ and $\sin(\theta_j)$, respectively. Ergo, these coefficients have to satisfy the Pythagorean identity $\mathbf{c}_j^2 + \mathbf{s}_j^2 = 1$.
After m steps have been carried out, these rotation matrices have the size $F_j \in \mathbb{R}^{(m+1)\times(m+1)}$ for $j \leq m$.
Now, let us assume that the application of the rotation matrices F_i, $i = 1, \ldots, j$, to $\tilde{H}_j$ from the left should have produced the upper-triangular matrix $R_j \in \mathbb{R}^{(j+1)\times j}$

$$R_j = \begin{pmatrix} \tilde{h}_{1,1} & \tilde{h}_{1,2} & \cdots & \tilde{h}_{1,j} \\ & \tilde{h}_{2,2} & \cdots & \tilde{h}_{2,j} \\ \vdots & & \ddots & \vdots \\ & & & \tilde{h}_{j,j} \\ 0 & \cdots & \cdots & 0 \end{pmatrix} \tag{4.5.81}$$

Accordingly, the coefficients $\mathbf{c}_j$ and $\mathbf{s}_j$ are chosen such that the entry $h_{j+1,j}$ will vanish in order to "upper-triangul-ize" the matrix.
In the next step ($\mapsto j+1$), we will append the last column of $\tilde{H}_{j+1}$ and last row with the entry $h_{j+2,j+1}$, in order to obtain the (non-triangular) matrix $\tilde{R}_{j+1}$. This results in

$$R_j = \begin{pmatrix} \tilde{h}_{1,1} & \tilde{h}_{1,2} & \cdots & \tilde{h}_{1,j} & h_{1,j+1} \\ & \tilde{h}_{2,2} & \cdots & h_{2,j} & \\ \vdots & & \ddots & \vdots & \vdots \\ & & & \tilde{h}_{j,j} & h_{j,j+1} \\ 0 & \cdots & \cdots & 0 & h_{j+1,j+1} \\ 0 & \cdots & \cdots & 0 & h_{j+2,j+1} \end{pmatrix}. \tag{4.5.82}$$

The sub-matrix $\tilde{R}_{j+1}(1:(j+1), 1:j)$ equals R_j.

In order to receive the (upper-triangular) matrix $R_{j+1} \in \mathbb{R}^{(j+2)\times(j+1)}$, the last column consisting of $(h_{1,j+1}, \ldots, h_{j+2,j+1})^t$ has to be successively multiplied from the left by all previous rotation matrices F_i, for $i = 1, \ldots, j$, (these applications do no effect the entry $h_{j+2,j+1}$). Afterwards, we extract the upper-triangular matrix R_{j+1} by applying the next rotation matrix F_{j+1} which will eliminate the entry $h_{j+2,j+1}$. Therefore, this rotation has to correspond to

$$\begin{aligned} \mathbf{c}_{j+1} &= \frac{\tilde{h}_{j+1,j+1}}{\sqrt{\tilde{h}_{j+1,j+1}^2 + h_{j+2,j+1}^2}} \\ \mathbf{s}_{j+1} &= -\frac{h_{j+2,j+1}}{\sqrt{\tilde{h}_{j+1,j+1}^2 + h_{j+2,j+1}^2}} \end{aligned} \tag{4.5.83}$$

in which $\tilde{h}_{j+1,j+1}$ denotes the entry $h_{j+1,j+1}$ of $\tilde{H}_{j+1}$ after the applications of the previous rotations.
With regards to the minimization problem 4.5.79, all the rotations that we have applied to $\tilde{H}_j$ must also be applied simultaneously to the other side ($\beta \mathbf{e}_1$).
After j steps, let $Q_j \in \mathbb{R}^{(j+1)\times(j+1)}$ be the accumulated product of these rotations

$$Q_j = \prod_{i=0}^{j-1} F_{j-i}. \tag{4.5.84}$$

The obtained decomposition equals

$$Q_j \tilde{H}_j = R_j \tag{4.5.85}$$

with the upper-triangular matrix $R_j \in \mathbb{R}^{(j+1)\times 1}$, whose last row only contains zeros.

Due to the Pythagorean identity, that is satisfied by the coefficients $\mathbf{c}_j$ and $\mathbf{s}_j$, every rotation matrix F_j is an unitary matrix, as well as their product Q_j. Therefore, the multiplication of Q_j does not affect the euclidean norm,

$$\left\|\beta \mathbf{e}_1 - \tilde{H}_j y_j\right\| = \left\|Q_j \left(\beta \mathbf{e}_1 - \tilde{H}_j y_j\right)\right\| = \left\|Q_j \beta \mathbf{e}_1 - R_j y_j\right\|. \tag{4.5.86}$$

And the minimization of this transformed problem with respect to y_j can be achieved by solving the upper-triangular system resulting from removing the last row of R_j (the $(j+1)^{\text{th}}$ row) and the last component of $Q_j \beta \mathbf{e}_1$, this is possible because the last row of R_j only contains zeros.
Afterwards, we can build the approximate solution u_j by the linear combination 4.5.74

$$u_j = u_0 + V_j y_j. \tag{4.5.87}$$

An interesting feature mentioned before is the way how the residual norm may be calculated. As shown in [SS86] (p. 862f), during the QR decomposition it is possible to receive the residual norm $\|r_j\|$ of the approximate solution u_j without calculating u_j explicitly.
Let us reconsider the minimization problem 4.5.79, which is equal to 4.5.86, and the initial formulation of the least squares problem 4.5.73 shows, that in fact the residual norm corresponds exactly to $\|Q_j \beta \mathbf{e}_1 - R_j y_j\|$. By the construction of y_j and the form of R_j, the residual norm equals to the absolute value of the last component of $Q_j \beta \mathbf{e}_1$.
So, premultiplying $\beta \mathbf{e}_1$ by the rotation matrices F_j, which were used in order to transform $\tilde{H}_j$ into the upper-triangular matrix R_j, automatically provides the residual norm r_j of the approximate solution u_j in its last (in the $(j+1)^{\text{th}}$) component.
Therefore, within every step of the QR decomposition the residual norm (of the corresponding iteration) is provided without any additional cost, because $Q_j \beta \mathbf{e}_1$ is updated at every step.

In the non-hermitian case the Arnoldi process depends on all previous basis vectors (unlike in the hermitian case or in case of the unsymmetric Lanczos process) in order to generate the next orthonormal basis vector. Due to this full recurrence, the storage requirements augment with every iteration and, correspondingly, the time needed to orthonormalize the next basis vector against all previous basis vectors grows simultaneously with every additional iteration.
If large linear non-hermitian systems are concerned, the "ever growing" storage requirements of GMRES may become a difficulty. Per iteration, the storage requirements grow linearly with the iteration count j and the amount of multiplications with $\frac{1}{2}nj^2$ ([SS86], p. 863ff). But this might be limited by a variation of the basic GMRES algorithm (4.5). The idea is to restart the

algorithm after a predefined number of iterations (say ℓ). So, if the approximation calculated by the algorithm does not comply with a given tolerance after ℓ steps, it will be restarted and the last approximation u_ℓ will become the "initial guess" and the first basis vector will be chosen as the scaled ℓ^{th} residual,

$$u_0 := u_m, \qquad v_1 := \frac{r_m}{\|r_m\|}. \tag{4.5.88}$$

As portrayed in [SS86], this leads to the following algorithm 4.8.

Algorithm 4.8 Restarted GMRES(ℓ)

Input: $A \in \mathbb{R}^{n\times n}$, right-hand-side $f \in \mathbb{R}^n$, initial guess $u_0 \in \mathbb{R}^n$

Output: approximation u_k of the solution of the linear system

compute *initial residual* $r_0 = f - Au_0$
initialize $v_1 = r_0 / \|r_0\|$
for $j = 1, 2, \ldots, \ell$
  for $i = 1, \ldots, j$
    $h_{i,j} = \langle Av_j, v_i \rangle$
  end
  $\tilde{v}_{j+1} = Av_j - \sum_{i=1}^{j} h_{i,j} v_i$
  $h_{j+1,j} = \|\tilde{v}_{j+1}\|$
  $v_{j+1} = \frac{\tilde{v}_{j+1}}{h_{j+1,j}}$
end
$u_j = u_0 + V_\ell y_\ell$ *where* $y_\ell \in \mathbb{R}^\ell$ *minimizes* 4.5.79
compute $r_j = f - Au_j$
if (r_j *is satisfying*) ***then*** *stop*
else *compute* $u_0 := u_j$, $v_1 = r_j / \|r_j\|$ *and jump to outer for-loop*

Neglecting the calculated orthonormal basis of the Krylov subspace after a predefined count of iterations, and start over again, slows down the convergence of the method. Some improvements can be achieved if a part of the previous basis is kept (e.g. see [EES00]).

Let us briefly conclude some features of the iterative solvers for non-hermitian problems.
The different minimization schemes have an impact on the methods. Whereas GMRES performs a full minimal residual minimization due to the storage of all consecutive vectors which yields a rather smooth convergence, the QMR method only performs a quasi-minimization, due to being based on the unsymmetric Lanczos process. Furthermore, the unsymmetric Lanczos process suffers from a severe loss of biorthogonality in finite arithmetic within its short recurrences.
Along with the loss of biorthogonality, possible breakdowns scenarios occur. To enhance the numerical stability of the unsymmetric Lanczos, some look-ahead techniques, in order to prevent breakdowns, may be incorporated, which may be useful in several cases, but, generally, their impact is limited ([LS12], p. 64, and references therein).
On the other hand, the methods based on the unsymmetric Lanczos process, e.g. BiCG and QMR, have a constant storage requirement and computational cost due to the short recurrences, in contrast to GMRES with a continuously increasing storage requirement, although it may be capped if it is restarted after several iterations ([K95], p. 4915ff).

Another way to improve GMRES would be to utilize another orthogonalization scheme. Previously, the Arnoldi Process was presented with the (modified) Gram-Schmidt orthogonalization. Using the Householder-Version of the Arnoldi process yields to a numerically more stable method ([S03], p. 173f).

5. Preconditioning

Preconditioning enhances the efficiency as well as the robustness of iterative solvers. SAAD even emphasizes, that generally, the quality of the preconditioner matters more than the choice of a specific Krylov subspace method in terms of the reliability of iterative techniques ([S03], p. 275).

In general, preconditioning transforms the linear system $A \cdot u = f$ (4.0.47), which ought to be solved, into another linear system with the same solution, but the iterative solver used should most likely have an easier time solving the transformed (preconditioned) system than solving the initial one.
But important aspects of the linear system should be retained. For example, if a hermitian problem is ought to be preconditioned, a symmetric preconditioning method should be chosen in order to keep this property.

The process of preconditioning happens mainly by the means of an application of a (preconditioning) Matrix $M \in \mathbb{R}^{n \times n}$.
If the preconditioner does not represent a linear operator, a flexible Krylov subspace should be used ([PP20], p. 4), e.g. resulting in FGMRES methods.
It may be distinguished between *left-* and *right-preconditioning* (regarding non-hermitian problems). In the former case the transformed linear system reads

$$M^{-1}Au = M^{-1}f. \tag{5.0.89}$$

In the latter case the preconditioning matrix is applied from the right, resulting in

$$AM^{-1}\tilde{u} = f, \quad \tilde{u} = Mu. \tag{5.0.90}$$

Some preconditioning matrices may be the product of factorization $M = M_1 \cdot M_2$ with $M_1, M_2 \in \mathbb{R}^{n \times n}$, for instance an LU-factorization.
In this case, a so called *split-preconditioning* can be applied ([S03], p. 285)

$$M_1^{-1}AM_2^{-1}\tilde{u} = M_1^{-1}f, \quad \tilde{u} = M_2u. \tag{5.0.91}$$

The convergence rate of an iterative solver depends (among other properties) on the size of the condition number of the matrix A of the underlying linear system, which ought to be solved. The spectral condition number of a regular matrix A is given by $\kappa(A) = \rho(A)\,\rho(A^{-1})$, if A is positive definite, the spectral condition number simplifies as follows $\kappa(A) = \dfrac{\lambda_{max}}{\lambda_{min}}$, in which λ_{max} describes the largest and λ_{min} describes the smallest eigenvalue of A.
Therefore, the spectral condition number of the preconditioned linear system should be lower than the spectral condition number of the original system matrix

$$\kappa\left(AM^{-1}\right) \leq \kappa(A), \tag{5.0.92}$$

exemplary, in the case of right-preconditioning.
Ideally, the eigenvalues of the matrix of the preconditioned linear system are clustered resulting in a condition number $\kappa\left(AM^{-1}\right) \approx 1$ in order to achieve fast convergence ([H16], p. 165).

Regarding a preconditioning matrix M applied to a linear system (4.0.47) which ought to be solved iteratively with GMRES (4.5), both, *left-* as well as right-preconditioning invoke the same affine subspace

$$u^0 + span\left\{\tilde{r}_0, M^{-1}A\tilde{r}_0, \left(M^{-1}A\right)^2 \tilde{r}_0, \ldots, \left(M^{-1}A\right)^{m-1} \tilde{r}_0\right\}, \tag{5.0.93}$$

from which the approximate solution of the linear system is obtained. But within GMRES the left-preconditioned variant minimizes the preconditioned residual $M^{-1}r$, whereas in the case of right-preconditioning the residual r gets minimized, in which r is taken from the same affine subspace ([S03], p. 285f).

Using a right-preconditioning strategy gives rise to another GMRES method, namely **F**lexible GMRES (FGMRES). The "flexibility" of this generalization of GMRES comes by the fact, that a varying preconditioner is permitted. The preconditioner is allowed to change at every step ([BD17], p. 85).

5.1. Block Preconditioning

The block preconditioners, which we consider in this work are the block Jacobi (BJ) and block Gauss-Seidel (BGS) preconditioner.
Those well-known preconditioning methods are rather easy to implement and computationally rather cheap, in several cases they yield a good performance. However, in a matrix-free context, in which neither the matrix A of the linear system as a whole nor single entries of it are directly accessible, may increase the cost of the application.

Let us consider a linear system $Au = f$, in case of the BJ preconditioner, the preconditioning matrix M results from the diagonal blocks of A. In a finite element context, it is a natural choice to choose the size of each block equal to the amount of degrees of freedom per element.

In the figure 5.1, we display a coefficient matrix A (for all interior nodes) corresponding to an exemplary grid, which is pictured on the left.

The preconditioning matrix M which is extracted by BJ only consists of the diagonal blocks of $A \in \mathbb{R}^{n\times n}$ (corresponding to the thick diagonal lines in the exemplary figure).
Let the number of degrees of freedom per element be given by $n^{dof} \in \mathbb{N}$ and n^{elem} denotes the number of elements within the grid. The discretization leads to the following block representation of A

$$A = \begin{pmatrix} A_{1,1} & A_{1,2} & \cdots & A_{1,n^{elem}} \\ A_{2,1} & A_{2,2} & \cdots & A_{2,n^{elem}} \\ \vdots & \vdots & \ddots & \vdots \\ A_{n^{elem},1} & A_{n^{elem},2} & \cdots & A_{n^{elem},n^{elem}} \end{pmatrix} \tag{5.1.94}$$

with $A_{i,j} \in \mathbb{R}^{n^{dof}\times n^{dof}}$.

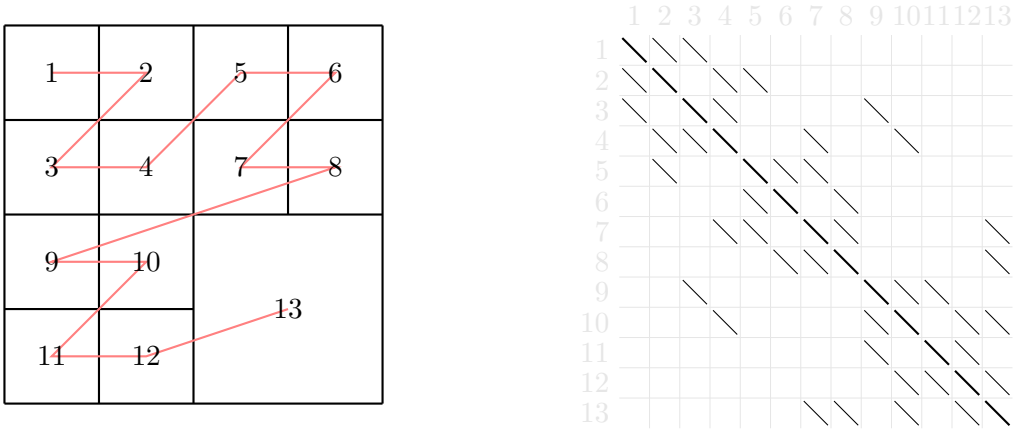


Figure 5.1: (Left:) Element ordering of the grid elements in form of the Morton Z-curve. (Right:) Sparsity pattern of a matrix A, based on the adjacency of the elements (for interior nodes). (For the ease of portraying, there are only diagonal lines colored in the sparsity pattern of the matrix A, although, all degrees of freedom in neighboring elements may be influential.)

So, $M \in \mathbb{R}^{n \times n}$ is also defined by blocks $M_{i,j}$ of the size $n^{dof} \times n^{dof}$

$$M_{i,j} = \begin{cases} A_{i,j}, & \text{if } i = j \\ \mathbf{0}, & \text{if } i \neq j \end{cases}, \tag{5.1.95}$$

for $i, j = 1, \ldots, n^{elem}$ and $\mathbf{0}$ denotes a matrix of size $n^{dof} \times n^{dof}$ full of zeros.

The block Gauss-Seidel preconditioner is defined in a similar way. It includes not only the diagonal blocks of A but also all blocks below (or above) the diagonal. Therefore, it equals the block-wise lower (or upper) triangular matrix of A.
We obtain the block Gauss-Seidel preconditioning matrix M by

$$M_{i,j} = \begin{cases} A_{i,j}, & \text{if } i \leq j \\ \mathbf{0}, & \text{if } i > j \end{cases} \tag{5.1.96}$$

for $i, j = 1, \ldots, n^{elem}$.

In order to apply these block preconditioners, we have to calculate the inverse of the preconditioning matrix. In this regard, obtaining the inverse of the BGS preconditioning matrix is more costly than in the case of BJ. However, for some problems BGS performs significantly better than BJ. The block Gauss-Seidel preconditioner depends on the ordering of the elements, this may have a substantial impact on the performance. Especially, if pure advection problems are considered, BGS (with an appropriate ordering) performs better than BJ, but for general problems rather small improvements are obtained by both block preconditioners ([PP08], p. 8).

5.2. Multigrid Preconditioning

Multigrid methods (MG) represent a popular and effective class of solvers as well as preconditioners which can be applied to linear systems of equations. These methods are capable of

solving a linear system of N unknowns in $\mathcal{O}(N)$ work and achieve a convergence order independently of the resolution of the grid ([F06]).
They may operate geometrically on a hierarchy of meshes or just algebraically on a linear system of equations without information of the underlying grid on which the discretization of the differential equation is considered. Especially, if (scalar) elliptic partial differential equations are of interest, MG methods will represent a very effective tool ([B96]).

Multigrid methods have been successfully applied as preconditioners to discretizations of linear advection-diffusion problems and as well in the context of discontinuous Galerkin methods, see e.g. [E19], [FM20], [BGM21], [PLH09].
The main idea, these methods are based on, is to reduce high-oscillatory errors on the initial underlying grid (also referred to as the *fine grid*) with a small number of smoothing iterations, performed by a basic iterative scheme (e.g. Richardson-, Jacobi-Iteration or even GMRES, etc.), which generally reduces the high-oscillatory components effectively. And to account for low-oscillatory errors on a much coarser grid (by further smoothing iterations or a so called *coarse grid correction*), because these low-oscillatory errors on the fine grid become "higher-oscillatory" on a coarser grid ([BHM00], p. 31ff).

MG methods consist essentially of three components, namely, smoothing, interpolation and coarse grid correction.
Geometrical multigrid methods (GMG) require at least a second (coarser) grid which emerged from the initial grid in order to operate.

The most basic variation of MG methods is a two level multigrid method in a so called *V-cycle*. A short description of this method may read:

Let us denote an initial guess of the linear system $Au = f$ (4.0.47) by u_0.
Based on this initial guess, we perform a predefined number of smoothing steps (on the fine grid) with a basic iterative method. After these (say $m \in \mathbb{N}$) steps, we calculate the residual $r_m = f - Au_m$ and interpolate/restrict it to the coarser grid ($r_m \mapsto r^{coarse}$). On this coarse grid, the (coarse) residual equation

$$A^{coarse} e^{coarse} = r^{coarse} \tag{5.2.97}$$

is solved for the coarse grid correction, denoted by e^{coarse}. Since the transformed (coarse) linear system has less unknowns, it is cheaper for us to solve than the initial system on the fine grid.

Afterwards, we have to interpolate/prolongate the calculated coarse grid correction back onto the fine grid ($e^{coarse} \mapsto e^{fine}$). The former obtained approximation u_m can be updated by this correction, $u_m = u_m + e^{fine}$.
Finally, we perform another small number of smoothing iterations based on our approximation which was updated by the coarse grid correction.

This procedure may be repeated iteratively in order to solve a linear system or it can operate as a preconditioner. In the latter case the "initial guess of the linear system" equals the residual obtained at the current step of the outer iterative solver which is preconditioned by this MG routine.

In this work geometrical multigrid preconditioners are considered in order to enhance the performance of iterative solvers like GMRES 4.5.
In our matrix free context of a discontinuous Galerkin discretization of the linear advection-diffusion equation, we may choose an L_2-projection as interpolation/restriction operator. During an interpolation/prolongation step, when a parent element within the mesh is replaced by its child elements, an interpolation matrix may be applied, in order to prolongate the degrees of freedom from the parent element onto the functionbasis of the child elements. Furthermore, we are focusing on Krylov subspace methods as smoothers. We will justify this choice in the next chapter (6).

One iteration of the two level MG algorithm can be written as displayed in algorithm 5.1 (the superscript ^{fine} refers to the initial grid and the superscript ^{coarse} describes the (interpolated) components of the coarse grid linear system).

Algorithm 5.1 2-Level MG

Input: $A^{fine} := A \in \mathbb{R}^{n \times n}$, right-hand-side $f^{fine} \in \mathbb{R}^n$, initial guess $u_0 \in \mathbb{R}^n$

Output: approximation u_{j+1} of the solution of the linear system

$\tilde{u}_j = \textit{Pre-Smoother}\left(A^{fine}, u_j, f^{fine}\right)$ //Perform a fixed number of Smoothing Iterations

$r^{fine} = f^{fine} - A^{fine} \cdot \tilde{u}_j$ //Calculate residual at the *fine level*

$r^{coarse} = R^{coarse}_{fine}\left(r^{fine}\right)$ //Restrict the residual from *fine* to *coarse level*

$e^{coarse} = (A^{coarse})^{-1} \cdot r^{coarse}$ //Calculate coarse grid correction

$e^{fine} = P^{fine}_{coarse}\left(e^{coarse}\right)$ //Prolongate the residual from *coarse* to *fine level*

$\tilde{u}_j = \tilde{u}_j + e^{fine}$ //Apply coarse grid correction

$u_{j+1} = \textit{Post-Smoother}\left(A^{fine}, \tilde{u}_j, f^{fine}\right)$ //Perform a fixed number of Smoothing Iterations

Let us denote the residual, corresponding to the current approximation u_k, by $r_k = f - Au_k$ and the algebraic error of the approximation by $e^{fine}_k := u_{exact} - u_k$. Based on this notation, we can formulate the residual equation,

$$\begin{aligned} A^{fine} e^{fine} = A^{fine}\left(u_{exact} - u_k\right) = f - A^{fine} u_k = r^{fine}_k \\ \Leftrightarrow A^{fine} e^{fine} = r^{fine}_k. \end{aligned} \tag{5.2.98}$$

Most of the time, the exact solution and subsequently the error is not accessible. The residual, however, is a computational measure of the quality of the approximation. But unfortunately, a small residual norm does not necessarily imply a small error norm ([BHM00], p. 7f). Therefore, an important relation of the residual and the algebraic error is provided by the equation above (5.2.98).

The initial linear system $Au = f$ and the linear system 5.2.98 are described by the same system matrix $\left(A \equiv A^{fine}\right)$, therefore, the solution of both systems would be equally expensive. Regarding the residual equation, it enforces that the error has to satisfy the same set of equations as the vector of the degrees of freedom ([BHM00], p. 7f).

Depending on the underlying problem, some basic iterative methods (like Jacobi, Gauss-Seidel,

SOR, etc.) have a so called *smoothing property*. Regarding large sparse linear systems, these methods act as a (pre- and post-) smoother and are capable of reducing high-oscillatory error components effectively in just a few iterations.
However, low-oscillatory error components are not efficiently captured ([HKM06], p. 4f). So, the smoothers eliminate peaks in the error very well and leave the smooth low-frequency "wave-like" components behind.
The (smooth) algebraic fine grid error, which remains, may be approximated efficiently on a coarser grid, i.e. the coarse residual equation 5.2.97.
The smooth modes on the fine grid become more oscillatory if they are projected onto a coarser grid (see [BHM00], p. 36ff). For example, if the (spatial) step size doubles from the initial grid to the coarse grid, the smooth error modes on the fine grid have to be represented on the coarse grid with just as half as many. Therefore, the smooth error tends to become more jagged or "higher-oscillatory" in its coarse grid representation and for that reason, the iterative schemes (i.e. smoothers) become more effective once again.
So, if the convergence of the smoother on the fine grid slows, because the error became smooth and it is predominated by low-oscillatory error components, it is advisable to switch to a coarser grid.

The reference to "V-cycle" comes from picturing the track the algorithm takes through the hierarchy of the meshes, as shown in figure 5.2.

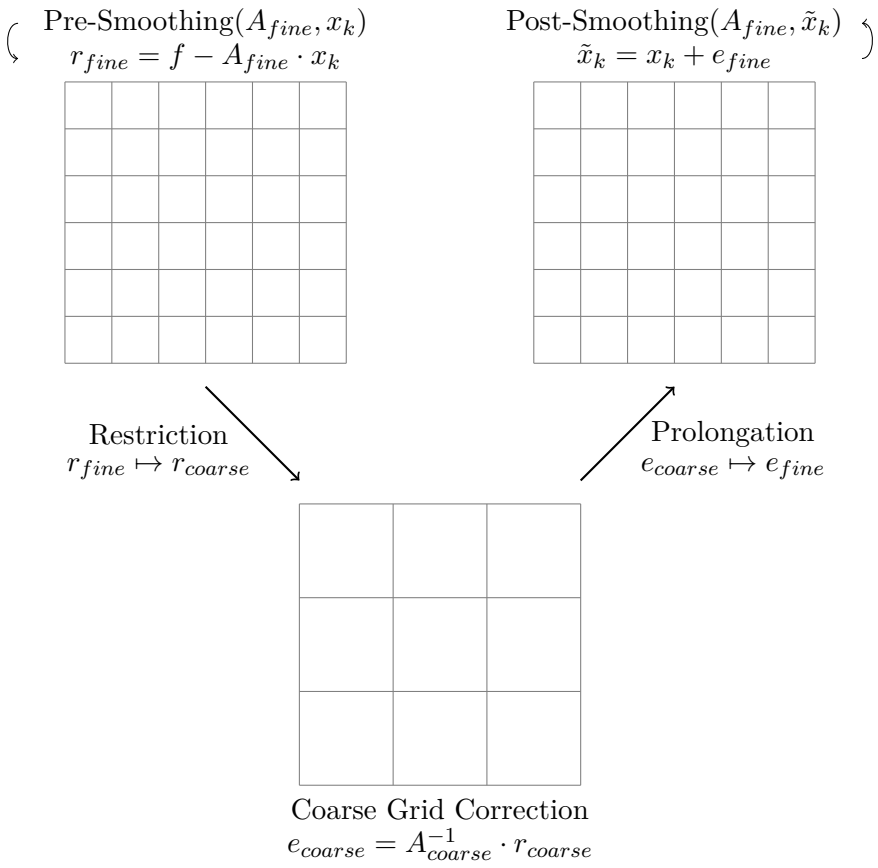


Figure 5.2: Multigrid V-cycle routine.

This routine is not bound to traverse just two grids, the grid hierarchy may be augmented to an arbitrary number of coarse grids.
The computational costs to manage additional coarse grids extends as it requires the generation of the next grid in the hierarchy and subsequently more interpolation and smoothing steps. On the other hand, the linear system on the coarsest mesh shrinks regarding the amount of unknowns, therefore, the calculation of the solution of the coarse grid correction gets continuously cheaper. A hierarchy of several coarse grids is shown in 5.3.

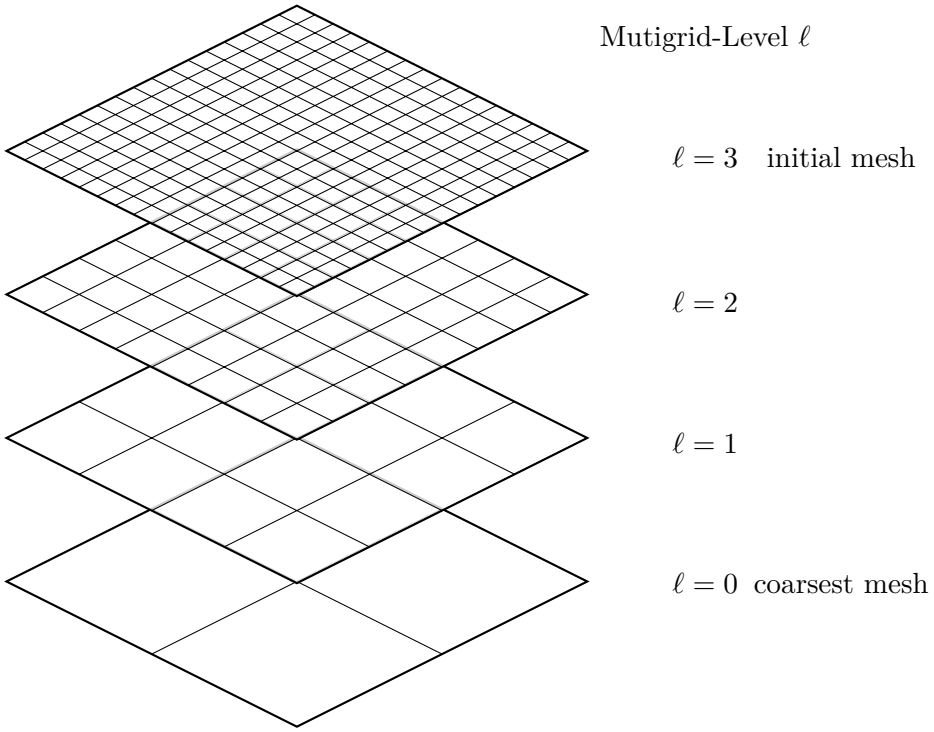


Figure 5.3: Multigrid hierarchy of coarse grids. From the initial mesh $\ell = 3$ down to the coarsest grid in the hierarchy at level $\ell = 0$.

On every grid in the hierarchy, except on the coarsest, on which the coarse grid correction is calculated, pre- and post- smoothing iterations are performed.

Let the superscript describe the multigrid level in the hierarchy of grids. So, $A^{(3)}$ (or A^{fine}) denotes the fine (initial) matrix of the linear system. And $A^{(0)}$ (or A^{coarse}) describes the (coarsened) matrix of the linear system on the coarsest level.
The general recursive form of the multigrid V-cycle is presented in algorithm 5.2 and corresponds to ([CHK20], p. 5).

Algorithm 5.2 MG V-Cycle

MG-V $\left(u^{(lvl)}, f^{(lvl)}\right)$

Input: initial guess $u_j^{(lvl)} \in \mathbb{R}^{n^{(lvl)}}$, right-hand-side $f^{(lvl)} \in \mathbb{R}^{n^{(lvl)}}$

Output: approximation $u_{j+1}^{(lvl)}$ of the solution of the linear system

if (*coarsest grid reached*) ***then***

  return $(A^{coarse})^{-1} f^{coarse}$

else

  $\tilde{u}_j^{(lvl)} \leftarrow$ *perform* $S_{pre} \in \mathbb{N}$ *Smoothing-iterations on* $A^{(lvl)} u_j^{(lvl)} = f^{(lvl)}$

  $f^{(lvl-1)} = R_{(lvl)}^{(lvl-1)} \left(f^{(lvl)} - A^{(lvl)} \tilde{u}_j^{(lvl)}\right)$

  $\tilde{u}_j^{(lvl-1)} = (0, \ldots, 0)^t$

  $\tilde{u}_j^{(lvl-1)} = \boldsymbol{MG\text{-}V}\left(\tilde{u}_j^{(lvl-1)}, f^{(lvl-1)}\right)$

  $\tilde{u}_{j+1}^{(lvl)} = u_j^{(lvl)} + P_{(lvl-1)}^{(lvl)} \tilde{u}_j^{(lvl-1)}$

  $u_{j+1}^{(lvl)} \leftarrow$ *perform* $S_{post} \in \mathbb{N}$ *Smoothing-iterations on* $A^{(lvl)} \tilde{u}_{j+1}^{(lvl)} = f^{(lvl)}$

  return u_{j+1}^{lvl}

end if

Additional to the number of grids used, even the scheme how the algorithm traverses the hierarchy may be adapted. In the previous paragraph, we have only shown the V-cycle. But once the hierarchy of the grids is established, we are able to introduce other traversing schemes. For example, if more than two grids are used within the algorithm, we can execute a so called *W-cycle* 5.4. The W-cycle comes with additional computing cost, but accelerates the convergence of MG in certain cases.

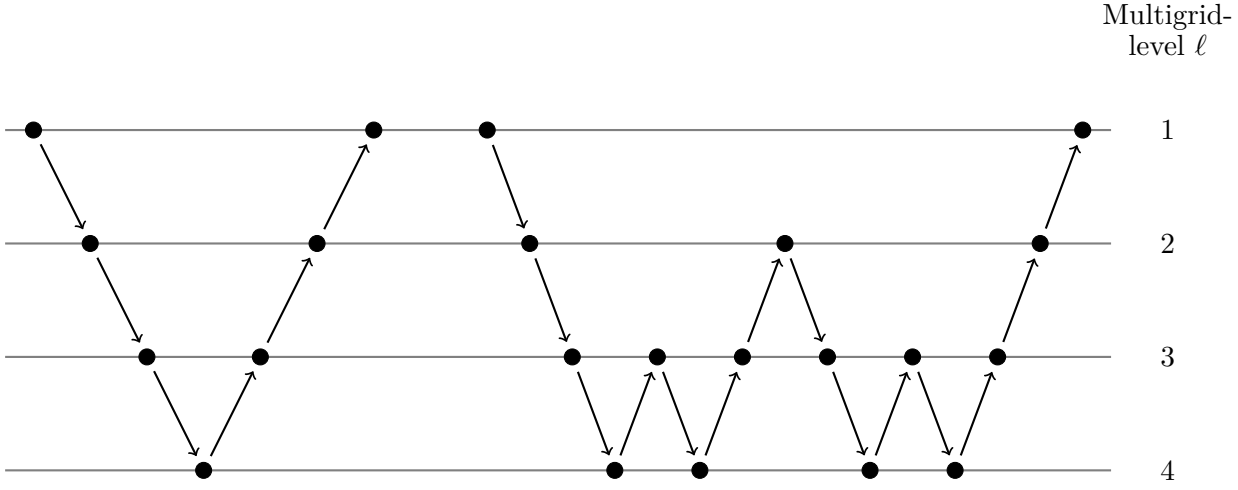


Figure 5.4: (Left:) Traversal of V-cycle, (Right:) Traversal of W-cycle.

Another traversal scheme is the *Full Multigrid V-cycle* (FMG). This multigrid cycle is made up of several V-cycles, in fact, each V-cycle is preceded by a coarse grid V-cycle. This provides a "good" initial guess for the successive V-cycles traversing the hierarchy all the way up to the (initial) fine grid.
Therefore, in the first place, the FMG needs to restrict the initial residual from the fine grid all the way down to the coarsest grid in the hierarchy. Afterwards, the successive V-cycles are performed. If the first V-cycle is finished, the calculated approximation is interpolated/prolongated one level up (towards the next finer grid) in the hierarchy, as an initial guess for the next consecutive V-cycle. In its recursive form the FMG can be displayed as in figure 5.5 ([BHM00], p. 43).

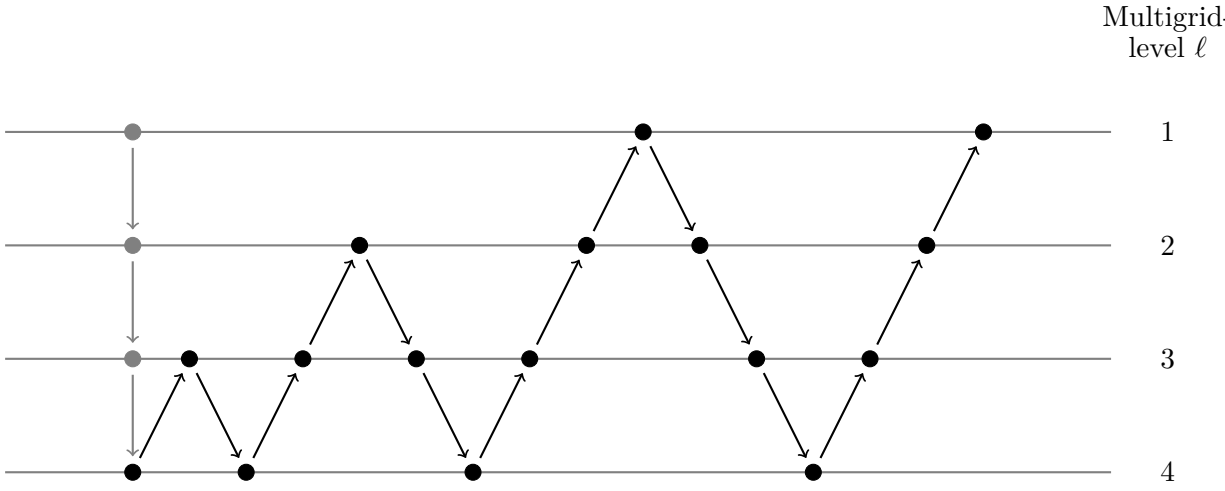


Figure 5.5: Traversal of the FMG cycle.

In contrast to the block preconditioning methods, it is not generally possible to formulate the application of the multigrid procedure as a single preconditioning matrix M. It is rather a constantly changing application. Because of this, a flexible GMRES method has to be used if we choose a multigrid preconditioner.

6. Numerical Results

Now, we connect the several components discussed in the previous chapters in order to simulate problems of the linear advection-diffusion equation. The computational numerical experiments were performed on a workstation, equipped with two 10-core Intel Xeon E5-2660V3 CPUs running at 2.6 GHz each and 128 GB RAM. If not stated otherwise, the experiments were performed on a single core.
On default and if not stated otherwise, PETSc utilizes a restarted version of GMRES/FGMRES, which is restarted after 30 iterations and iterates until a convergence rate of 10^{-5} in the residual norm is achieved. Accordingly, in the course of this chapter, we examine the impact of several preconditioners on iterative solvers, like GMRES (4.5) and BiCGstab (4.4), applied to a linear system of equations arising from an implicit time-stepping scheme (3) in order to perform the

time integration of a semi-discretization of the linear advection-diffusion equation by the means of the LDG method (2).
Regarding preconditioners, we are focusing on geometric multigrid preconditioners but also take a look on block preconditioners.
As mentioned in the introduction, for our LDG spatial discretization, we are using `t8dg`, it is implemented in a matrix-free fashion, therefore, we have to adapt the preconditioners and the iterative solvers to this matrix-free context. For those reasons, we have to exclude iterative solvers which only operate on hermitian problems, as well as such that are using the transpose of the system matrix A within the calculation.
On default, `t8dg` chooses the upwind flux in a 1D case and the Lax-Friedrich flux in a 2D and a 3D case as the numerical flux.

If not stated otherwise, we are concerning a hypercube $[0,1]^d$, for all dimensions $d = 1, 2, 3$, as our physical domain Ω. Furthermore, we are assuming periodic boundary conditions in all cases. In the LDG approach, other boundary conditions may be added by adjusting the numerical fluxes on the boundary of the physical domain.
With the help of `t8code` we obtain an underlying computational grid Ω^h consisting of the elements. We are focusing on quadrilaterals and hexahedrons as elements in 2D and 3D, respectively. And, especially, we want to observe the iterative solvers and preconditioners on adaptive meshes.

But, let us first examine the different time-stepping methods by briefly comparing explicit and implicit methods.
Therefore, we need to consider an initial value problem. In the following example, we would like to observe a 2D problem. As the initial spatial function at time $t_0 = 0.0$, we are choosing the cosine product

$$u\left(\vec{x}, t_0\right) \equiv u_{init}\left(\vec{x}\right) = \cos\left(2\pi x_1\right)\cos\left(2\pi x_2\right). \tag{6.0.99}$$

Because, we are looking at a constant (divergence free) velocity field $\vec{c}$ in x_1-direction with magnitude 1.0, the analytical solution of the linear advection-diffusion equation equals, in this case, at an end time of $T = 1.0$, to the analytical solution of pure diffusion, which is given by

$$u_{exact}^{\text{diff}}\left(\vec{x}, t\right) = e^{-\mathbf{d}(2\pi)^2 d \cdot t}\cos\left(2\pi x_1\right)\cos\left(2\pi x_2\right). \tag{6.0.100}$$

Here, $\mathbf{d}$ represents the diffusion coefficient. Let us first consider a relatively small diffusion coefficient (compared to the advection coefficient) of $\mathbf{d} = 0.0001$.
We are able to compare the L^2-error at the time $T = 1.0$ which arises by advancing the simulation with different time-stepping methods. In order to do so, we choose a small uniform 2D grid and increase the CFL number continuously, because the underlying grid does not change, we have a constant mesh size, consequently, higher CFL numbers lead to bigger time steps. As approximations we choose polynomials of order three on each element. Hence, with 64 elements within the grid and a tensor product structure of 3 LGL points per element per dimension, it leads us to a small system with 576 degrees of freedom.

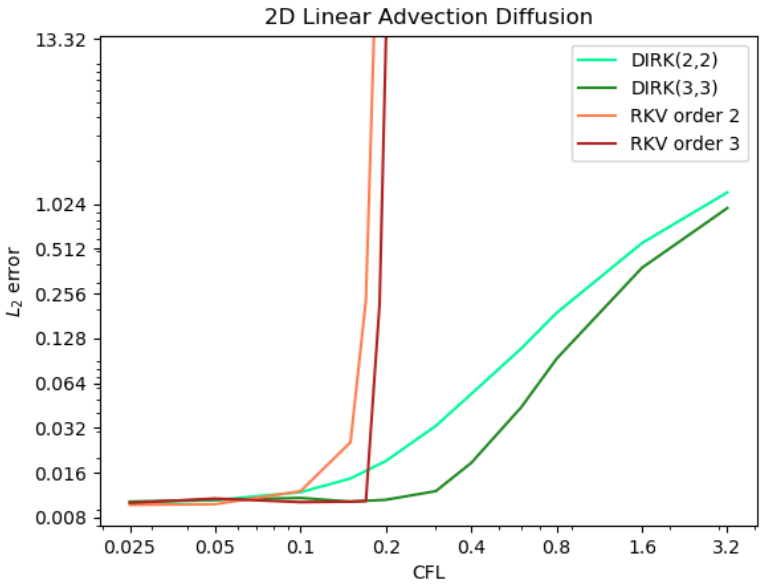


Figure 6.1: L^2-errors at the end time $T = 1.0$ of explicit Runge Kutta methods of order 2 ("RKV order 2") and order 3 ("RKV order 3"), as well as of the *DIRK(2,2)* of order 2 and *DIRK(3,3)* of order 3 in dependency of the CFL number.

In figure 6.1 we see that the errors lie within the magnitude of the discretization error up to a CFL number of approximately $CFL \leq 0.1$ for the different time-stepping schemes. But for a CFL number within the interval $[0.1, 0.2]$, the explicit Runge Kutta methods become numerically unstable, at the latest from a CFL number of approximate 0.19 onward.
In contrast to the implicit DIRK methods, which still produce stable approximations, although the accuracy suffers from bigger time steps ($\rightarrow$ higher CFL numbers). Because the spatial discretization in this example is of order 3, the explicit Runge Kutta method of order 2 and the DIRK(2,2) method produce higher errors from lower CFL numbers on, than the time stepping methods of order 3, because the error in the time integration predominates earlier. The initial values and the approximate solution obtained by applying the explicit and implicit time-stepping methods is displayed (in the appendix) in figure A.1.

As we mentioned earlier in chapter 4.4, this work mainly focuses on GMRES, although other iterative solvers may be applicable too in order to solve the linear system of equations.
Therefore, let us compare the performance of GMRES (4.5) and BiCGstab (4.4). We consider a three dimensional problem with a sphere step function as initial value function on the unit cube. The underlying adaptive mesh consists of $10,088$ hexahedrons. The linear advection-diffusion problem which we consider provides an advection velocity field in x_1 direction and a diffusion coefficient of $\mathbf{d} = 0.001$. The resulting systems of equations of the application of the DIRK(2,2) time-stepping method is solved by GMRES as well as BiCGstab. The iterative solvers run until a convergence tolerance of 10^{-5} of the residual norm is reached.
In figure 6.2 we notice that the iteration count needed by BiCGstab in order to solve the systems is considerably lower than the iterations needed by GMRES. However, its computational time

always exceeds the computational time needed by GMRES, although, BiCGstab has a constant iteration time, in contrast to GMRES. As we stated in chapter 4.4, the computational complexity per iteration within BiCGstab is higher than within GMRES.

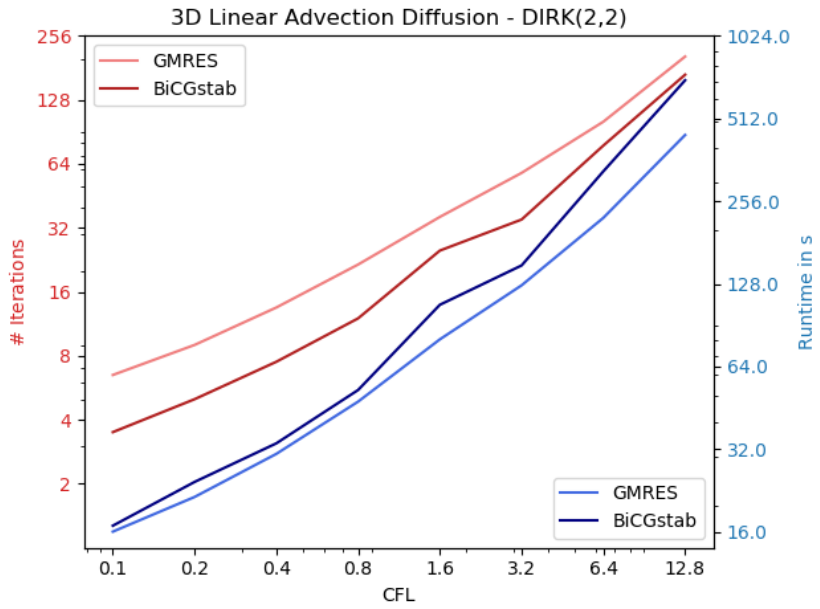


Figure 6.2: Comparison of the iteration count and the runtime of GMRES and BiCGstab in dependency of a rising CFL number. A linear advection-diffusion example on a 3D adaptive mesh is considered. The time-stepping method of choice was the DIRK(2,2) method. The plotted values of the iteration count are the averaged results per stage, whereas the runtime corresponds to the overall runtime of the iterative solver per time step.

From now on, we are focusing on GMRES as our iterative solver of choice.
Let us first examine different multgrid preconditioners. We want to examine the effect of a different number of coarse grids. Therefore, we need a hierarchy of coarse grids.

As our model problem, we choose an adaptive 2D case with a two dimensional rotating advection velocity field. As approximations on each element we choose polynomials of order 2 and use the implicit Euler method as our time-stepping scheme. We choose a CFL number of 2.0 for our simulation of a single time step.

We are establishing the following hierarchy of coarse grids, as shown in figure 6.3, on which the multigrid preconditioners operate. The initial mesh of the problem is shown in the top-left corner followed by the computed coarse grids. Starting from $5,092$ elements of which the initial mesh consists, the coarsening process leads us to a coarsest grid with only 16 elements.

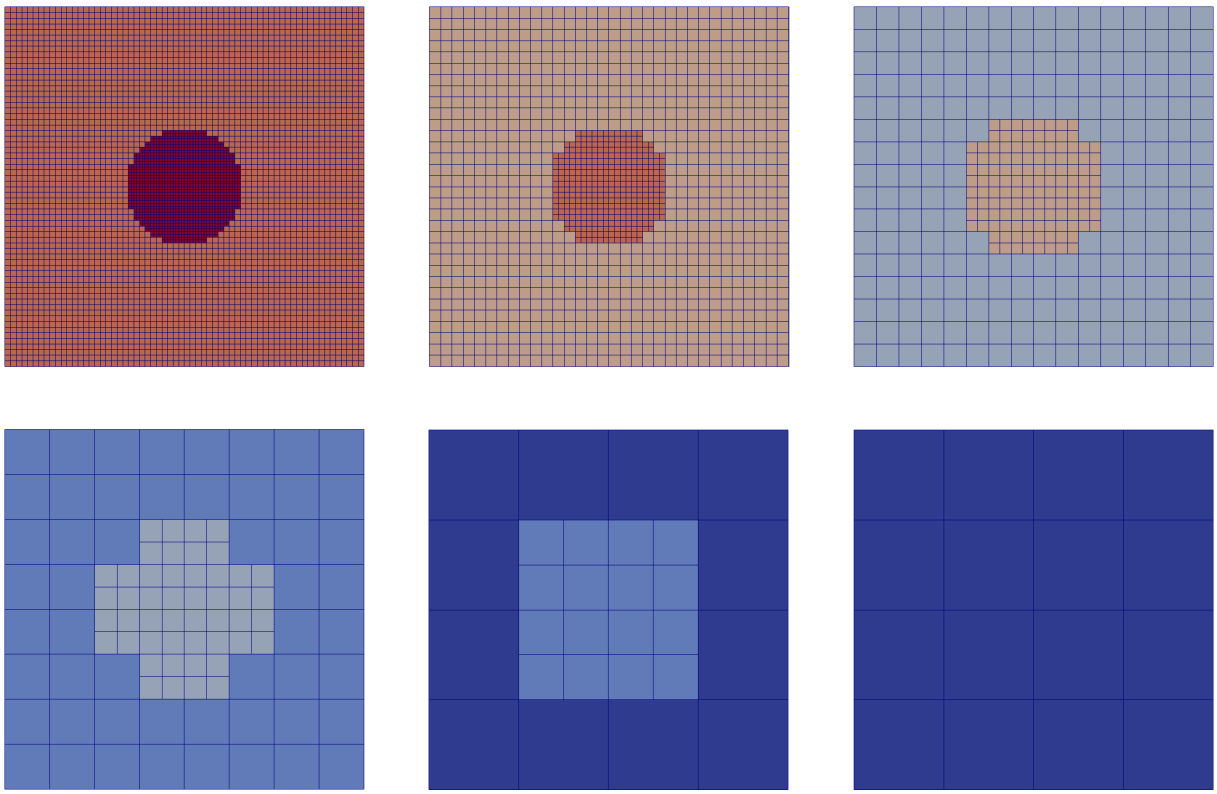


Figure 6.3: Coarse grid hierarchy starting from the initial mesh (top-left) to the coarsest grid in the hierarchy (bottom-right). The color corresponds to the refinement level of each element (red: refinement level = 7 to blue: refinement level = 2). The coarsening process consisted of replacing a family of elements (in 2D: four child elements) by its parent element in order to reduce this elements refinement level by one.

In the next step, we disaggregate the runtime of the different multigrid preconditioners in their subroutines. Additional to the initial mesh, the "2-LVL MG" preconditioner establishes one coarse grid, the "3-LVL MG" preconditioner establishes two coarse grids, and so on.
The portrayed runtime always includes the complete setup and solution of the (preconditioned) linear system of a (whole) single time step of the considered time-stepping method. Especially, the generation of the coarse meshes is also included within the displayed runtimes.
We choose 3 pre- and post-smoothing iterations, performed by a GMRES method, for each multigrid preconditioner on each level, except on the coarsest, on which the coarse grid correction is calculated.

In the figure 6.4 we see different multigrid preconditioners with a varying number of coarse grids in comparison, plotted against the diffusion coefficient (starting from no diffusion $\mathbf{d} = 0.0$ at all, up to a high diffusion coefficient of $\mathbf{d} = 1.28$).
The runtime is disaggregated into the different components of the multigrid procedure (smoothing, interpolation, coarse grid correction). A detailed table with iteration counts and runtimes can be found in B.2.

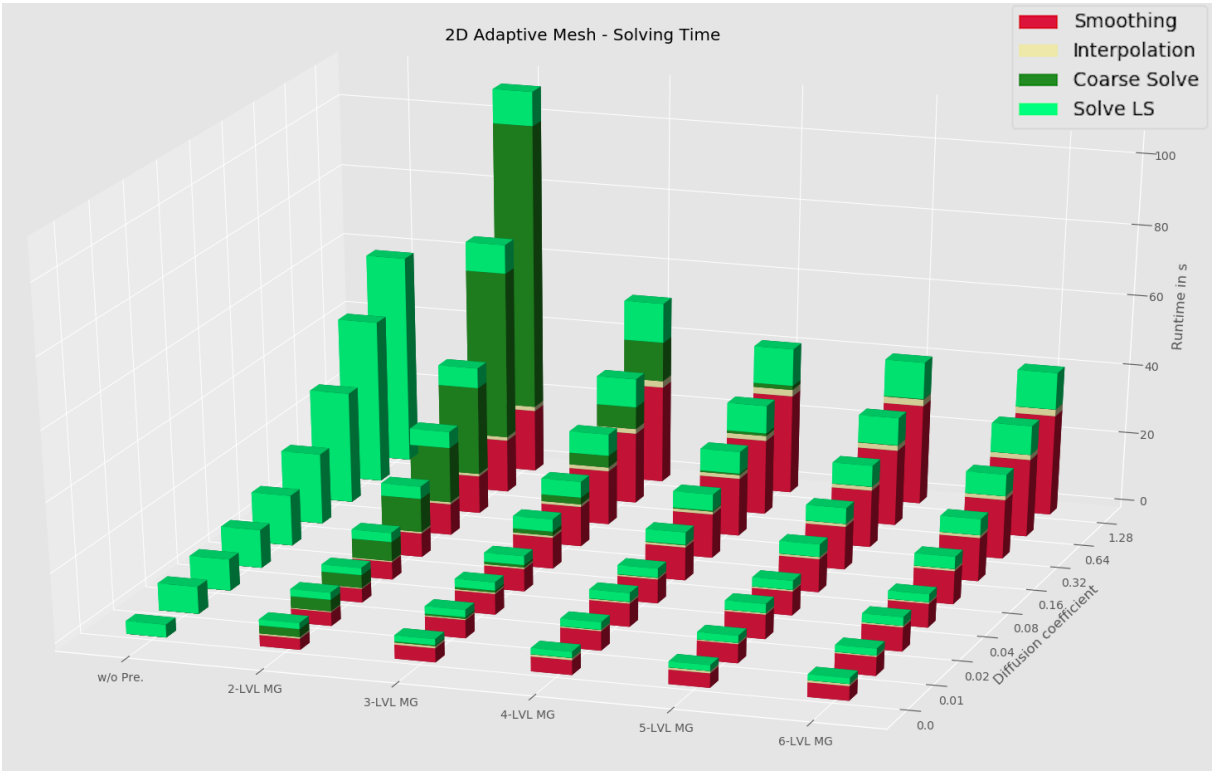


Figure 6.4: Runtime of one implicit time step using different multigrid preconditioners (with a varying count of coarse grids; from left to right: without preconditioning, "2-LVL MG"-,"3-LVL MG"-, ..., "6-LVL MG"-preconditioner). The runtime is disaggregated into the preconditioners' subroutines and plotted against a varying diffusion coefficient. Light green represents the time spent on solving the system on the initial mesh and dark green corresponds to the time needed in order to solve the coarse grid systems. Red collects the time spent on pre- and post-smoothing. Last but not least, yellow corresponds to the interpolation, i.e. restriction and prolongation between the grids in the hierarchy.

In the first place, it may be interesting to notice, that from a diffusion coefficient of $\mathbf{d} \geq 0.04$ onward the multigrid preconditioners with at least two additional coarse grids ("3-LVL MG", ..., "6-LVL MG") always lead to an overall faster solution of the whole time step, than solving the system without preconditioning.
The most eye-catching component of the figure 6.4 is the coarse grid correction (colored in dark green) of the "2-LVL MG" preconditioner. It even solely exceeds the computational time needed in order to solve the non-preconditioned system (most-left row). We notice that the coarse grid correction component significantly reduces if more than one coarse grid is incorporated, until it almost vanishes within in the "5-LVL MG" and "6-LVL MG" preconditioner.
This is due to the fact, that if the next coarse level is added the multigrid preconditioner does not loose performance and nearly provides a constant iteration count "independently" of the number of coarse grids. GMRES is also a good smoother which helps keeping up the convergence for very coarse grids. If the table of these results is regarded, an approximately eight-fold reduction

in the computational time of the coarse grid correction is gained if an additional coarse level is considered.

Overall, the interpolation (colored in yellow), consisting of restriction and prolongation between the multigrid levels, is rather cheap, as the least time is spent within this sub-routine. Even scaling up the amount of coarse levels, yields only to a small increase in the interpolation time.

The time spent in the Smoothing process (colored in red), i.e. pre- and post-Smoothing, grows as expected when additional coarse grids are considered. It is an important subroutine worth to be optimized during the implementation, as it takes up the biggest part of the computational time if two or more coarse grids are considered.

As a summarizing remark, we may want to choose more than one coarse grid when applying a multigrid preconditioner, due to the immense cost of solving a large coarse grid system at every iteration if only one coarse level is used. However, building up a hierarchy of grids all the way down to the coarsest grid possible is also not worthy, because at a certain mesh size a further efficiency boost by adding another coarse grid fails to appear, i.e. compare "5-LVL MG" and "6-LVL MG" in 6.4 and B.2.

Birken et al. ([BBJ16]) stated that symmetric Gauss-Seidel and, especially, a 2-step GMRES method are effective and interesting smoothers for multigrid methods if high order finite volume discretizations of the linear advection-diffusion equation are regarded. These smoothers are assumed to be also "attractive candidates" within a discontinuous Galerkin context.
Kanschat ([K08]) examined smoothers for multigrid preconditioners for high order discontinous Galerkin discretizations for the linear advection-diffusion equation. He concluded that block Gauss-Seidel smoothers yield efficient and robust multigrid preconditioners if the ordering of the degrees of freedom complies with the flow of the advection-velocity field. Furthermore, a block Jacobi smoother is not robust if problems with slight diffusion are considered and a point Gauss-Seidel smoother deteriorates the effectiveness for higher orders.
In our framework we inherit the element ordering and subsequently the ordering of the degrees of freedom from `t8code` and `t8dg`. Generally, this ordering does not comply with the flow of the velocity field of a considered problem and we are rather limited in regards to reordering.
Therefore, we stick to a Krylov subspace method as a smoother within the multigrid preconditioner, namely GMRES.

In this example at hand, we chose three GMRES iterations within our smoothing process, because it led us to more efficient results. However, if higher order approximations are considered, a reduction to two smoothing iterations may be practical. Also, because the system becomes substantially larger, when higher orders are considered. This may lead to a significantly increase in the computational time of the smoothing subroutines.

Therefore, let us examine the effect of high order discretizations and a change in the amount of smoothing iterations. We regard the above problem and fix the diffusion coefficient at $\mathbf{d} = 0.001$ and see how the "4-LVL MG" preconditioner with 3 coarse grids performs if one, two, three, or four pre- and post-smoothing iterations are applied.

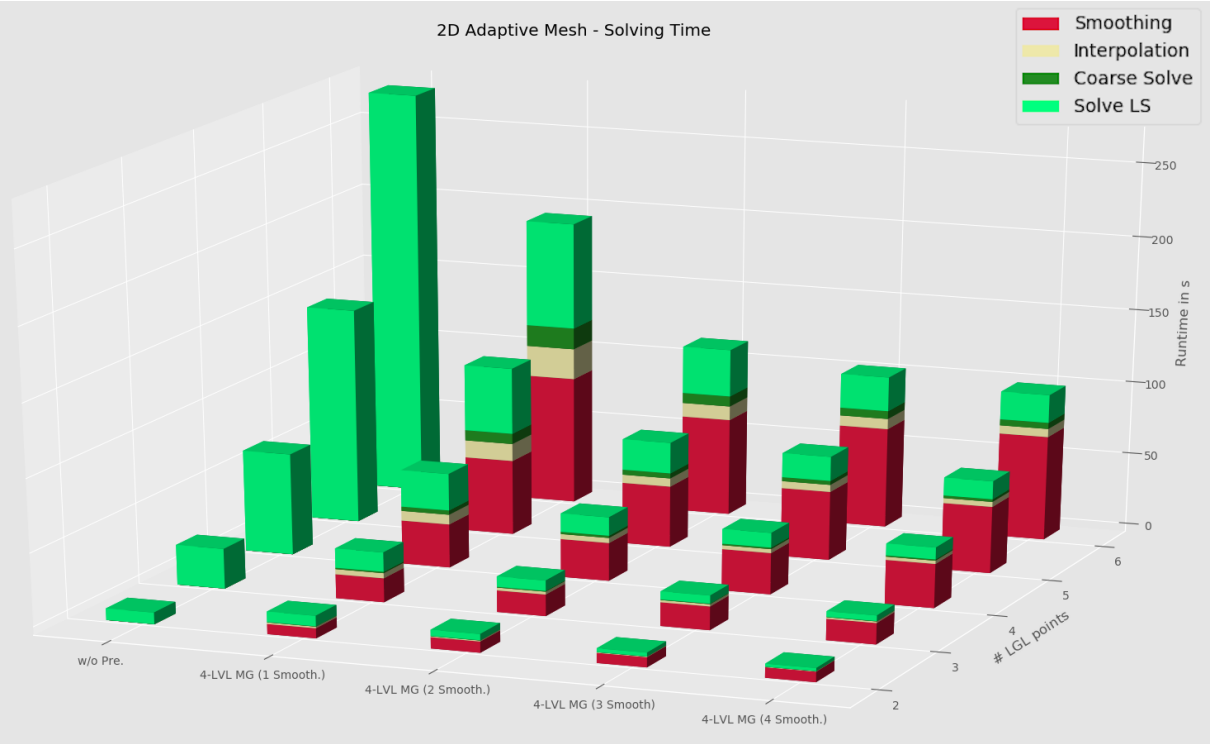


Figure 6.5: Comparison of a "4-LVL MG" multigrid preconditioner with a varying amount of smoothing iterations (from left to right: without preconditioning, "4-LVL MG" with 1 smoothing iteration, ..., "4-LVL MG" with 4 smoothing iterations), in dependency of the order of the approximation (number of LGL points per element per dimension).

In figure 6.5 and the table B.1 we compared GMRES as a smoother with a varying amount of pre- and post-smoothing steps; from one smoothing iteration up to four smoothing iterations. Obviously, the more smoothing iterations we perform, the fewer outer iterations are needed by the iterative solver. We notice that from an order of 3 onward (from order 4 onward in case of one smoothing iteration), the preconditioner effectively reduces the runtime.
For high orders the time spent on smoothing reaches its maximum in case of only one smoothing iteration, this is due to the relatively high amount of outer iterations, in contrast to the smoothers with more than one smoothing iteration. This also drastically increases the computational time for the interpolation steps and coarse grid corrections.
In the context of a multigrid smoother, the more smoothing iterations are performed, the stronger high-oscillatory error components in the residual are smoothed out.
The resulting fewer outer iterations needed by the iterative solver, when many smoothing iterations are performed, remedy for the higher smoothing cost per iteration. Also, less overall outer iterations yield fewer interpolation steps and fewer coarse grid solutions.
Generally, we experienced three smoothing iterations with GMRES as a good "black-box" amount of smoothing iterations throughout the experiments, although, a GMRES smoother with four smoothing iterations led to the best performance corresponding to the example B.1.

Until now, all multigrid preconditioners performed a V-cycle through the grid hierarchy, but we

have seen that also other gird-traversing schemes exist and may be advantageous due to a faster convergence. Therefore, we compare the multigrid preconditioners in a V-cycle, W-cycle, and FMG-cycle.
Therefore, we are considering a highly adaptive underlying mesh whose elements' refinement levels differ up to three levels. The grid consists of 6160 elements. The grid as well as the initial value function are displayed in figure 6.6.

Let us consider a similar problem of the linear advection-diffusion equation as above with a diffusion coefficient of $\mathbf{d} = 0.0001$ and a rotating advection velocity field. We choose polynomials of order three as approximation on each element.
In figure 6.7 we displayed the iteration count and the runtime that the iterative solver needed in order to solve the system of a single time step arising from the implicit Euler method. The obtained approximations at some given CFL numbers are displayed in C.1.
We compare the case in which no preconditioning is applied with a case in which a "5-LVL MG" preconditioner is applied in a V-cycle, W-cycle, and, FMG-cycle. In each case we chose three pre- and post-smoothing iterations.

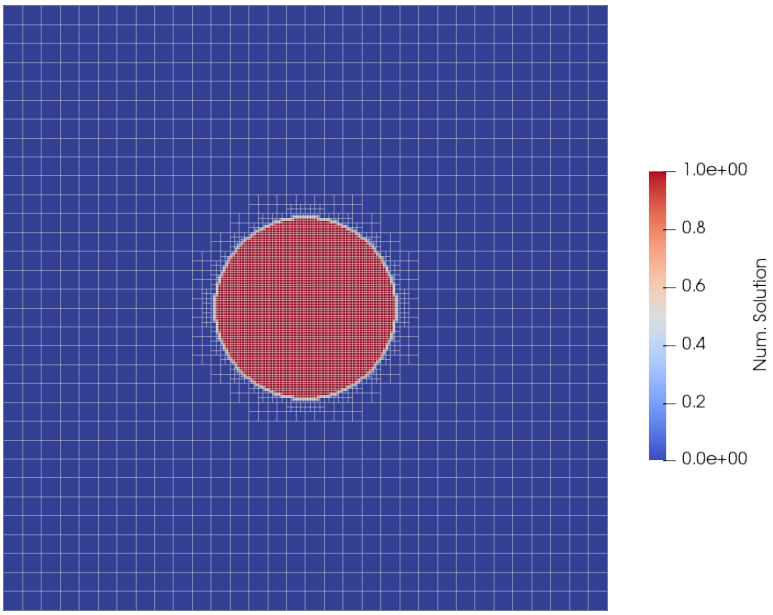


Figure 6.6: 2D adaptive grid with 6160 elements. The mesh contains elements whose refinement level differ from 3 to 6. The initial value function resembles a circle step function.

In figure 6.7 we are displaying the iteration count and runtime of an FGMRES iterative solver applied to linear system which is preconditioned by a multigrid preconditioner using different cycle types. We notice that up to a $CFL \leq 2.0$ the multigrid preconditioner results in the same iteration count for the V- and W-cycle. If an FMG-cycle is chosen, the outer iteration count decreases even further and leads for small CFL numbers to the fewest iterations. Due to the several additional smoothing steps, interpolation steps and coarse solves which are performed within a W-cycle and FMG-cycle, the runtime is considerably higher than in their V-cycle counterpart.

From a CFL number of $CFL \geq 4.0$, both, the multigrid preconditioner with a W-cycle and with a FMG-cycle lead to fewer outer iterations due to their faster convergence than the V-cycle, but the runtime still substantially exceeds the MG V-cycle runtime for any given CFL.

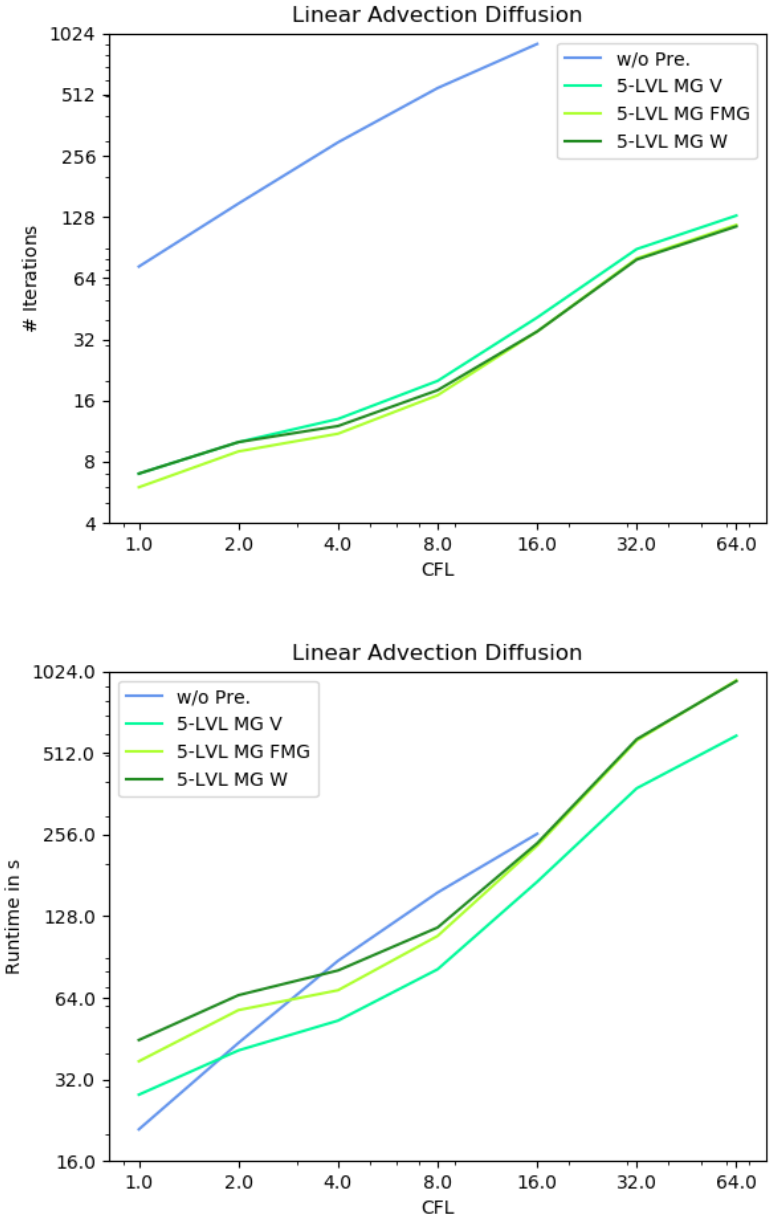


Figure 6.7: (Top:) Comparison of iterations needed by the outer iterative solver if the specified preconditioning techniques are applied (i.e. without preconditioning, "5-LVL MG" in a V-cycle, "5-LVL MG" in a FMG-cycle, "5-LVL MG" in a W-cycle)
(Bottom:) The corresponding runtime in order to obtain the solution of the linear system if the specified preconditioning techniques are applied.

From a CFL number of $CFL \geq 8.0$ onward, the W-cycle and the FMG-cycle result (more or less) in the same iteration count and runtime.
At the latest from a $CFL \geq 16.0$ onward, the iterative solver diverged if applied to the not-preconditioned system, whereas the MG preconditioners still yield a robust application.
Tests with a W-cycle/FMG-cycle with fewer smoothing iterations and/or fewer coarse levels did not lead to more efficient results.
Generally, disregarding the fewer outer iterations needed if a W-cycle MG or a FMG-cycle MG is applied (especially, for high CFL numbers), the application of a plain V-cycle MG yield substantially faster results.
Even from a $CFL \geq 1.8$ onward the application of a multigrid V-cycle to the linear system results in runtime advantages concerning the iterative solver, in comparison to the not-preconditioned system. While the application of a multigrid preconditioner in W- or FMG-cycle yield runtime advantages not before a $CFL \geq 4.0$.

However, interesting to notice is that the FMG-cycle performs not only in terms of the iteration count but also in terms of the runtime better than the W-cycle. And although, the V-cycle has a slower convergence, the solution of the linear system is obtained significantly faster.

After we have examined multigrid preconditioners in several variations, let us now see how the different preconditioners behave throughout the dimensions.

Therefore, we are starting in 1D with a line and a step function as initial value function and consider periodic boundary conditions. We discretize the physical domain into 32 uniform line segments. Every time a dimension is added, we consider a tensor-product structure of the 1D case. Ergo, we obtain 32^2 and 32^3 elements in 2D and 3D, respectively. The meshes and initial functions of the 2D and 3D case are shown in figure 6.8.

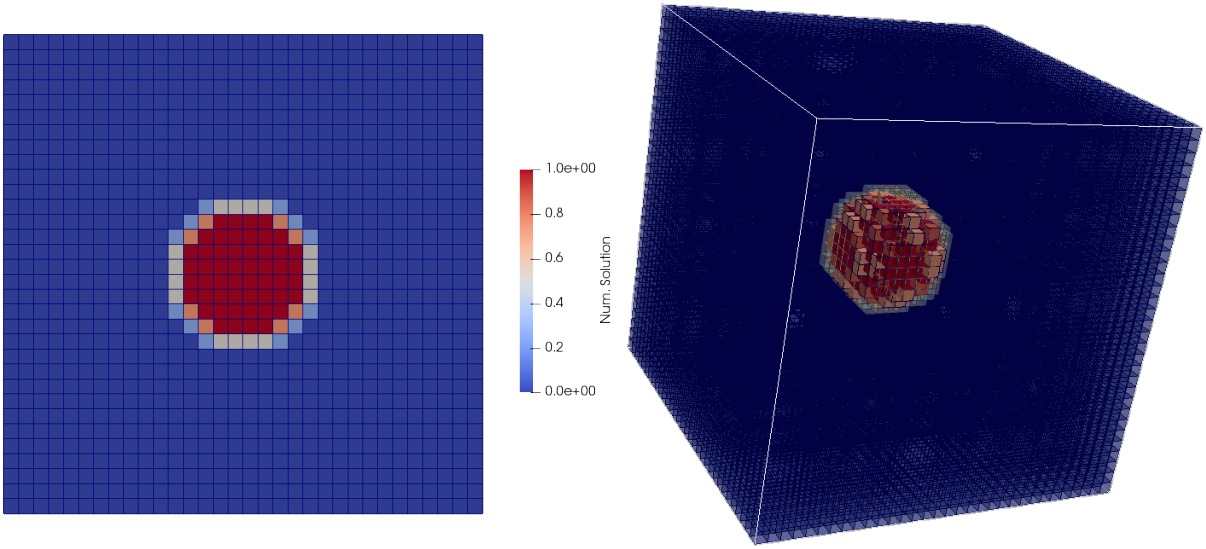


Figure 6.8: (Left:) 2D Mesh with $1,024$ elements and an initial circle step function.
(Right:) 3D Mesh with $32,768$ elements and an initial sphere step function.

We choose 3 LGL points per element per dimension, such that we obtain polynomials of order 3 as approximations on each element. We compare the (averaged) iterations needed by the iterative solver (GMRES) in order to solve a single time step. The implicit systems of this step result from the DIRK(2,2) method. In the figure 6.9 the average iteration count per stage, regarding the two stages of the DIRK(2,2) method, is displayed in dependence of the considered dimension. The stride of the simulated time step was chosen subject to a $CFL = 1.0$.

As we see in figure 6.9, the preconditioners do well in reducing the iterations needed by GMRES. Especially, in the case of pure advection, Block-Jacobi, Block-Gauss-Seidel, and the 3-Level multigrid preconditioner reduce the iterations by a significant amount. However, if diffusion is added, only the multigrid preconditioner retains the effectiveness as in the purely advective case. The application of the Block-Jacobi preconditioner becomes significantly more ineffective while the performance of the Block-Gauss-Seidel preconditioner only slightly suffers in comparison to the case of pure advection.
We notice, that the multigrid preconditioner captures the diffusion really good, whereas the Block-Jacobi preconditioner is mostly affected by the additional diffusion.

In the above paragraph we have examined the preconditioners' behavior when we are augmenting the dimension of the problem, i.e. 1D to 3D. Nevertheless, we have kept a constant velocity field $\vec{c}$ "flowing" only in x_1-direction, independently of the dimension of the problem.
In figure 6.10 we want to explore how another velocity field $\vec{c} = (1.0, 0.25, 0.25)^t$ may affect the preconditioners. Therefore, we extend the velocity field if we augment the dimension of the problem by a corresponding component in x_2- and x_3-direction, respectively.

We observe in figure 6.10, that the Block-Jacobi preconditioner is mainly affected by the change of the velocity field. The other variants adapt easily to the change and result in a slightly higher iteration count as in the previous case.
If the velocity field flows in each spatial dimension, and not only in x_1-direction, the elements are stronger coupled among each other and more information is transferred via the numerical flux, This information is not captured well by the block Jacobi preconditioner and, therefore, the effect of its application is rather weak. Especially in the 3D case, it has a deteriorating effect.

We want to emphasize that the exemplary simulations above were performed with a CFL number of only $CFL = 1.0$, which results in rather small time steps. Since we are examining implicit methods, we want to choose large time steps, which correspond to high CFL numbers.

Therefore, we are considering the prior conducted linear advection-diffusion problem, but let us ease the complexity of the system by decreasing the order of the approximations on the elements by one. In the figure 6.11 we analyze the effect of high CFL numbers of $CFL = 10.0,\ 100.0$.
In fact, for a CFL number ≥ 10.0 the outer GMRES iterative solver diverged if the system was preconditioned by the Block-Jacobi method. Aside from that, the application of the Block-Gauss-Seidel preconditioner led also to divergence for a CFL number ≥ 100.0. Only the multigrid preconditioner remains as a reliable as well as effective option.
We notice that the Block-Jacobi and the Block-Gauss-Seidel preconditioner deteriorate the solvability of the system (regarding the iterative solver), the higher we choose the CFL number. In case of the very high CFL number of $CFL = 100.0$ only the multigrid preconditioner preserves the solvability and decreases the outer iterations needed by the iterative solver drastically.

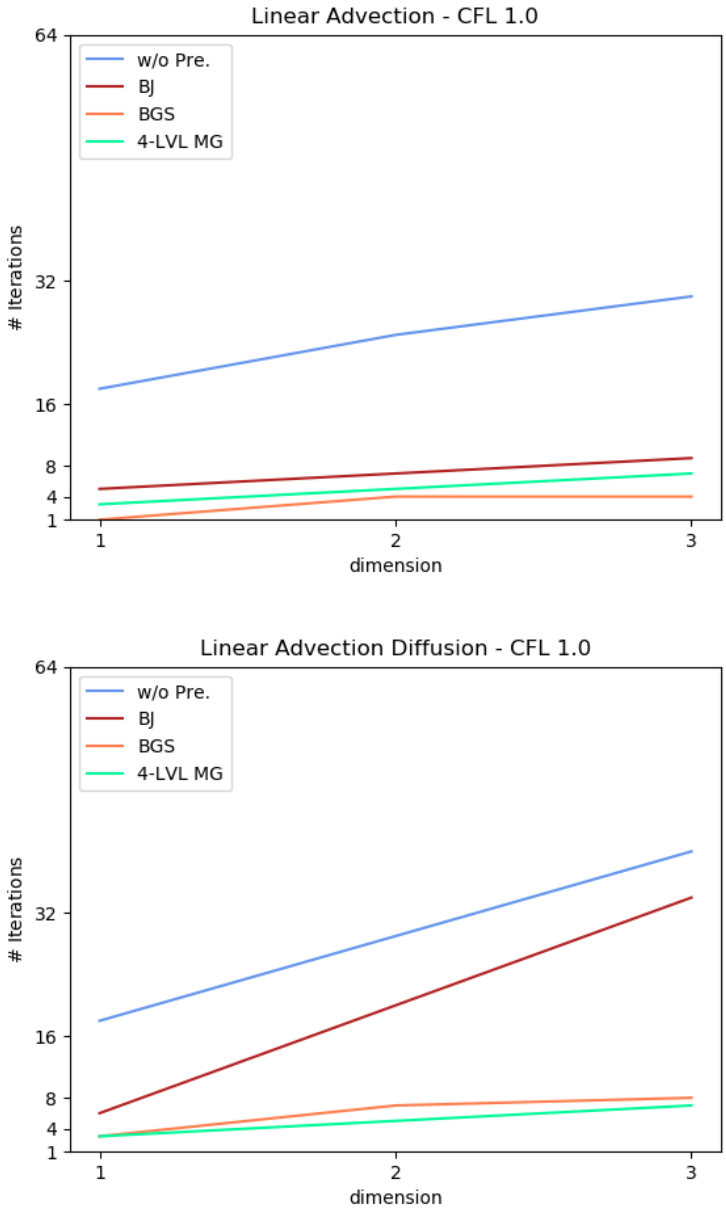


Figure 6.9: Average iteration count needed by GMRES in order to solve the implicit systems resulting from the DIRK(2,2) method.
(Top:) Linear advection problem (constant velocity field $\vec{c} = (1.0, 0.0)^t$).
(Bottom:) The same linear advection problem extended by an additional diffusion coefficient of $\mathbf{d} = 0.001$.

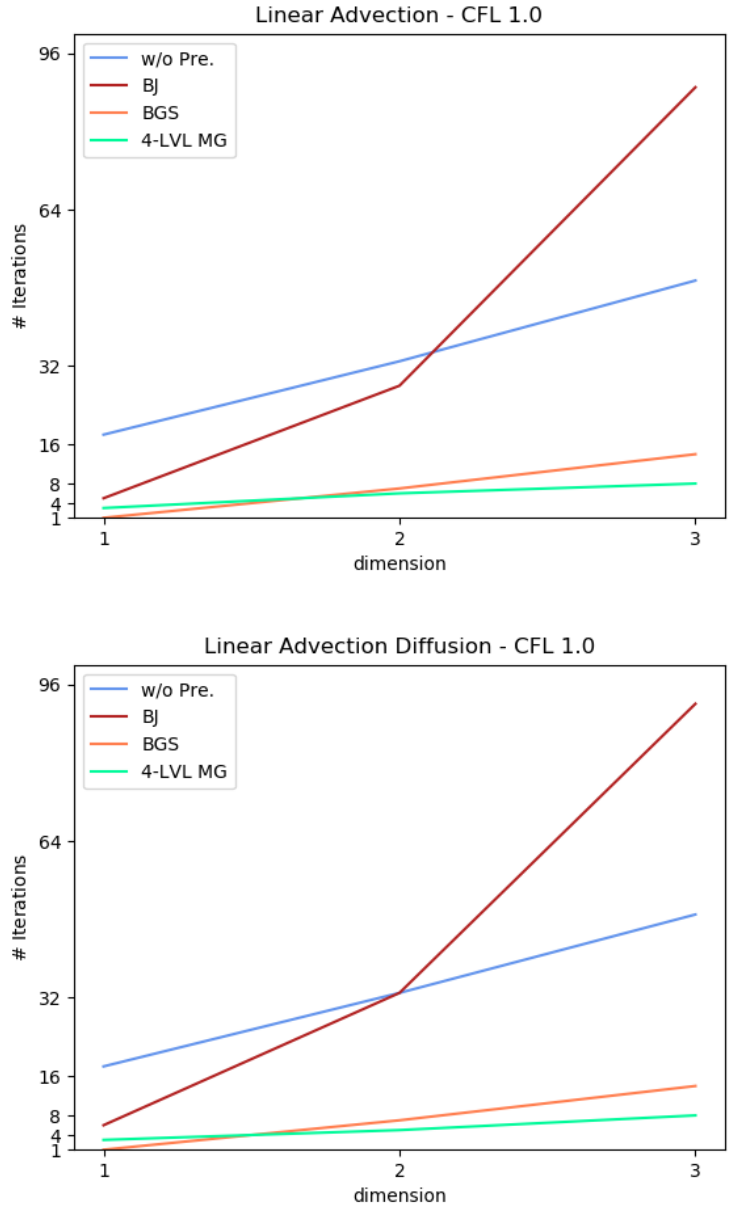


Figure 6.10: Average iteration count needed by GMRES in order to solve the implicit systems resulting from the DIRK(2,2) method.
(Top:) Linear advection problem (velocity field with flow in every spatial dimension).
(Bottom:) Linear advection-diffusion problem, with an additional diffusion coefficient of $\mathbf{d} = 0.001$.

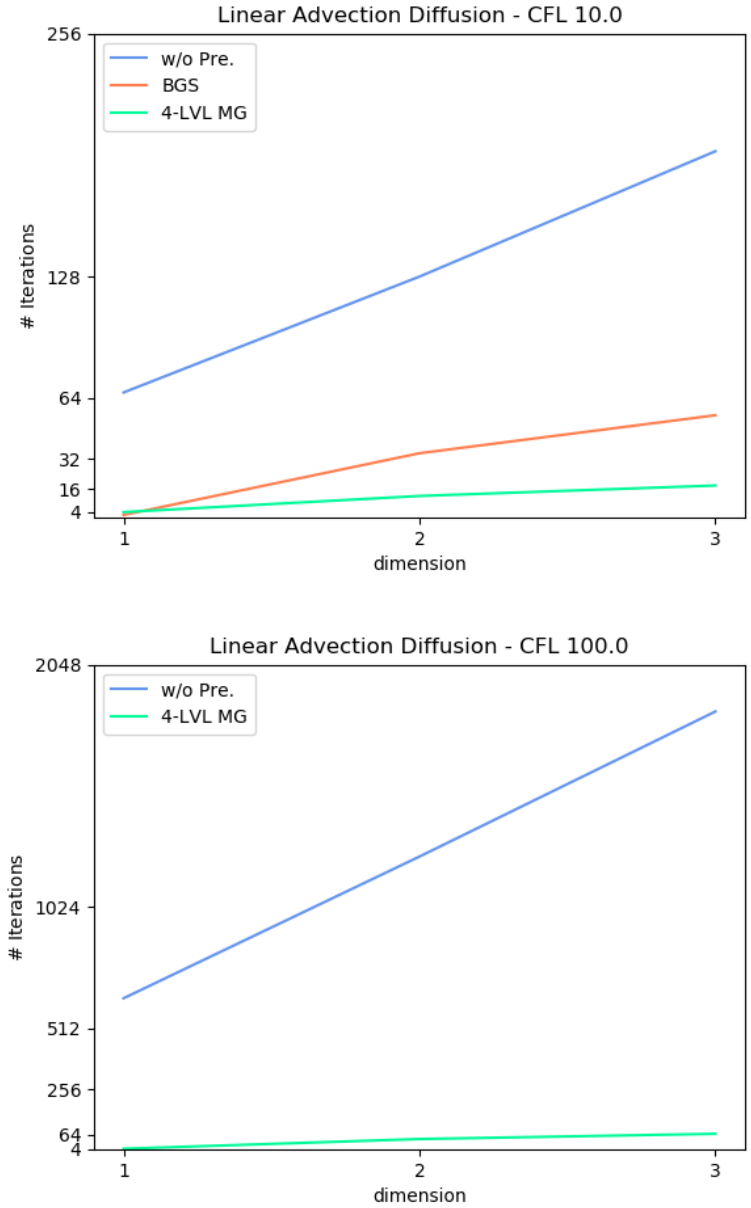


Figure 6.11: Average iteration count needed by GMRES in order to solve the implicit systems resulting from the DIRK(2,2) method.
(Top:) Linear advection-diffusion problem with a $CFL = 10.0$ (the BJ preconditioned system leads to divergence).
(Bottom:) Linear advection-diffusion problem with a $CFL = 100.0$ (the BJ preconditioned system as well as the BGS preconditioned system leads to divergence).

As a conclusion of the previous examination, we remark, that if we consider a linear advection-diffusion problem with a (potentially) non-trivial advection-velocity field (which not only flows in one spatial direction) and if we are interested in high CFL numbers, only the multigrid preconditioners yield reliable results by reducing the iteration count (as well as runtime in several

cases) and even enhance the robustness of the outer iterative solver (as we will see in the next section).

Therefore, we are comparing the application of a "4-LVL" multigrid preconditioner (with 3 additional coarse grids) in a V-cycle to the not-preconditioned system. We are also interested in how the multigrid prconditioner performs if we increase the order of the time stepping scheme from order 1 (implicit Euler) to order 3 (DIRK(3,3)).
The DIRK(2,2) method employs 2 stages and the DIRK(3,3) method employs 3 three stages, respectively. We compare the average stage of these methods with the ("single stage" of the) implicit Euler method.

We are considering a 3D problem of the linear advection-diffusion problem on an adaptive grid with $6,896$ elements (whose elements differ in the refinement level by three levels) and an polynomial order of 2 for the approximation on each elements, resulting in $55,168$ degrees of freedom. We choose an advective velocity field which constantly flows in each spatial direction ($\vec{\mathbf{c}} = (1.0, 0.25, 0.25)$) and a diffusion coefficient of $\mathbf{d} = 0.0001$.
The adaptive mesh with the initial value function is displayed in figure 6.12.

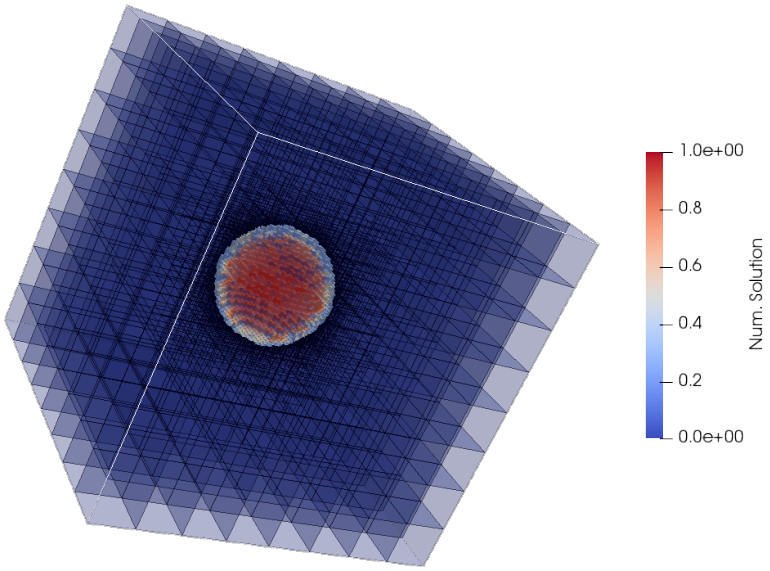


Figure 6.12: 3D adaptive mesh consisting of $6,896$ elements. An initial sphere step function of a linear advection-diffusion problem is shown.

In figure 6.13, we are displaying the iteration count needed by the (outer) iterative solver (FGMRES) in order to solve a single time step of the problem. The iteration count portrayed for the DIRK methods is averaged over two and three stages, respectively. This hints at the complexity of the (average) linear system which is solved at each stage of these methods in comparison to the system resulting from the implicit Euler method.

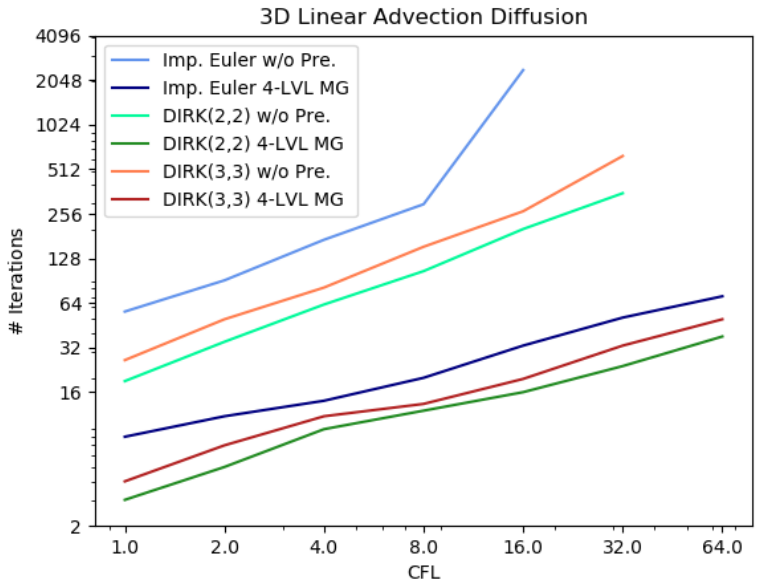


Figure 6.13: Iteration count of FGMRES applied to linear system applied to linear systems arising from all considered time-stepping methods. Comparison between the non-preconditioned systems and the application of a "4-LVL MG" preconditioner.

In 6.13 we notice that the iteration count of an average DIRK(2,2) stage is the lowest in both cases, preconditioned as well as not-preconditioned, while the iterative solver needs the most iterations for a time step of the implicit Euler method.

Within the DIRK methods, the whole time step is split into "smaller" time steps which are calculated at the respective stages within the methods.
Especially, if high CFL number are chosen, this reduces the complexity of the system in contrast to the implicit Euler method, which calculates the whole time step at once.
Although the DIRK(3,3) methods splits the time step into even more stages than the DIRK(2,2) method, the average stage of the DIRK(3,3) method is "harder" to solve than the average stage of the DIRK(2,2) method.

If we observe 6.14, the first thing to notice is that the DIRK(2,2) method overall leads to the fewest runtime needed by the iterative solver (FMGRES) if a whole time step is considered, for the preconditioned as well as for the not-preconditioned system.
We also observe that (for all time stepping methods) the preconditioned system enhances the robustness of the solver, since the not-preconditioned system leads to an earlier divergence of the solver. The solver diverges at the latest for a $CFL \geq 16.0$ regarding the implicit Euler method and at the latest for a $CFL \geq 32.0$ regarding both DIRK methods.
However, although the multigrid preconditioner effectively reduces the iteration count 6.13, real runtime advantages of the application of the "4-LVL MG" preconditioner occur from a $CFL \geq 4.0$ onward regarding the implicit Euler method and from about a $CFL \geq 16.0$ onward if one of the DIRK methods is considered.

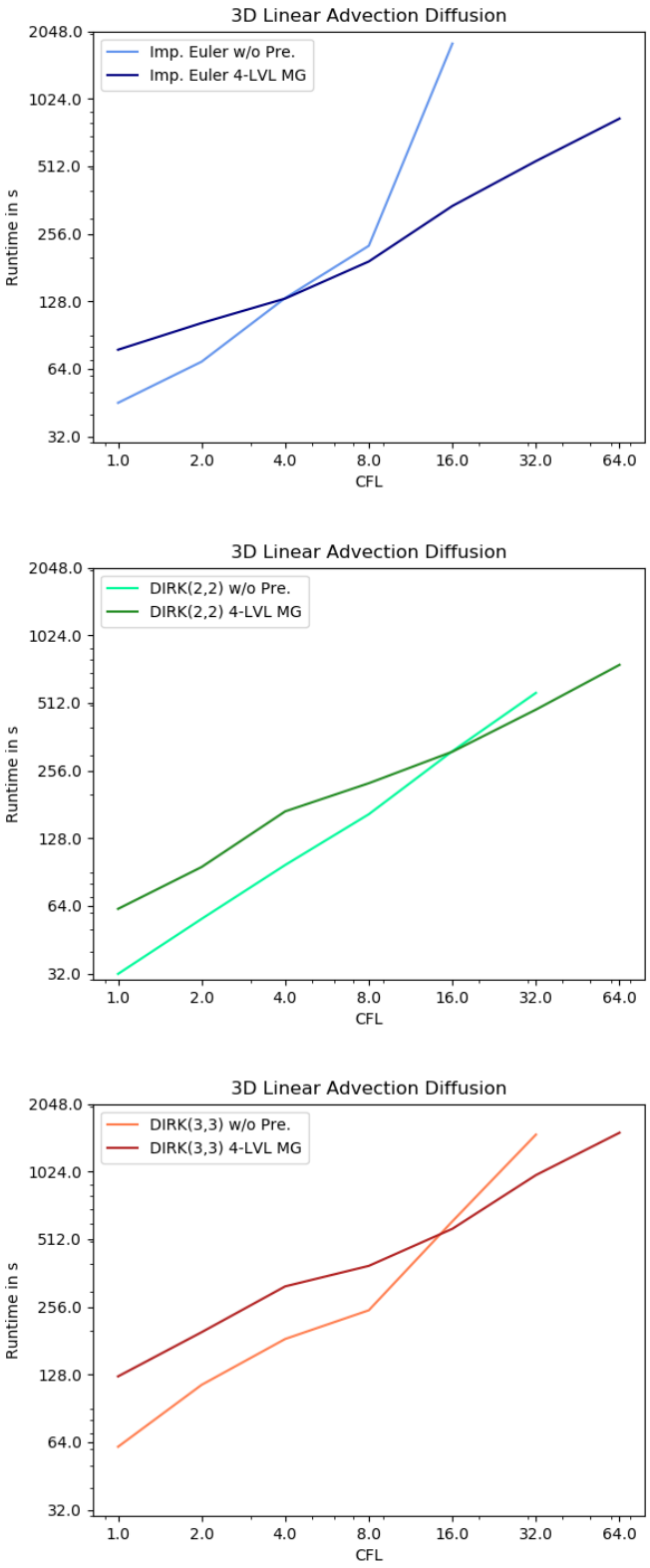


Figure 6.14: The runtime (in s) needed by the iterative solver (FMGRES) in order to solve a whole time step if different time integration methods are used (all stages accumulated). Comparison between the preconditioned and not-preconditioned system. (Top:) Implicit Euler method, (Middle:) DIRK(2,2), (Bottom:) DIRK(3,3).

7. Conclusion and Outlook

We extended `t8dg`, a matrix-free discontinuous Galerkin based solver for the linear advection-diffusion equation, by implicit time-stepping methods and block preconditioners as well as geometric multigrid preconditioners. The source code originated from this thesis is found in [t8dgIS].

We performed several experiments in 1D, 2D, and 3D with block preconditioning methods, i.e. Block-Jacobi and Block-Gauss-Seidel, as well as with different geometric multigrid preconditioners.
We experimented with uniform as well as adaptive meshes and different advection velocity fields and varying diffusion coefficients.

Let us first summarize the impressions we observed regarding the block preconditioning methods. Without an explicitly given matrix A of a linear system $Au = f$, extracting and inverting the preconditioning matrix becomes more costly, especially, in case of the Block-Gauss-Seidel preconditioner.
The Block-Gauss-Seidel preconditioner (BGS) affects the linear system and the iterative solver well in terms of the iteration count, the iterative solver needs in order to solve the linear system. This effect appears not only in case of linear advection but also in case of additional diffusion and if more complex advection velocity fields are considered. However, in our context, we experienced the BGS application just too expensive in terms of runtime.
The application of the Block-Jacobi preconditioner (BJ) is relatively cheap. But generally, the impact of BJ is limited. However, in a purely advective context it does well enhancing the iterative solver in terms of the iteration count as well as the runtime.
If the advection velocity field becomes more complex and/or diffusion is added to the system, the performance of BJ declines, in some cases it even deteriorates the performance of the iterative solver. Complex advection velocity fields as well as diffusion probably lead to a stronger inter-element coupling which is not captured by the BJ preconditioner, because its preconditioning matrix only consists of the diagonal blocks of a global system matrix.
We obtained the best performance of the block Jacobi preconditioner on uniform grids. This results probably from the fact, that adaptive meshes introduce "larger" off-diagonal-block dependencies in the global system matrix, because if for example a coarse element is bordered by finer elements, this results in a lot more adjacent elements than on a uniform grid.

We have implemented the implicit time-stepping methods as well as the preconditioners to run in parallel and even on more complex grids than a hypercube. The parallel framework of the grid and data management was provided by `t8code` and the parallel framework of the iterative solvers was provided by PETSc. We plan a thoroughly examination of the parallel performance in the future. At the moment, we just performed a small test set with up to eight MPI ranks.

We considered an example on a 3D cube with $4,096$ elements of a linear advection problem with a constant advection velocity field in one spatial dimension. As approximation we chose polynomials of order four and a CFL number of $CFL = 4.0$.

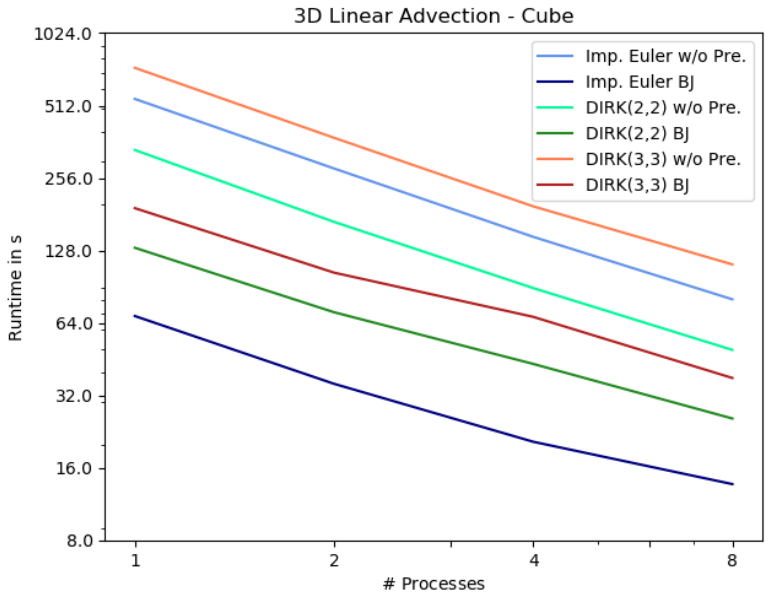


Figure 7.1: Runtimes of GMRES applied to systems arising from the implicit Euler method, DIRK(2,2) method, and DIRK(3,3) method. These systems were solved with and without the application of a Block-Jacobi preconditioner. The underlying mesh corresponds to a 3D cube consisting of $4,096$ elements.

Overall, geometric multigrid preconditioners offer a great opportunity in order to enhance iterative solvers. In all cases the iteration count was substantially decreased. Especially, if stiff problems and very high CFL numbers are concerned, they not only boost the performance of the iterative solver in terms of the iteration count as well as runtime but also add robustness to the solver and prevents (early) divergence of the iterative solver.
In several linear advection-diffusion problems, we measured runtime advantages in comparison to the non-preconditioned linear system in 1D, 2D, and even 3D.
In cases of pure diffusion, we mainly did not observe runtime advantages in comparison to the non-preconditioned system, but as soon as a diffusive part is added to the system, multigrid preconditioners perform even in regards of the runtime very well.
Generally, we experienced that diffusion is well captured by the geometric multigrid preconditioners. Moreover, they react relatively "insensitive" if the advection velocity field becomes more complex.
Choosing more than one coarse mesh within the multigrid preconditioner usually augments the performance of the preconditioner, because the calculation of the coarse grid correction becomes substantially cheaper, the coarser the mesh becomes, while the convergence of the iterative solver remains mainly the same.
Even for high order DG discretizations, we obtained good results from the application of the multigrid preconditioner, due to the usage of several coarse meshes.
A V-cycle within the multigrid preconditioner offered us the best results in terms of the runtime, although other cycle types, i.e. W-cycle and FMG-cycle, decrease the iteration count even further than a V-cycle, but the additional traversals throughout the grid hierarchy adds substantially to the runtime of the application.

In summary, geometric multigrid preconditioners provide great potential in preconditioning linear advection-diffusion problems. If high diffusion coefficients are considered and the parabolic part of the linear advection-diffusion equation predominates the system, multigrid preconditioners perform very well. But even in advection-dominated problems with small diffusion coefficients, we observed runtime advantages in several problems of all dimensions.

As stated before, we have implemented the time-stepping schemes as well as the preconditioners to run in parallel and on more complex geometries. in the following we present a small parallel examination of the multigrid preconditioner.
Therefore, we have tested a "5-LVL" multigrid preconditioner on a two dimensional circle ring domain with $16,384$ elements. The considered linear advection-diffusion problem consisted of a circular rotating advection velocity field and a diffusion coefficient of $\mathbf{d} = 0.0001$. As approximations on each element, we chose polynomials of order four. The CFL number in this example equals $CFL = 4.0$. The initial value function and the second coarse level mesh within the multigrid preconditioner (distributed over eight MPI ranks) is shown in figure 7.2.
Currently, `t8code` does not implement "partition for coarsening". This means, that the grids which are managed in parallel are not partitioned such that child elements are kept together on one process, ergo, a family of elements may be divided onto multiple processes eventually. In this case, the child elements cannot be coarsened to their parent element. Therefore, the coarse meshes may not be identical in the sequential and parallel case.

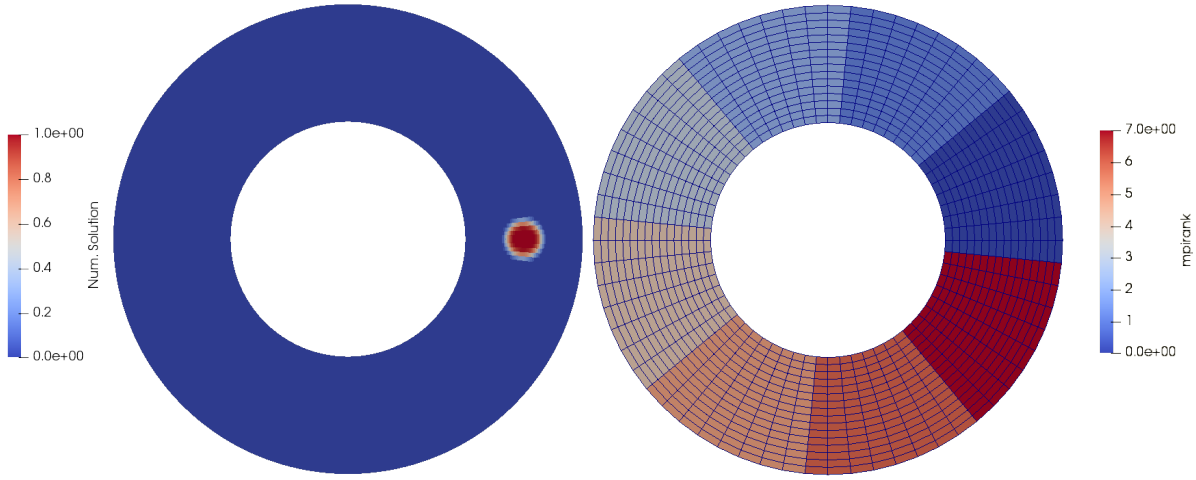


Figure 7.2: (Left:) Initial value function on a circle ring domain. (Right:) Circle ring mesh distributed over eight MPI ranks; this mesh corresponds to a second level coarse mesh with 1024 elements within a multigrid preconditioner which is derived from an initial mesh consisting of $16,384$ elements.

The runtimes of the iterative solver in dependency of the amount of processes is shown in figure 7.3. The runtimes include the whole setup and solution of the system of a single time step as well as the setup and generation of the coarse grids of the multigrid preconditioner. The iterative solver (FGMRES) was applied to systems arising from all considered time-stepping method. The figure displays the comparison between the non-preconditioned case and the case in which these systems were preconditioned with a "5-LVL MG" preconditioner (in a V-cycle). A simulation of the aforementioned problem portrayed at some given points in time is found in D.1.

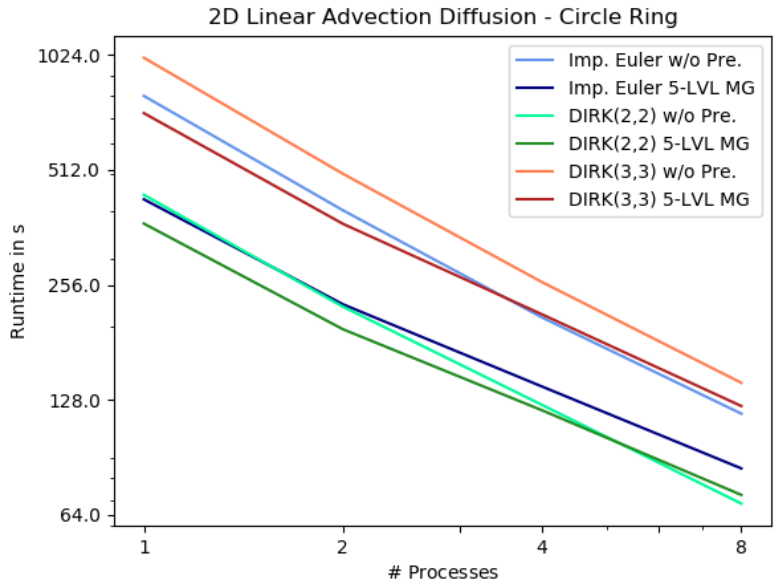


Figure 7.3: Runtimes of FGMRES applied to systems arising from the implicit Euler method, DIRK(2,2) method, and DIRK(3,3) method. These systems were solved with and without the application of a "5-LVL MG" preconditioner. The underlying mesh corresponds to a 2D circle ring consisting of 16, 384 elements.

Besides enhancing and examining the parallel performance of the preconditioners, there are many possible extensions and further investigations, we may consider in the future. This ranges from time-stepping methods to preconditioning techniques.
There are many alternative time-stepping methods which may be considered, like ESDIRK methods or even fully implicit Runge Kutta methods.

Furthermore, there is a broad range of different preconditioners, which can be examined in this context. DG methods naturally implement high order discretizations, therefore, p-multigrid preconditioners offer an interesting and probably effective approach.

Apart from different time-stepping methods and preconditioners, enabling the preconditioners to work on hybrid meshes, for example, meshes consisting of quadrilaterals and triangles, would also be a subsequently interesting feature, which may be examined.

8. Table of Notations

Notations that have been commonly used throughout this thesis are displayed below.

$\square$	Represents an arbitrary value, index, index set, vector, or matrix
$\dot{\square},\ \square_t$	Time-derivative of $\square$
$\square^t$	Transpose of $\square$
t_0	Initial point in time
T	End point in time
Δt	Size of the time step
$\vec{\mathbf{c}}$	Advection velocity field
$\mathbf{d}$	Diffusion coefficient
ε	Square root of the diffusion coefficient
u_0	Initial guess to a linear system
u_{init}	Initial (spatial) value function
$u_\square$	Approximation/Solution of a PDE (at step $\square$)
Ω	Physical domain of an underlying problem
Ω^h	Mesh resulting from the discretization of Ω
Q_ℓ	An element within the mesh Ω^h
$r,\ r_\square$	Residual
$\mathcal{I}_\square$	Identity matrix of size $\mathbb{R}^\square$
$\mathbf{e}_\square$	$\square^{\text{th}}$ column or row of an identity matrix
$\mathcal{K}_\square, \mathcal{L}_\square$	Krylov subspaces
$V_\square,\ W_\square$	Matrices that contain basis vectors generated by the Arnoldi process/ (unsymmetric) Lanczos process
$H_\square$	Upper Hessenberg matrix resulting from the Arnoldi process
$T_\square$	Triangular matrix describing the recurrence of the unsymmetric Lanczos process
$\langle \square, \square \rangle$	Canonical dot product
$\|\square\|$	Norm of $\square$ (induced 2-Norm if not stated otherwise)

9. Table of Algorithms

10. Table of Figures

References

[A03] Ayachour, E.H. 2003, *A fast implementation for GMRES method*, Journal of Computational and Applied Mathematics, Volume 159, Issue 2, 269-283.
https://doi.org/10.1016/S0377-0427(03)00534-X.

[A51] Arnoldi, W.E. 1951, *The principle of minimized iterations in the solution of the matrix eigenvalue problem*, Quarterly of Applied Mathematics, Volume 9, Issue 1, 17-29.
https://doi.org/10.1090/qam/42792.

[A77] Alexander, R. 1977, *Diagonally Implicit Runge–Kutta Methods for Stiff O.D.E.'s*, SIAM Journal on Numerical Analysis, Volume 14, Issue 6, 1006-1021.
https://doi.org/10.1137/0714068.

[A95] Anderson, J.D. 1995, *Computational Fluid Dynamics: The Basics with Applications*, McGraw-Hill International Editions: Mechanical Engineering.

[AB17] Aref, H. & Balachandar, S. 2017, *A First Course in Computational Fluid Dynamics*, Cambridge University Press.
https://doi.org/10.1017/9781316823736.

[AG11] Ascher, U.M. & Greif, C. 2011, *A First Course in Numerical Methods*, SIAM.
https://doi.org/10.1137/9780898719987.

[B00] Black, K. 2000, *Spectral element approximation of convection–diffusion type problems*, Applied Numerical Mathematics, Volume 33, Issues 1–4, 373-379.
https://doi.org/10.1016/S0168-9274(99)00104-X.

[B21] Balay, S., Abhyankar, S., Adams, M.F., Benson, S., Brown, J., Brune, P., Buschelman, K., Constantinescu, E.M., Dalcin, L., Dener, A., Eijkhout, V., Gropp, W.D., Hapla, V., Isaac, T., Jolivet, P., Karpeev, D., Kaushik, D., Knepley, M.G., Kong, F., Kruger, S., May, D.A., Curfman McInnes, L., Tran Mills, R., Mitchell, L., Munson, T., Roman, J.E., Rupp, K., Sanan, P., Sarich, J., Smith, B.F., Zampini, S., Zhang, H. & Zhang, J. 2021, *PETSc/TAO Users Manual*, https://petsc.org/, Argonne National Laboratory, ANL-21/39 - Revision 3.16.

[B96] Braess D. 1986, *On the combination of the multigrid method and conjugate gradients*, In: Hackbusch W., Trottenberg U. (eds) *Multigrid Methods II. Lecture Notes in Mathematics*, Volume 1228, Springer.
https://doi.org/10.1007/BFb0072641.

[BBB16] Blom, D.S., Birken, P., Bijl, H. et al. 2016, *A comparison of Rosenbrock and ESDIRK methods combined with iterative solvers for unsteady compressible flows*, Advances in Computational Mathematics, Volume 42, 1401–1426.
https://doi.org/10.1007/s10444-016-9468-x.

[BBJ16] Birken, P., Bull, J. & Jameson, A. 2016, *A STUDY of multigrid smoothers used in compressible CFD based on the convection diffusion equation*, In: *ECCOMAS Congress 2016 - Proceedings of the 7th European Congress on Computational Methods in Applied Sciences and Engineering*, National Technical University of Athens, Volume 2, 2648-2663.

[BD17] Bertaccini, D. & Durastante, F. 2017, *Iterative Methods and Preconditioning for Large and Sparse Linear Systems with Applications* (1st ed.), CRC Press.
https://doi.org/10.1201/9781315153575.

[BGM21] Betteridge, J.D., Gibson, T.H., Graham, I.G. & Müller, E.H. 2021, *Multigrid preconditioners for the hybridized Discontinuous Galerkin discretisation of the shallow water equations*, Journal of Computational Physics, Volume 426, 109948.
https://doi.org/10.1016/j.jcp.2020.109948.

[BH16] Burstedde, C. & Holke, J. 2016, *A tetrahedral space-filling curve for nonconforming adaptive meshes*, SIAM Journal on Scientific Computing, Volume 38, Issue 5:C471–C503.

[BHG18] Bonev, B., Hesthaven, J.S., Giraldo, F.X. & Kopera, M.A. 2018, *Discontinuous Galerkin scheme for the spherical shallow water equations with applications to tsunami modeling and prediction*, Journal of Computational Physics, Volume 362, 425-448.
https://doi.org/10.1016/j.jcp.2018.02.008.

[BHM00] Briggs, W.L., Henson, V.E. & McCormick, S.F. 2000, *A Multigrid Tutorial* (2^{nd}), SIAM.
https://doi.org/10.1137/1.9780898719505.

[BL11] Brandt, A. & Livne, O.E. 2011, *Multigrid Techniques: 1984 Guide with Applications to Fluid Dynamics, Revised Edition*, SIAM Classics in Applied Mathematics.
https://doi.org/10.1137/1.9781611970753.

[BL89] Berger, M. & LeVeque, R. 1989, *An adaptive Cartesian mesh algorithm for the Euler equations in arbitrary geometries*, In: *9th Computational Fluid Dynamics Conference*, AIAA.
https://doi.org/10.2514/6.1989-1930.

[BO84] Berger, M.J. & Oliger, J. 1984, *Adaptive mesh refinement for hyperbolic partial differential equations*, Journal of Computational Physics, Volume 53, Issue 3, 484-512.
https://doi.org/10.1016/0021-9991(84)90073-1.

[BTB15] Bauer, P., Thorpe, A. & Brunet, G. 2015, *The quiet revolution of numerical weather prediction*, Nature, Volume 525, Issue 7567, 47–55.
https://doi.org/10.1038/nature14956.

[C09] Shu, C.W. 2009,*Discontinuous Galerkin Methods: General Approach and Stability*, In: *Numerical Solutions of Partial Differential Equations*, Birkhäuser Verlag, Advanced Courses in Mathematics - CRM Barcelona, 149-201.
https://doi.org/10.1007/978-3-7643-8940-6.

[CDG17] Cangiani, A., Dong, Z., Georgoulis, E.H. & Houston, P. 2017, *hp-Version Discontinuous Galerkin Methods on Polygonal and Polyhedral Meshes*, SpringerBriefs in Mathematics.
https://doi.org/10.1007/978-3-319-67673-9.

[CH21] Clevenger, T.C. & Heister, T. 2021, *Comparison between algebraic and matrix-free geometric multigrid for a Stokes problem on adaptive meshes with variable viscosity*, Numerical Linear Algebra with Applications, Volume 28, Issue 5, e2375.
https://doi.org/10.1002/nla.2375.

[CHK20] Clevenger, T.C., Heister, T., Kanschat, G. & Kronbichler, M. 2020, *A Flexible, Parallel, Adaptive Geometric Multigrid Method for FEM*, ACM Transactions on Mathematical Software, Volume 47, Issue 1, 1-27.
https://doi.org/10.1145/3425193.

[CHS90] Cockburn, B., Hou, S. & Shu, C.W. 1990, *The Runge-Kutta Local Projection Discontinuous Galerkin Finite Element Method for Conservation Laws. IV: The Multidimensional Case*, Mathematics of Computation, Volume 54, No. 190, 545-581.
https://doi.org/10.2307/2008501.

[CS98] Cockburn, B. & Shu, C.W. 1998, *The Local Discontinuous Galerkin Method for Time-Dependent Convection-Diffusion Systems*, SIAM Journal on Numerical Analysis, Volume 35, Issue 6, 2440-2463.
https://doi.org/10.1137/S0036142997316712.

[CS981] Cockburn, B. & Shu, C.W. 1998, *The Runge–Kutta Discontinuous Galerkin Method for Conservation Laws V: Multidimensional Systems*, Journal of Computational Physics, Volume 141, Issue 2, 199-224
https://doi.org/10.1006/jcph.1998.5892.

[CW91] Cockburn, B. & Shu, C.W. 1991, *The Runge-Kutta local projection P^1-discontinuous-Galerkin finite element method for scalar conservation laws*, ESAIM: Mathematical Modelling and Numerical Analysis, ESAIM: M2AN 25 (3), 337-361.
https://doi.org/10.1051/m2an/1991250303371.

[D21] Dreyer, L. 2021, *The Local Discontinuous Galerkin Method for the Advection-Diffusion Equation on adaptive meshes*, Master's Thesis, Rheinische Friedrich-Wilhelms-Universität Bonn.

[E19] Evstigneev, N.M. 2019, *Numerical analysis of Krylov multigrid methods for stationary advection-diffusion equation*, Journal of Physics: Conference Series, Volume 1391, 012080.
https://doi.org/10.1088/1742-6596/1391/1/012080.

[EES00] Eiermann, M., Ernst, O.G. & Schneider, O. 2000, *Analysis of acceleration strategies for restarted minimal residual methods*, Journal of Computational and Applied Mathematics, Volume 123, Issues 1–2, 261-292.
https://doi.org/10.1016/S0377-0427(00)00398-8.

[F06] Falgout, R.D. 2006, *An introduction to algebraic multigrid*, Computing in Science & Engineering, Volume 8, Issue 6, 24-33.
https://doi.org/10.1109/MCSE.2006.105.

[F76] Fletcher, R. 1976, *Conjugate gradient methods for indefinite systems*, In: Watson, G.A. *Numerical Analysis. Lecture Notes in Mathematics*, Volume 506, Springer.
https://doi.org/10.1007/BFb0080116.

[F93] Freund, R.W. 1993, *A Transpose-Free Quasi-Minimal Residual Algorithm for Non-Hermitian Linear Systems*, SIAM Journal on Scientific Computing, Volume 14, Issue 2, 470-482.
https://doi.org/10.1137/0914029.

[FM20] Franciolini, M. & Murman, S.M. 2020, *Multigrid preconditioning for a space-time spectral-element discontinuous-Galerkin solver*, AIAA Scitech 2020 Forum, AIAA 2020-1314.
https://doi.org/10.2514/6.2020-1314.

[FN91] Freund, R.W. & Nachtigal, N.M. 1991, *QMR: a quasi-minimal residual method for non-Hermitian linear systems*, Numerische Mathematik, Volume 60, 315–339.
https://doi.org/10.1007/BF01385726.

[G20] Gassner, G.J. 2020, Lecture *Wissenschaftliches Rechnen I: Discontinuous Galerkin Methoden*, Universität zu Köln.

[GBG05] Germaschewski K., Bhattacharjee A., Grauer R., Keyes D. & Smith B. 2005, *Using Krylov-Schwarz methods in an adaptive mesh refinement environment*, In: Plewa T., Linde T., Gregory Weirs V. (eds) *Adaptive Mesh Refinement - Theory and Applications. Lecture Notes in Computational Science and Engineering*, Volume 41, Springer.
https://doi.org/10.1007/3-540-27039-6_8.

[GR08] Giraldo, F.X. & M. Restelli, M. 2008, *A study of spectral element and discontinuous Galerkin methods for the Navier–Stokes equations in nonhydrostatic mesoscale atmospheric modeling: Equation sets and test cases*, Journal of Computational Physics, Volume 227, Issue 8, 3849-3877.
https://doi.org/10.1016/j.jcp.2007.12.009.

[GW21] Gassner, G.J. & Winters, A.R. 2021, *A Novel Robust Strategy for Discontinuous Galerkin Methods in Computational Fluid Mechanics: Why? When? What? Where?*, Frontiers in Physics, Volume 8, Article 500690.
https://doi.org/10.3389/fphy.2020.500690.

[GWK16] Gassner, G.J., Winters, A.R. & Kopriva, D.A. 2016, *A well balanced and entropy conservative discontinuous Galerkin spectral element method for the shallow water equations*, Applied Mathematics and Computation, Volume 272, Part 2, 291-308.
https://doi.org/10.1016/j.amc.2015.07.014.

[GZ99] Griebel, M. & Zumbusch, G. 1999, *Parallel multigrid in an adaptive PDE solver based on hashing and space-filling curves*, Parallel Computing, Volume 25, Issue 7, 827-843.
https://doi.org/10.1016/S0167-8191(99)00020-4.

[H09] Hermann, M. 2009, *Numerik gewöhnlicher Differentialgleichungen: Anfangs- und Randwertprobleme*, München: Oldenbourg Wissenschaftsverlag.
https://doi.org/10.1524/9783486594980.

[H16] Hackbusch, W. 2016, *Iterative Solution of Large Sparse Systems of Equations*, Springer Volume 95.
https://doi.org/10.1007/978-3-319-28483-5.

[H18] Holke, J. 2018, *Scalable algorithms for parallel tree-based adaptive mesh refinement with general element types*, PhD thesis, Rheinische Friedrich-Wilhelms-Universität Bonn.
https://bonndoc.ulb.uni-bonn.de/xmlui/handle/20.500.11811/7661.

[HGA12] Hindenlang, F., Gassner, G.J., Altmann, C., Beck, A. Staudenmaier, M. & Munz, C.D. 2012, *Explicit discontinuous Galerkin methods for unsteady problems*, Computers &

Fluids, Volume 61, 86-93.
https://doi.org/10.1016/j.compfluid.2012.03.006.

[HHG15] Heng, Y., Hoffmann, L., Griessbach, S., Rößler, T. & Stein, O. 2015, *Inverse transport modeling of volcanic sulfur dioxide emissions using large-scale simulations*, Geoscientific Model Development, Volume 9, Issue 4, 1627–1645.
https://doi.org/10.5194/gmd-9-1627-2016.

[HKB21] Holke, J., Knapp, D. & Burstedde, C. 2021, *An Optimized, Parallel Computation of the Ghost Layer for Adaptive Hybrid Forest Meshes*, SIAM Journal on Scientific Computing, Volume 43, Issue 6, C359–C385.
https://doi.org/10.1137/20M1383033.

[HKM06] Hülsemann F., Kowarschik M., Mohr M., Rüde U. 2006, *Parallel Geometric Multigrid*, In: Bruaset A.M., Tveito A. (eds) *Numerical Solution of Partial Differential Equations on Parallel Computers*, Lecture Notes in Computational Science and Engineering, Volume 51, Springer.
https://doi.org/10.1007/3-540-31619-1_5.

[HRG16] Hoffmann, L., Rößler, T., Griessbach, S., Heng, Y. & Stein, O. 2016, *Lagrangian transport simulations of volcanic sulfur dioxide emissions: Impact of meteorological data products*, Journal of Geophysical Research: Atmospheres, Volume 121, Issue 9, 4651-4673.
https://doi.org/10.1002/2015JD023749.

[HRX19] Heister, T., Rebholz, L.G. & Xue, F. 2019, *Numerical Analysis: An Introduction*, De Gruyter.
https://doi.org/10.1515/9783110573329.

[HS52] Hestenes, M.R. & Stiefel, E. 1952, *Methods of Conjugate Gradients for Solving Linear Systems*, Journal of Research of the National Bureau of Standards, Volume 49, Issue 6, 409-435.
https://doi.org/10.6028/JRES.049.044.

[K08] Kanschat, G. 2008, *Robust smoothers for high-order discontinuous Galerkin discretizations of advection–diffusion problems*, Journal of Computational and Applied Mathematics, Volume 218, Issue 1, 53-60.
https://doi.org/10.1016/j.cam.2007.04.032.

[K17] Knapp, D. 2017, *Adaptive Verfeinerung von Prismen*, Bachelor's thesis, Rheinische Friedrich-Wilhelms-Universität Bonn.

[K20] Knapp, D. 2020, *A space-filling curve for pyramidal adaptive mesh refinement*, Master's thesis, Rheinische Friedrich-Wilhelms-Universität Bonn.

[K95] Karlsson, H.O. 1995, *"The quasi-minimal residual algorithm applied to complex symmetric linear systems in quantum reactive scattering*, The Journal of Chemical Physics, Volume 103, Issue 12, 4914-4919.
https://doi.org/10.1063/1.470627.

[KBB21] Krais, N., Beck, A., Bolemann, T., Frank, H., Flad, D., Gassner, G.J., Hindenlang, F., Hoffmann, M., Kuhn, T., Sonntag, M. & Munz, C.D. 2021, *FLEXI: A high order discontinuous Galerkin framework for hyperbolic–parabolic conservation laws*, Computers

& Mathematics with Applications, Volume 81, 186-219.
https://doi.org/10.1016/j.camwa.2020.05.004.

[KC16] Kennedy, C. & Carpenter, M. 2016, *Diagonally Implicit Runge-Kutta Methods for Ordinary Differential Equations. A Review.*, NASA Technical Memorandum (NASA/TM-2016-219173).

[KMS20] Katsafados, P., Mavromatidis, E. & Spyrou, C. 2020, *Numerical Weather Prediction and Data Assimilation*, iSTE, Volume 6.
https://doi.org/10.1002/9781119560463.

[L02] LeVeque, R. 2002, *Finite Volume Methods for Hyperbolic Problems*, Cambridge University Press, Cambridge Texts in Applied Mathematics.
https://doi.org/10.1017/CBO9780511791253.

[L06] Skvortsov, L.M. 2006, *Diagonally implicit Runge-Kutta methods for stiff problems*, Comput. Math. and Math. Phys., Volume 46, 2110–2123.
https://doi.org/10.1134/S0965542506120098.

[L50] Lanczos, C. 1950, *An Iteration Method for the Solution of the Eigenvalue Problem of Linear Differential and Integral Operators*, Journal of Research of the National Bureau of Standards, Volume 45, Issue 4, 255-282.
https://dx.doi.org/10.6028/jres.045.026.

[L52] Lanczos, C. 1952 *Solution of Systems of Linear Equations by Minimized Iterations*, Journal of Research of the National Bureau of Standards, Volume 49, Issue 1, 33-53.
https://dx.doi.org/10.6028/jres.049.006.

[LS12] Liesen, J. & Strakos, Z. 2012, *Krylov Subspace Methods: Principles and Analysis*, Oxford University Press.
https://doi.org/10.1093/acprof:oso/9780199655410.001.0001.

[O98] Owen, J. 1998, *A Survey of Unstructured Mesh Generation Technology*, International Meshing Roundtable, 239-267.

[p4est] Burstedde, C., Wilcox, L.C. & Ghattas. O 2011, *P4est: Scalable algorithms for parallel adaptive mesh refinement on forests of octrees*, SIAM Journal on Scientific Computing, Volume 33, Issue 3, 1103–1133.
https://doi.org/10.1137/100791634.

[PLH09] Prill, F., Lukáčová-Medviďová, M. & Hartmann, R. 2009, *Smoothed Aggregation Multigrid for the Discontinuous Galerkin Method*, SIAM Journal on Scientific Computing, Volume 31, Issue 5, 3503–3528.
https://doi.org/10.1137/080728457.

[PP08] Persson, P.O. & Peraire, J. 2008, *Newton-GMRES Preconditioning for Discontinuous Galerkin Discretizations of the Navier-Stokes Equations*, SIAM Journal on Scientific Computing, Volume 30, Issue 6, 2709-2733.
https://doi.org/10.1137/070692108.

[PP20] Perasson, J.W. & Petsana, J. 2020, *Preconditioners for Krylov subspace methods: An overview*, GAMM-Mitteilungen, Volume 43, Issue 4, e202000015.
https://doi.org/10.1002/gamm.202000015.

[PS75] Paige, C.C. & Saunders, M.A. 1975, *Solution of Sparse Indefinite Systems of Linear Equations*, SIAM Journal on Numerical Analysis, Volume 12, Issue 4, 617–29.
https://doi.org/10.1137/0712047.

[RH73] Reed, W.H. & Hill, T.R. 1973, *Triangular mesh methods for the neutron transport equation.*
https://www.osti.gov/servlets/purl/4491151.

[RVH13] Reiter, S., Vogel, A., Heppner, I., Rupp, M. & Wittum, G. 2013, *A massively parallel geometric multigrid solver on hierarchically distributed grids*, Computing and Visualization in Science, Volume 16, Issue 4, 151–164.
https://doi.org/10.1007/s00791-014-0231-x.

[S03] Saad, Y. 2003, *Iterative Methods for Sparse Linear Systems*, SIAM.
https://doi.org/10.1137/1.9780898718003.

[S15] Si, H. 2015, *TetGen, a Delaunay-Based Quality Tetrahedral Mesh Generator*, ACM Transactions on Mathematical Software, Volume 41, Issue 2, Article 11.
https://doi.org/doi.org/10.1145/2629697.

[S84] Sonneveld, P. 1984, *CGS, A Fast Lanczos-Type Solver for Nonsymmetric Linear systems*, SIAM Journal on Scientific and Statistical Computing, Volume 10, Issue 1, 36–52.
https://doi.org/10.1137/0910004.

[SS86] Saad, Y. & Schultz, M.H. 1986, *GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems*, SIAM Journal on Scientific and Statistical Computing, Volume 7, Issue 6, 856-869.
https://doi.org/10.1137/0907058.

[SV00] Saad, Y. & van der Vorst, H.K. 2000, *Iterative solution of linear systems in the 20th century*, Journal of Computational and Applied Mathematics, Volume 123, Issues 1–2, 1-33.
https://doi.org/10.1016/S0377-0427(00)00412-X.

[SWR21] Schlottke-Lakemper, M., Winters, A.R., Ranocha, H. & Gassner, G.J. 2021, *A purely hyperbolic discontinuous Galerkin approach for self-gravitating gas dynamics*, Journal of Computational Physics, Volume 442.
https://doi.org/10.1016/j.jcp.2021.110467.

[T09] Toro, E.F. 2009, *Riemann Solvers and Numerical Methods for Fluid Dynamics*, Springer.
https://doi.org/10.1007/b79761.

[t8code] Holke, J. & Burstedde, C. 2015, *t8code: Parallel AMR on hybrid non-conforming meshes*, https://github.com/holke/t8code, last accessed on 06.01.2022.

[t8dg] Dreyer, L. 2021, *t8dg: DG solver on parallel adaptive meshes*, https://github.com/lukasdreyer/t8dg, last accessed on 06.01.2022.

[t8dgIS] Dreyer, L., Holke, J. & Böing, N. 2022, t8dg, https://github.com/Niklas997/t8dg/tree/feature-implicit_solver, last accessed on 06.01.2022, (currently private).

[TCF11] Tsunematsu, K., Chopard, B., Falcone, J., & Bonadonna, C. 2011, *Comparison of Two Advection-Diffusion Methods for Tephra Transport in Volcanic Eruptions*, Communications in Computational Physics, Volume 9, Issue 5, 1323-1334.
https://doi.org/10.4208/cicp.311009.191110s.

[TK19] Teunissen, J. & Keppens, R. 2019, *A geometric multigrid library for quadtree/octree AMR grids coupled to MPI-AMRVAC*, Computer Physics Communications, Volume 245, 106866.
https://doi.org/10.1016/j.cpc.2019.106866.

[V92] Van der Vorst, H.A. 1992, *Bi-CGSTAB: A Fast and Smoothly Converging Variant of Bi-CG for the Solution of Nonsymmetric Linear Systems*, SIAM Journal on Scientific and Statistical Computing, Volume 13, Issue 2, 631-644.
https://dx.doi.org/10.1137/0913035.

[W07] Watkins, D.S. 2007, *The matrix eigenvalue problem - GR and Krylov subspace methods*, SIAM.
https://doi.org/10.1137/1.9780898717808.

A. Simulation Results Explicit and Implicit Runge Kutta Methods

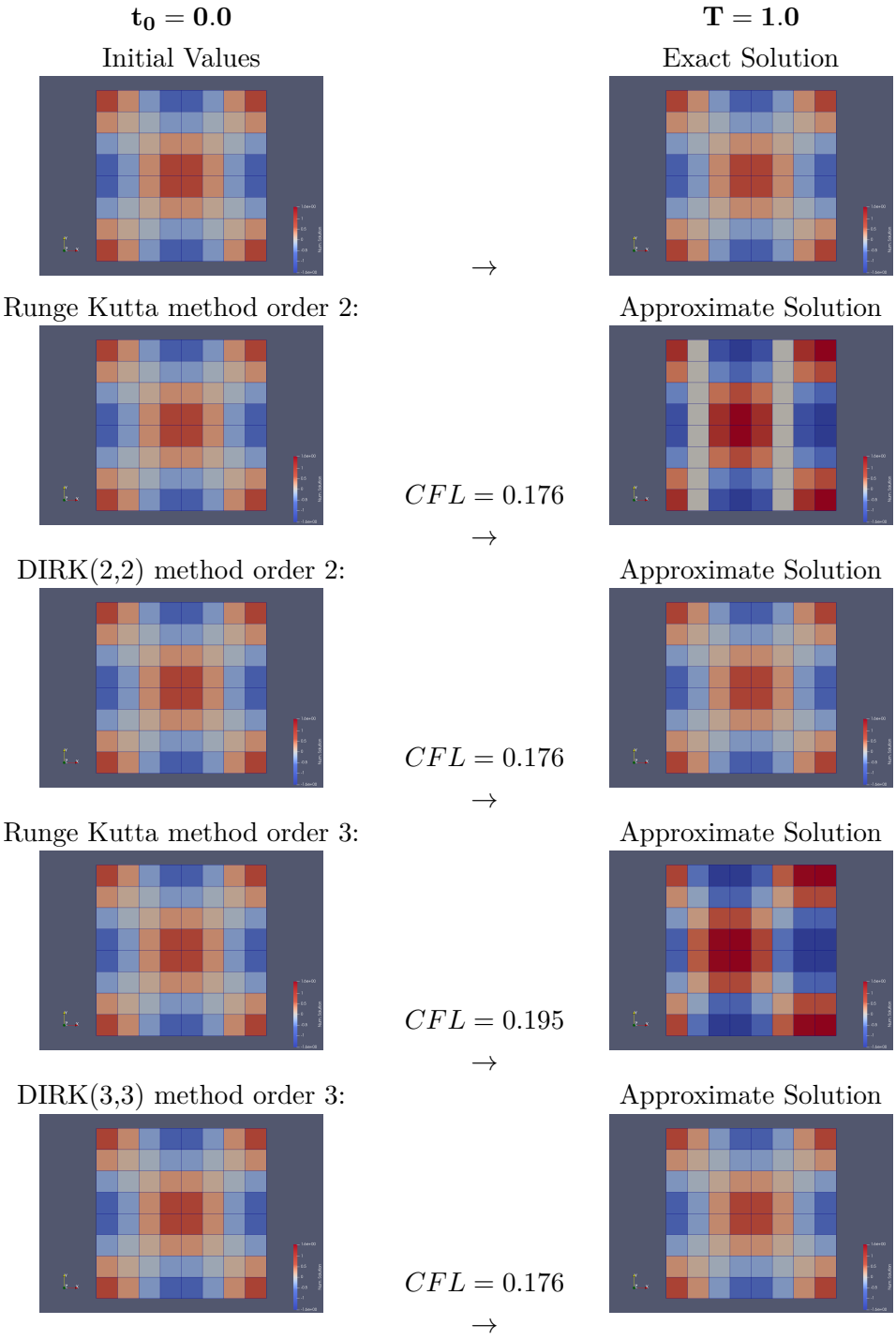


Figure A.1: Comparison of explicit and implicit Runge Kutta methods. The simulation with the explicit schemes starts to become numerically unstable at the pictured CFL, in contrast to the implicit methods.

B. Comparison: Runtime of Multigrid Subroutines

A cell of the table is structured as

# Iterations	Overall Solving time Smoothing time Interpolation time Coarse Solve time

.

-LGL	w/o Pre.	4-LVL MG 1 Smooth.		4-LVL MG 2 Smooth.		4-LVL MG 3 Smooth		4-LVL MG 4 Smooth.	
2	56 | 7.91	12	15.33 6.88 1.12 0.21	8	12.67 7.62 0.71 0.14	5	9.92 6.83 0.47 0.06	4	9.47 7.05 0.36 0.03
3	113 | 27.47	16	34.24 16.49 3.57 0.89	9	24.36 14.8 1.98 0.47	7	23.62 16.18 1.57 0.34	5	19.67 14.53 1.12 0.21
4	186 | 69.80	21	65.00 30.02 6.78 2.94	11	44.02 26.24 3.54 1.5	8	42.36 28.49 2.76 1.00	7	41.55 30.34 2.27 0.81
5	288 | 147.89	28	115.79 51.49 11.97 7.10	13	72.82 42.36 5.78 3.48	10	71.80 47.48 4.92 2.72	8	63.68 46.02 3.51 1.77
6	398 | 276.17	33	195.36 87.37 21.08 14.54	15	115.91 66.78 9.64 7.04	11	105.04 68.91 7.18 5.42	9	101.07 71.91 5.72 4.21

Figure B.1: Comparison of the "4-LVL MG" preconditioner with a varying amount of smoothing iterations plotted against the order of the polynomial approximation on each element, on a two dimensional adaptive mesh for a linear advection-diffusion problem. The runtime is disaggregated into the multigrid subroutines.

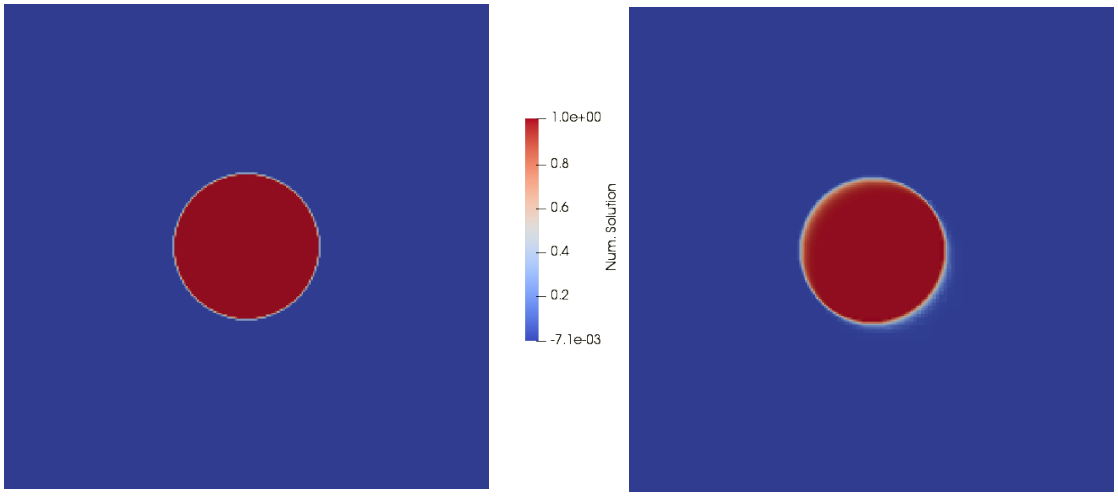
-d	w/o Pre.	2-LVL MG		3-LVL MG		4-LVL MG		5-LVL MG		6-LVL MG	
0.0	66 | 3.68	7	7.69 3.21 0.48 2.29	7	6.67 4.05 0.62 0.27	7	6.65 4.25 0.65 0.03	7	6.48 4.15 0.64 ≈0.0	7	6.47 4.16 0.63 ≈0.0
0.01	53 | 7.82	4	9.62 3.97 0.27 3.33	4	8.19 5.27 0.35 0.46	4	8.19 5.63 0.36 0.05	4	7.89 5.48 0.37 ≈0.0	4	7.82 5.46 0.39 ≈0.0
0.02	62 | 9.08	4	10.03 3.98 0.28 3.71	5	9.35 6.00 0.42 0.56	5	9.77 6.72 0.46 0.06	5	10.15 7.07 0.47 ≈0.0	5	9.92 6.97 0.46 ≈0.0
0.04	75 | 11.06	5	13.58 5.10 0.35 9.66	5	10.28 6.54 0.44 0.70	5	10.11 6.96 0.46 0.09	5	10.06 6.98 0.46 0.01	5	9.67 6.92 0.45 ≈0.0
0.08	102 | 14.60	7	20.59 6.89 0.48 9.66	7	14.41 9.04 0.60 1.18	7	14.00 9.61 0.65 0.15	7	13.94 9.65 0.64 0.01	7	13.79 12.61 0.83 ≈0.0
0.16	140 | 20.38	9	30.20 9.01 0.62 15.90	9	18.93 11.57 0.77 1.97	9	18.54 12.71 0.85 0.25	9	17.92 12.48 0.85 0.02	9	18.09 12.61 0.83 ≈0.0
0.32	217 | 32.04	11	42.53 10.87 0.75 25.2	12	26.45 15.68 1.11 3.43	12	24.59 16.75 1.11 0.42	12	23.85 16.60 1.10 0.04	12	24.55 17.15 1.15 ≈0.0
0.64	326 | 46.82	15	72.14 15.22 1.03 47.44	16	36.48 20.50 1.38 6.39	16	31.83 21.59 1.45 0.73	16	31.48 21.95 1.50 0.09	16	32.35 22.49 1.50 0.01
1.28	416 | 59.69	18	109.85 17.87 1.25 81.18	21	52.52 28.00 1.88 11.31	21	42.34 28.55 1.91 1.19	21	41.53 28.85 1.93 0.14	21	41.65 29.08 1.93 0.02

Figure B.2: Disaggregation of the runtime into the multigrid preconditioners' subroutines, i.e. Smoothing, Interpolation, solving the system on the coarse(st) grid. Mutigrid preconditioners with a varying count of coarse grids are compared on an adaptive 2D linear advection-diffusion problem.

C. 2D Simulation on an Adaptive Mesh with a Varying CFL Number

Initial Values:

Time step with $CFL = 2.0$:



Time step with $CFL = 8.0$:

Time step with $CFL = 32.0$:

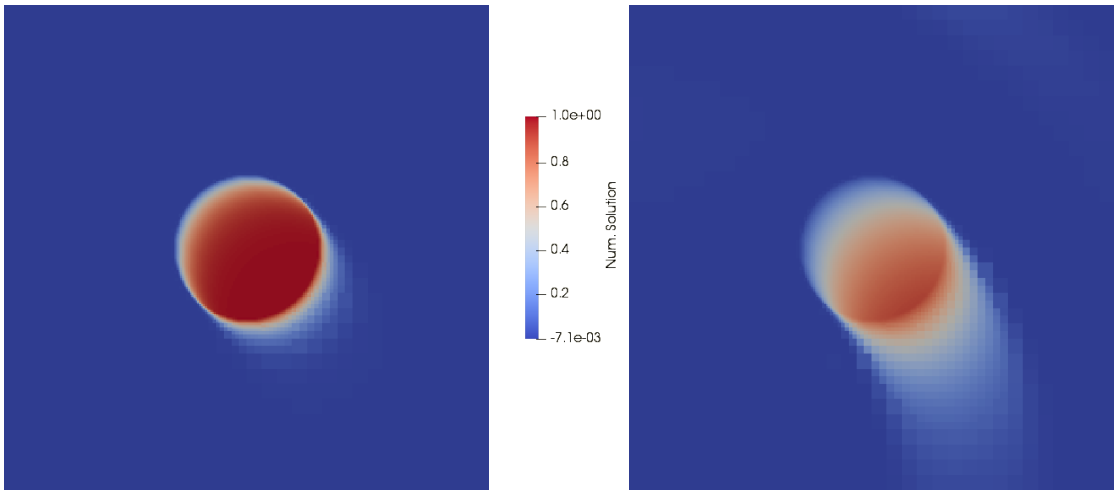
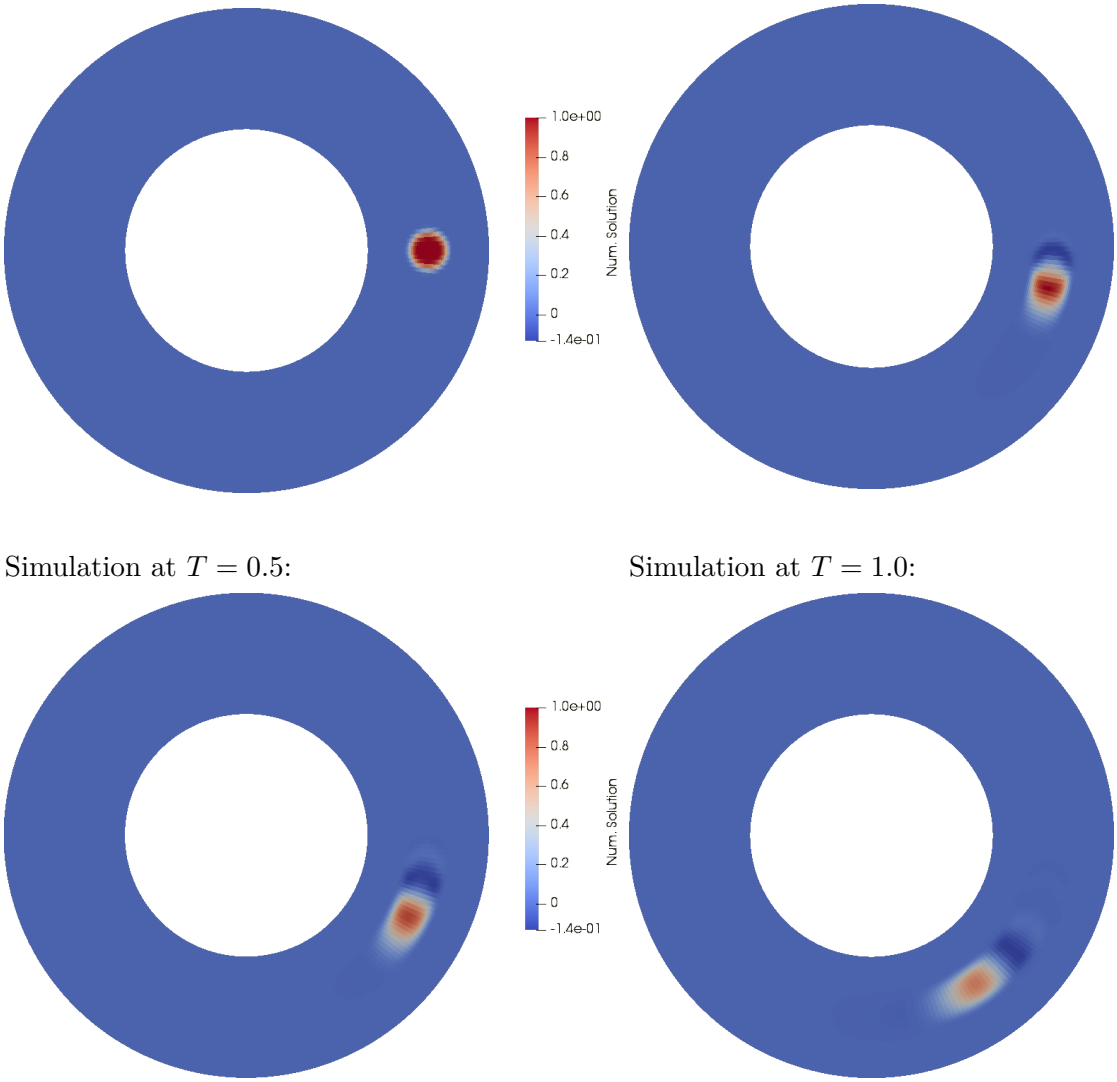


Figure C.1: Approximation of a linear advection-diffusion problem on a 2D adaptive grid with $6,160$ elements. A rotating advection velocity field and a diffusion coefficient of $\mathbf{d} = 0.0001$ was considered. The approximation corresponds to the solution of a linear system arising from the implicit Euler method, after one time step with different CFL numbers have been performed (higher CFL numbers correspond to bigger time steps). At the top-left the initial value function is displayed. In the figures top-right, bottom-left, and bottom-right the calculated approximations with a $CFL = 2.0$, $CFL = 8.0$, and $CFL = 32.0$ are shown.

D. 2D Circle Ring Simulation

Initial Values ($t_0 = 0.0$):

Simulation at $T = 0.25$:



Simulation at $T = 0.5$:

Simulation at $T = 1.0$:

Figure D.1: Approximation of a linear advection-diffusion problem on a 2D circle ring mesh with 16,384 elements. A rotating advection velocity field and a diffusion coefficient of $\mathbf{d} = 0.0001$ was considered. The approximation corresponds to the solutions of a linear systems arising from the DIRK(3,3) method at the time $t_0 = 0.0$ (initial values), $T = 0.25$, $T = 0.5$, and $T = 1.0$ (from top-left to bottom-right).

Erklärung

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne die Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten und nicht veröffentlichten Schriften entnommen wurden, sind als solche kenntlich gemacht. Die Arbeit ist in gleicher oder ähnlicher Form oder auszugsweise im Rahmen einer anderen Prüfung noch nicht vorgelegt worden. Ich versichere, dass die eingereichte elektronische Fassung der eingereichten Druckfassung vollständig entspricht.

Ort, Datum Unterschrift