Subgrid finite volume preconditioner for Discontinuous Galerkin implemented in the DUNE framework

Johannes Kasimir

Master's thesis
2021:E61

Lund University

Faculty of Engineering
Centre for Mathematical Sciences
Numerical Analysis

Centrum Scientiarum Mathematicarum

Master's Theses in Mathematical Sciences 2021:E61
ISSN 1404-6342

LUTFNA-3051-2021

Numerical Analysis
Centre for Mathematical Sciences
Lund University
Box 118, SE-221 00 Lund, Sweden

http://www.maths.lu.se/

Subgrid finite volume preconditioner for Discontinuous Galerkin implemented in the DUNE framework

Johannes Kasimir

August 31, 2021

Abstract

A Jacobian free multigrid preconditioner for linear problems arising from Implicit Discontinuous Galerkin (DG) discretizations is implemented. The preconditioner is based on a multigrid method for a low order Finite volume (FV) discretization on a subcell grid. L^2 projections are introduced as a strategy to translate between the discretizations. Numerical tests on advection dominated linear advection diffusion in two dimensions, performed using the DUNE framework, shows there is potential in this approach when the FV problem can be solved efficiently to low error.

Popular summary

What does the shifting weather have in common with an electricity generating gas turbine, the blood pumping appliances in our chests, and with the formation of stars in nebulae? These are some examples of the vast variety of systems that can be modeled using fluid dynamics.

Even though the basic mechanical laws of liquids and gases have been known for centuries, and even though a detailed mathematical model of fluid systems was developed already in the 1840s, accurately simulating the motion of liquids and gases in realistic settings still proves a formidable challenge. Despite the prevalence of ever more powerful computers.

A key reason why fluid systems are tough to simulate is that they have important dynamics on many scales of magnitude. A wave crashing against the hull of a boat has a wave length of about a meter. But in the aftermath, the tumbling mess of chaotic swirls and bubbles might be anywhere from decimeters to millimeters in size. In practice it is not possible to directly keep track of everything that is going on. And in many cases it is sufficient to resolve only the largest and most energetic aspects of the flow.

But promising future applications in science and industry, ranging from more accurate weather forecasting to fuel efficient vehicle design, requires increasingly detailed simulations - which is why this is an important goal for research.

Commonly used methods for problems in fluid dynamics are the so called "Finite Volume" methods. They divide the environment containing the simulated situation into many small pieces. Small enough that the important quantities of the problem, such as pressure, wind velocity or temperature, can be approximately treated as being constant inside every piece. If the pieces are made smaller, the resolution and accuracy of the method increases. How much more accurate the simulation becomes depends on the kind of method used. For a "first order method" (the Finite Volume method is usually in this category), the increase in accuracy is proportional to the shrinkage of the length of the pieces. In other words, pieces half as long makes a solution twice as good. Perhaps this does not sound too bad, but when the pieces are three dimensional the problem quickly becomes worse! If every piece is made half as long, and wide, and thick, the total number of pieces has increase $2 \times 2 \times 2 = 8$ times, and this makes the solution only twice as accurate. Imagine if we need a 10 times more accurate solution, how many more pieces do we need? (ɯɐ ɹɐǝd ⅂000 ɟıɯɹǝs ɯoɹ6i). This is the reason why first order methods do not scale well to achieve high accuracy.

This motivates efforts to use alternative methods that can increase accuracy without using a huge number of pieces. Collectively such methods are referred to as "high order methods". But in contrast to the low order Finite Volume methods, it is not yet clear how the high order methods can be implemented efficient enough for large real world problems. Researchers at LTH, my master thesis supervisors Philipp Birken and Lea Versbach, have been looking into the question of how to realize effective high order methods. A potentially fruitful approach they identified is to use the well studied and efficient Finite volume methods to quickly

find approximate "coarse grained" solutions. Although blurry and lacking in detail, the coarse solutions quickly determine how the bulk of the matter has moved in some interval of time, and from there it is possible to successively resolve finer details to acquire a much more accurate high order solution.

My work has been to further develop and investigate this method. A fascinating realization was that in order to make best use of the low order method, we must be careful when translating the problem from the high order representation to the low order representation. Making this translation more sophisticated significantly increased the effectiveness of the method in the experiments. But it remains to see if the improvements can be realized in practice for more complicated problems.

Contents

Chapter 1

Introduction

Many natural phenomena interesting for applications in science and engineering allow themselves to be modeled by the partial differential equations (PDEs) of fluid dynamics, the Navier-Stokes equations. Efficient and accurate algorithms suitable for industrial instances of such problems is still an open problem. The overall goal of research in this area is to perform Large Eddy Simulation (LES) of wall bounded and turbulent flows of interest in many important applications, such as the design of engines and wind turbines. LES is a turbulence model with high demands on the resolution of the simulation, requiring high order methods to keep the modeling error low [1]. The high order scheme considered in [2] is the Discontinuous Galerkin Spectral Element Method (DGSEM), which approximates the solution in every grid element using a high order polynomial basis, but allows discontinuities between elements. The polynomial basis in every element is of Lagrange kind with nodes in the Legandre-Gauss-Lobatto quadrature points. This kind of discretization has a large number of degrees of freedom, on the order of hundreds of millions in practical problems.

Discretizations based on DGSEM has great potential for LES because of their high order, the possibility to construct entropy stable and kinetic energy preserving discretizations and because they are well suited for domain decomposition parallelization [2]. However, DGSEM discretizations leads to large stiff ordinary differential equation systems that require implicit time stepping methods to avoid restrictive Courrant Friedrich Lewy (CFL) conditions on the size of the time steps. Efficient implicit time stepping can only be achieved if the linear and nonlinear systems that arises can be solved efficiently [2].

The DGSEM systems have a block structure, with interactions between blocks along the element interfaces. The sparse interaction between elements is what makes them suitable for domain decomposition parallelization, but the size of the dense blocks grows fast with the order of the polynomial basis in the element and with the dimension of the domain. Storing the Jacobian is therefore expensive and the solvers for the systems that arise in the implicit time stepping should to be memory efficient and not require storing the entries of the Jacobian. This property is called being *Jacobian free*. The lack of fast and memory efficient solvers is an obstacle that needs to be solved for DGSEM discretizations to be effectively used on practical problems in industry [2].

To solve the equation systems arising from a DGSEM space discretization and implicit time stepping, Versbach and Birken suggests in [2] and [3] to use Jacobian Free Newton Krylov solvers (JFNK) for their low memory usage. A Krylov subspace method requires only Jacobian vector multiplications and can therefore be implemented Jacobian free. To accelerate the convergence of the Krylov method they develop a preconditioner based on a multigrid method for a finite volume (FV) discretization of the same PDE problem. The purpose of this thesis is to continue that work by implementing the preconditioner in the

Distributed and Unified Numerics Environment (DUNE) framework [4] and test it on a *linear advection diffusion* problem in two dimensions. The goal is that the preconditioner makes the Krylov subspace method converge in a number of iterations not dependent on the grid density, and that the cost of each application of the preconditioner is $\mathcal{O}(n)$ where n is the total number of degrees of freedom (dofs).

The preconditioner suggested in [2] and [3] embeds a subgrid inside the DGSEM elements and formulates a FV discretization on the subgrid. The FV discretization has the same number of unknowns as the original DGSEM discretization. Efficient multigrid methods can be constructed for solving the FV discretized problem and a solver of this kind is used to precondition the DGSEM problem. Since memory minimization is desirable it is important that the multigrid method is also implemented Jacobian free. For this reason the smoother (a linear iterative method) used inside the multigrid method is chosen to be a pseudo time iteration based on a low storage Runge-Kutta time integration method. The preconditioner in [3] exploits that the DGSEM basis consists of Lagrange polynomials with nodes in the Legandre-Gauss-Lobatto (LGL) quadrature nodes. Because of the Lagrange polynomial basis, the degrees of freedom of the DGSEM method corresponds to the function values in the LGL points in the cell. The FV dofs corresponds to the average function values in the cells. Since the dof coefficients in both domains corresponds to function values in the cells, the coefficients of the solution to the FV problem approximates the coefficients of the DGSEM solution to the same problem.

In this thesis, the low order FV discretization *replacement operator* suggested in [2] was implemented for a linear advection diffusion problem in 2D. The replacement operator was defined on a subgrid inserted in every DGSEM element with the same number of cells as the number of degrees of freedom (dofs) in each element. A multigrid method from [2] and [1] was implemented to solve the FV problem. The multigrid method was applied as a preconditioner for the high order DGSEM problem, as was done in [2] and [3]. By introducing functions translating between the high order and low order discretization in a more sophisticated way, the FV multigrid solver could be utilized better, and the performance of the preconditioner improved. The operators used to transfer data between the domains, called *transfer functions*, aims to represent the problem from the high order DGSEM domain in the low order domain as if it had been discretized there from the beginning. The main focus of this investigation was on transfer functions based on L^2 projections between the function spaces. It was also investigated if the FV multigrid preconditioner with transfer functions could be combined with linear iterative methods in the DGEM domain, and if this could further improve the preconditioner.

The main contribution from this thesis is the implementation of the method developed in [2] and [3] in the Distributed and Unified Numerics framework (DUNE) and the investigation of using projection based transfer functions between the DGSEM function space and the low order FV function space.

1.1 Organization of the thesis

In chapter 2 the PDE problem and the discretization used is presented and an overview of the tools employed in the thesis is provided. In chapter 3 the details in the construction of the preconditioner that is the focus of this thesis are presented. In chapter 4 the numerical experiments are described, the results of the experiments are presented as well as a discussion about the results and how the method performs. In the last chapter a summary with conclusions and outlook is presented.

Chapter 2

Background

The chapter starts with the definition of the PDE model problem. Then there is a short introduction to Discontinous Galerkin methods, followed by a description of the DGSEM and FV discretizations that will be used throughout the rest of this thesis. After that, a brief description of the components of the JFNK solver, the Krylov subspace method GMRES and the concept of preconditioning linear systems. In the end there is an overview of multigrid methods for solving linear systems and lastly the multigrid method used in chapter 3 to construct the preconditioner is defined.

2.1 The equation - linear advection diffusion

The conservation law for a convecting and diffusing quantity with a scalar density function $u(t, \mathbf{x})$ is [5]

$$u_t(t, \mathbf{x}) + \nabla \cdot \mathbf{F}(u(t, \mathbf{x}), \nabla u(t, \mathbf{x})) = g(t, \mathbf{x}), \quad \forall \mathbf{x} \in \Omega \subset \mathbf{R}^2, t \in [0, T] \tag{2.1}$$

$$u(0, \mathbf{x}) = u_0(\mathbf{x}), \quad \forall \mathbf{x} \in \Omega \tag{2.2}$$

$$u(t, \mathbf{x}) = u_d(t, \mathbf{x}), \quad \forall \mathbf{x} \in \Gamma_d \subset \partial\Omega \tag{2.3}$$

$$\mathbf{n} \cdot \mathbf{F}(u(t, \mathbf{x}), \nabla u(t, \mathbf{x})) = F_n(t, \mathbf{x}), \quad \forall \mathbf{x} \in \Gamma_n = \partial\Omega \setminus \Gamma_d \tag{2.4}$$

where $\mathbf{F}$ is the flux of the flow, dependent on u and ∇u. t is the time variable and u_t is the derivative of u with respect to time. $\mathbf{x}$ is the space coordinate and g is a function describing the production or destruction of the quantity in the domain. u_0 is the initial state, u_d is a Dirichlet boundary condition on $\Gamma_d \subset \partial\Omega$ and F_n a specified flux boundary condition through the subset of the boundary $\Gamma_n = \partial\Omega \backslash \Gamma_d$. In the linear case $\mathbf{F}(u, \nabla u) = \mathbf{b}u - D\nabla u$, with $\mathbf{b}$ being a constant vector denoting the advection velocity and D a scalar diffusion coefficient. In this case, equation (2.1) can be rewritten as

$$u_t(t, \mathbf{x}) + \mathbf{b} \cdot \nabla u(t, \mathbf{x}) = D\Delta u(t, \mathbf{x}) + g(t, \mathbf{x}). \tag{2.5}$$

Numerically approximating solutions to equation (2.5) with boundary and initial conditions is the goal of this thesis.

2.2 Discontinuous Galerkin discretization for linear advection diffusion

*Discontinuous Galerkin Spectral Element Method*s (DGSEM) are high order schemes that generalizes the finite volume method and borrows features from Finite element (FE) methods. Like traditional continuous Galerkin methods they are derived from a weak form of the PDE. Inside every cell they use high order polynomial approximations, but in contrast

to most FE methods they allow solutions that are discontinuous between grid cells [1]. This is preferable when solving hyperbolic conservation laws since the solutions are typically discontinuous, and imposing global continuity causes problems [2]. In the limit of low polynomial order approximations in every cell (0-th order) DGSEM becomes equivalent to a finite volume scheme.

Equation (2.5) is discretized by a method of lines approach. First the problem is discretized in the space domain to an ordinary differential equation (ODE) and then the ODE is discretized in time using the implicit Euler time stepping method. The following two sections contains definitions used when formulating the discretization.

Domain partition and notation

The domain Ω is partitioned into quadrilateral conforming elements Ω^e for $e \in I$ where I is a set of indices. For convenience, define Γ^0 to be the union of all interior facets, $\Gamma^0 = \cup_{e\in I}\partial\Omega^e \backslash \Gamma$, also called the "skeleton" of the partition. The "jump" $[\cdot]$ and "average" $\{\cdot\}$ on a facet γ are defined to be $[v] = v^+ - v^-$ and $\{v\} = (v^+ + v^-)/2$ where v^+ is the value on the positive side of the facet and v^- is the value on the negative side. The positive and negative sides of the facet are defined by the normal on the facet, the normal points from the positive to the negative side.

The L^2 inner product between two functions $\varphi, \theta \in L^2(V)$ on a domain V is denoted by $\langle\varphi, \theta\rangle_V := \int_V \varphi\theta dv$.

Function space and basis functions

The function space $V_{hp} = \{u_h \in L^2(\Omega); u_h|_{\Omega^e} \in P_p(\Omega^e); \forall e \in I\}$ for a DGSEM discretization of order p on the domain partition $\{\Omega^e\}$ is the space of piecewise (inside every Ω^e) polynomials of order p.

In local coordinates on a reference rectangle $\Omega^{ref} = [0,1] \times [0,1]$ the DGSEM approximation [1][2] u_h^{ref} is written in a polynomial basis

$$u_h^{ref}(\varepsilon, \eta; \mathbf{u}) = \sum_{\mu=0}^{p}\sum_{\nu=0}^{p} u_{\mu\nu}\phi_{\mu\nu}(\varepsilon, \eta) = \mathbf{\Phi}^T(\varepsilon, \eta)\mathbf{u} \tag{2.6}$$

where $\{\phi_{\mu\nu}(\varepsilon, \eta))\}$ is a set of basis functions spanning $P_p(\Omega^{ref})$ and $u_{\mu\nu}$ are coefficients associated with them. The basis functions are collected in the vector $\mathbf{\Phi}(\varepsilon, \eta)$ and the coefficients in the vector $\mathbf{u}$. $\varepsilon, \eta \in [0,1]$ are the spatial coordinates on the reference element. In this thesis the polynomial basis is the same as is used in [2], a Lagrange polynomial basis with nodes in the Gauss-Lobatto quadrature nodes. Let $\{q_i; i = 0 \dots p\}$ be Gauss-Lobatto quadrature nodes on the interval $[0,1]$, then the Lagrange polynomials $\{\phi_j\}$ associated with those nodes are

$$\phi_j(\varepsilon) = \prod_{i=0, i\neq j}^{p} \frac{\varepsilon - q_i}{q_j - q_i}, \quad \varepsilon \in [0,1], \tag{2.7}$$

and in 2D the Lagrange-Gauss-Lobatto (LGL) basis functions are

$$\phi_{\mu\nu}(\varepsilon, \eta) = \phi_\mu(\varepsilon)\phi_\nu(\eta) = \prod_{i=0, i\neq \nu}^{p} \frac{\eta - q_i}{q_\nu - q_i} \prod_{i=0, i\neq \mu}^{p} \frac{\varepsilon - q_i}{q_\mu - q_i}, \quad \varepsilon, \eta \in [0,1]. \tag{2.8}$$

For every $u_h \in V_{hp}$ and every Ω^e there exists a vector valued function $\mathbf{u}^e(t)$ such that $\forall \mathbf{x} \in \Omega^e$ and $\forall t \in \mathbf{R}$, $u_h(t, \mathbf{x})|_{\Omega^e} = u_h^{ref}(\varepsilon^e(\mathbf{x}), \eta^e(\mathbf{x}); \mathbf{u}^e(t))$ where ε^e and η^e are transformations from the spatial coordinates in the element Ω^e to the coordinates on the reference element. $\mathbf{u}^e(t)$ are the time dependent coefficients of the approximation $u_h(t, \mathbf{x})$ inside the element Ω^e. When the domain partition Ω^e is a regular rectangular grid then ε^e and η^e are affine functions.

2.2.1 Derivation of a form approximating the weak form

The following description of the DGSEM discretization of equation (2.5) is taken partly from [2] and partly from [6]. The overarching structure will be the same as in [6] and [7]. In the end we will arrive at a description of the discretization that can be expressed in the domain specific *unified form language*[8] (UFL) and easily parsed by the PDE software package Distributed and Unified Numerics Environment (DUNE). If the description varies slightly from what is normally used it serves the purpose to describe the system the same way it is implemented.

To formulate the discretization of the problem, equation (2.5) is multiplied by a test function $\varphi \in V_{hp}$. And the true solution u is approximated by $u_h \in V_{hp}$

$$\sum_e \int_{\Omega^e} \varphi(u_{ht} + \nabla \cdot \mathbf{b}u_h - D\Delta u_h - g)dV = 0, \tag{2.9}$$

by applying integration by parts to shift the derivative form u_h to φ we get boundary terms

$$\sum_e \int_{\Omega^e} \varphi u_{ht} dV + \int_{\partial\Omega^e} \varphi(u_h\mathbf{b} - D\nabla u_h)\cdot\mathbf{n}dS - \int_{\Omega^e} \nabla\varphi\cdot(u_h\mathbf{b} - D\nabla u_h)dV - \int_{\Omega^e} \varphi g dV = 0. \tag{2.10}$$

On Γ_n the flux is already known because of the flux boundary condition

$$\int_{\Gamma_n} \varphi(u_h\mathbf{b} - D\nabla u_h)\cdot\mathbf{n}dS = \int_{\Gamma_n} \varphi F_n dS \tag{2.11}$$

which lets us rewrite equation (2.10) as

$$A^*(u_h,\varphi) + B^*(u_h,\varphi) + (F_n,\varphi)_{\Gamma_n} + \langle u_{ht},\varphi\rangle_\Omega - \langle g,\varphi\rangle_\Omega = 0 \tag{2.12}$$

where

$$A^*(u_h,\varphi) = \sum_e \int_{\partial\Omega^e\setminus\Gamma_n} \varphi u_h\mathbf{b}\cdot\mathbf{n}dS - \int_{\Omega^e} \nabla\varphi\cdot u_h\mathbf{b}dV, \tag{2.13}$$

$$B^*(u_h,\varphi) = \sum_e \int_{\partial\Omega^e\setminus\Gamma_n} -\varphi D\nabla u_h\cdot\mathbf{n}dS + \int_{\Omega^e} \nabla\varphi\cdot D\nabla u_h dV, \tag{2.14}$$

$$\langle u,v\rangle_\Omega = \int_\Omega uv dV. \tag{2.15}$$

A^* discretizes the advection component in equation (2.5) and B^* discretizes the diffusion component. Since $u_h, \varphi \in V_{hp}$ are discontinuous on the element boundaries, the integrands in the surface integrals in equation (2.13) and (2.14) are not well defined. So the operators A^* and B^* are modified to be defined for functions with discontinuities on the element boundaries. This is done using numerical flux functions, see [1] and [2]. They define the integrands on the interfaces when φ and u_h are discontinuous. While leaving them unchanged for continuous φ and u_h. The numerical flux function used for A^* is the upwind flux used in [3]. The flux function is

$$u_h\mathbf{b}\cdot\mathbf{n}|_\gamma \approx f^*(u_h^+,u_h^-,\mathbf{n}) = \frac{\mathbf{b}\cdot\mathbf{n} + |\mathbf{b}\cdot\mathbf{n}|}{2}u_h^+ + \frac{\mathbf{b}\cdot\mathbf{n} - |\mathbf{b}\cdot\mathbf{n}|}{2}u_h^- \tag{2.16}$$

where γ is a surface between two elements Ω^e and Ω^f where $e, f \in I$. It is convenient to write $f^*(u_h^+,u_h^-,\mathbf{n}) = [\hat{b}u_h]_\gamma$, where $\hat{b} = (\mathbf{b}\cdot\mathbf{n} + |\mathbf{b}\cdot\mathbf{n}|)/2$ because that makes it easy to implement in UFL. A is the operator A^* extended to the space of functions that are discontinous on the element interfaces. It is defined as

$$A(u_h,\varphi) := \sum_e \int_{\partial\Omega^e\setminus\Gamma_n} \varphi f(u_h^+,u_h^-,\mathbf{n})dS - \int_{\Omega^e} \nabla\varphi\cdot u_h\mathbf{b}dV, \tag{2.17}$$

which can be rewritten on the skeleton Γ^0 and the external Dirichlet boundary Γ_d as

$$A(u_h, \varphi) = \int_{\Gamma_d} \varphi f(u_h, u_d, \mathbf{n}) dS + \int_{\Gamma^0} [\varphi][\hat{b} u_h] dS - \int_\Omega \nabla\varphi \cdot u_h \mathbf{b} dV. \tag{2.18}$$

The flux function used for B^*, the diffusion component of the discretization, is the symmetrical interior penalty flux from [6]. The flux function is

$$\varphi D \nabla u_h \cdot \mathbf{n}|_\gamma \approx \hat{f}(u_h^+, \nabla u_h^+, u_h^-, \nabla u_h^-, \mathbf{n})\varphi \tag{2.19}$$

where

$$\hat{f}(u^+, \nabla u^+, u^-, \nabla u^-, \mathbf{n}) = D\frac{\nabla u^+ + \nabla u^-}{2} \cdot \mathbf{n} - D\frac{\mu}{2h_\gamma}(u^+ - u^-), \tag{2.20}$$

and γ is a surface. h_γ is the *local mesh width* defined on a surface between two grid elements $e, f \in I$ as $h_\gamma = (|\Omega^e| + |\Omega^f|)/(2|\gamma|)$ and on a surface $\partial\Omega^e \cup \Gamma_d$ overlapping the Dirichlet boundary Γ_d as $h_\gamma = |\Omega^e|/|\gamma|$. The empirically chosen penalty constant μ is dependent on the order p of the DGSEM method, and we use the same values as in [6]

$$\mu = \begin{cases} 1, & p = 0 \\ 10p^2, & otherwise. \end{cases} \tag{2.21}$$

Using the numerical flux in equation (2.19), and adding a term to preserve the symmetry of the differential operator, we get the operator B extending B^* to the space of functions in V_{hp}. The added term is proportional to $[u_h]$ on all cell boundaries, and is zero for continuous functions. The definition for B is

$$B(u_h, \varphi) := \sum_e \int_{\partial\Omega^e \setminus \Gamma_n} -\varphi \hat{f}(u_h^+, \nabla u_h^+, u_h^-, \nabla u_h^-, \mathbf{n}) - u_h \hat{f}(\varphi^+, \nabla\varphi^+, \varphi^-, \nabla\varphi^-, \mathbf{n}) dS \tag{2.22}$$

$$+ \int_{\Gamma_d} u_d \hat{f}(\varphi, \nabla\varphi, 0, \mathbf{0}, \mathbf{n}) dS + \int_{\Omega^e} \nabla\varphi \cdot D\nabla u_h dV \tag{2.23}$$

which can be rewritten on the skeleton Γ^0 as

$$B(u_h, \varphi) = -\int_{\Gamma_d} \varphi \hat{f}(u_h, \nabla u_h, u_d, \nabla u_h, \mathbf{n}) + (u_h - u_d)\hat{f}(\varphi, \nabla\varphi, 0, \mathbf{0}, \mathbf{n}) dS \tag{2.24}$$

$$- \int_{\Gamma^0} D[\varphi]\{\nabla u_h\} \cdot \mathbf{n} + D[u_h]\{\nabla\varphi\} \cdot \mathbf{n} - D\frac{\mu}{h}[\varphi][u_h] dS \tag{2.25}$$

$$+ \int_{\Omega^e} \nabla\varphi \cdot D\nabla u_h dV. \tag{2.26}$$

Finally, replacing A^* and B^* in (2.12) with our extended operators A and B we have

$$\langle u_{ht}, \varphi\rangle_\Omega = \int_\Omega \nabla\varphi \cdot u_h \mathbf{b} - \nabla\varphi \cdot D\nabla u_h + \varphi g dV \tag{2.27}$$

$$+ \int_{\Gamma^0} D[\varphi]\{\nabla u_h\} \cdot \mathbf{n} + D[u_h]\{\nabla\varphi\} \cdot \mathbf{n} - D\frac{\mu}{h}[\varphi][u_h] - [\varphi][\hat{b} u_h] dS \tag{2.28}$$

$$+ \int_{\Gamma_d} D\varphi\nabla u_h \cdot \mathbf{n} + D(u_h - u_d)\nabla\varphi \cdot \mathbf{n} - D\frac{\mu}{h_\gamma}\varphi(u_h - u_d) - \varphi(\hat{b} u_h + (|\mathbf{b} \cdot \mathbf{n}| - \hat{b})u_d) dS \tag{2.29}$$

$$- \int_{\Gamma_n} \varphi F_n dS =: G(u_h, \varphi) + H(u_d, \varphi) - \langle F_n, \varphi\rangle_{\Gamma_n} + \langle g(t), \varphi\rangle_\Omega \tag{2.30}$$

where G is a bilinear form and

$$H(u_d, \varphi) = \langle u_d, (D\mu/h_\gamma + (\hat{b} - |\mathbf{b} \cdot \mathbf{n}))\varphi - D\nabla\varphi \cdot \mathbf{n} \rangle_{\Gamma_d}. \tag{2.31}$$

Note that the H form implements the Dirichlet boundary condition u_d on Γ_d.

Discretizing the time derivative with implicit Euler

$$\langle u_{ht}(t_{k+1}, \mathbf{x}), \varphi \rangle_\Omega \approx \Big\langle \frac{u_h(t_{k+1}, \mathbf{x}) - u_h(t_k, \mathbf{x})}{\Delta t}, \varphi \Big\rangle_\Omega \tag{2.32}$$

produces an equation system

$$\begin{aligned} &\langle u_h(t_{k+1}, \mathbf{x}), \varphi \rangle_\Omega - \Delta t G(u_h(t_{k+1}, \mathbf{x}), \varphi) = \\ &\quad \langle u_h(t_k, \mathbf{x}), \varphi \rangle_\Omega + \Delta t \Big[H(u_d(t_{k+1}), \varphi) + \langle f(t_{k+1}), \varphi \rangle_{\Gamma_n} + \langle g(t_{k+1}), \varphi \rangle_\Omega \Big], \quad \forall \varphi \in V_{hp}, \end{aligned} \tag{2.33}$$

that can be expressed in the coefficients of the approximating function u_h. Collect all coefficients $\mathbf{u}^e$ of the local LGL polynomial approximations into one global coefficient vector $\mathbf{u}$. Then in matrix form we have

$$\mathbf{A}\mathbf{u}^{k+1} := (\mathbf{I} - \Delta t \mathbf{M}^{-1}\mathbf{G}\mathbf{M})\mathbf{u}^{k+1} = \mathbf{u}^k + \mathbf{d}^{k+1} + \mathbf{f}^{k+1} + \mathbf{g}^{k+1} =: \mathbf{s}^k \tag{2.34}$$

where

$$\mathbf{G} = \big(G(\phi^e_{ij}, \phi^f_{kl})\big)_{s_{ije}, s_{klf}}, \tag{2.35}$$

$$\mathbf{M} = \big(\langle \phi^e_{ij}, \phi^f_{kl} \rangle_\Omega\big)_{s_{ije}, s_{klf}}, \tag{2.36}$$

$$\mathbf{u}^k = \mathbf{M}^{-1}\big(\langle u_h(t_k, \mathbf{x}), \phi^e_{ij} \rangle_\Omega\big)_{s_{ije}}, \tag{2.37}$$

$$\mathbf{d}^k = \Delta t \mathbf{M}^{-1}\big(H(u_d(t_{k+1}), \phi^e_{ij})\big)_{s_{ije}}, \tag{2.38}$$

$$\mathbf{f}^k = \Delta t \mathbf{M}^{-1}\big(\langle -F_n(t_k), \phi^e_{ij} \rangle_{\Gamma_n}\big)_{s_{ije}}, \tag{2.39}$$

$$\mathbf{g}^k = \Delta t \mathbf{M}^{-1}\big(\langle g(t_k), \phi^e_{ij} \rangle_\Omega\big)_{s_{ije}} \tag{2.40}$$

where s_{ije} associates an index to every tuple (i, j, e) for $i \in 0 \dots p, j \in 0 \dots p$ and $e \in I$. The notation $\big(\cdot_{ij}\big)_{S,T}$ denotes a matrix with the element $\cdot_{ij}$ equal to the content of the curly brackets for all $i \in S$ and all $j \in T$. Similarly, $\big(\cdot_i\big)_S$ denotes a column vector. Sometimes the index set S is clear from the context, such as if it references all basis function in some discretization. In such cases the subfix denoting the index set might be skipped and replaced only by the index variable (usually i).

DGSEM with polynomial order 0 is equivalent to a finite volume discretization [2], thus equation (2.34) can define finite volume discretizations as well as higher order DGSEM discretizations. In the following chapters the operators and vectors defined in (2.34)-(2.40) might be referenced by subscripts such as $\mathbf{A}_{dg}$ or $\mathbf{M}_{fv}$. This denotes different realizations of the discretization, based on different grids and varying order p of the spectral basis functions.

2.2.2 Finite volume replacement operator

The preconditioner constructed in chapter 3 uses a FV discretization internally. In [2] and [3] this discretization is called the *replacement operator*, and it has the same number of dofs as the high order DGSEM discretization it is used as a preconditioner for. As will be seen later, this is not strictly necessary. One could for example demand a finite volume discretization with fewer dofs than the original high order discretization, or conversely an even finer discretization with more dofs. The finer naturally incurring a higher cost in

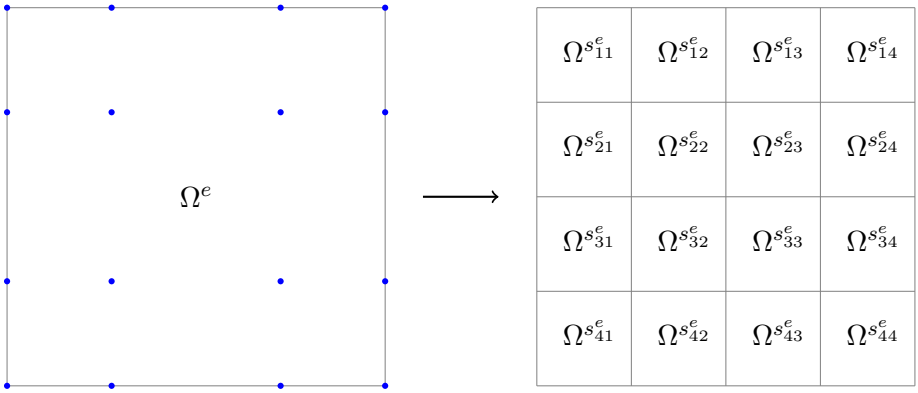

Figure 2.1: DGSEM ($p = 3$) cell with nodes in the Gauss-Lobatto quadrature points (left picture) is refined to 16 finite volume cells after the subgrid is introduced (right picture), $s^e_{jk} \in I^e_{fv} \subset I_{fv}$ for $j, k = 1 \dots p+1$ and $e \in I$.

terms of computation, but perhaps to the benefit of being more effective. That is a trade off between cost and effectiveness that will not be investigated in this thesis. Instead, the replacement operator will have the same number of dofs as the original DGSEM discretization, as it is done in [2].

For a FV discretization, the number of dofs in every cell is 1. The number of dofs in every DGSEM cell is $(p+1)^2$, where p is the order of the polynomials in the spectral basis. Because of this, the domain partitioning used to define the replacement operator must be finer than the domain partitioning (the set $\{\Omega^e\}_{e\in I}$) used to define the DGSEM discretization. The simplest way to do this is to introduce a subgrid [3][2] in the reference element, and let that induce a subgrid in every DGSEM cell. Denote by I^e_{fv} the set of indices for the subgrid cells in the DGSEM cell Ω^e. Every cell in each of the subgrids becomes a cell in the FV discretization. The set of indices for all cells in the FV discretization is denoted I_{fv}. See figure 2.1 for an illustration.

For the domain partition above, the FV or DGSEM($p = 0$) basis functions can be written as

$$v_i(\mathbf{x}) = \mathbf{1}_{\Omega^i}(\mathbf{x}), \quad i \in I_{fv}, \tag{2.41}$$

where $\mathbf{1}_V$ is the indicator function on the set V.

The system in equation (2.34) simplifies for the FV discretization. All terms containing gradients of the basis functions vanishes since $\nabla v_i = 0$ for all $i \in I_{fv}$.

$$\begin{aligned}
\mathbf{G}_{fv} &= \big(G(v_i, v_j)\big)_{I_{fv}, I_{fv}} \\
&= \left(-\int_{\Gamma^0} \frac{D}{h}[v_j][v_i] + [v_j][\hat{b}v_i]dS - \int_{\Gamma_d} \big(\frac{D}{h_\gamma} + \hat{b}\big) v_j v_i dS \right)_{I_{fv}, I_{fv}}, && (2.42)\\
\mathbf{M}_{fv} &= \big(\langle v_i, v_j\rangle_\Omega\big)_{I_{fv}, I_{fv}} = \big(|\Omega^i|\delta_{ij}\big)_{I_{fv}, I_{fv}}, && (2.43)\\
\mathbf{u}^k_{fv} &= \mathbf{M}^{-1}_{fv}\big(\langle u_h(t_k, \mathbf{x}), v_i\rangle_\Omega\big)_{I_{fv}} \\
\mathbf{d}^k_{fv} &= \Delta t \mathbf{M}^{-1}_{fv}\big(H(u_d(t_{k+1}), v_i)\big)_{I_{fv}}, && (2.44)\\
&= \Delta t \mathbf{M}^{-1}_{fv}\big(\langle u_d, v_i(D/h_\gamma + (\hat{b} - |\mathbf{b}\cdot\mathbf{n}|))\rangle_{\Gamma_d}\big)_{I_{fv}}, && (2.45)\\
\mathbf{f}^k_{fv} &= \Delta t \mathbf{M}^{-1}_{fv}\big(\langle -F_n(t_k), v_i\rangle_{\Gamma_n}\big)_{I_{fv}}, && (2.46)\\
\mathbf{g}^k_{fv} &= \Delta t \mathbf{M}^{-1}_{fv}\big(\langle g(t_k), v_i\rangle_\Omega\big)_{I_{fv}} && (2.47)
\end{aligned}$$

where δ_{ij} is the Kronecker delta function.

2.3 Finite difference Jacobian

To avoid calculating and storing the Jacobians, the methods are implemented Jacobian free. This is done using the finite difference approximation from [2].

$$\left[\frac{d\mathbf{F}}{d\mathbf{x}}(\mathbf{u})\right]\mathbf{v} \approx \frac{\mathbf{F}(\mathbf{u}+\epsilon\mathbf{v}) - \mathbf{F}(\mathbf{u})}{\epsilon}, \tag{2.48}$$

where $\epsilon = \sqrt{eps}/||\mathbf{v}||_2$, $\mathbf{F}$ is an in general nonlinear vector valued function and $\frac{d\mathbf{F}}{d\mathbf{x}}(\mathbf{u})$ is the Jacobian of $\mathbf{F}$ at $\mathbf{u}$. The choice of *eps* depends on two sources of errors, the finite difference approximation error and the numerical error introduced by scaling a difference between similar quantities in inexact arithmetic. Depending on the balance between the error terms *eps* is chosen in the interval $[10^{-14}, 1.0]$. In the experiments section in the thesis all problems are linear, and therefore $\mathbf{F}(\mathbf{u}) = \mathbf{A}\mathbf{u} - \mathbf{s}^k$ is affine and in exact arithmetic the approximation in equation (2.48) is exact for any choice of *eps*, and the choice reducing the numerical error is $eps = 1.0$ so this will be used. For nonlinear problems a smaller value should be chosen to limit the approximation error.

2.4 Solving algebraic equation systems

Implicit time stepping methods requires fast solvers for the equation systems that arise from the discretization of the PDE [2]. In general these are nonlinear but in this thesis they are linear since equation (2.5) is linear. Similar tools can however be used to deal with both situations since a nonlinear system can be solved using Newtons method [1]. This reduces the nonlinear system to a short sequence of linear systems, which can be solved efficiently if each of the linear systems can be solved efficiently [1].

The strategy suggested in [3] and [2] for solving linear systems from DGSEM discretizations, e.g. (2.34), is to use a Krylov subspace method preconditioned by a finite volume based multigrid method. In this chapter the components of this solver will be reviewed. Beginning with the Krylov subspace solver GMRES (Generalized minimal residual method), followed by the concept of preconditioning, and finishing with multigrid methods for solving linear systems from finite volume discretizations.

2.4.1 GMRES

GMRES approximates the solution to a linear algebraic problem by finding the minimal norm residual solutions in a sequence of successively larger Krylov subspaces [1] [9].

Specifically, for a given linear equation system $\mathbf{A}\mathbf{x} = \mathbf{b}$ where $\mathbf{A} \in \mathbf{R}^{m\times m}$, GMRES solves

$$\min_{\mathbf{x}\in\mathcal{K}_n(\mathbf{A},\mathbf{b})} ||\mathbf{A}\mathbf{x} - \mathbf{b}||_2 \tag{2.49}$$

for successively larger n, where $\mathcal{K}_n(\mathbf{A},\mathbf{b}) = span\{\mathbf{b}, \mathbf{A}\mathbf{b}, \mathbf{A}^2\mathbf{b} \dots \mathbf{A}^{n-1}\mathbf{b}\}$. An orthonormal basis of the subspace $\mathbf{A}K_n(\mathbf{A},\mathbf{b})$ is generated using the Arnoldi iteration. It constructs the basis for the Krylov subspace by repeatedly applying the operator $\mathbf{A}$ to the starting value $\mathbf{b}$, and using the modified Gram-Schmidt procedure to produce orthonormal basis vectors [9]. On every iteration $\mathbf{b}$ is projected on the new orthogonal basis vector of $\mathbf{A}K_n(\mathbf{A},\mathbf{b})$ and the 2-norm of the residual (equation (2.49)) is measured. When it is deemed small enough the iteration is terminated. An approximation for $\mathbf{x}$ is then constructed from the basis and the projection coefficients.

Convergence

The sequence of residual norms $||\mathbf{r}_n||_2 := ||\mathbf{A}\mathbf{x}_n - \mathbf{b}||_2$ can never increase since every subsequent Krylov subspace contains the previous. But in the worst case the residual

stays constant until the $m-1$:th iteration (for a matrix of size m), when at that point it directly becomes zero [1]. The hope is this will not happen, and that the iteration reaches a sufficiently low residual (determined by the error tolerance of the problem) in a number of iterations much lower than and *independent* of m. The GMRES algorithm performes well when the eigenvalues of the operator $\mathbf{A}$ are clustered away from the origin [1]. Since this is not usually the case for discretizations of unbounded linear operators, such as differential operators, preconditioning (see section 2.4.2) is required to achieve fast convergence [1].

Cost

Since the orthogonal basis has to be stored, storage and work per GMRES iteration increases linearly with the number of iterations n [1]. In the n:th iteration one matrix vector product (matvec) with the system operator $\mathbf{A}$ and n scalar products with basis vectors must be calculated. For small n the matvec dominates this cost [1]. If the number of iterations to reach a sufficiently low residual is small and invariant to the size of the system, the computational cost of the whole procedure is a constant times the cost of the matvec, which in most PDE-discretizations is $\mathcal{O}(m)$ [1]. In this case the GMRES convergence is called *grid independent*.

2.4.2 Preconditioners

As discussed in the previous section, and more detailed in [1], the convergence speed of GMRES depends on the spectrum of the operator, and operators from PDE-discretizations have the property that GMRES converges slowly. It is therefore neccessary to *precondition* [1] the system with invertible matrices $\mathbf{P}_l$ and $\mathbf{P}_r$. This is a transformation of the system $\mathbf{Ax} = \mathbf{b}$ into the system

$$\tilde{\mathbf{A}}\mathbf{z} = \mathbf{P}_l^{-1}\mathbf{A}\mathbf{P}_r^{-1}\mathbf{z} = \mathbf{P}_l^{-1}\mathbf{b} = \mathbf{c}, \quad \mathbf{x} = \mathbf{P}_r^{-1}\mathbf{z}. \tag{2.50}$$

The matrices $\mathbf{P}_l$ and $\mathbf{P}_r$ should be such that the eigenvalues of the operator $\mathbf{P}_l^{-1}\mathbf{A}\mathbf{P}_r^{-1}$ are clustered away from the origin. This is achieved if the combined effect of the operators $\mathbf{P}_l^{-1}$ and $\mathbf{P}_r^{-1}$ is similar to multiplying $\mathbf{A}$ with its inverse. The best preconditioner would therefore be setting either $\mathbf{P}_l$ or $\mathbf{P}_r$ to $\mathbf{A}$ and the other to identity, in which case the GMRES iteration would converge on step 1. But of course the inverse of $\mathbf{A}$ is not available, and other strategies must be applied. In the rest of the thesis only right preconditioning will be considered ($\mathbf{P}_l = \mathbf{I}$) since that is the kind of preconditioning used in [3].

2.4.3 Stationary linear methods

Stationary linear methods [1][3] are a class of fixed point iterations for solving linear equation systems $\mathbf{Ax} = \mathbf{b}$ where $\mathbf{A} \in \mathbf{R}^{m\times m}$ is assumed to be nonsingular, and $\mathbf{x}, \mathbf{b} \in \mathbf{R}^m$. The methods have the form of a "correction"

$$\mathbf{x}^{k+1} = \mathbf{x}^k - \mathbf{N}^{-1}(\mathbf{Ax} - \mathbf{b}) = \mathbf{x}^k - \mathbf{N}^{-1}\mathbf{r}_k, \tag{2.51}$$

where $\mathbf{r}_k = \mathbf{Ax}_k - \mathbf{b}$ is the residual at the k:th iteration. As long as $\mathbf{N}^{-1}$ is invertible, this method has a single fixed point at the solution $\mathbf{x}^*$ since $\mathbf{Ax}^* - \mathbf{b} = 0$. In this case the method is called *consistent*. If $\mathbf{N}^{-1}\mathbf{r}$ is a good approximation of the error of the current solution candidate $\mathbf{x}^k$, then an approximation of the error is removed on every iteration, and hopefully this leads to convergence. The approximation is good if $\mathbf{N}^{-1} \approx \mathbf{A}^{-1}$, this is the same criteria that makes $\mathbf{N}^{-1}$ a good preconditioner. Therefore any preconditioner for the system $\mathbf{Ax} = \mathbf{b}$ is a reasonable choice of $\mathbf{N}^{-1}$, and any stationary linear method also provides a preconditioner in the form of the matrix $\mathbf{N}^{-1}$. These schemes are not usually fast, but form a building block for developing multigrid methods and preconditioners [1].

The linear stationary scheme in equation (2.51) can be rewritten as

$$\mathbf{x}^{k+1} = (\mathbf{I} - \mathbf{N}^{-1}\mathbf{A})\mathbf{x}^k + \mathbf{N}^{-1}\mathbf{b} = \mathbf{M}\mathbf{x}^k + \mathbf{N}^{-1}\mathbf{b}, \tag{2.52}$$

where the matrix $\mathbf{M} = \mathbf{I} - \mathbf{N}^{-1}\mathbf{A}$, called the *iteration matrix*, determines the convergence of the method [1]. Expressing the error of the next iteration as

$$\begin{aligned}\mathbf{e}^{k+1} &= \mathbf{x}^{k+1} - \mathbf{x}^* = \mathbf{M}\mathbf{x}^k + \mathbf{N}^{-1}\mathbf{b} - \mathbf{x}^* \\ &= \mathbf{M}(\mathbf{e}^k + \mathbf{x}^*) + \mathbf{N}^{-1}\mathbf{b} - \mathbf{x}^* = \mathbf{M}\mathbf{e}^k,\end{aligned} \tag{2.53}$$

shows the asymptotic growth rate/decrease is determined by the spectral radius $\rho(\mathbf{M})$.

Slow convergence

For many PDE discretizations, the common category of stationary iterative schemes called *relaxation methods* converges slower when the number of dofs in the discretization increases. Even if some components of the error are quickly reduced there are usually components not reduced well, which makes the asymptotic convergence rate bad for this category of methods [3][1].

Another category of linear iterative schemes are methods where the $\mathbf{N}^{-1}$ matrix is a polynomial $p(\mathbf{A})$ of the system matrix $\mathbf{A}$. The pseudo time iteration in the section below is an example of this kind of method. Because system operators from PDE discretizations discretizes differential operators, their influence becomes increasingly *local* when the resolution of the discretization increases. Therefore the influence of a polynomial of $\mathbf{A}$ also becomes increasingly local, and when the grid is refined the method requires more steps for the residual to influence the global solution of the system. In conclusion, many common/simple linear stationary schemes are not on their own very effective for solving systems from PDE discretizations.

2.4.4 Pseudo time iteration

A class of linear stationary methods that do not require storing the entries of the system matrix $\mathbf{A}$ are *pseudo time stepping methods* described in [10] and in [1]. Since they only require mat-vec operations (matrix vector products) with the system matrix they can be implemented Jacobian free, which is one of the goals of the method developed in [2] and [3].

The iteration is based on a general iterative method to solve an (potentially nonlinear, but in our case linear) algebraic problem $\mathbf{f}(\mathbf{x}) = \mathbf{b} - \mathbf{A}\mathbf{x} = 0$. The idea is that the solution to the algebraic problem is also the stationary solution to an initial value problem (IVP) of the form

$$\mathbf{x}_{t^*} = \mathbf{f}(\mathbf{x}) = \mathbf{b} - \mathbf{A}\mathbf{x}, \quad \mathbf{x}(t^*) = \mathbf{x}_0. \tag{2.54}$$

Assuming all eigenvalues of $-\mathbf{A}$ have negative real part (they do) [1], any numerical time integration method can be used to solve the algebraic system of equations by time stepping until the stationary solution is reached. If the discretization approach we have taken when constructing the matrix $\mathbf{A}$ is stable then the pseudo time ODE (2.54) is stable and unconditionally converges to the solution. In the nonlinear case the method should also work if the starting point $\mathbf{x}_0$ is sufficiently close to the solution $\mathbf{x}^*$ and if all eigenvalues of $\frac{d\mathbf{f}}{d\mathbf{x}}(\mathbf{x}^*)$ have negative real part.

Because memory usage is an important factor for performance, the particular time integration method used should maintain a small memory footprint [2]. For this reason the method used is a low storage explicit Runge-Kutta scheme from [10] with optimized

s	α_1	α_2	α_3	c
1	0.33			0.98
2	0.145	0.395		1.495
3	0.0867	0.2133	0.433	1.84

Table 2.1: Smoother parameters used in equation (2.55) for a 1, 2 and 3 stage method. The parameters are taken from [10] where they were optimized for linear advection with a CFL number of 24. In experiments where $D > 1e-4$ the c parameter was changed to 1.1 instead of 1.495 because it was found to work better for the more diffusive problem.

parameters. The method has s stages, and can be expressed as

$$\mathbf{x}^0 := \mathbf{x}_k \tag{2.55}$$

$$\mathbf{x}^i := \mathbf{x}_k + a_i \Delta t^* \mathbf{f}(\mathbf{x}^i), \quad i = 1 \dots s \tag{2.56}$$

$$\mathbf{x}_{k+1} := \mathbf{x}_k + \Delta t^* \mathbf{f}(\mathbf{x}^s) \tag{2.57}$$

where $\mathbf{x}_k$ is the approximation of $\mathbf{x}^*$ at the k:th pseudo time step. As in [10] the method is parameterized by $\{\alpha_i\}$ and c, where c defines the pseudo time step $\Delta t^* := c/CFL$ where CFL corresponds to the CFL number in the implicit Euler method when solving the original (not pseudo time) problem. The CFL numbers are expressed in the section below. Table 2.1 contains the parameters used for three different pseudo time stepping methods with different number of stages. Note that the parameters used were optimized in the setting of finite volume and linear advection and might not be optimal in the presence of diffusion or for higher order space discretizations.

CFL numbers

These CFL numbers are used to decide the pseudo time step in the Runge-Kutta smoothers. They depend on the grid spacing Δx, the time step Δt and on the order p of the Discontinous Galerkin method. The CFL numbers are taken from [1]. For a (regular grid) finite volume space discretization of the linear advection diffusion equation, the implicit Euler CFL number is

$$CFL_{fv} = \frac{|b|\Delta t}{\Delta x} + \frac{2|D|\Delta t}{\Delta x^2} \tag{2.58}$$

where b, D are the parameters in equation (2.5). Δx is the grid element width or height. Δt is the time step. For Discontinuous Galerkin the situation is slightly different. For a DG method of order $p =: N - 1$ the CFL number is

$$CFL_{dg} = (2N+1)\frac{|b|\Delta t}{\Delta x} + N^2 \frac{2|D|\Delta t}{\Delta x^2}. \tag{2.59}$$

2.4.5 Multigrid methods

Geometrical multigrid methods is a large class of methods for solving algebraic problems that arise from discretized PDEs [1]. They have been developed for many kinds of problems and can be very efficient, with convergence rates independent of mesh size [2].

They are based on the property that a PDE can be discretized on grids with different resolutions, and some component of the error might be cheaper to calculate on a coarser grid. The fundamental idea is to divide the error into high frequency and low frequency components, resolving the low frequency components on a coarser grid where the problem is easier to solve since it has fewer degrees of freedom, and reduce the high frequency components on the fine grid using a stationary linear method especially good for those components [1][2].

The step of projecting a vector from finer to coarser grids is called *restricting* and interpolating from the coarser to finer is called *prolongating*, the linear operators responsible for this are the restriction operator $\mathbf{R}$ and the prolongation operator $\mathbf{P}$. The stationary linear method that suppresses the high frequency components is called the *smoother* [1][2]. Pseudo code for the multigrid algorithm can be found in algorithm 1.

Two grid correction

The method described above but without the smoother, called the *two-grid correction*, can be expressed as a stationary iterative scheme with $\mathbf{N}^{-1} = \mathbf{P}\mathbf{A}_c^{-1}\mathbf{R}$, where $\mathbf{P}$ is the prolongation, $\mathbf{R}$ is the restriction and $\mathbf{A}_c$ is the operator from a linear PDE, same as $\mathbf{A}$, but defined on a *coarser grid*. The stationary linear method corresponding to this choice of $\mathbf{N}^{-1}$ is

$$\mathbf{x}^{k+1} = \mathbf{x}^k - \mathbf{P}\mathbf{A}_c^{-1}\mathbf{R}(\mathbf{A}\mathbf{x}^k - \mathbf{b}) \tag{2.60}$$

$$= (\mathbf{I} - \mathbf{P}\mathbf{A}_c^{-1}\mathbf{R}\mathbf{A})\mathbf{x}^k + \mathbf{P}\mathbf{A}_c^{-1}\mathbf{R}\mathbf{b} \tag{2.61}$$

and it has the iteration matrix $\mathbf{M} = \mathbf{I} - \mathbf{N}^{-1}\mathbf{A}$. But the spectral radius of $\mathbf{M}$ is at least 1, and the method is therefore not convergent. This is true since the range of $\mathbf{R}$ lies in a lower dimensional vector space than its domain, therefore its kernel is nonzero, and since $\mathbf{A}$ is nonsingular the matrix $\mathbf{I} - \mathbf{P}\mathbf{A}_c^{-1}\mathbf{R}\mathbf{A}$ has an eigenvector with the eigenvalue 1.

Applying the two-grid correction twice to a guess $\mathbf{x}^0$ the second iterate $\mathbf{x}^2$ is calculated as

$$\mathbf{x}^2 = (\mathbf{I} - \mathbf{P}\mathbf{A}_c^{-1}\mathbf{R}\mathbf{A})((\mathbf{I} - \mathbf{P}\mathbf{A}_c^{-1}\mathbf{R}\mathbf{A})\mathbf{x}^0 + \mathbf{P}\mathbf{A}_c^{-1}\mathbf{R}\mathbf{b}) + \mathbf{P}\mathbf{A}_c^{-1}\mathbf{R}\mathbf{b} \tag{2.62}$$

$$= (\mathbf{I} - 2\mathbf{P}\mathbf{A}_c^{-1}\mathbf{R}\mathbf{A} + \mathbf{P}\mathbf{A}_c^{-1}\mathbf{R}\mathbf{A}\mathbf{P}\mathbf{A}_c^{-1}\mathbf{R}\mathbf{A})\mathbf{x}^0 \tag{2.63}$$

$$+ (2\mathbf{P}\mathbf{A}_c^{-1}\mathbf{R} - \mathbf{P}\mathbf{A}_c^{-1}\mathbf{R}\mathbf{A}\mathbf{P}\mathbf{A}_c^{-1}\mathbf{R})\mathbf{b}. \tag{2.64}$$

If the *Galerkin condition* $\mathbf{A}_c = \mathbf{R}\mathbf{A}\mathbf{P}$ holds for the coarse grid operator $\mathbf{A}_c$ then the equation above reduces to

$$\mathbf{x}^2 = (\mathbf{I} - \mathbf{P}\mathbf{A}_c^{-1}\mathbf{R}\mathbf{A})\mathbf{x}^0 + \mathbf{P}\mathbf{A}_c^{-1}\mathbf{R}\mathbf{b} = \mathbf{x}^1. \tag{2.65}$$

We see that if the Galerkin condition holds then the two-grid correction converges (possibly to a non-solution!) after one iteration, and further applications does not improve the guess. This does not seem great, but fullfilling the Galerkin condition is a desirable thing because we already know that the two-grid correction is not consistent but if the Galerkin condition holds then convergence is at least fast.

Two grid method

The problem with the two-grid-correction in the previous section can be solved by combining it with a convergent stationary iterative method (the smoother) which reduces *high frequencey* components (defined as the components in the kernel of the restriction) quickly [1] [2].

Let $S(\mathbf{x}) = \mathbf{M}\mathbf{x} + \mathbf{N}^{-1}\mathbf{b}$, then the *two-grid method* is

$$\begin{aligned}\mathbf{x}^{k+1} &= S(\mathbf{x}^k) - \mathbf{P}\mathbf{A}_c^{-1}\mathbf{R}(\mathbf{A}S(\mathbf{x}^k) - \mathbf{b}) \\ &= (\mathbf{I} - \mathbf{P}\mathbf{A}_c^{-1}\mathbf{R}\mathbf{A})\mathbf{M}\mathbf{x}^k + (\mathbf{N}^{-1} - \mathbf{P}\mathbf{A}_c^{-1}\mathbf{R}(\mathbf{A}\mathbf{N}^{-1} - \mathbf{I}))\mathbf{b}\end{aligned} \tag{2.66}$$

which if S is a consistent method, is another consistent linear method. The spectral radius of the iteration matrix $\mathbf{M}_{2gc}\mathbf{M}$, where $\mathbf{M}_{2gc}$ is the iteration matrix of the two-grid correction and $\mathbf{M}$ is the iteration matrix of the smoother, is small according to the assumption that all components not reduced well by two-grid correction are reduced well by the smoother (there are some caveats here, but this is the general idea) [1].

Multigrid method

When a two grid method has been constructed, the same method can be applied recursively on the coarser level instead of solving the coarse system exactly [1]. Instead of denoting the coarser level with $_c$, define a hierarchy of operators $\mathbf{A}_l$, $\mathbf{R}_l$, $\mathbf{P}_l$ and smothers S_l where $l = L, \ldots, 0$ from the finest level to the coarsest. At the coarsest level, $l = 0$, the problem is solved exactly, and at every higher level it is solved by applying the multigrid scheme with $\mathbf{x}_{l-1} = 0$ and $\mathbf{b}_{l-1} = \mathbf{R}_l\mathbf{r}_l$. The iterations matrices can be expressed recursively [1][2]. Denote the recursively defined methods by $MG_l(\mathbf{x}, \mathbf{b}) = \mathbf{M}_l^{MG}\mathbf{x} + \mathbf{N}_l^{MG-1}\mathbf{b}$. On the coarsest level $l = 0$ the method is defined as

$$MG_0(\mathbf{x}, \mathbf{b}) = \mathbf{A}_0^{-1}\mathbf{b} \implies \mathbf{M}_0^{MG} = 0, \quad \mathbf{N}_0^{MG-1} = \mathbf{A}_0^{-1}, \tag{2.67}$$

and on the higher levels we have

$$\begin{aligned}
\mathbf{x}^{k+1} &= MG_l(\mathbf{x}_k, \mathbf{b}) = \mathbf{M}_l^{MG}\mathbf{x}^k + (\mathbf{N}_l^{MG})^{-1}\mathbf{b} \\
&= S_l(\mathbf{x}_k, \mathbf{b}) - \mathbf{P}_l MG_{l-1}(0, \mathbf{R}_l(\mathbf{A}S_l(\mathbf{x}_k, \mathbf{b}) - \mathbf{b})) \\
&= \mathbf{M}_l^S\mathbf{x}_k + \mathbf{N}_l^{S-1}\mathbf{b} - \mathbf{P}_l\mathbf{N}_{l-1}^{MG-1}\mathbf{R}_l(\mathbf{A}\mathbf{M}_l^S\mathbf{x}_k + \mathbf{A}\mathbf{N}_l^{S-1}\mathbf{b} - \mathbf{b}) \\
&= (\mathbf{I} - \mathbf{P}_l\mathbf{N}_{l-1}^{MG-1}\mathbf{R}_l\mathbf{A})\mathbf{M}_l^S\mathbf{x}_k + (\mathbf{N}^{S-1} - \mathbf{P}_l\mathbf{N}_{l-1}^{MG-1}\mathbf{R}_l(\mathbf{A}\mathbf{N}^{S-1} - \mathbf{I}))\mathbf{b},
\end{aligned}$$

the multigrid method is also a stationary iterative scheme with

$$\mathbf{M}_l^{MG} = (\mathbf{I} - \mathbf{P}_l\mathbf{N}_{l-1}^{MG-1}\mathbf{R}_l\mathbf{A})\mathbf{M}_l^S, \tag{2.68}$$

$$\mathbf{N}_l^{MG-1} = \mathbf{N}^{S-1} - \mathbf{P}_l\mathbf{N}_{l-1}^{MG-1}\mathbf{R}_l(\mathbf{A}\mathbf{N}^{S-1} - \mathbf{I}). \tag{2.69}$$

Algorithm 1 $\mathrm{MG}_l(\mathbf{x}_l, \mathbf{b}_l)$

Require: $l \geq 0$
  if $l = 0$ **then**
    $\mathbf{x}_l \leftarrow \mathbf{A}_l^{-1}\mathbf{b}_l$ (exact solve on the coarsest grid)
    return $\mathbf{x}_l$
  end if
  $\mathbf{x}_l = \mathbf{M}_l\mathbf{x}_l + \mathbf{N}_l^{-1}\mathbf{b}_l$ (presmoothing)
  $\mathbf{r}_{l-1} \leftarrow \mathbf{R}_l(\mathbf{A}_l\mathbf{x}_l - \mathbf{b}_l)$ (restriction)
  $\mathbf{v}_{l-1} = \mathbf{0}$
  for $i \leftarrow 1 \ldots \gamma$ **do**
    $\mathbf{v}_{l-1} \leftarrow \mathrm{MG}_{l-1}(\mathbf{v}_{l-1}, \mathbf{r}_{l-1})$ (approximate solve using multigrid)
  end for
  $\mathbf{x}_l \leftarrow \mathbf{x}_l - \mathbf{P}_l\mathbf{v}_{l-1}$ (fine-grid correction)
  $\mathbf{x}_l = \mathbf{M}_l\mathbf{x}_l + \mathbf{N}_l^{-1}\mathbf{b}_l$ (postsmoothing)

Multigrid preconditioning

Like any other stationary iterative scheme, the $\mathbf{N}_l^{MG-1}$ operator can be used as a preconditioner for the problem $\mathbf{A}\mathbf{x} = \mathbf{b}$. This is a matrix-free preconditioner if the smoother used in the multigrid method is matrix-free.

2.4.6 Multigrid for the finite volume method

To define the multigrid method to be used later in the experiments we need to define the different multigrid components, restriction, prolongation and smoother. As restriction and prolongation the commonly used and conservative [1] agglomeration and injection operators will be chosen because they are suitable for the FV replacement operator and

because they are the ones used in [2] and [3]. For completeness their definitions are included here.

Assuming we have a fine grid and a finite volume discretization based on the grid. The agglomeration operator joins adjacent grid cells to make a coarser grid and assigns the value of the coarse grid FV function in the cell to be the mean of the fine FV function in the union of the joined cells. The injection operator reverts the joining of cells, splitting every coarse cell into its original constituent cells, and assigns the function values in the fine grid cells to be equal to the value in the coarse cell. To express this mathematically it is practical to use the *children* function: $C(e) = \{f \in I_{l+1} | \Omega^f \subset \Omega^e, e \in I_l\}$ where I_l is the set indexing the grid cells in the l:th grid, $\{\Omega^i\}$ are the grid cells in the partition of the domain. In other words, the children are the set of cells in the next finer grid that are contained in a particular coarse grid cell e. The agglomeration operator can be defined in the following way

$$\mathbf{R}_l = \left(R_l^{ec}\right)_{ec}, \quad R_l^{ec} = \begin{cases} |\Omega^c|/|\Omega^e|, & c \in C(e) \quad and \quad e \in I_{l-1} \\ 0, & otherwise \end{cases} \tag{2.70}$$

where e and c are elements in the level $l-1$:th and l:th grids respectively, by $|\Omega^e|$ is meant the volume of the element. The injection operator is defined as

$$\mathbf{P}_l = \left(P_l^{ce}\right)_{ce}, \quad P_l^{ce} = \begin{cases} 1, & c \in C(e) \quad and \quad e \in I_{l-1} \\ 0, & otherwise. \end{cases} \tag{2.71}$$

The hierarchy of operators $\mathbf{A}_l, \quad l = 0, \ldots, L$ are chosen to be the finite volume discretization on the respective grids, as is done in [2]. This choice fulfills the Galerkin condition. The Runge-Kutta pseudo time iterations from section 2.4.4 are used as smoothers. As shown in the previous section, the whole multigrid method is a stationary iterative scheme, and the particular scheme described in this section will be denoted as MG_{fv}

$$MG_{fv}(\mathbf{b}, \mathbf{x}) = \mathbf{M}_{fv}^{MG}\mathbf{x} + (\mathbf{N}_{fv}^{MG})^{-1}\mathbf{b}, \tag{2.72}$$

here $\mathbf{M}_{fv}^{MG}$ and $(\mathbf{N}_{fv}^{MG})^{-1}$ are the operators defined recursively in (2.69) and (2.68) but with the particular choice of restriction, prolongation, smoother and hierarchy of operators $\{A_l\}$ described in this section.

Chapter 3

Construction of the preconditioner

In this chapter the components from the previous chapter will be assembled to a preconditioner for the DGSEM problem in equation (2.34). First the preconditioner suggested in [3] is described and then a couple of improvements and generalizations are introduced. *Transfer functions*, methods for reinterpreting the DGSEM coefficients as FV coefficients and vice versa, are constructed in section 3.1. In section 3.2 smoothing is added in the DGSEM domain as the analogue to the *presmoothing* and *postsmoothing* steps in a multigrid method.

The starting point is the preconditioner suggested in [3], where the multigrid method constructed in the previous chapter (see equation (2.72)) is applied as a preconditioner for the DGSEM Jacobian matrix A_{dg} from (2.34). Using the preconditioner for the low order replacement operator as a preconditioner for the high order DGSEM problem is motivated by an equivalence between DGSEM and FV methods with high order flux functions [2][3]. The preconditioner suggested is thus

$$(\mathbf{N}_{fv}^{MG})^{-1} \tag{3.1}$$

from equation (2.72).

Note that one assumption necessary for using the preconditioner in equation (3.1) is that the order of the dofs in the coefficient vectors $\mathbf{u}_{dg}$ and $\mathbf{u}_{fv}$ matches. Naturally, any reordering of the equations in either of the discretizations is still a valid discretization of the problem. But the equivalence between the DGSEM and FV with high order flux functions only holds when the dofs are ordered appropriately. See figure 3.1, the DGSEM coefficients are the function values in the Gauss-Lobatto nodes, the FV coefficients are the average values in the respective subcells in the grid. For the two discretizations to match, the dofs could for example be ordered lexicographically. So that the coefficient in $\mathbf{u}_{dg}$ corresponding to the function value in the top left Gauss-Lobatto node occurs in the same position as the coefficient corresponding to the average in the top-left subcell occurs in $\mathbf{u}_{fv}$, and so on. This is a detail. But it forces us to consider what it means to reinterpret the state in one discretization as a state in another, and raises the question if we can we do better.

3.1 Transfer functions

In line with the discussion above, the reinterpretation of the dofs in the different discretizations is explicitly delegated to two linear operators $\mathbf{T}_r$ and $\mathbf{T}_u$. Such that $\mathbf{T}_r\mathbf{s}_{dg}^k = \hat{\mathbf{s}}_{fv}^k$ where $\mathbf{s}_{dg}^k$ is the right-hand-side (rhs) in equation (2.34) and $\hat{\mathbf{s}}_{fv}^k$ is an estimate of the rhs of

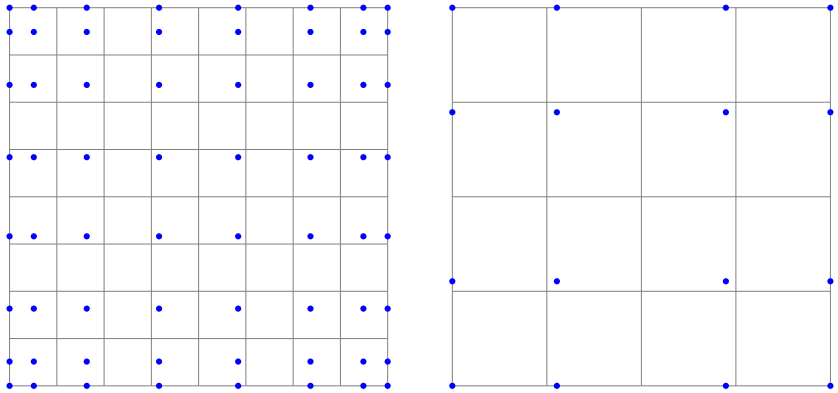

Figure 3.1: Two DGSEM grid reference elements with the Gauss-Lobatto quadrature nodes marked, and an inlaid finite volume subgrid for comparison. To the left is a DGSEM cell with a polynomial basis of order 7 ($p = 7$), to the right with $p = 3$. In the $p = 7$ case the Lagrange interpolation ignores some of the finite volume cells.

the same problem if it had been discretized using the FV discretization. $\mathbf{T}_u \mathbf{u}_{fv}^{k+1} = \hat{\mathbf{u}}_{dg}^{k+1}$ estimates the DGSEM solution given the FV solution to the same problem. This is the main methodological change in this thesis compared to [3] and [2].

Using these estimates, the preconditioner in equation (3.1) is generalized to

$$\mathbf{T}_u (\mathbf{N}_{fv}^{MG})^{-1} \mathbf{T}_r. \tag{3.2}$$

The notation for $\mathbf{T}_r$ respectively $\mathbf{T}_u$ should be read as "transfer function for the residual" respectively "transfer function for u".

Constraints when choosing $\mathbf{T}_r$ and $\mathbf{T}_u$

Arbitrary matrices should not be allowed. If they are, a choice making the best preconditioner would be $\mathbf{T}_r = \mathbf{I}$ and $\mathbf{T}_u = \mathbf{A}_{dg}^{-1} \mathbf{N}_{fv}^{MG}$, but that is **not** a useful choice. (Making this substitution in equation 3.2 we directly get the inverse $\mathbf{A}_{dg}^{-1}$.)

To guide us towards more sensible sensible choices, here are two constraints:

1. The *transfer functions* $\mathbf{T}_r$ and $\mathbf{T}_u$ are restricted to only allow coupling between dofs associated with basis functions that are nonzero in the same DGSEM cell. That is, $\mathbf{T}_r$ only considers the DGSEM basis functions overlapping with a particular FV subcell when estimating the FV coefficient value for that cell. Respectively, $\mathbf{T}_u$ only considers the values in the subcells that a particular LGL polynomial overlaps when estimating the DGSEM coefficient corresponding to that LGL polynomial basis function. The two matrices $\mathbf{T}_r$ and $\mathbf{T}_u$ are therefore block-diagonal, assuming the dofs of the discretizations are ordered in the way described in the section above. If the grid defining the DGSEM discretization is affine, the diagonal blocks should all be equal.

2. For the transfer function to be sound, as the grid is refined, their estimates should become arbitrarily good. In other words $\lim_{h \to 0} \mathbf{T}_r \mathbf{s}_{dg}^k = \mathbf{s}_{fv}^k$ and $\lim_{h \to 0} \mathbf{T}_u \mathbf{u}_{fv}^{k+1} = \mathbf{u}_{dg}^{k+1}$. This may seem a trivial condition since both discretizations converge to the true solution when the grid is refined. But actually neither the ad hoc transfer function or the $L^2(\Omega)$ projection based transfer functions (see section 3.1.1) satisfies this condition when nonzero boundary conditions are present.

3.1.1 Projection based transfer functions

One promising choice of $\mathbf{T}_r$ and $\mathbf{T}_u$ is to base them on the L^2 projection. In fact, for a problem with zero boundary conditions and no sources in the domain ($u_d = 0$, $F_n = 0$, $g = 0$, see equation (2.1)), under the assumption that the true solution $u(\mathbf{x}, t)$ can be exactly represented either by the DGSEM discretization or by the FV discretization, it is possible to show that a L^2 projection correctly estimates the right-hand-side for the finite volume replacement operator. In other words, that $\mathbf{T}_r$ is an L^2 projection.

Assume a problem without any sources or sinks in the domain ($g = 0$) and zero boundary conditions ($F_n = 0$ and $u_d = 0$). The true solution to the PDE at the time t_k is $u_k(\mathbf{x}, t_k)$. The vector $\mathbf{\Phi} = \big(\phi_i\big)_i$ contains the LGL basis functions of the DGSEM function space, $\mathbf{V} = \big(v_i\big)_i$ contains the finite volume basis functions. Then for the DGSEM discretization of order p, equation (2.34) is

$$\mathbf{A}_{dg}\mathbf{u}_{dg}^{k+1} = \mathbf{u}_{dg}^{k} \tag{3.3}$$

where

$$\mathbf{u}_{dg}^{k} = \mathbf{M}_{dg}^{-1}\big(\langle u(\mathbf{x}, t_k), \phi_i\rangle_\Omega\big)_i. \tag{3.4}$$

Remember that the notation $\big(\langle u(\mathbf{x}, t_k), \phi_i\rangle_\Omega\big)_i$ is a column vector containing the inner products between u and all different ϕ_i.

Using the FV discretization, the system is

$$\mathbf{A}_{fv}\mathbf{u}_{fv}^{k+1} = \mathbf{u}_{fv}^{k} \tag{3.5}$$

where

$$\mathbf{u}_{fv}^{k} = \mathbf{M}_{fv}^{-1}\big(\langle u(\mathbf{x}, t_k), v_i\rangle_\Omega\big)_i. \tag{3.6}$$

The two discretizations both converge to the true solution $u(\mathbf{x}, t_{k+1})$ as the grid is refined [1]. If we are given the vector $\mathbf{u}_{dg}^{k}$, the DGSEM right hand side *data*, and want to approximate the DGSEM solution $\mathbf{u}_{dg}^{k+1}$ by solving the same problem (having the same PDE, the same boundary and the same initial condition) using a finite volume discretization, we must infer the finite volume data $\mathbf{u}_{fv}^{k}$ from the DGSEM data $\mathbf{u}_{dg}^{k}$.

Assume the true solution at the previous time step is contained in the DGSEM function space so that

$$u(\mathbf{x}, t_k) = \mathbf{\Phi}^T\mathbf{u}_{dg}^{k}, \tag{3.7}$$

then equation (3.6) becomes

$$\begin{aligned}\mathbf{u}_{fv}^{k} = \mathbf{M}_{fv}^{-1}\big(\langle u_k(\mathbf{x}, t_k), v_i\rangle_\Omega\big)_i &= \mathbf{M}_{fv}^{-1}\big(\langle \mathbf{\Phi}^T\mathbf{u}_{dg}^{k}, v_i\rangle_\Omega\big)_i \\ &= \mathbf{M}_{fv}^{-1}\big(\langle \phi_j, v_i\rangle_\Omega\big)_{ij}\mathbf{u}_{dg}^{k}.\end{aligned} \tag{3.8}$$

Note that we got the FV right hand side as a function of the DGSEM right hand side. This is the kind of estimate we want for $\mathbf{T}_r$.

If we instead assume the true solution at the previous time step is contained in the FV function space

$$u(\mathbf{x}, t_k) = \mathbf{V}^T\mathbf{u}_{fv}^{k}, \tag{3.9}$$

then equation (3.4) becomes

$$\begin{aligned}\mathbf{u}_{dg}^{k} = \mathbf{M}_{dg}^{-1}\big(\langle u_k(\mathbf{x}, t_k), \phi_i\rangle_\Omega\big)_i &= \mathbf{M}_{dg}^{-1}\big(\langle \mathbf{V}^T\mathbf{u}_{fv}^{k}, \phi_i\rangle_\Omega\big)_i \\ &= \mathbf{M}_{dg}^{-1}\big(\langle v_j, \phi_i\rangle_\Omega\big)_{ij}\mathbf{u}_{fv}^{k}.\end{aligned} \tag{3.10}$$

This gives us two different choices for $\mathbf{T}_r$. We should use

$$\mathbf{T}_r = \mathbf{M}_{fv}^{-1}\big(\langle \phi_j, v_i\rangle_\Omega\big)_{ij} =: \mathbf{P}_{fv} \tag{3.11}$$

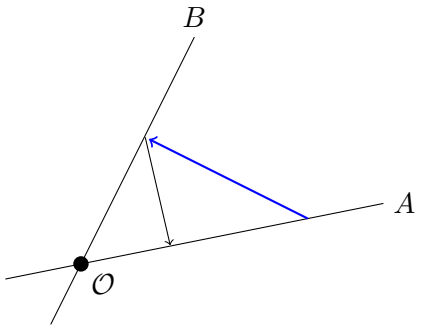

Figure 3.2: Projections between two linear subspaces A and B. As can be seen in the picture, the projection is here not a singular operator, as one might have guessed by the name. They have inverses, the reversed arrows in the image. But projecting $A \to B$ is not the inverse of projecting $B \to A$. In the experiment section 4.4, different combinations of the projection arrows and their inverses are tested as transfer functions $\mathbf{T}_r$ and $\mathbf{T}_u$. For example $\mathbf{T}_u$ could be either the projection of the FV functions on the LGL space, or the inverse of the projection of the LGL functions on the FV space.

if we believe the true solution $u(\mathbf{x}, t_k)$ at the previous time step is in the span of the DGSEM basis functions, and

$$\mathbf{T}_r = \left(\langle v_j, \phi_i \rangle_\Omega\right)_{ij})^{-1} \mathbf{M}_{dg} =: \mathbf{P}_{dg}^{-1} \tag{3.12}$$

if we believe the true solution at the previous time step is in the span of the FV basis functions. The first option is more plausible since the accuracy of high order methods is the motivation for using them to begin with.

The matrix $\mathbf{P}_{fv}$ is the L^2 projection of functions in the DGSEM space onto the FV function space, acting on the coefficients of DGSEM functions. The matrix $\mathbf{P}_{dg}$ is the L^2 projection of functions in the FV space onto the DGSEM function space, acting on the coefficients of FV functions.

As is required by transfer functions, both $\mathbf{P}_{fv}$ and $\mathbf{P}_{dg}$ are block diagonal and only couple dofs inside every DGSEM cell. If $\varphi_i^e(\mathbf{x})$ and $\varphi_j^e(\mathbf{x})$ for $i, j \in 1 \dots (p+1)^2$ are basis functions on the element Ω^e with corresponding basis function $\tilde{\varphi}_i(\mathbf{x})$ and $\tilde{\varphi}_j(\mathbf{x})$ on the reference element, and $\mathbf{m}(\mathbf{x}) = \begin{bmatrix} \varepsilon^e(\mathbf{x}) \\ \eta^e(\mathbf{x}) \end{bmatrix}$ is the mapping from Ω^e to the reference element. Then the inner product can be calculated as

$$\begin{aligned} \langle \varphi_i^e(\mathbf{x}), \varphi_j^e(\mathbf{x}) \rangle_{\Omega^e} &= \int_{\Omega^e} \varphi_i^e(\mathbf{x}) \varphi_j^e(\mathbf{x}) dV = \int_{\Omega^e} \tilde{\varphi}_i(\mathbf{m}(\mathbf{x})) \tilde{\varphi}_j(\mathbf{m}(\mathbf{x})) dV \\ &= \left[\mathbf{m}(\mathbf{x}) = \mathbf{y}\right] = \int_{\Omega^{ref}} \tilde{\varphi}_i(\mathbf{y}) \tilde{\varphi}_j(\mathbf{y}) \left| \frac{d\mathbf{m}^{-1}}{d\mathbf{x}} \right| (\mathbf{y}) dV. \end{aligned} \tag{3.13}$$

Generally $\mathbf{P}_{fv}$ respectively $\mathbf{P}_{dg}$ do not have repeated blocks along the diagonal. However, if the grid is affine then $\frac{d\mathbf{m}^{-1}(\mathbf{x})}{d\mathbf{x}}$ is constant and

$$\langle \varphi_i^e(\mathbf{x}), \varphi_j^e(\mathbf{x}) \rangle_{\Omega^e} = \left| \frac{d\mathbf{m}^{-1}}{d\mathbf{x}} \right| \langle \tilde{\varphi}_i(\mathbf{x}), \tilde{\varphi}_j(\mathbf{x}) \rangle_{\Omega^{ref}}. \tag{3.14}$$

For the matrices $\mathbf{P}_{fv}$ and $\mathbf{P}_{dg}$ where every entry is a fraction of two inner products, this implies block diagonality since the only factor dependent on the element e cancels.

Benefits of projection based transfer functions

- The L^2 projection based transfer function can be calculated between almost any two function spaces. In principle, this allows using the preconditioner for any choice of

DGSEM spectral basis, not only for Lagrange type basis functions. This makes it more flexible. The function spaces could even have different numbers of degrees of freedom.

- When solving the PDE for successive time steps. The solution at the previous time step is guaranteed to live in the DGSEM space, since that is how the solution is represented. It is also what is used to define the next time step solution, in that sense it is the true previous time step solution. And since we are using Galekin methods, using the L^2 projection to move it to the FV space gives us the true FV right hand side. Which puts us in a good position to find an approximate solution to the DGSEM problem.
- in the experiments it improves the performance of the preconditioner.

3.1.2 Transfer functions selected for the experiments

Using the transfer functions described above, there are some different options for how to construct the preconditioner in equation (3.2). No good way of motivating how to choose $\mathbf{T}_u$ was found. But in practice the best choice was to use the inverse of $\mathbf{T}_r$ whatever choice was made for that function.

The options for selecting $\mathbf{T}_r$, maps from DGSEM coefficients to FV coefficients, are

- $\mathbf{P}_{fv}$: the L^2 projection onto the FV space,
- $\mathbf{P}_{dg}^{-1}$: the inverse of the L^2 projection onto the DGSEM space,
- $\mathbf{I}$: the ad hoc transfer,

and the options for selecting $\mathbf{T}_u$, maps from FV coefficients to DGSEM coefficients, are

- $\mathbf{P}_{fv}^{-1}$: the inverse of the L^2 projection onto the FV space,
- $\mathbf{P}_{dg}$: the L^2 projection onto the DGSEM space,
- $\mathbf{I}$: the ad hoc transfer.

These four combinations for selecting $(\mathbf{T}_r, \mathbf{T}_u)$ are tested in the experiments section

- FV projection: $(\mathbf{P}_{fv}, \mathbf{P}_{fv}^{-1})$
- DG projection: $(\mathbf{P}_{dg}^{-1}, \mathbf{P}_{dg})$
- Projection: $(\mathbf{P}_{fv}, \mathbf{P}_{dg})$
- Ad hoc: $(\mathbf{I}, \mathbf{I})$

3.1.3 Computational cost

The ad hoc transfer function is equivalent to doing sparse matrix multiplications in every DGSEM cell. Therefore applying it requires $\mathcal{O}(n(p+1)^d)$ operations, where n is the number of gridcells, p is the order of the polynomial basis, and d is the dimension. The projection based transfer functions are dense in every DGSEM cell, and therefore they are more expensive to apply. The work required to apply them is $\mathcal{O}(n(p+1)^{2d})$, which is 16 respectively 64 times more expensive for DGSEM($p = 3$) respectively DGSEM($p = 7$) in 2D compared to the ad hoc transfer function.

However, the transformations between discretizations was not a large part of the total cost to solve in the experiments. It is dominated by evaluating the quadrature on the grid. For DGSEM($p = 3$) the time computing one finite difference Jacobian matvec was several times larger than the cost of one $\mathbf{P}_{fv}$ matrix multiplication. It is reasonable that the work

should be on the same order of magnitude, since a Jacobian matvec is a slightly denser matrix multiplication than a block diagonal matrix.

On affine grids the projection based transfer function is a matrix with the same diagonal block repeated for every cell. The block can be precomputed and reused through all time steps since it does not depend on anything but the geometry of the problem and the basis functions. On non affine grids a projection based transfer function (for example the $\mathbf{P}_{fv}$ matrix) is a block diagonal matrix with in general a different block associated with every DGSEM cell. Storing it therefore requires $\mathcal{O}(n)$ memory, same as storing the Jacobian. But in contrast to the Jacobian, the transfer matrix can be precomputed and stored throughout the whole computation.

3.2 Adding smoothing in the DGSEM domain

The preconditioners in equation (3.2) uses a FV multigrid based preconditioner $(\mathbf{N}_{fv}^{MG})^{-1}$ together with suitable transfer functions $\mathbf{T}_r$ and $\mathbf{T}_u$ to precondition the DGSEM problem. It is

$$\mathbf{P}^{-1}_{MG(p,q,\gamma)} := \mathbf{T}_u(\mathbf{N}_{fv}^{MG})^{-1}\mathbf{T}_r, \tag{3.15}$$

where p, q, γ denotes the number of presmoothing steps postsmoothing steps and the number of cycles in the multigrid method.

As described in section 2.4.3, a preconditioner $\mathbf{P}^{-1}$ can be used to define a stationary iterative method of the type in equation (2.51) by making the assignment $\mathbf{N}^{-1} := \mathbf{P}^{-1}$. Using the preconditioner in equation (3.15) to define a stationary iterative scheme, we get

$$\mathbf{x}^{k+1} = \mathbf{x}^k - \mathbf{P}^{-1}_{MG(p,q,\gamma)}(\mathbf{A}_{dg}\mathbf{x}^k - \mathbf{b}) \tag{3.16}$$

$$= (\mathbf{I} - \mathbf{P}^{-1}_{MG(p,q,\gamma)}\mathbf{A}_{dg})\mathbf{x}^k + \mathbf{P}^{-1}_{MG(p,q,\gamma)}\mathbf{b}. \tag{3.17}$$

A stationary iterative scheme such as equation (3.16) can be combined with other iterative methods, by first applying one and then the next. Hopefully this constructs a better iterative method.

Assume we have another iterative method $(\mathbf{M}, \mathbf{N}^{-1})$. We chain it with equation (3.16) by first applying $(\mathbf{M}, \mathbf{N}^{-1})$ in a "presmoothing" step and then applying (3.16). This gives us a new iterative scheme

$$\begin{aligned}\mathbf{x}^{k+1} =&(\mathbf{M} - \mathbf{P}^{-1}_{MG(p,q,\gamma)}\mathbf{A}_{dg}\mathbf{M})\mathbf{x}^k \\ &+ (\mathbf{N}^{-1} - \mathbf{P}^{-1}_{MG(p,q,\gamma)}(\mathbf{A}_{dg}\mathbf{N}^{-1} - \mathbf{I}))\mathbf{b}.\end{aligned} \tag{3.18}$$

Note that the iterative scheme $(\mathbf{M}, \mathbf{N}^{-1})$ is applied in the DGSEM domain. Before the transfer function $\mathbf{T}_r$ has converted the DGSEM rhs to a FV rhs, or after the $\mathbf{T}_u$ transfer function has estimated the DGSEM solution from the FV approximate solution. See the pseudo code 2 for a detailed description.

As for any other linear stationary scheme, the matrix

$$(\mathbf{N}^{-1} - \mathbf{P}^{-1}_{MG(p,q,\gamma)}(\mathbf{A}_{dg}\mathbf{N}^{-1} - \mathbf{I})) \tag{3.19}$$

can be used as a preconditioner. If the smoother $(\mathbf{M}, \mathbf{N}^{-1})$ can remove error components the preconditioner $\mathbf{P}^{-1}_{MG(p,q,\gamma)}$ do not remove, then the combined preconditioner will be more effective than the original preconditioner, at the cost of requiring more Jacobian matvecs.

The following benefits are hypothesised to make the preconditioner in equation (3.19) with smoothing in the DGSEM domain speed up the GMRES convergence compared to the original preconditioner $\mathbf{P}^{-1}_{MG(p,q,\gamma)}$ without extra smoothing:

1. If the smoother is developed for the DGSEM problem directly, it can be made more effective than smoothing in the finite volume domain.
2. Postsmoothing can help dampen the large residual components of the error, the components contributing disproportionately to the 2-norm of the residual. Typically for discretizations of differential operators these would be high frequency components. A low norm error, but with large residual might prevent the GMRES algorithm from accepting an otherwise mostly correct component of the suggested solution.

To represent different versions of this scheme, we need seven parameters, $(a, b, p_f, q_f, p, q, \gamma)$ where a, b are the number of pre and postsmoothing in the DGSEM domain, p_f, q_f are the pre and post smoothings on the finest finite volume level, and p, q, γ are parameters of the finite volume multigrid method below the finest level. The preconditioner is

$$\mathbf{P}^{-1}_{(a,b,p_f,q_f,p,q,\gamma)} := (\mathbf{N}^{-1} - \mathbf{P}^{-1}_{MG(p,q,\gamma)}(\mathbf{A}_{dg}\mathbf{N}^{-1} - \mathbf{I})) \tag{3.20}$$

where $\mathbf{N}^{-1}$ is the smoother matrix in the DGSEM domain. Without pre- and postsmoothing, $\mathbf{N}^{-1} = 0$, and $\mathbf{P}^{-1}_{(0,0,p,q,p,q,\gamma)} = \mathbf{P}^{-1}_{MG(p,q,\gamma)}$.

In the results chapter 4 when refering to different methods, the seven parameters are written (ab, $p_f q_f$, $pq\gamma$). Or if $a = b = 0$ they may be written ($p_f q_f$, $pq\gamma$).

Below can be found pseudocode for the implementation of this preconditioner.

Algorithm 2 Apply preconditioner, calculate $\mathbf{x} := \mathbf{P}^{-1}_{(a,b,p_f,q_f,p,q,\gamma)}\mathbf{s}^k_{dg}$

$\mathbf{x} \leftarrow \mathbf{0}$
for $i \leftarrow 1 \dots a$ **do**
  $\mathbf{x} \leftarrow \mathbf{M}\mathbf{x} + \mathbf{N}^{-1}\mathbf{s}^k_{dg}$ ▷ Presmoothing in DGSEM domain
end for
$\mathbf{r} \leftarrow \mathbf{A}_{dg}\mathbf{x} - \mathbf{s}^k_{dg}$
$\mathbf{v} \leftarrow \mathbf{P}^{-1}_{(p,q,\gamma)}\mathbf{r}$ ▷ Apply multigrid preconditioner
$\mathbf{x} \leftarrow \mathbf{x} - \mathbf{v}$
for $i \leftarrow 1 \dots b$ **do**
  $\mathbf{x} \leftarrow \mathbf{M}\mathbf{x} + \mathbf{N}^{-1}\mathbf{s}^k_{dg}$ ▷ Postsmoothing in DGSEM domain
end for
$\mathbf{x}$

Chapter 4

Numerical experiments

4.1 Problems

This section contains the details of the problems solved in the experiments. All problems are instances of equation (2.5) in a square domain, but with different boundary values, initial conditions and diffusion coefficients.

The domain is discretized using DGSEM with varying polynomial degrees (0, 3 and 7) on a cartesian quadrilateral grid. The domain Ω is depicted in figure 4.1.

4.1.1 P1: Linear advection diffusion with zero boundary conditions

The first test problem has a zero Dirichlet boundary condition along the whole boundary. The flow direction is not aligned with the cartesian grid. The initial value is a Gaussian pulse in the center, its width is determined by the parameter c. The diffusion constant is varied between $1e-4$ and $1e-6$. In the form of equation (2.1), the problem is

$$\frac{du}{dt} + \mathbf{b} \cdot \nabla u = D\Delta u, \tag{4.1}$$

$$\Gamma_d = \partial\Omega, \quad u_d(\mathbf{x}, t) = 0, \tag{4.2}$$

$$u_0(\mathbf{x}) = \exp(-c||\mathbf{x} - \mathbf{1}\pi||^2), \tag{4.3}$$

where

$$\mathbf{b} = [1, 1], \quad D \in [1\text{e}{-4}, 1\text{e}{-6}], \quad c = 1. \tag{4.4}$$

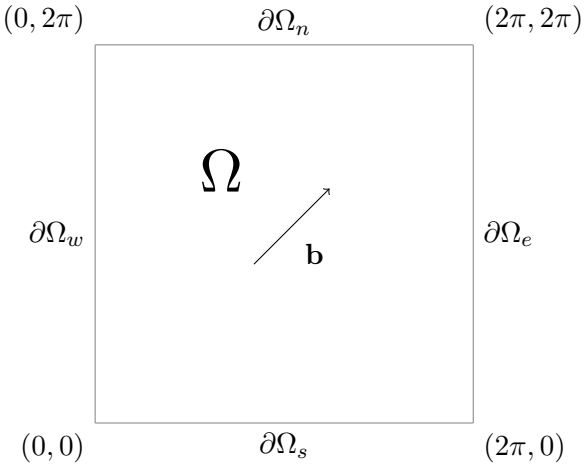

Figure 4.1: The domain used in the experiments, with some flow direction $\mathbf{b}$.

4.1.2 P2: Linear Advection Diffusion with a nonzero time dependent boundary condition

The second test problem has a time dependent Dirichlet boundary condition. In the form of equation (2.1), the problem is

$$\frac{du}{dt} + \mathbf{b} \cdot \nabla u = D\Delta u, \tag{4.5}$$

$$\Gamma_d = \partial\Omega, \tag{4.6}$$

$$u_d(\mathbf{x}, t) = \sin(\mathbf{b}^T(\mathbf{x} - \mathbf{b}t)) \exp(-2Dt), \tag{4.7}$$

$$u_0(\mathbf{x}) = \sin(\mathbf{b}^T\mathbf{x}), \tag{4.8}$$

where

$$\mathbf{b} = [1, 1], \quad D \in [1\text{e}{-4}, 1\text{e}{-6}]. \tag{4.9}$$

This problem has a closed form solution, which is used to determine the asymptotic rate of convergence for the time and space discretizations, the solution is $u(\mathbf{x}, t) = \sin(\mathbf{b}^T(\mathbf{x} - \mathbf{b}t)) \exp(-2Dt)$.

4.2 Distributed and Unified Numerics Environment

The Distributed and Unified Numerics Environment [4] (DUNE) is a collection of tools for solving PDEs with grid based methods. It contains structured and unstructred grid implementations with adaptivity, linear and nonlinear solvers, and a high level Python interface that allows specifying weak forms of PDEs (equation (2.27)-(2.30)) using the Unified Form Language (UFL) [8][11].

The forms can be evaluated using quadrature rules that are compiled for particular choices of approximation and test function spaces, for example Finite Volume or Discontinous Galerkin with some choice of polynomial basis. UFL is a common interface used by several PDE software packages for parsing PDE weak forms to discretizations. Given a geometry, and a domain partition such as a mesh or the quadrilateral grid used in the experiments in this thesis, UFL lets you define weak forms, integrals over the grid elements and their boundary surfaces. This is a powerful language for separating the description of the PDE discretization from the geometry and choice of approximation function basis. Not to mention, from the implementation of the numerical quadrature.

In 2.2.1 the integrals needed for describing our discretized problem in UFL are derived, the implementation of these integrals in UFL can be found in the code section on the next page.

```
from ufl import (
    TestFunction, TrialFunction, SpatialCoordinate,
    FacetNormal, dx, ds, dS, CellVolume, FacetArea,
    avg, jump, grad, dot, as_vector,
)
from dune.ufl import Constant

from dune.fem.scheme import molGalerkin
from dune.fem.space import finiteVolume

def linear_advection_diffusion_problem(
        *, boundary_value_function,
        initial_value_function,
        space_discretization=finiteVolume,
        dt=1e-2, b_direction, epsilon):

    space = space_discretization
    u = TrialFunction(space)
    v = TestFunction(space)
    n = FacetNormal(space)
    he = avg(CellVolume(space)) / FacetArea(space)
    hbnd = CellVolume(space) / FacetArea(space)
    x = SpatialCoordinate(space)

    eps = Constant(epsilon, name='epsilon')
    dt = Constant(dt, name='dt')
    t = Constant(0, name='time')

    b = as_vector(b_direction)
    hatb = (dot(b, n) + abs(dot(b, n))) / 2.0
    g = boundary_value_function(x, t + dt)
    u_h = space.interpolate(
        initial_value_function(x, 0),
        name='u_h')
    # penalty parameter
    beta = 10 * space.order**2 if space.order > 0 else 1

    def form(u):
        F = dot(b * u - eps * grad(u), grad(v)) * dx
        # diffusion skeleton
        F += eps * dot(avg(grad(u)), n('+')) * jump(v) * dS
        F += eps * dot(avg(grad(v)), n('+')) * jump(u) * dS
        F -= eps * beta / he * jump(u) * jump(v) * dS
        F += eps * dot(grad(u), n) * v * ds
        F += eps * dot(grad(v), n) * (u - g) * ds
        F -= eps * beta / hbnd * (u - g) * v * ds
        # advection skeleton
        F -= jump(u * hatb) * jump(v) * dS
        F -= (hatb * u + (dot(b, n) - hatb) * g) * v * ds
        return F

    scheme = molGalerkin((u - u_h) * v * dx == dt * form(u))
    return scheme, u_h
```

4.3 Multigrid method convergence baseline

In this section the efficiency of the FV multigrid method described in section 2.4.6, using 4 grid levels, is investigated. The linear problem solved arises from problem P1 with $D = 1e - 4$ from the experiments section 4 and solving for the solution at the next timestep. Δt is chosen to make the CFL number of the problem approximately 100.

The multigrid method is tested on two different grids and for different parameters corresponding to

- (1) different number of pre and post smoothings,
- (2) the kind of multigrid cycle applied (V or W),
- (3) using an exact linear solver (the spsolve method from the sparse linear algebra routines in Scipy) on the coarsest FV level or only applying the smoother on the finest level.

Note about notation The multigrid methods in this section are described by three numbers $(pq\gamma)$, the first is the number of presmoothings (see algorithm 1), the second is the number of postmoothings and the third denotes the kind of cycle used. Where 1 is a V cycle and 2 is an W cycle. If postfixed with "!" they do not use the exact FV solver on the coarsest grid level, instead they only apply smoothing. For the experiments further down in this chapter, the multigrid method might apply a different number of smoothings on the finest and the coarser levels. In that case the parameters are written $(p_f q_f, pq\gamma)$ where p_f and q_f are pre and post smoothings on the finest grid level, and $pq\gamma$ have the same meaning as before for the coarser grid levels. If DGSEM domain smoothing is applied, the parameters are written $(ab, p_f q_f, pq\gamma)$ where a and b are the number of pre and post smoothings in the DGSEM domain.

To calculate the Jacobian vector multiplications, the finite difference Jacobian approximation from equation (2.48) is used with $\epsilon = 1.0$. As smoother in the multigrid method, the two staged Runge-Kutta smoother (see equation (2.55)) with optimized parameters (see table 2.1) from [10] is used.

For the case of a 128×128 grid (16384 dofs), the results are presented in figure 4.2, and for the 256×256 grid (65536 dofs) in figure 4.3. Convergence is fast in both cases and largely unaffected by the grid resolution, which is the great advantage of multigrid methods [2][1]. In this case the test with higher grid resolution even converge somewhat faster than the lower resolution test. The reason for this could have something to do with the particular problem solved, or with the chosen smoother parameters which maybe could have been chosen better in the lower resolution case.

Conclusions about the different versions of the multigrid method

The number of smoothings on the fines grid level clearly influenced the asymptotic convergence. But for the first few iterations, factors such as cycle type or coarse grid solver played a larger role. During the first 3-4 iterations, method (012) with only a single smoothing on the finest grid level reached the same error norm as the method (221), with 4 smoothings on the finest grid level. Methods with W cycles converged faster during the first few iterations. If they had both pre- and postsmoothing the convergence continued to be fast without leveling of much. Methods with W cycles did not converge faster if they also had an exact solver on the coarse grid. Method (011!) with few smoothings was more affected by the lack of a coarse grid solver.

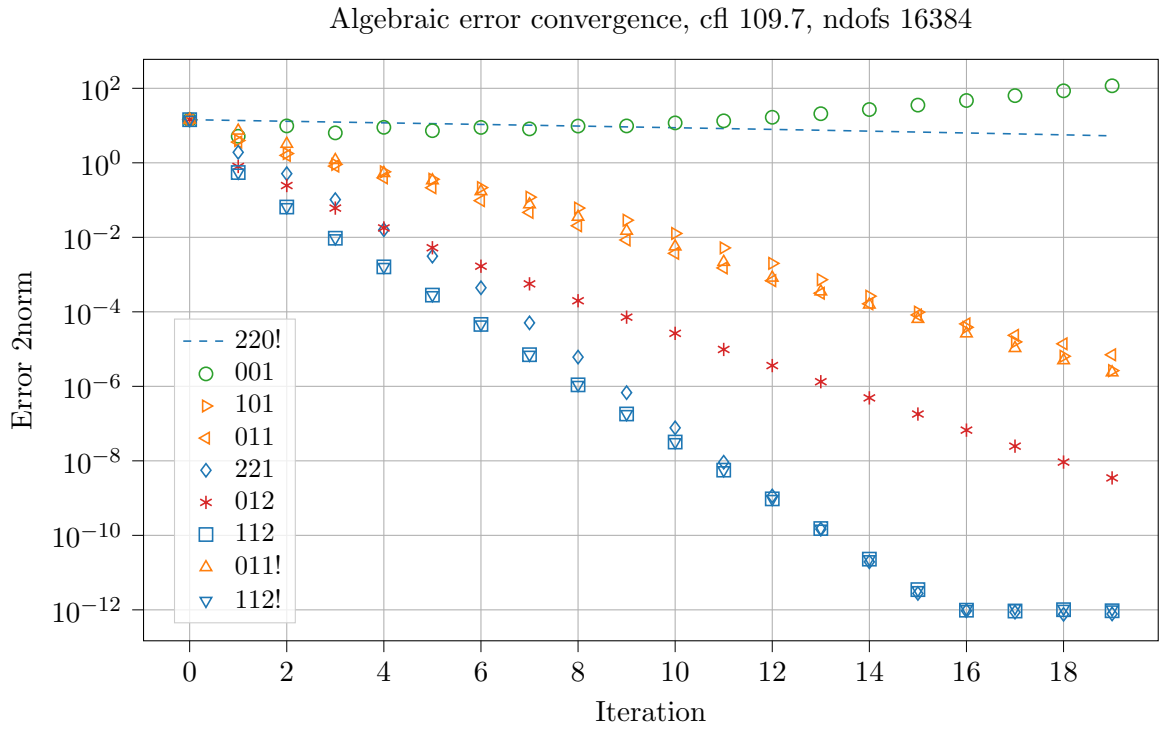

Figure 4.2: Solving one timestep in problem P1, linear advection diffusion, discretized with FV and implicit Euler, solved using the multigrid iteration in equation (2.72), $MG_{FV}(p, q, \gamma)$ on 4 grid levels, with or without exact coarse level solver. The labels in the figure denotes $pq\gamma$, number of pre- and postsmoothing steps, and $\gamma = 1$ for V cycle and $\gamma = 2$ for W cycle, and ! if the problem was not solved exactly on the coarsest grid. Methods with several smoothing applications are colored blue, methods with one smoothing and v cycle are colored yellow.

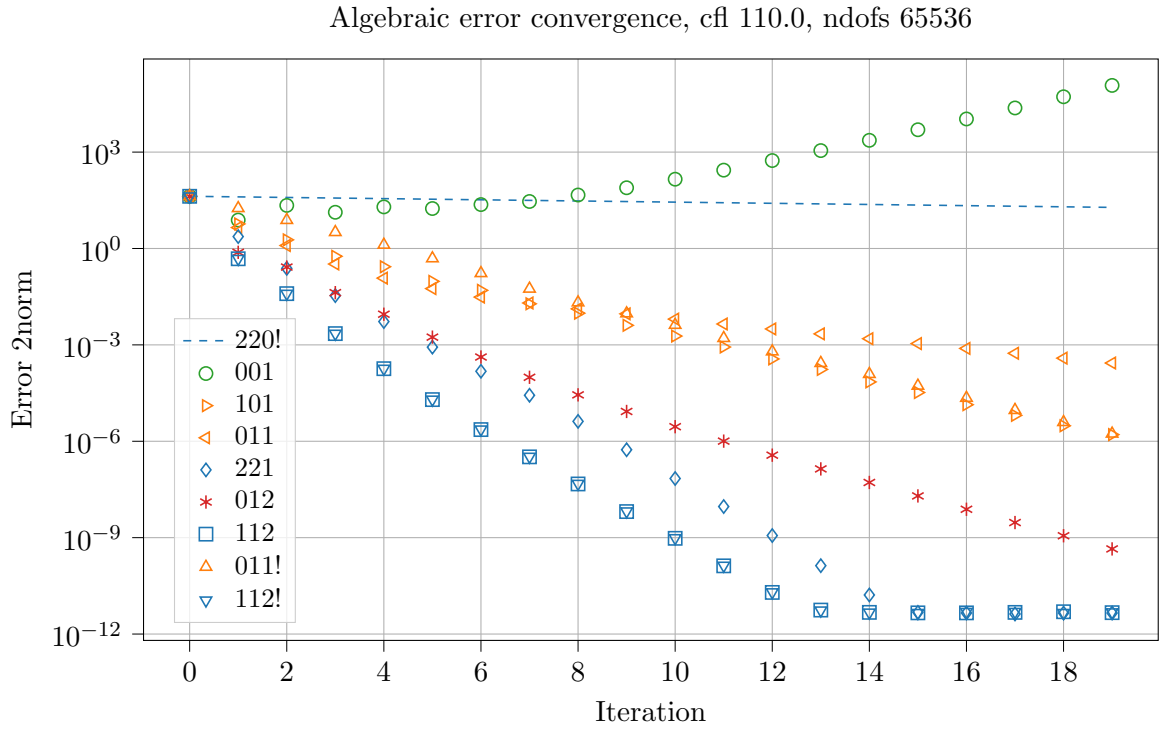

Figure 4.3: Same as 4.2 but number of dofs increased by a factor 4. Convergence rates are similar except for the pure smoothing method (does not use multigrid, dashed line) and the method with only post smoothing (left pointing triangle). The labels in the figure denotes $pq\gamma$, number of pre- and postsmoothing steps, and $\gamma = 1$ for V cycle and $\gamma = 2$ for W cycle. The label is postfixed with ! if the problem was not solved exactly on the coarse grid. Methods with several smoothing applications are colored blue, methods with one smoothing and V cycle are colored yellow.

These results serve to demonstrate that the multigrid method for the FV problem works as expected, and to establish a baseline for the performance of the preconditioner for a discretization of polynomial order 0. In the following sections we will see how the preconditioner performs when used within GMRES for higher order discretizations.

4.4 Transfer function comparison

In this section the effect on the preconditioner of using different transfer functions is investigated. Transfer functions are block diagonal matrices that are part of the preconditioner, see section 3.1 for more information.

Four different transfer functions are considered here. The operators inside the parentesis explains what should substitute $\mathbf{T}_r$ and $\mathbf{T}_u$ in equation (3.2) to form the preconditioner.

- **FV proj.** ($\mathbf{P}_{fv}$ and $\mathbf{P}_{fv}^{-1}$) Uses the projection on the FV space to transfer to the FV space, and the inverse of the projection on FV to transfer back to the DGSEM space.
- **DG proj.** ($\mathbf{P}_{dg}^{-1}$ and $\mathbf{P}_{dg}$) Uses the inverse of the projection on the DGSEM space to transfer to the FV space, and the inverse of the projection on DGSEM to transfer back to the DGSEM space.
- **Proj.** ($\mathbf{P}_{fv}$ and $\mathbf{P}_{dg}$) Uses the projection on FV to transfer to the FV space, and the projection on DGSEM to transfer to the DGSEM space.

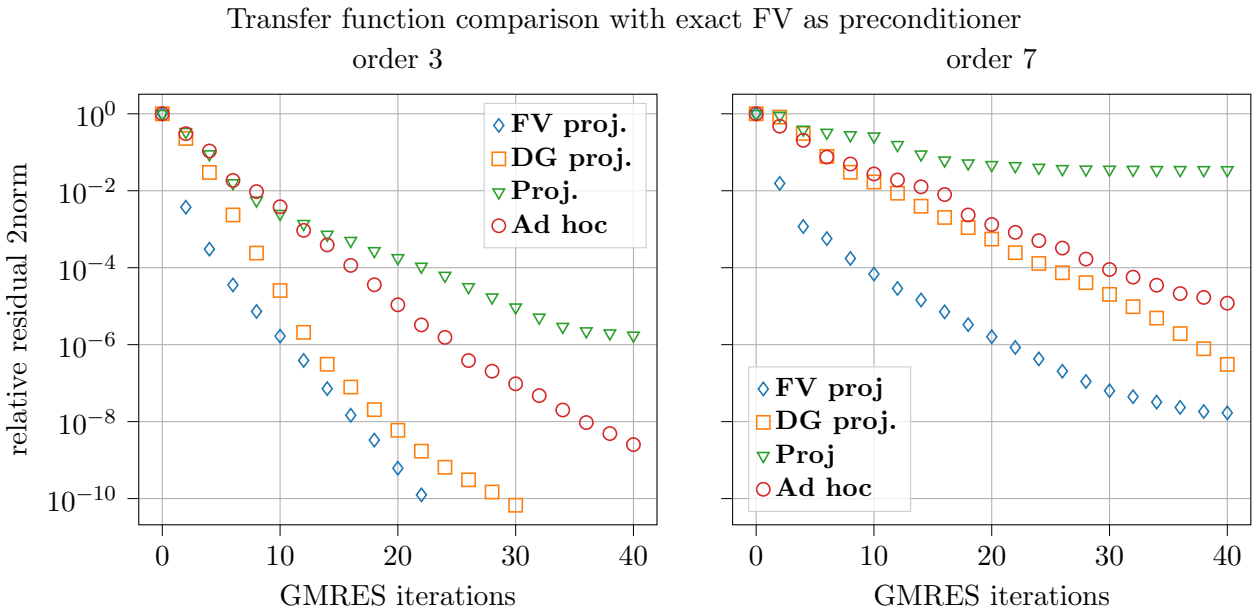

Figure 4.4: The figure shows the GMRES convergence history using the preconditioner in equation (3.2) with an exact FV solver instead of the multigrid method. For different choices of transfer functions $\mathbf{T}_r$ and $\mathbf{T}_u$.

- **Ad hoc** (**I** and **I**) Uses the Ad hoc transfer (identity) to transfer between the spaces.

The effect of the different transfer functions varies between different versions of the preconditioner, especially between those that uses smoothing in the DGSEM domain and the versions that do not (compare figure 4.4 and figure 4.6). Therefore three different cases are studied, one where the FV solver in the preconditioner is exact (see figure 4.4), one where the FV solver in the preconditioner is a multigrid V cycle (see figure 4.5), and one using the inexact FV solver but combined with smoothing in the DGSEM domain (see figure 4.6).

The L^2 projection based transfer functions where the transfers are each others inverses generally make the GMRES iteration converge fast, specially for DGSEM order 3. In the higher order case the difference between method 2 (DG proj.) and method 3 (Ad hoc) is less pronounced. Method 1 (FV proj.) do however converge much faster than the other methods for the first few iterations. See figure 4.4. The difference between the transfer methods disappears almost completely in the DGSEM order 7 case with inexact FV solution, see figure 4.5. This indicates that higher order makes the projection transfer functions more sensitive to errors in the FV solution.

When the problem is solved exactly in the FV domain (see figure 4.4) the L^2 projection on FV (method 1, blue diamond symbol) is a better transfer function than the projection on the DGSEM space (method 2, orange square symbol) for the first few iterations, their respective relative residuals differs by a factor of about $1e-2$ after 4 GMRES iterations. This is the case for both the low and high order DGSEM method, but at about 20 GMRES iterations (respectively 40 iterations for DGSEM of order 7) the two different methods achieve approximately the same reduction of the residual.

The slowest converging method in this case is the one using the L^2 projection in both directions (method 3), much worse than the Ad hoc transfer function (method 4). This is particularly interesting because it shows projecting the FV solution onto the DGSEM space does not approximate the DGSEM solution well, since in that case it should converge more similar to method 1. The difference between method 1 and 3 is only the transfer from FV to DGSEM, and method 1 converge faster than 3 in most cases, with the exception of

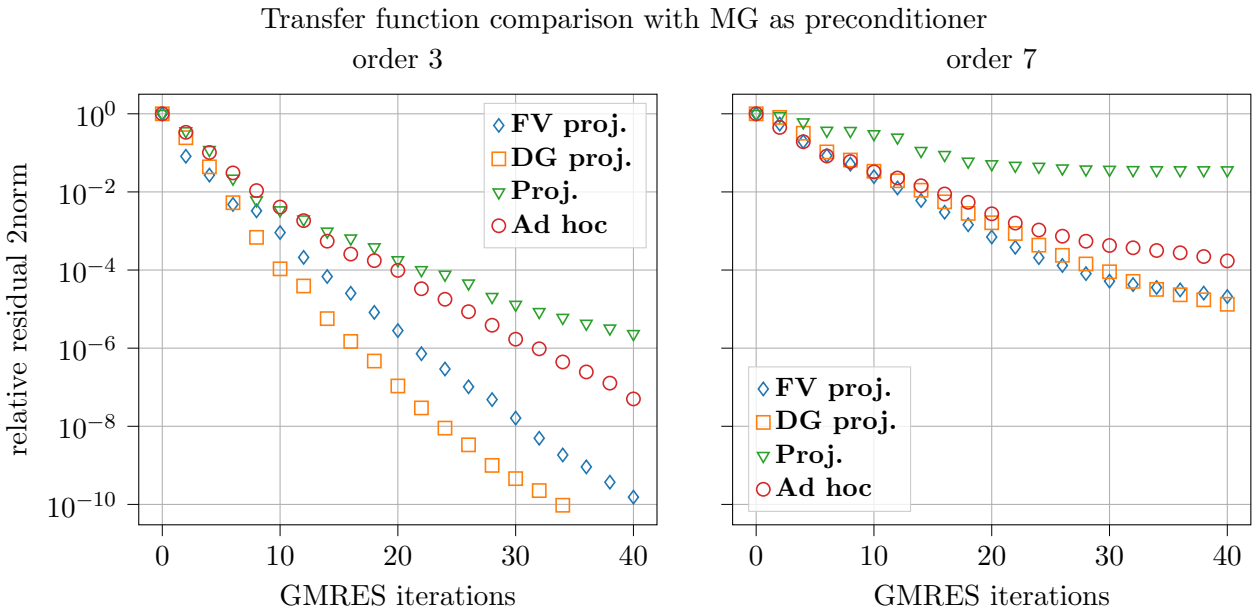

Figure 4.5: Same as figure 4.4 but using the multigrid method to compute the solution in the FV domain instead of the exact solver.

the case with inexact FV solution and smoothing in the DGSEM domain, see figure 4.6. This indicates that either the errors introduced by method 3 are particularly difficult for the GMRES algorithm to remove compared to the errors introduced by the other transfer methods, or it is comparably more easy to remove for the DGSEM domain smoother.

Using the multigrid method (inexact finite volume solution) as preconditioner (see figure 4.5) changes the situation. The inexact solver makes method 2 the one requiring fewest GMRES iterations to reach convergence, possibly because the map it uses to go back to DGSEM is contracting, and does not amplify the errors introduced by the inexact FV solution. Method 1 interestingly becomes considerably worse, and requires almost twice as many iterations compared to the exact FV case (see figure 4.4, but is still the fastest the first 5 iterations. The performance of the Ad hoc transfer (method 4) and the projection in both directions (method 3) is relatively unaffected by the inexact FV solution, but both converge slower than with an exact FV solution.

Using an inexact FV solution as before but moving two of the smoother applications to the DGSEM domain, the preconditioner **mg and dg** with parameters (11, 11, 111), the situation changes considerably again (see figure 4.6). All transfer functions converge faster compared to when using the multigrid preconditioner without DGSEM domain smoothing. Surprisingly, the fastest transfer function is now method 3 which was the slowest in both the previous experiments.

The other most notable difference is that all transfer functions now need about the same number of GMRES iterations, it seems no matter the errors components introduced by different transfer functions, they are apparently all reduced about equally well by the Runge-Kutta pseudo time smoother in the DGSEM domain. However, the FV solver is now very inexact and we know this has the effect of making all methods perform worse, but particularly method 1, so it is likely that would converge faster if the FV solver was more exact.

Compared to the previous experiments in figure 4.5 and 4.4 method 3 and method 4 converge much faster, even when compared to using the exact FV solver as preconditioner. This indicates that a cheap FV solution, a cheap transfer function and a cheap DGSEM domain smoother could combine to get relatively fast convergence. But the fast results

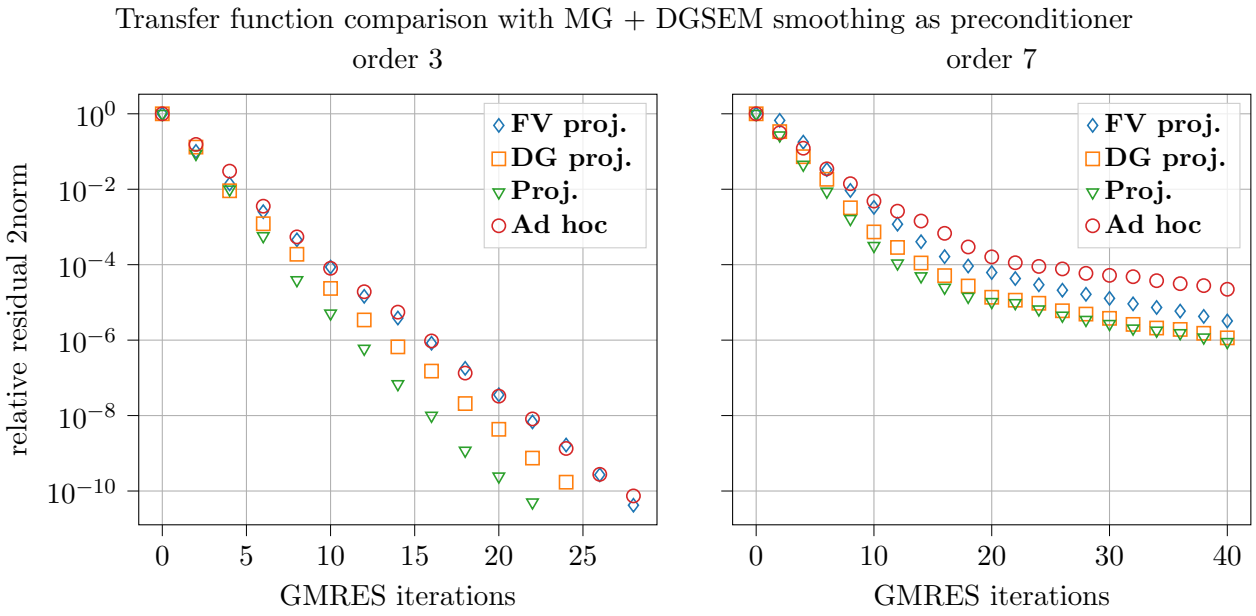

Figure 4.6: Same as figure 4.4 but using the multigrid method to compute the solution in the FV domain instead of the exact solver, and dividing the smoothings on the finest grid between the FV domain and the DGSEM domain.

are only for DGSEM order 3, see figure 4.6). For order 7 the convergence stagnates, the reason for this could be that the DGSEM domain smoother does not work as well for DGSEM order 7.

4.4.1 Evaluating the choice of transfer function

In figures 4.4-4.6 a comparison is made between using transfer functions. When the FV solver is exact, the transfer function based on projection on FV performs well, especially for the first few GMRES iterations. This confirms the guess in section 3.1.1 that method 1 should work better. The motivation for that guess was that method 1 is designed for the situation when the DGSEM solution is much closer in norm to the true solution than the FV solution is. And we know the DGSEM solution should be more accurate because it is of higher order.

In the rest of the experiments the transfer function used in the preconditioner is always method 1 (FV Proj.) if nothing else is specified.

4.5 DGSEM domain smoothing

The effect of smoothing in the DGSEM domain is illustrated in figure 4.7 where four different preconditioners are compared, all using the transfer FV Proj. (method 1).

Notation for the different preconditioner versions tested

- **exact and dg** Uses the exact finite volume solver instead of the multigrid method. Applies one presmoothing and one postsmoothing in the DGSEM domain.
- **exact** Uses the exact finite volume solver instead of the multigrid method. No smoothing in the DGSEM domain.
- **mg and dg** Multigrid method with parameters (11, 111) as finite volume solver. Apllies one presmoothing and one postsmoothing in the DGSEM domain. For a total of 4 smoother applications on the fine grid.

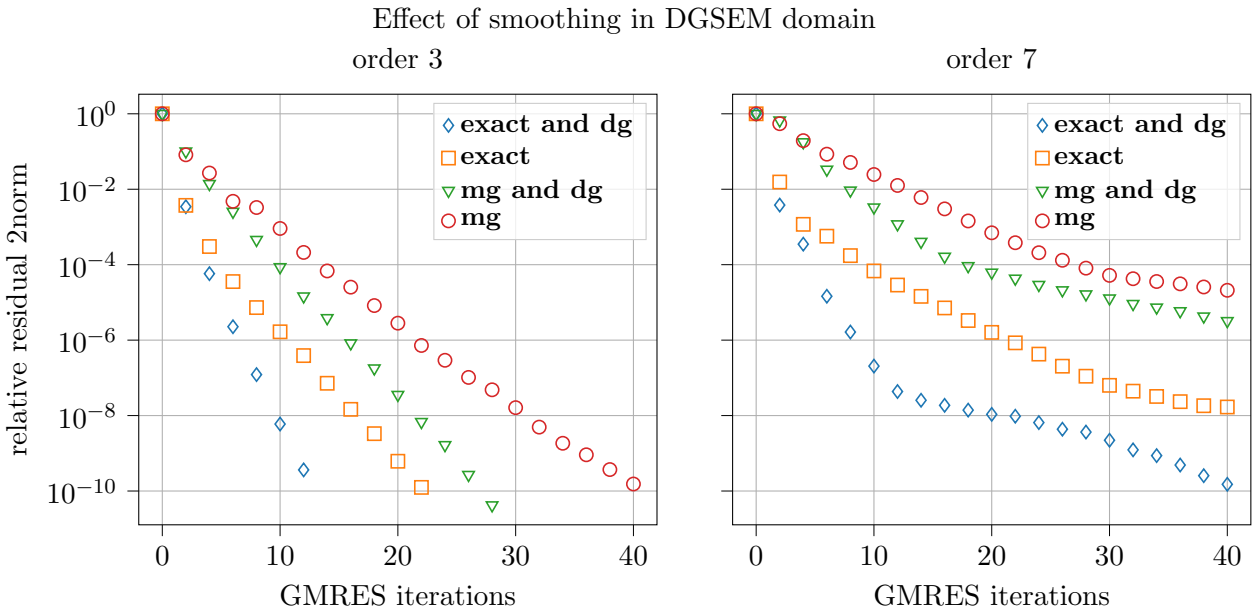

Figure 4.7: GMRES convergence history for different versions of the preconditioner. All use the method 1 transfer function, but differ on how the FV problem is solved, and on the number of smoothings applied in the FV respectively the DGSEM domain.

- **mg** Multigrid method with parameters (22, 111) as finite volume solver. No smoothing in the DGSEM domain. For a total of 4 smoother applications on the fine grid.

Two of the preconditioners uses an exact FV solver and shows how much the other methods could potentially gain by adding more smoothing or improving the smoother in the FV multigrid method. The two other methods use the multigrid V-cycle to solve the FV problem, but differs in that one only uses smoothing in the FV domain and the other applies the Runge-Kutta smoother both in the DGSEM and the FV domain.

Applying smoothings in the DGSEM domain makes all the methods converge faster compared to applying more smoothings in the FV domain. Looking at the preconditioner **mg and dg**, the number of GMRES iterations required is about 30% lower than the **mg** preconditioner in the case when solving the FV problem inexactly. Note here that the two preconditioners have the same number of smoothings on every grid level, so they are equal in cost assuming applying the operators are equally expensive in the DGSEM and the FV domain, which makes the faster convergence gained by smoothing in the DGSEM domain "free of charge". Generally it is not the case that the operator applications are equally expensive, the DGSEM operator should be more expensive since it corresponds to a denser matrix multiplication. But it is not always clear cut. In this project for example, the operator from the DGSEM($p = 3$) discretization was actually faster to apply than the operator from the FV discretization with the same number of dofs. See section 4.9 for more discussion on this topic.

4.6 Problems with boundary conditions

The performance of the preconditioner varies a lot between problem P1 and P2. When the problem has a nonzero Dirichlet boundary condition (P2), the GMRES iteration stagnates and the relative residual is not reduced much beyond $1e-4$, see figure 4.8. In this sections this issue is analyzed and a potential remedy presented.

Assume a problem without sources or sinks in the domain ($g = 0$) and only Dirichlet boundary conditions ($\Gamma_d = \partial\Omega$). Then for the two different discretizations, FV and

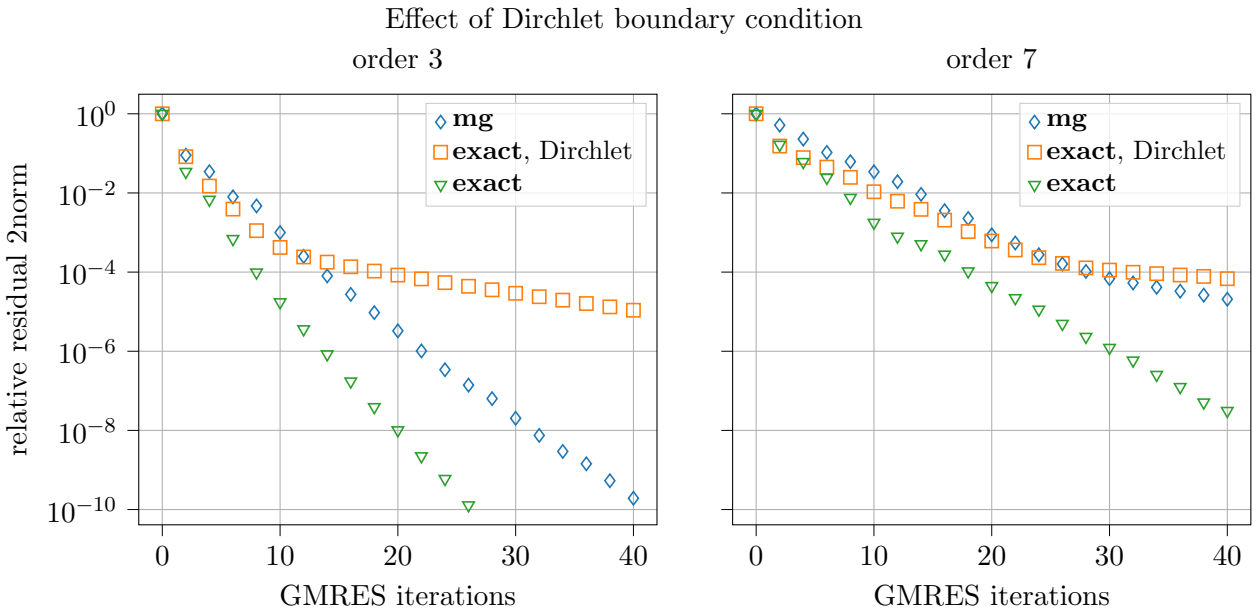

Figure 4.8: The GMRES convergence comparison betweeen using the preconditioner to solve a problem with a zero Dirichlet boundary conditions, and a problem with non-zero Dirichlet boundary (denoted "Dirichlet" in the labels). The Dirichlet boundary slows down the convergence considerably.

DGSEM of order p, (2.34) gives us the system

$$\mathbf{A}_{dg}\mathbf{u}_{dg}^{k+1} = \mathbf{u}_{dg}^{k} + \mathbf{d}_{dg}^{k+1} = \mathbf{s}_{dg}^{k} \tag{4.10}$$

for the DGSEM discretization, and the system

$$\mathbf{A}_{fv}\mathbf{u}_{fv}^{k+1} = \mathbf{u}_{fv}^{k} + \mathbf{d}_{fv}^{k+1} = \mathbf{s}_{fv}^{k} \tag{4.11}$$

for the FV discretization. $\mathbf{A}_{dg}$ and $\mathbf{A}_{fv}$ are the Jacobians for the DGSEM problem respectively the FV problem, $\mathbf{d}_{dg}^{k+1}$ respectively $\mathbf{d}_{fv}^{k+1}$ are the components of the rhs that arise from the Dirichlet boundary condition, see equation (2.38).

As the grid is refined, the solutions from the different discretizations converge to the same true solution if the data (the $\mathbf{s}_{dg}^{k}$ and $\mathbf{s}_{fv}^{k}$ vectors) corresponds to the same problem (the same initial condition and the same boundary condition). If the initial condition $u(\mathbf{x}, t_k)$ and the boundary condition $u_d(\mathbf{x}, t_{k+1})$ are known, then $\mathbf{s}_{fv}^{k}$ can be calculated the way it would if we discretized the problem using FV from the beginning. But if they are not known, then as discussed in section 3.1 we want to estimate them using transfer functions.

Assuming we know $\mathbf{d}_{dg}^{k+1}$. By making assumptions about $u_d(\mathbf{x}, t_{k+1})$ it is possible to find an estimate for $\mathbf{d}_{fv}^{k+1}$. From the discretization in section 2.2.1, equation (2.38), we get that

$$\mathbf{d}_{dg}^{k+1} = \Delta t \mathbf{M}_{dg}^{-1}\big(H(u_d(,\mathbf{x}, t_{k+1}), \phi_j)\big)_{j\in I}, \tag{4.12}$$

$$\mathbf{d}_{fv}^{k+1} = \Delta t \mathbf{M}_{fv}^{-1}\big(H(u_d(t_{k+1}), v_j)\big)_{j\in I_{fv}}, \tag{4.13}$$

$$H(u, \varphi) = \langle u, (D\mu/h_\gamma + (\hat{b} - |\mathbf{b}\cdot\mathbf{n}|))\varphi - D\nabla\varphi\cdot\mathbf{n}\rangle_{\Gamma_d}. \tag{4.14}$$

Assume $u_d(\mathbf{x}, t_{k+1})$ is in the span of the DGSEM basis functions on the boundary, then

$$u_d(\mathbf{x}, t_{k+1}) = \mathbf{\Phi}^T\mathbf{u}_d, \quad \forall\mathbf{x}\in\Gamma_d \tag{4.15}$$

for the unique coefficient vector $\mathbf{u}_d$.

Using (4.13) and (4.15),

$$\mathbf{d}_{fv}^{k+1} = \Delta t \mathbf{M}_{fv}^{-1}\big(H(u_d(\mathbf{x}, t_{k+1}), v_j)\big)_{j\in I_{fv}} \tag{4.16}$$

$$= \Delta t \mathbf{M}_{fv}^{-1}\big(H(\mathbf{\Phi}^T\mathbf{u}_d, v_j)\big)_{j\in I_{fv}} \tag{4.17}$$

$$= \Delta t \mathbf{M}_{fv}^{-1}\big(H(\phi_i, v_j)\big)_{ji}\mathbf{u}_d \tag{4.18}$$

and using (4.12) and (4.15),

$$\mathbf{d}_{dg}^{k} = \Delta t \mathbf{M}_{dg}^{-1}\big(H(u_d(\mathbf{x}, t_{k+1}), \phi_j)\big)_{j} \tag{4.19}$$

$$= \Delta t \mathbf{M}_{dg}^{-1}\big(H(\mathbf{\Phi}\mathbf{u}_d, \phi_j)\big)_{j} \tag{4.20}$$

$$= \Delta t \mathbf{M}_{dg}^{-1}\big(H(\phi_i, \phi_j)\big)_{ji}\mathbf{u}_d \tag{4.21}$$

implies that we can estimate (under the assumption (4.15))

$$\mathbf{d}_{fv}^{k+1} = \mathbf{M}_{fv}^{-1}\big(H(\phi_i, v_j)\big)_{ji}\big(H(\phi_i, \phi_j)\big)_{ji}^{-1}\mathbf{M}_{dg}\mathbf{d}_{dg}^{k}. \tag{4.22}$$

In equation (4.22) we have an estimate of $\mathbf{d}_{fv}^{k}$ provided that $\mathbf{d}_{dg}^{k}$ is known. As can be seen, it is not the $L^2(\Omega)$ projection that transfers information about the boundary component. It is also not the $L^2(\Gamma)$ projection, even if it has some similarities to it (H involves an inner product on Γ). This explains why the convergence slows down in the presence of boundary conditions, see figure 4.8. The transfer functions we use do not handle the boundary component $\mathbf{d}_{dg}^{k}$ well. Therefore, the FV problem is not given the same data as the DGSEM problem and does not produce a similar solution, which makes the preconditioner perform worse.

Separate transfer functions for different components of the rhs

If the reason for the worsened convergence in presence of boundary conditions is the issue described above. It is expected that applying different transfer functions for $\mathbf{u}_{dg}^{k}$ and the boundary component $\mathbf{d}_{dg}^{k+1}$ separately, the FV representation of the DGSEM problem would improve and the issue be resolved. Figure 4.9 indicates that this is indeed the case, since the solution obtained by treating $\mathbf{u}^k$ and $\mathbf{d}_{dg}^{k+1}$ separately is better than transferring the whole rhs using the same transfer function.

The values of the rhs components are individually unknown

The problem with using different transfer function on different components is that we usually do not have access to the previous solution component $\mathbf{u}_{dg}^{k}$ and the boundary component $\mathbf{d}_{dg}^{k+1}$ separately throughout the GMRES iteration. They are only available as separate vectors before the first GMRES iteration. For the later iterations we only observe the sum $\mathbf{s}_{dg}^{k}$, which can be interpreted as the sum of a "boundary term" and an "initial value term" that are separately unknown. It is then not clear how to extract the data to set up a representative FV problem.

But the problem is simpler than it seems at the first glance because the boundary term $\mathbf{d}_{dg}^{k}$ can only have non-zero entries for the DGSEM basis functions that live in cells adjacent to the boundary. Therefore all other coefficients in $\mathbf{s}_{dg}^{k}$, those not associated with basis functions in boundary adjacent cells, are known to be the coefficients of the the previous time step component $\mathbf{u}_{dg}^{k}$, and can be transferred accurately to the FV discretization by the $L^2(\Omega)$ projection.

On the boundary there is ambiguity about how to interpret the data. Imagine a situation where the true but unknown $\mathbf{u}_{dg} = \mathbf{0}$ is all zero, corresponding to the solution at the

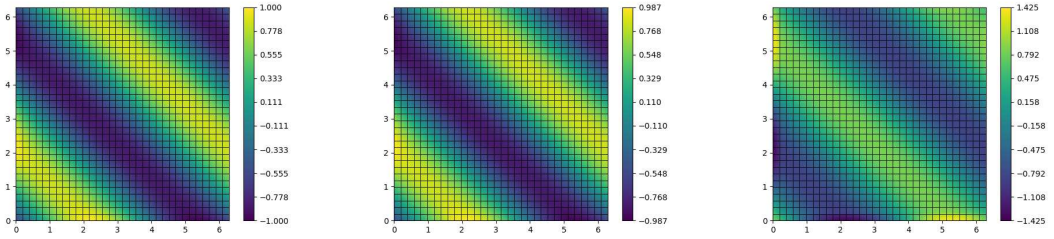

Figure 4.9: Three DGSEM solutions to an advection diffusion problem with Dirichlet boundary conditions. (left) Is the exact solution. (middle) Is an approximate solution obtained by transferring a FV solution to the DGSEM grid. Where the approximate FV rhs is $\hat{\mathbf{s}}^k_{fv} := \mathbf{P}_{fv}\mathbf{u}^k_{dg} + \mathbf{d}^{k+1}_{fv}$. The $L^2(\Omega)$ projection was used to transfer $\mathbf{u}^k_{dg}$, but then the exact FV $\mathbf{d}^{k+1}_{fv}$ was added. (right) Is an inexact solution transferred from a FV solution, with approximate rhs obtained by projecting the entire DGSEM rhs to the FV space, $\hat{\mathbf{s}}^k_{dg} = \mathbf{P}_{fv}\mathbf{s}^k_{dg}$. The last method (right) gives a bad solution because the $L^2(\Omega)$ projection is not a good transfer function for the boundary component $\mathbf{d}^{k+1}_{dg}$.

previous timestep $u(\mathbf{x}, t_k) = 0$. And where the true but unknown $\mathbf{d}^{k+1}_{dg}$ corresponds to a large constant value of $u_d = C$ on the boundary. We only observe the vector $\mathbf{s}^k_{dg} = \mathbf{u}^k_{dg} + \mathbf{d}^{k+1}_{dg}$. The $\mathbf{s}^k_{dg}$ vector could either be interpreted as a very thin mass distribution in the boundary adjacent cells, or as a boundary condition (the true interpretation), or a combination thereof. The interpretation decides what transfer function should be applied to estimate $\mathbf{s}^k_{fv}$ used to obtain a FV solution approximating the DGSEM problem. But how should the interpretation be made?

A very simple interpretation could for example be estimating all entries of $\mathbf{u}^k_{dg}$ in cells adjacent to the boundary as constant. The difference between the observed rhs and the estimated $\mathbf{u}^{k+1}_{dg}$ would then be the estimate for $\mathbf{d}^{k+1}_{dg}$. Then the two components could each be transferred to to FV domain using appropriate, different, transfer functions.

4.7 Effect of increasing the diffusion coefficient

In this section the effect of changing the diffusion coefficient is investigated. The problem solved is P1, the advection diffusion problem with zero Dirichlet boundary conditions. Two cases are considered, one with smoothing in the DGSEM domain (see figure 4.11, and one without (see figure 4.10). In both cases, the preconditioning methods using an exact FV solver converge slower when the problem is more diffusive, the effect is largest for the high order DGSEM problem solved without smoothing on the DGSEM domain. Preconditioners using multigrid to solve the FV problem are only marginally affected by changing the diffusion coefficient.

4.8 Effect of increasing grid resolution

Increasing the grid resolution makes the GMRES iteration converge faster for both high and low order DGSEM, and for exact and inexact FV solutions in the preconditioner (see fig 4.12 respectively 4.13). This is resonable because when the grid is refined both the FV solution and the DGSEM solution converge to the true solution, and in the limit of a very dense grid the solution is constant on each grid element, and then any transfer function that preserves constant functions (all transfer functions considered in this thesis does this) will transfer solutions exactly between the grids.

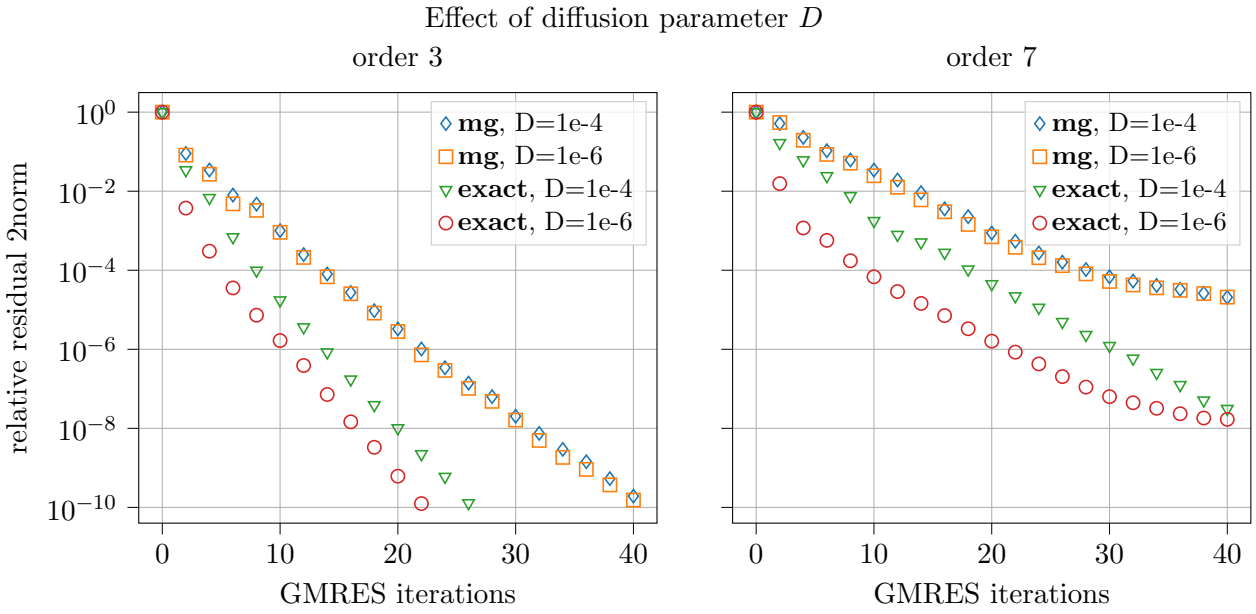

Figure 4.10: GMRES convergence history when using the preconditioner for different values of the diffusion coefficient D. Method 1 is used as transfer function, the convergence is tested for $D = 1e-4$ and $D = 1e-6$, and for two versions of the preconditioner, one with the exact FV solver and one using multigrid to solve the FV problem approximately.

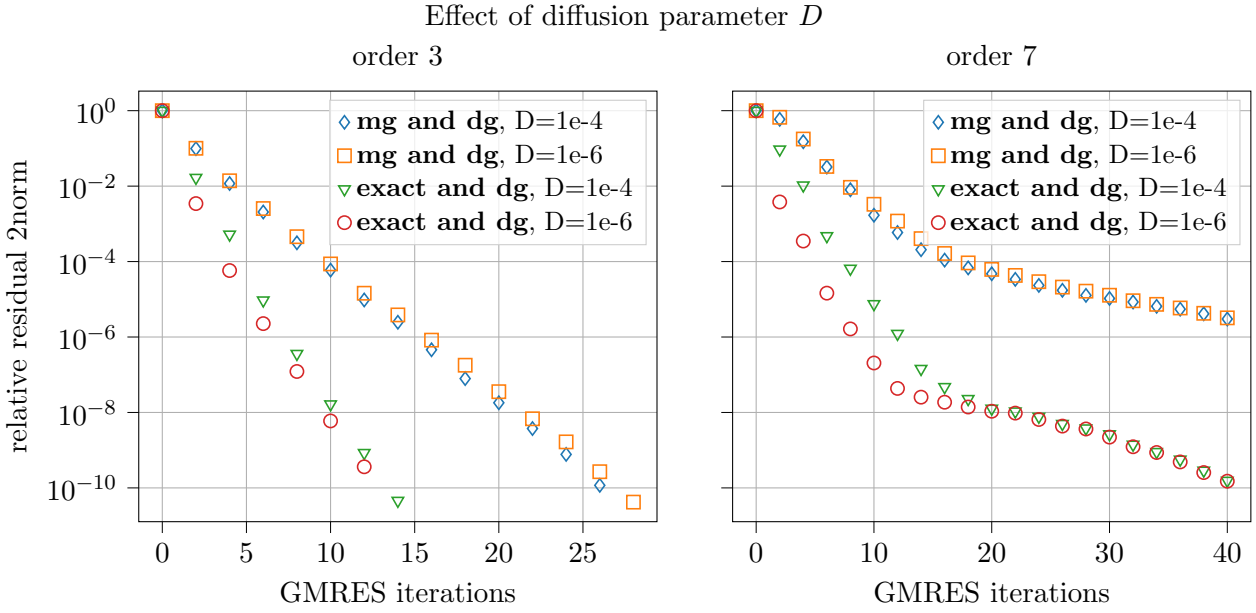

Figure 4.11: Same as figure 4.10 but using preconditioner methods with smoothing also in the DGSEM domain.

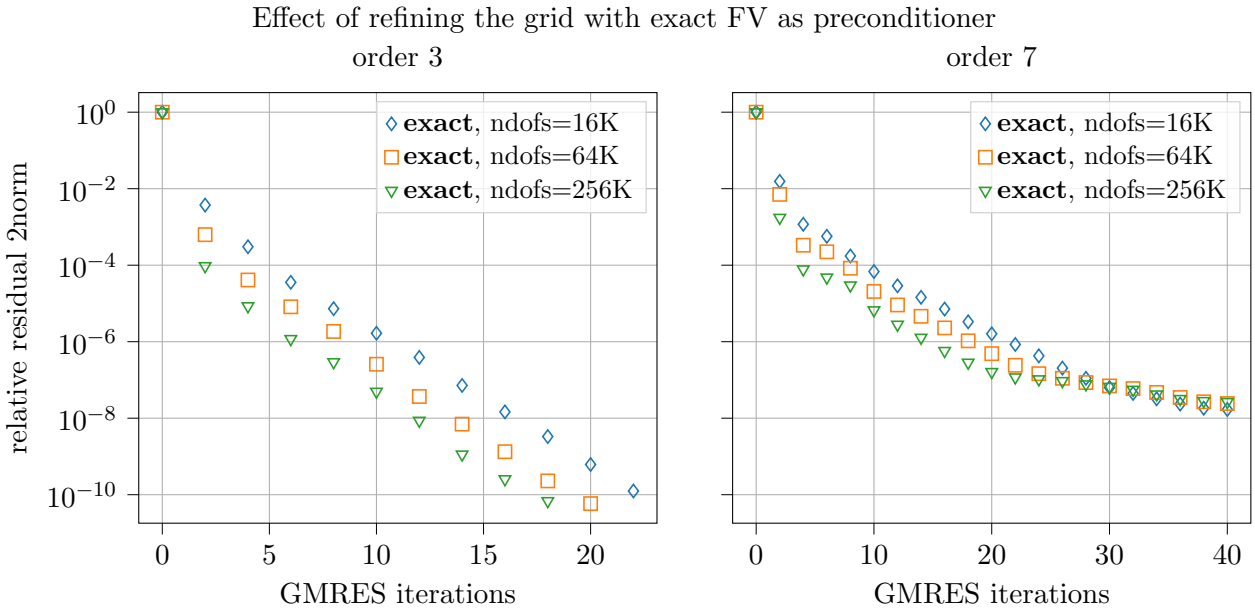

Figure 4.12: This figure shows how the GMRES convergence changes with increasing grid resolution. All preconditioners use the exact FV method instead of the multigrid method. For DGSEM($p = 7$), the convergence stagnate after about 20 iterations.

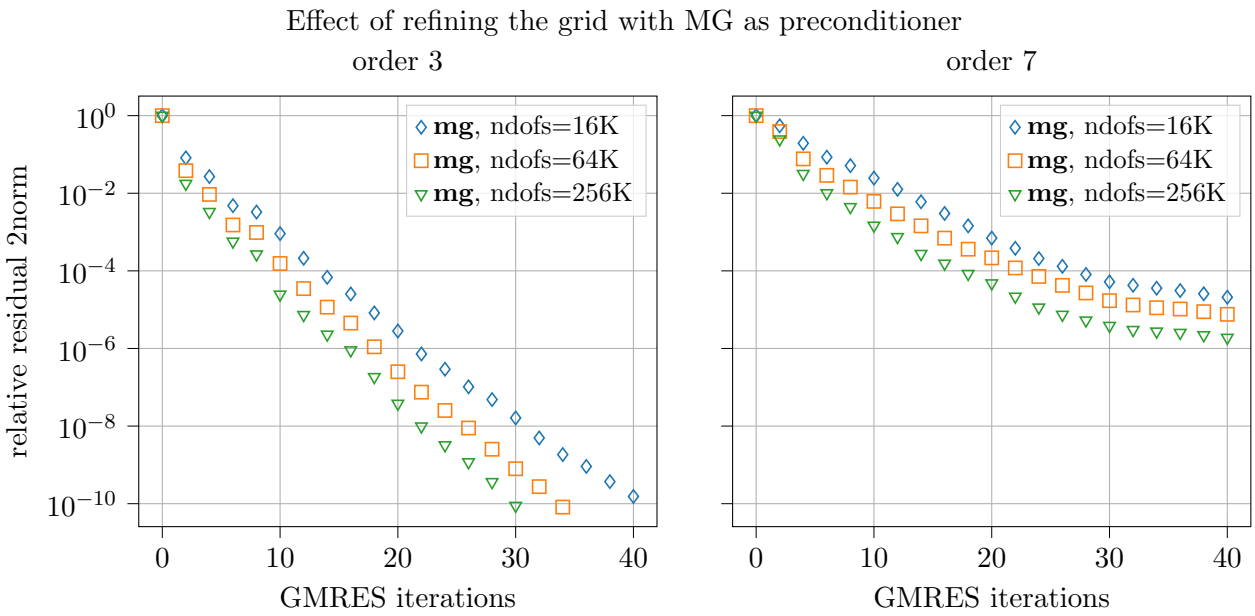

Figure 4.13: Shows how the GMRES convergence changes with increasing grid resolution. All preconditioners uses the multigrid method and no DGSEM domain smoothing.

4.9 CPU timing analysis

In this section an analysis of the CPU timings is performed to get an idea of how the different methods compare. Only the MG method based preconditioner with DGSEM domain smoothings is analysed since it realizes the other methods for particular choices of parameters.

The main computational cost in the program comes from the evaluations of the forms in DUNE, which occur in the finite difference Jacobian matvecs which are needed when applying smoothers in the multigrid based preconditioner. These dominate the cost for the sizes of problems in this thesis, and presumably also beyond that since in theory the program should scale as $\mathcal{O}(n)$ where n is the number of unknowns. With the exception of the coarse problem solver in the multigrid method, which scales as $\mathcal{O}(n^2)$, but that cost can be reduced by adding more levels to the multigrid scheme and is assumed to be negligible.

Notation:

- C_f: cost of evaluating the finite volume scheme.
- C_d: cost of evaluating the discontinuous galerkin scheme.
- n_S: number of Jacobian vector products in the smoother
- a: DG grid smoothing applications
- b: FV fine grid number of smoothing applications
- c: Number of smoother applications on every coarser MG level

Under the assumptions that the prolongation, restriction and discretization transformations do not cost anything in terms of operations, and that the number of GMRES iterations are low enough that the cost per iteration is constant (Jacobian matvec applications dominate), then the cost per GMRES iteration, C_{tot} is approximately

$$C_{tot} \approx n_s(C_f(b + \frac{c}{3}) + C_d a) + C_d.$$

How well does this approximation hold in practice? For a problem with 16800 dofs, $C_f \approx 83.6ms$ and $C_d \approx 16.8ms$. Three methods were tested, (1) MG with DG domain smoothings with one pre and one post smoothing in the DG domain, and one post smoothing on the finest FV level, and two post smoothings on the coarser FV levels, $C_{tot} \approx 7C_d + 5C_f$, (2) MG(01), $C_{tot} \approx 1C_d + 4C_f$, and (3) MG(02), $C_{tot} \approx 1C_d + 8C_f$. All with 3 multigrid levels and exact solving on the coarsest level. The timings measured are collected in Table 4.1.

The timings in Table 4.1 look especially competitive for the method with DGSEM domain smoothings, but we can now see that one reason that method has lower cost is because C_d is lower than C_f in this implementation. Assuming $C_d =: C_f$ the cost for MG with DG domain smoothings in the example above instead becomes $38.1s$ (according to the approximation C_{tot}), which is comparable to the $MG(01)$ method. So in this case it's not clear that any benefit is gained by using MG with DG domain smoothings over the simpler MG, unless the parameters are tuned well for the problem at hand, which gives an advantage to the MG with DG domain smoothings method since it has more parameters available. It also seems that the smoother used is more effective in the finite volume domain, where its parameters are optimized. Using a smoother more adapted to the DG discretization (for example by optimizing the smoother parameters, or by using a different method) might make MG with DG domain smoothings more competitive. Since MG with DG domain smoothings is a generalisation of MG it is always at least as efficient by definition, at a cost of being slightly more complex, and with the advantage that it adds more flexibility to construct better methods.

Method	GMRES iterations, k	kC_{tot} [s]	Time measured [s]
MG with DG domain smoothings	38	20.35	20.57
MG(01)	98	34.4	38.0
MG(02)	78	53.5	53.1

Table 4.1: Performance experiment for problem 3 with 16K dofs. The estimated times are close to the measured, this shows the assumptions made in the analysis are sound and that the time to solve really is dominated by the smoother applications. The difference between expected and measured is largest for the MG(01) method, this is probably because the overhead costs (the not smoother application costs) assumed to be zero makes up a comparably larger part of the cost for that method.

Chapter 5

Summary and Outlook

A multigrid based Jacobian free preconditioner for a two-dimensional advection-diffusion problem discretized by Discontinuous Galerkin method with Legandre-Gauss-Lobatto basis and implicit time stepping has been implemented following the work in [2] and [3]. Two strategies have been tested to further improve the performance of the preconditioner.

1. Using more sophisticated transfer functions to reinterpret the data on the DGSEM grid as a problem in the finite volume domain with the same or a very similar solution as the DGSEM problem, and
2. combining the finite volume multigrid based preconditioner with smoothing in the DGSEM domain.

The preconditioner does not handle problems with nonzero boundary conditions well and a theoretical explanation and possible remedy have been suggested, see section 4.6. For problems with zero boundary conditions, smoothing in the DGSEM domain improved the GMRES convergence, it also decreased the influence of the transfer functions since different methods performed more similarly when smoothing was applied in the DGSEM domain.

Conclusions

For the preconditioning method discussed in this thesis there is much to gain by designing the transfer functions between the different discretizations carefully, especially if the finite volume problem can be solved with high accuracy. For a test problem with zero Dirichlet boundary conditions, depending on the level of accuracy in the finite volume solution, the norm of the relative residual in the test problem could be reduced to $1e-10$ in less than 40 GMRES iterations for DGSEM with polynomial basis of order 3 and to below $1e-4$ in 40 GMRES iterations for DGSEM order 7 using the preconditioner compared to less than $1e-1$ in 40 iterations for the unpreconditioned problem.

If efficient smoothers can be constructed in the DGSEM domain they can speed up the convergence further. When smoothing was shifted from the FV domain to the DGSEM domain the residual could be reduced to $1e-10$ in less than 30 GMRES iterations for the DGSEM order 3 discretization, while being more robust to the choice of transfer function.

For a problem with nonzero Dirichlet boundary conditions the preconditioners were observed to perform considerably worse for the tested transfer functions. Using the same preconditioner for a problem with and without boundary conditions the relative residuals differed by a factor of $1e5$ after 40 GMRES iterations. The reason for this is hypothesized to be that the component in the right-hand-side corresponding to the boundary condition

is not well approximated in the finite volume domain by the transfer functions used. An alternative transfer function to handle problems with boundary conditions is suggested in section 4.6. The question if transfer functions handling data from the bulk and data from the boundary can be combined to work well generally is left for further study to determine.

Outlook

Similar preconditioning strategies to the ones investigated in [2] and in this thesis are used in [12]. Among other things the authors of that article precondition DGSEM problems using a finite volume discretization with modified numerical fluxes defined on the same grid. Combining it with a powerful but expensive smoother in the DGSEM domain they achieves fast Jacobian free GMRES convergence. One large benefit of the approach compared to ours is that no transfer functions are required, since the finite volume basis is coarser and is contained in the DGSEM basis. On the other hand, the DGSEM domain smoother used is costly and the cost increases quickly with the number of unknowns in the grid cells.

Bibliography

[1] P. Birken. *Numerical Methods for the Unsteady Compressible Navier-Stokes Equations*. Habilitation Thesis. University of Kassel, Germany, 2012.

[2] Lea M. Versbach. *Multigrid Preconditioners for the Discontinuous Galerkin Spectral Element Method*. PhD thesis, Lund University, 2020.

[3] Philipp Birken, Gregor J. Gassner, and Lea M. Versbach. Subcell finite volume multigrid preconditioning for high-order discontinuous galerkin methods. *International Journal of Computational Fluid Dynamics*, 33(9):353–361, 2019.

[4] Peter Bastian, Markus Blatt, Andreas Dedner, Nils-Arne Dreier, Christian Engwer, René Fritze, Carsten Gräser, Christoph Grüninger, Dominic Kempf, Robert Klöfkorn, Mario Ohlberger, and Oliver Sander. The Dune framework: Basic concepts and recent developments. *Computers Mathematics with Applications*, 81:75 – 112, 2021. Development and Application of Open-source Software for Problems with Numerical PDEs.

[5] Convection-Diffusion equation. https://en.wikipedia.org/wiki/Convection%E2%80%93diffusion_equation. [Online; accessed October-December-2020].

[6] S. Brdar, A. Dedner, and R. Klöfkorn. Compact and stable discontinuous galerkin methods for convection-diffusion problems. *SIAM Journal on Scientific Computing*, 34(1):A263–A282, 2012.

[7] Douglas N. Arnold, Franco Brezzi, Bernardo Cockburn, and L. Donatella Marini. Unified analysis of discontinuous galerkin methods for elliptic problems. *SIAM Journal on Numerical Analysis*, 39(5):1749–1779, 2002.

[8] Martin S. Alnæs, Anders Logg, Kristian B. Ølgaard, Marie E. Rognes, and Garth N. Wells. Unified form language: A domain-specific language for weak formulations of partial differential equations. *ACM Trans. Math. Softw.*, 40(2), March 2014.

[9] GMRES. https://en.wikipedia.org/wiki/Generalized_minimal_residual_method. [Online; accessed October-December-2020].

[10] P. Birken. Optimizing runge-kutta smoothers for unsteady flow problems. *ETNA*, 39:298–312, 2012.

[11] Martin S. Alnaes, Anders Logg, Kristian B. Oelgaard, Marie E. Rognes, and Garth N. Wells. Unified form language: A domain-specific language for weak formulations of partial differential equations, 2013.

[12] Peter Bastian, Eike Hermann Müller, Steffen Müthing, and Marian Piatkowski. Matrix-free multigrid block-preconditioners for higher order discontinuous galerkin discretisations. *Journal of Computational Physics*, 394:417–439, 2019.