Institute of Applied Analysis and Numerical Simulation (IANS)

University of Stuttgart
Allmandring 5b
D–70569 Stuttgart

Masterarbeit Nr. 68

Preconditioning Techniques for Coupled Stokes Darcy Systems

Jenny Schmalfuß

Course of Study: Simulation Technology

Examiner: Prof. Dr. Dominik Göddeke

Commenced: December, 3, 2019

Completed: June 8, 2020

Abstract

An efficient and fast simulation of coupled porous media flows is the key to addressing a number of scientific and applied questions. The numerical modeling of coupled porous media flows is often performed with specialized software packages like DUMU^{X} [28, 34] that take care of the system's discretization and the numerical treatment of the resulting system of equations. In the context of fluid dynamical modeling, the resulting systems are often large and sparse, which requires fast solvers in order to calculate the numerical solution within a reasonably short time frame.

Iterative solvers are preferred for the large, sparse matrices, but often suffer from stagnation due to the ill-conditioning of matrices that stem from discretized flow equations. A common approach to increase the robustness of iterative solvers and to improve their convergence is to apply preconditioning to the system. However, due to the special block structure of matrices that originate from discretized coupled porous media flows, many standard preconditioning techniques are not directly applicable.

In this thesis, we present a flexible block preconditioning strategy that significantly improves the convergence of iterative GMRES solvers [44] for coupled Stokes-Darcy systems. The approach takes advantage of the system's block structure and allows to precondition the different blocks based on their own structural characteristics. Our numerical evaluation shows that tailored block preconditioning does not only improve the convergence of iterative GMRES solvers for the tested Stokes-Darcy systems; Especially on large problems with many degrees of freedom it also solves the system up to ten times faster than a sparse direct solver provided by UMFPACK [19].

Kurzfassung

Eine effiziente und schnelle Simulation von gekoppelten Flüssen in porösen Medien ist der Schlüssel zur Beantwortung vieler wissenschaftlicher und angewandter Fragestellungen. Die numerische Modellierung von gekoppelten Strömungen in porösen Medien wird oft mit spezialisierten Softwarepaketen wie DUMU^{X} [28, 34] durchgeführt, welche die Diskretisierung und numerische Behandlung des resultierenden Gleichungssystems übernehmen. Die Gleichungssysteme, die im Zusammenhang mit strömungsmechanischer Modellierung entstehen, sind häufig groß und dünn besetzt, was schnelle Löser erforderlich macht, um die numerische Lösung in einer angemessen kurzen Zeit zu berechnen.

Iterative Löser werden für große, dünn besetzte Matritzen bevorzugt, stagnieren jedoch häufig Aufgrund schlechter Konditionierung für solche Matritzen, die durch die Diskretisierung von Strömungsgleichungen entstehen. Ein gebräuchlicher Ansatz um die Robustheit von iterativen Lösern zu erhöhen und um deren Konvergenzverhalten zu verbessern, ist die Vorkonditionierung des Systems. Aufgrund der speziellen Blockstruktur, die aus der Diskretisierung von gekoppelten porösen Medienströmungen resultiert, sind viele Standard-Vorkonditionierungstechniken jedoch nicht direkt anwendbar.

In dieser Arbeit stellen wir einen flexiblen Blockvorkonditionierungsansatz vor, der das Konvergenzverhalten von iterativen GMRES Lösern [44] für gekoppelte Stokes-Darcy Systeme deutlich verbessert. Der Ansatz nutzt die Blockstruktur des Systems aus und erlaubt es, die unterschiedlichen Blöcke basierend auf ihren strukturellen Eigenschaften vorzukonditionieren. Unsere numerische Auswertung zeigt, dass problemangepasse Blockvorkonditionierung nicht nur die Konvergenzgeschwindigkeit iterativer GMRES Löser drastisch erhöht; insbesondere für große Probleme mit vielen Freiheitsgraden wird das System damit ebenfalls bis zu zehn Mal schneller gelöst als mit einem Sparse Direct Löser, welcher durch das Softwarepaket UMFPACK [19] bereitgestellt wird.

Contents

Chapter 1

Introduction

Fluid mechanical modeling is an important tool to answer a number of questions in academia and industry, reaching from environmental modeling over medicine to technical applications like fuel cells. Due to the omnipresence of porous materials in our environment, the simulation of flows interacting with porous media is of special interest.

A classic example for porous media interaction is the simulation of fluids in and around soil. Models have been developed to study water evaporation from soil [53], the root system of plants and its interactions with the soil via root growth, evaporation or solute transport [33], the processes that occur during soil salinization [49] or the effect of heavy rains on dry soil [57], to give a few examples.

Many processes within the human body can be regarded as porous media fluid interaction too, like the blood flow through microvascular networks [55]. We emphasize that the applicability of porous media models is not limited to environmental and medical modeling. They can also be applied to questions that arise in industrial contexts, for example to model airflow and heat transport during the refrigerated storage of food bulks [54]. Further technical applications are fuel cells [31] or transpiration cooling with matrix composite materials [17].

For all these complex applications it is often necessary to define multiple flow types in different coupled domains. This leads to coupled systems of equations to describe the porous medium and its coupled flows, like fluid or heat flows. A well known equation to model the flow in porous media is Darcy's law, while the most famous equations to model free flows are the Stokes or Navier-Stokes equations. To numerically solve such coupled problems, the selected model equations and coupling conditions have to be discretized, which often leads to sparse and high-dimensional linear and nonlinear systems. Those are solved numerically, often with specialized numerics environments like the porous media flow simulator DuMu$^{\text{x}}$ [28, 34].

For any application of simulation tools like DuMu$^{\text{x}}$ it is crucial to supply them with efficient solver techniques that enable them to solve large systems of equations within an acceptable time frame and with modest memory requirements. A special challenge in this context are the sparse, high-dimensional and often ill-conditioned matrices that originate from coupled porous media flow simulations. Sparse direct solvers like the one that is provided by the software package UMFPACK [20, 21] are good and robust options, but may be slow and memory intense as they are not well suited to take advantage of the system's sparsity and produce fill-in during the solution process. An alternative are iterative solvers such as GMRES [44], which can be implemented efficiently for sparse matrices. Often, iterative approaches have lower memory requirements and are easier to parallelize than direct solvers [46, Chap. 9.1].

In the context of coupled fluid models, the matrices to solve have a special blocked structure that includes a saddle point problem and are often ill-conditioned. Due to the ill-conditioning, iterative solvers tend to show a poor convergence rate or even fail to converge to the correct solution. This makes their application very expensive, as many iterations are required to solve the system. To remedy the slow convergence of

iterative solvers for ill-conditioned matrices, preconditioning can be applied to the system to improve its condition.

Research outline The goal of this thesis is two-fold. In a first step, we develop a preconditioning strategy for linear systems that stem from a two-phase Stokes-Darcy flow model, which describes the interaction of between a free flow phase and a porous medium. The resulting linear system exhibits a clear block structure that corresponds to the physical quantities pressure and velocity, as well as the phases of the flow model. This overall block structure together with the structural characteristics of the sub-blocks poses a challenge to traditional preconditioning methods like Gauss-Seidel [7] or ILU(p) [45]. We therefore consider specialized preconditioning strategies that use knowledge about the problem's (block) structure to develop tailored preconditioning strategies for the Stokes-Darcy problem. Based on [14] we design a block Gauss-Seidel preconditioning approach that allows to flexibly incorporate prior knowledge about the underlying physical model and its resulting matrix structure.

Our second contribution is to implement this universal block preconditioning approach into the Iterative Template Solver Library (ISTL) [8] of the widely used numerics environment DUNE [3, 10]. By using the general preconditioner interface, other porous media flow models that are implemented with the porous media package DUMU$^{\text{X}}$ can profit from our flexible block preconditioning. Therefore, the application of such preconditioners is not limited to the considered Stokes-Darcy problems.

We find that using explicit knowledge about the problem in the preconditioner construction greatly improves the convergence rates of existing preconditioned GMRES solver strategies. With block preconditioned iterative solvers we can now solve Stokes-Darcy problems that are simulated with the porous media simulation package DUMU$^{\text{X}}$, which was previously impossible with iterative strategies. On top of that, our preconditioned iterative solver strategy yields a better performance in terms of runtime for the two-phase flow problem compared to the sparse direct alternative UMFPACK. This illustrates that using problem specific knowledge for the preconditioning of a coupled Stokes-Darcy system is a promising path to its efficient iterative solution.

This thesis is structured as follows.

Chapter 2 – Continuum-mechanical modeling introduces the stationary, coupled Stokes-Darcy system that serves as our model problem. After defining and discretizing the model equations, we obtain a block structured stiffness matrix with a saddle point structure in the Stokes block. This sparse, ill-conditioned system motivates the need for capable preconditioners in order to apply iterative solvers.

Chapter 3 – Solvers and preconditioners builds the theoretical background that is used to develop preconditioned solvers for the Stokes-Darcy model. It first introduces direct and iterative solvers along with a selection of preconditioning techniques, with special emphasis on GMRES as flexible iterative solver and the Uzawa iteration as a matrix-structure aware preconditioner. Last, we give a left preconditioned formulation for GMRES.

Chapter 4 – Preconditioning for the Stokes-Darcy system develops tailored block preconditioning techniques for the stiffness matrix. Inspired by Jacobi and Gauss-Seidel preconditioning, flexible formulations for block Jacobi and block Gauss-Seidel are designed. We also discuss the applicability of previously introduced preconditioning techniques within the block framework.

Chapter 5 – Implementation introduces the numerics environment DUNE and the porous media package DUMU$^{\text{X}}$ to simulate the coupled Stokes-Darcy system. We give details about the preconditioner interface of DUNE's Iterative Solver Template Library ISTL and discuss how to implement the developed block preconditioning techniques in order to comply with existing interfaces.

Chapter 6 – Results evaluates the developed preconditioning techniques. A first part of this chapter is concerned with comparing different block preconditioning strategies and identifies especially capable block preconditioner setups. The second part compares those well performing iterative solvers to the direct solver UMFPACK in terms of runtime and efficiency.

Chapter 7 – Conclusion and outlook summarizes our work and discusses further research.

Chapter 2

Continuum-mechanical modeling

Fluid mechanics plays an essential role in many scientific and industrial applications, and the development of mathematical formulations to describe interaction with and between fluids dates as far back as the 9$^{\text{th}}$ century [42]. Today, fluid simulations in two- or three-dimensional domains are normality in most applications. In combination with fine discretizations, this quickly leads to many degrees of freedom and high-dimensional systems of equations. Fast solver techniques are therefore required to compute the solution to such systems in a viable time.

This chapter introduces a stationary, coupled Stokes-Darcy problem that is our test case for the exploration of preconditioned iterative solver techniques within this thesis. We first motivate the physical idea behind the model and introduce the equations to describe free flows, flow through a porous medium and their coupling in Section 2.1. Within this chapter we consider a two-phase flow model, where a single free flow phase is coupled to one porous media phase.

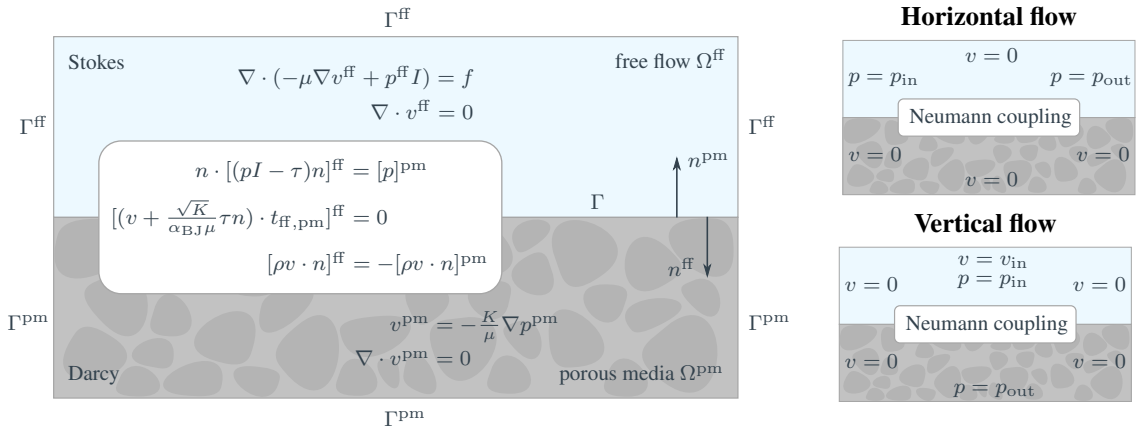
After introducing the governing equations, the problem is discretized in Section 2.2. We describe how those continuous equations are discretized using finite volumes for the space to obtain a solvable system of equations. Coupling conditions are treated monolithically, which means they are not decoupled. Finally, we discuss the matrix structure of the resulting linear system, which underlines the need for tailored, fast iterative solvers.

2.1 Stationary, coupled Stokes-Darcy problem

In this section we discuss the model for a stationary system that consists of a free flow and a porous media phase. We call it Stokes-Darcy problem, as the Stokes equations are used to describe the free flow and Darcy's law is used to model the flow through the porous medium. Additional coupling conditions are introduced to link the phases. Because the problem is not time dependent, it is called stationary.

With those ingredients we model two main scenarios that we call *horizontal* and *vertical flow*. For them, all governing equations to describe the fluids are the same, while they differ in the boundary conditions. For both scenarios a free flow domain Ω^{ff} is placed on top of a porous media domain Ω^{pm}, as shown in Figure 2.1a. To emphasize the distinction between the two coupled components, we denote quantities associated with the free flow with $^{\text{ff}}$ and those in the porous media phase are indexed with $^{\text{pm}}$. The horizontal flow scenario models a fluid running horizontally through a channel, with a porous medium on the bottom. In the vertical flow scenario, a fluid is pushed through a pipe from the top into the porous medium.

We first introduce the model equations for both problem versions, and explain the differences in the scenarios when this common model basis is set. Before we cover the model discretization in the next



(a) Coupled Stokes-Darcy problem **(b)** Boundary conditions

Figure 2.1: Illustration of the stationary, coupled Stokes-Darcy problem, and the boundary conditions for the variants horizontal and vertical flow.

Stokes-Darcy illustration, left: schematic sketch of the free flow Ω^{ff} and the porous media domain Ω^{pm} with their respective boundaries Γ^{ff} and Γ^{pm} along with the shared coupling interface Γ. The Stokes (top) and Darcy equations (bottom) are used to describe the velocity v and pressure p in free flow and porous media domain, the interface is modeled via three coupling conditions (white box). For a detailed description of the used quantities, please refer to the descriptions of Equations (2.1)-(2.7) in the text below.
Horizontal and vertical flow boundary conditions, right: Neumann coupling is used on the shared boundary. On all other boundaries, Neumann or Dirichlet boundary conditions are prescribed for either pressure or velocity. In the vertical flow scenario, the boundary condition for the upper free flow boundary can be a Dirichlet boundary condition for the pressure *or* the velocity. The quantities p_{in}, p_{out} and v_{in} are in- or out-flow Dirichlet boundary values, velocity Neumann boundaries are described with $v = 0$.

section, we also discuss how to choose the free parameters and show the resulting pressure and velocity fields for the models.

Following [39], we begin to describe the governing equations for the free flow phase Ω^{ff}. Boundaries that are not shared with the porous media domain are denoted by Γ^{ff}, the shared boundary is Γ. The free flow normal n^{ff} is orthogonal to Γ and points towards the porous media domain Ω^{pm}. For the stationary, coupled Stokes-Darcy problem we use the Stokes equations to model the free flow. The *Stokes equations* are a specialization of the Navier Stokes equations for low Reynolds numbers [6, Chap. 3], and are used for flows where the velocities are either small, or the viscosity of the fluid is high. Equation (2.1) is the well known *momentum equation*, Equation (2.2) the *continuity equation*:

$$\nabla \cdot (-\mu \nabla v^{\mathrm{ff}} + p^{\mathrm{ff}} I) = f \qquad \text{in } \Omega^{\mathrm{ff}} \tag{2.1}$$

$$\nabla \cdot v^{\mathrm{ff}} = 0 \qquad \text{in } \Omega^{\mathrm{ff}} \tag{2.2}$$

Here, v^{ff} and p^{ff} denote the velocity and pressure in the free flow domain, with f being an external force like gravity or coriolis force, μ the fluid viscosity and I is a suitably shaped identity matrix.

For the porous media domain Ω^{pm}, too, its boundaries that are not shared with Ω^{ff} are denoted by Γ^{pm}. The porous media domain normal n^{pm} is orthogonal to Γ, pointing towards the free flow domain. For the flow description in Ω^{pm}, *Darcy's law* (2.3) is used along with the *continuity assumption* in Equation (2.4) in the porous domain:

$$v^{\mathrm{pm}} = -\frac{K}{\mu}\nabla p^{\mathrm{pm}} \qquad \text{in } \Omega^{\mathrm{pm}} \tag{2.3}$$

$$\nabla \cdot v^{\mathrm{pm}} = 0 \qquad \text{in } \Omega^{\mathrm{pm}} \tag{2.4}$$

In analogy to the free flow domain, v^{pm} and p^{pm} are velocity and pressure in the porous media domain. The constant K in Darcy's law is the permeability of the porous medium, which describes how unhindered water can flow through the medium.

Furthermore, a number of coupling conditions are required to link the two domains. Those are the *the continuity of the normal stresses* (2.5), the *Beavers-Joseph-Saffman condition* (2.6) and the *continuity of the normal mass fluxes* (2.7) [39]:

$$n \cdot [(pI - \tau)n]^{\mathrm{ff}} = [p]^{\mathrm{pm}} \qquad \text{on } \Gamma \tag{2.5}$$

$$[(v + \frac{\sqrt{K}}{\alpha_{\mathrm{BJ}}\mu}\tau n) \cdot t_{\mathrm{ff,pm}}]^{\mathrm{ff}} = 0 \qquad \text{on } \Gamma \tag{2.6}$$

$$[\rho v \cdot n]^{\mathrm{pm}} = -[\rho v \cdot n]^{\mathrm{ff}} \qquad \text{on } \Gamma \tag{2.7}$$

Above, we use n for the normal of the respective flow component, τ for the viscous stresses and ρ as density of the phases. Further, the basis of the tangent plane that describes the interface between free and porous media flow is denoted by $t_{\mathrm{ff,pm}}$. The coefficient α_{BJ} is the Beavers-Joseph coefficient and has to be determined either experimentally or numerically.

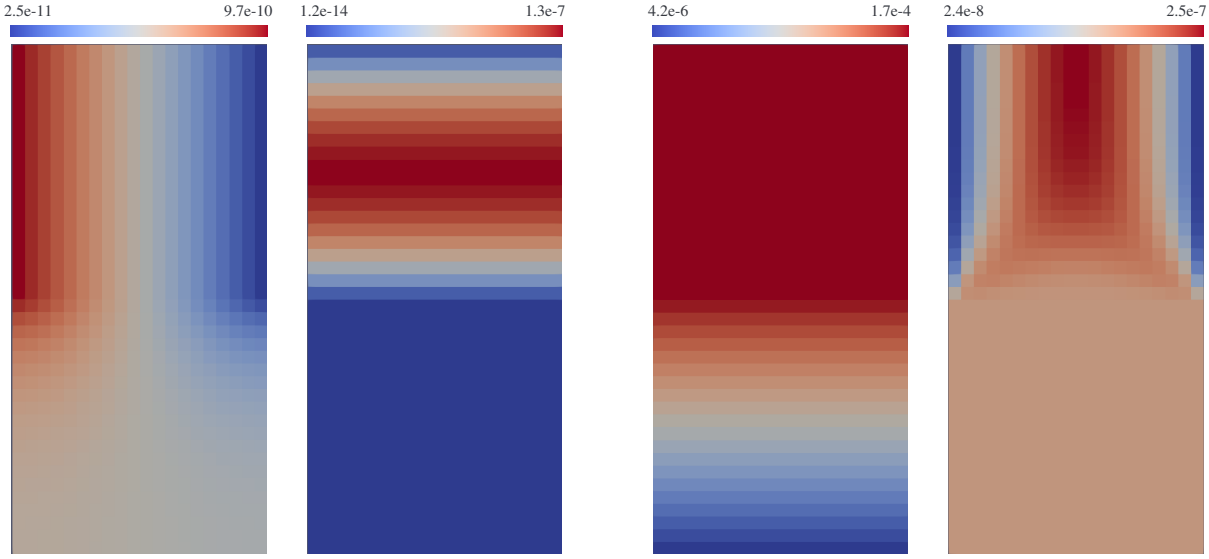
To complete the mathematical problem definitions, boundary conditions for the nonshared domain boundaries are required. As indicated before, the problem's variants horizontal flow and vertical flow differ in the types of boundary conditions that are set on the domain boundaries. In the horizontal flow, liquid is streaming horizontally in the free flow domain with a wall on the upper boundary, causing a pressure gradient in the porous medium. The vertical flow models liquid flowing into the porous medium, with walls on the left and right boundaries of the free flow domain. In both cases, we use a mix of Dirichlet and Neumann boundary conditions to induce the desired streaming behavior. Their distribution to the boundaries is illustrated in Figure 2.1b.

To mathematically prescribe the upper free flow boundary for the vertical flow scenario we have two options: either we initiate the free flow from the top boundary into the porous media by setting a velocity boundary condition, or we apply a suitably chosen pressure to obtain the same behavior. Both cases are shown in Figure 2.1, but the velocity boundary case is pre-implemented in the porous media simulation software DuMu$^{\mathrm{x}}$ [28, 34] that we use to model the coupled Stokes-Darcy problem. However, using pressure Dirichlet boundary conditions avoids numerical problems that arise from pure pressure Neumann boundary conditions on the free flow domain. The matter is discussed in greater detail during the evaluation in Chapter 6 and specifically in Section 6.1.2. Both boundary condition options yield the same pressure and velocity fields, with small differences in the minimal pressure that is reached. Unless otherwise noted, we therefore use the second, pressure-defined vertical flow setup.

Due to its importance, the Stokes-Darcy problem received a lot of scientific attention. Layton, Schieweck, and Yotov [35] show the well-posedness of the model Equations (2.1)-(2.7) and in [22] the existence and uniqueness of a weak solution for the Stokes-Darcy problem is proven.

As an illustration for the resulting flows, we show examples for the pressure and velocity fields for both flow types in Figure 2.2. The velocity field shows the magnitude of the velocities in x- and y-direction. Those exemplary calculations use a permeability $K = 10^{-6}$, a Beavers-Joseph coefficient α_{BJ} of 1.0 and an external force $f = 0$. The horizontal flow is defined over a pressure difference of 10^{-9} between left and right free flow boundary, which initiates a flow through the free flow domain. Pressure and velocity are constant or zero in the porous media domain. In the vertical flow scenario, a pressure of $1.6 \cdot 10^{-4}$ is applied to the upper free flow boundary. There, it causes a constant velocity and a pressure gradient that takes its highest value at the boundary to the free flow domain.

Except for the free flow pressure Dirichlet boundary condition, all values above come from pre-implemented horizontal and vertical flow test cases in DuMu$^{\mathrm{x}}$. The original vertical flow test case uses a Dirichlet velocity value of $v_y = -10^{-6}$ on the upper free flow boundary, the pressure value



(a) Horizontal flow **(b)** Vertical flow

Figure 2.2: Pressure (left) and velocity magnitude (right) for horizontal and vertical flow.

of $1.6 \cdot 10^{-4}$ is selected to yield an almost identical flow behavior. Those values are also used to perform all numerical experiments throughout this thesis.

2.2 Model discretization

To convert the previous model into a solvable system of equations, we consider the coupled system and treat it in a monolithic fashion. An alternative approach is to decouple the two domains in order to solve the Stokes and Darcy equations separately with adequate conditions that guide updates and information exchange across the domain boundaries. This leads to smaller problems that are often easier and faster to solve. The monolithic approach, in contrast, has to solve a bigger problem but potentially preserves the underlying physics of the problem better. This is the main reason to use the monolithic approach within this thesis.

Because our problem is stationary, we only need to discretize the space. The discretization approach used by the numerics environment DUNE [3, 10] is to partition the space into rectangular shaped finite volumes and to apply different types of finite volume methods. For all models described within this thesis, the Stokes and Darcy domains both have a dimensionless size of 1×1 that is divided in $M \times M$ volumes with side length $\frac{1}{M}$. The number of volumes per dimension M is specified in the application. In the following, we outline the finite volume scheme for our model equations and describe how the resulting system is solved. We then analyze the structure of the linear system and discuss the applicability of iterative solvers to it.

From finite volumes to a linear system

The main idea of the finite volume method is to calculate fluxes into and from a finite volume to progressively update the monitored quantities in time [27, Chap. 1]. Because we consider a stationary problem, a time discretization is not necessary. For the space discretization, the space is divided in a set of finite volumes that have the same shape properties like a common control volume. All model equations for the

free flow and porous media flow are equations of the form

$$-\nabla \cdot g(u) = w, \tag{2.8}$$

where u is the monitored quantity and g a vector function that describes the transport of u. The vector function g is defined over the domain Ω. The monitored quantity in our equations are the velocity v or pressure p, Ω can be either Ω^{ff} or Ω^{pm}. For the continuity Equations (2.2) and (2.4) we see that w is zero, and for the momentum Equation (2.1) it equals the external force f. In a first step towards the finite volume scheme, we integrate the momentum and continuity Equations (2.1), (2.2) and (2.4) over a control volume $\mathcal{V}$. We then use the Gauss divergence theorem to transform the integral over $\mathcal{V}$ into an integral over the flux $g(u)$ through the volume boundary $\partial\mathcal{V}$. For our general equation, Equation (2.8), this yields

$$-\int_{\partial\mathcal{V}} g(u) n_{\mathcal{V}} = \int_{\mathcal{V}} w. \tag{2.9}$$

Therefore the expression $-\int_{\partial\mathcal{V}} g(u) n_{\mathcal{V}}$ measures the amount of the monitored quantity that is exchanged between two neighboring volumes. For our three model equations described above, the reformulation is straight forward. In the case of the mass conservation in the porous media domain, Darcy's law is used to obtain a formulation that depends on p^{pm} instead of v^{pm}:

$$\int_{\mathcal{V}} 0 \,\mathrm{d}x = \int_{\mathcal{V}} \nabla \cdot v^{\mathrm{pm}} \,\mathrm{d}x \tag{2.10}$$

$$\overset{\text{Gauss}}{=} \int_{\partial\mathcal{V}} v^{\mathrm{pm}} \cdot n \,\mathrm{d}\Gamma \tag{2.11}$$

$$\overset{(2.3)}{=} \int_{\partial\mathcal{V}} (-\frac{\mathcal{V}}{\mu} \nabla p^{\mathrm{pm}}) \cdot n \,\mathrm{d}\Gamma \tag{2.12}$$

If we switch back to the general Equation (2.9), the first step to its discretization is to discretize the monitored quantity. We begin by replacing the continuous u by averaged values over the volumes that are placed in the center of each volume, and also transition from a general finite volume $\mathcal{V}$ to a specific volume $\mathcal{V}_i$:

$$u_i \coloneqq \frac{1}{|\mathcal{V}_i|} \int_{\mathcal{V}_i} g(u)$$

The value u_i is called a *degree of freedom (DoF)* because it is the discrete solution of u in $\mathcal{V}_i$ that we seek to compute. For the finite volume method, the flux $g(u)$ through the interface between the volumes i and j is not integrated over the boundary but approximated as $\bar{f}_{ij}(u_i, u_j)$, which uses the averages u_i and u_j. So far, the procedure was identical for free flow and porous media flow equations, because the difference between the schemes lies in the used approximations for the flux.

In the Darcy domain it is sufficient to use the simple two-point flux approximation, which works well for diffusive equations like Darcy's law [30, Chap. 4]:

$$\bar{f}_{\mathrm{diff},ij}(u_i, u_j) \coloneqq \frac{u_i - u_j}{|x_i - x_j|}$$

The expression $|x_i - x_j|$ computes the euclidean distance between the volume centers x_i and x_j. With this approximation, consistency is guaranteed for K-orthogonal grids, which is the case for all grids used in the porous media phase. For the advective velocities in the Stokes domain using the two-point flux alone may not lead to a stable scheme [27, Chap. 7]. Therefore, a staggered grid is used to calculate the discretized quantities pressure and velocity. Scalar quantities like pressure and density are assigned to the cell centers, while the velocity is defined on the volume faces. Further, an upwind scheme is used to approximate the fluxes [30, Chap. 4] [34, 48]:

$$\bar{f}_{\mathrm{up},ij}(u_i, u_j, v_{ij}) \coloneqq \begin{cases} u_i & \text{if} \quad v_{ij}(x_i - x_j) < 0 \\ u_j & \text{else} \end{cases}$$

Here, v_{ij} is the velocity through the face between the volumes i and j, which is used to determine the transport direction of the monitored quantity u due to the velocity. For more details on the finite volume discretization in the free flow domain, we refer the reader to [48].

Using the fluxes for Equation (2.9), the fluxes from and into every volume of the discretized domain should sum up to the negative integral over w. Bringing all terms to one side of the equation, this yields

$$\sum_{j \in \mathcal{N}_i} \left(\bar{f}_{\text{diff},ij}(u_i, u_j) + \bar{f}_{\text{up},ij}(u_i, u_j, v_{ij}) \right) + \int_{\mathcal{V}} w = 0 \tag{2.13}$$

and neighboring volumes around $\mathcal{V}_i$ are in $\mathcal{N}_i$. We also recall that for two out of three equations the integral over w equals zero. In this formulation, boundary conditions are introduced as fluxes over the domain boundary $\partial\Omega$. An additional flux $\bar{f}_{Ni}$ into Ω_i across $\partial\Omega$ is used to model Neumann boundary conditions, while Dirichlet boundary conditions are realized over so called ghost elements Ω_i^{g}. Those additional elements mirror the element inside the domain, while the value u_i^{g} is chosen such that linear interpolation between u_i^{g} and u_i leads to the desired Dirichlet value.

So far we only discussed the discretization for the model equations within the free flow and porous media domain. The coupling conditions in Equations (2.5)-(2.7) are realized as boundary conditions on the boundary Γ between the two domains. We now discuss which coupling conditions are used for which balance equation. In the Stokes domain we have the momentum and continuity equation. The momentum equation in Equation (2.1) is coupled with Equation (2.5), which is implemented as Neumann boundary condition for the normal momentum. If Equation (2.5) is fulfilled, v^{ff} is already coupled which means we do not have to implement an additional coupling for the continuity Equation (2.2) as it only depends on v^{ff}. The Beavers-Joseph-Saffman condition in Equation (2.6) is not a coupling condition in the sense that it links the main quantities pressure and velocity of the free flow and porous media domain, it rather poses a boundary condition to the material parameters of the free flow domain. Therefore it is implemented as solution dependent Neumann boundary condition [30, Chap. 4]. For the only equation on the Darcy domain, Equation (2.12), we use the continuity of normal mass fluxes, Equation (2.7), as Neumann boundary condition. For more details on the implementation and a detailed discussion about the consequences of coupling with the staggered grid we refer to [30, Chap. 4].

With the insight that the coupling conditions, too, are implemented as boundary conditions and therefore realized as fluxes, we continue to discuss how the system of fluxes on volumes is solved numerically. For every volume, we want to solve the flux balance from Equation (2.13) with the porous media coupling package DuMu$^{\text{x}}$ [28, 34]. The standard approach for porous media models that are simulated with DuMu$^{\text{x}}$ is wrap them into the DuMu$^{\text{x}}$ solver routine, which uses Newton's method to solve the system irrespective of its linearity. Therefore, to comply with the existing implementation and even though the Stokes-Darcy problem is linear, we have to use one step of Newton's method in order to obtain the approximate solution vector. The solution vector consists of all degrees of freedom for the primary variables from both subdomains $u := (u_{v_1}^{\text{ff}}, u_{v_2}^{\text{ff}}, \dots, u_{p_{2M}}^{\text{pm}})$. To apply Newton's method, we define the right hand side $b_i = 0$ to Equation (2.13) on every volume, which directly leads to the following element-wise residual definition:

$$r_i(u_i) = \sum_{j \in \mathcal{N}_i} \left(\bar{f}_{\text{diff},ij}(u_i, u_j) + \bar{f}_{\text{up},ij}(u_i, u_j, v_{ij}) \right)$$

With the residual, we can define the Jacobian or *stiffness matrix* for the residual as

$$J_r(u) := \left(\frac{\partial}{\partial u_j} r_i(u_i) \right)_{i=0\dots\text{DoF},\ j=0\dots\text{DoF}},$$

which can be used to compute an update $\hat{u}$ to an initial solution u^{init} by solving

$$J_r(u)\hat{u} = u^{\text{init}}. \tag{2.14}$$

One step of Newton's algorithm then gives the approximate solution $\tilde{u}$ as

$$\tilde{u} = u^{\text{init}} - \hat{u}.$$

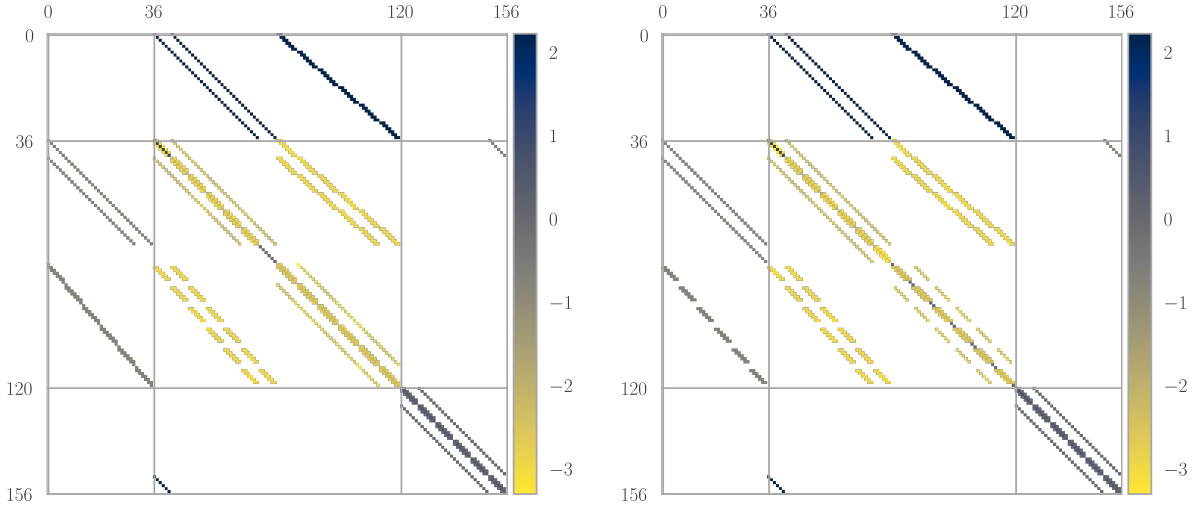
Stiffness matrix structure and solver applicability

This finite volume discretization leaves us with a sparse and potentially high-dimensional stiffness matrix of the form

$$J_r(u) \coloneqq A = \begin{pmatrix} A' & B' \\ C' & D' \end{pmatrix} = \begin{pmatrix} \begin{bmatrix} 0 & B \\ C & V \end{bmatrix} & \begin{bmatrix} 0 \\ B'_1 \end{bmatrix} \\ \begin{bmatrix} 0 & C'_1 \end{bmatrix} & D' \end{pmatrix},$$

where A' models the free flow with the variables v^{ff} and p^{ff}, and D' the porous media flow with p^{pm}. Coupling conditions between them are given by the matrices B' and C'. The matrix dimensions are discussed at the end of this section. The Stokes equation in the free flow domain leads to a saddle point structure of $A' = \begin{bmatrix} 0 & B \\ C & V \end{bmatrix}$. For the two-dimensional case, V describes the velocity components in x- and y-direction, v_x^{ff} and v_y^{ff}. In theory, we could split V again into $V = \begin{bmatrix} V_x & V_{xy} \\ V_{yx} & V_y \end{bmatrix}$, where V_x and V_y are the blocks associated with the velocity degrees of freedom in x- and y-direction and their coupling blocks are V_{xy} and V_{yx}. However, we do not usually use this additional blocking.

Figure 2.3 illustrates the structure of the sparse stiffness matrix A for the stationary, coupled Stokes-Darcy problem in its versions horizontal and vertical flow and with blocks for p^{ff}, v^{ff} and p^{pm}. The entry color reflects the $\log_{10}$ magnitude of the matrix entries. Additionally, Figure 2.4 shows approximated eigenvalues for the respective matrices. The special structure of A originates from the previous modeling as well as from the implementation in DuMu$^{\text{x}}$.



Horizontal flow **Vertical flow**

Figure 2.3: Stiffness matrix structure for horizontal and vertical flow.

Matrix structure for the coupled, stationary Stokes-Darcy problem for a discretization of Ω^{ff} and Ω^{pm} with 6×6 cells each. The color reflects the $\log_{10}$ of the entry magnitude, and the matrix blocks belonging to p^{ff}, v^{ff} and p^{pm} are marked with lines. Note that there is no coupling between the pressures p^{ff} and p^{pm}, free flow and porous media phase are only coupled via v^{ff} and p^{pm} (lower middle and middle right block).

We begin with an explanation of the resulting saddle point structure of the matrix A'. As the upper row of A' is associated with the pressure variables p^{ff}, the upper zero diagonal block originates from the incompressibility of the fluid, which sets the pressure derivatives to zero.

One source for the unsymmetry of A' is the implementation of velocity Dirichlet boundary conditions in DuMu$^{\text{x}}$. Recalling the scenarios horizontal and vertical flow, the velocity on the upper (horizontal flow) or left and right (vertical flow) free flow boundaries is set to zero. To include those zero boundary conditions, a one is written to the stiffness matrix's diagonal for the respective degree of freedom, the remaining entries in the matrix row are replaced by zeros and the right hand side is adapted. In Figure 2.3, this causes the single entries in the middle of the horizontal flow matrix because the degrees of freedom are ordered row wise, which places all v_x degrees of freedom for the upper boundary in a consecutive order. In the vertical flow scenario the same pattern occurs for the left and right boundary v_y values, because no velocity is allowed through the walls. In the vertical flow matrix, those are distributed in the lower part of the velocity matrix, but not in consecutive order as for the upper boundary. This unsymmetry caused by the Dirichlet velocity boundary conditions is also responsible for the structural unsymmetry the coupling blocks B and C. Those structural unsymmetries could be avoided by a different implementation of the boundary conditions in DuMu$^{\text{x}}$.

But there is another reason for the unsymmetric velocity block in the horizontal case. If we compare the blocks that couple v_x and v_y, we observe that both exhibit a two banded structure that is unsymmetrically interrupted in the lower coupling block. This effect, too, originates from the problem definition. In the horizontal flow case, the flow is induced by a pressure Dirichlet boundary condition that is applied to the left and right free flow boundary. This implies a so called "do-nothing" boundary condition for the velocity, where the gradient normal to the boundary of any quantity except the pressure has to be zero. On the right and left boundary, this yields $\frac{\partial v_y}{\partial x} = 0$ and gives zero entries in the lower v_x-v_y coupling block. In the nonnormal direction of the upper v_x-v_y coupling block we have in general $\frac{\partial v_x}{\partial y} \neq 0$, which yields the two bands without interruptions.

In addition to the structural unsymmetry of the coupling blocks B and C that is caused by the DuMu$^{\text{x}}$ Dirichlet boundary implementation, the scales of the coupling entries differ. However, the entries are otherwise identical up to a constant. This is a special case that applies to incompressible fluids, where the density is independent of the pressure. That leads to coupling terms that are symmetric up to a scaling factor. A similar argument explains the relation between the blocks B'_1 and C'_1, which couple free flow velocity and porous media pressure.

The missing coupling between the pressures in the free and porous media flow is due to the continuity of normal mass fluxes in Equation (2.7), because it defines the free flow and porous media coupling without the free flow pressure. The porous media pressure block that stems from a Laplace problem is symmetric for both scenarios.

Summing up, this leaves us to solve the system

$$Ax = b \quad = \quad \begin{pmatrix} \begin{bmatrix} 0 & B \\ C & V \end{bmatrix} & \begin{bmatrix} 0 \\ B'_1 \end{bmatrix} \\ \begin{bmatrix} 0 & C'_1 \end{bmatrix} & D' \end{pmatrix} \begin{pmatrix} \begin{bmatrix} p^{\text{ff}} \\ v^{\text{ff}} \end{bmatrix} \\ p^{\text{pm}} \end{pmatrix} = \begin{pmatrix} \begin{bmatrix} 0 \\ g \end{bmatrix} \\ 0 \end{pmatrix}, \tag{2.15}$$

where A is unsymmetric due to the treatment of boundary conditions - both on a model and implementation level. Because all our experiments are run without external forces, which means $f = 0$ in Equation (2.1), the right hand side g is theoretically zero. However, due to the explicitly prescribed velocities on the Dirichlet boundaries, the right hand side for those value is not zero. For our simple incompressible fluid model, the blocks B, C, B'_1 and C'_1 would be symmetric up to linear scaling, was it not for the Dirichlet boundary implementation of DuMu$^{\text{x}}$. The velocity block V, however, is always unsymmetric.

To solve the system of equations above, we have to choose a solver that does not only solves the system reasonably fast but can also handle its unsymmetry and size. Above, we suggested that A is potentially

very large. If N_x^{ff} and N_y^{ff} are the number of cells x- and y-direction for the free flow, with N_x^{pm} and N_y^{pm} for the porous media domain, then A is a square matrix with the following dimensions that are equal to the number of degrees of freedom:

$$\mathrm{DoF}(A) = \underbrace{N_x^{\mathrm{ff}} N_y^{\mathrm{ff}}}_{p^{\mathrm{ff}}} + \underbrace{(N_x^{\mathrm{ff}}+1) N_y^{\mathrm{ff}}}_{v_x^{\mathrm{ff}}} + \underbrace{N_x^{\mathrm{ff}} (N_y^{\mathrm{ff}}+1)}_{v_y^{\mathrm{ff}}} + \underbrace{N_x^{\mathrm{pm}} N_y^{\mathrm{pm}}}_{p^{\mathrm{pm}}} \tag{2.16}$$

For 5000 cells in each direction and for free flow and porous media, this already leaves us with about 100 million degrees of freedom.

As we discuss in the next chapter, using iterative solvers is the preferred strategy for large, sparse systems like the stiffness matrices produced for the horizontal and vertical flow Stokes-Darcy problem. Unlike for direct solvers, the performance of well converging iterative solvers scales more favorably with the matrix size and for sparse matrices. However, directly applying iterative solvers like PD-GMRES, which is introduced in Section 3.1.2, is problematic due to the eigenvalue distribution of the matrices and their resulting condition numbers. The eigenvalue distribution of the matrices as shown in Figure 2.4 is very scattered and leads to condition numbers of $\kappa_2(A_{\mathrm{hf}}) = 6.0 \times 10^6$ and $\kappa_2(A_{\mathrm{vf}}) = 1.5 \times 10^6$. Those values are for exemplary horizontal and vertical flow matrices with a discretization of 20×20 cells per domain. Also for other discretizations, the horizontal and vertical flow stiffness matrices are very ill-conditioned.

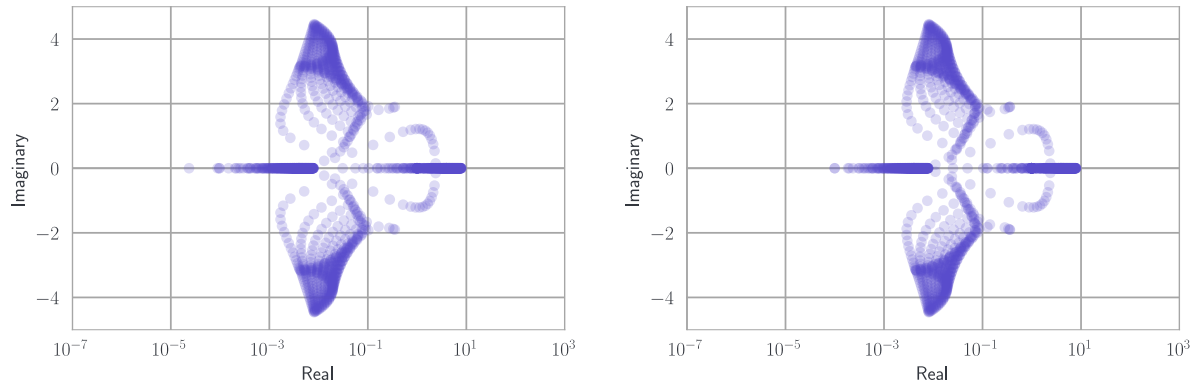


Figure 2.4: Eigenvalue distribution for horizontal flow and vertical flow.

Spectrum for the coupled, stationary Stokes-Darcy problem and a discretization of Ω^{ff} and Ω^{pm} with 20×20 cells each. Note the log scale on the real axis.

This poses a problem for iterative solvers, which often exhibit poor convergence on ill-conditioned matrices. Figure 2.5 showcases the behavior of the iterative solver PD-GMRES when applied to Equation (2.15). In over 500 iterations the solver does not reduce the residual below its initial value and is therefore not a practicable solver choice. Without modification, this only leaves direct solvers for the problem. Those, in turn, can require significantly more memory than is necessary for storing the sparse matrix alone.

This demonstrates that solving Equation (2.15) is challenging for two reasons: the system matrix is ill-conditioned, and the system size becomes very large as the discretization becomes finer. The remainder of this thesis explores preconditioning as a strategy to improve the condition of the system in Equation (2.15), in order to render iterative solvers applicable and to obtain iterative strategies that are competitive to direct solvers.

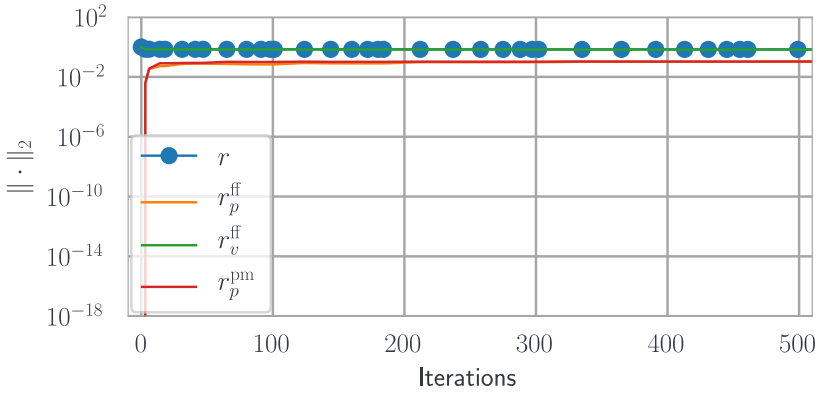


Figure 2.5: PD-GMRES residual reduction on the horizontal flow problem.

Residual reduction for PD-GRMES(-3,3,3,5) over the number of performed iterations, applied to the horizontal flow Stokes-Darcy problem with a discretization of 50×50 on both domains. The residual over all degrees of freedom is marked in blue, residuals for the degrees of freedom of p^{ff}, v^{ff} and p^{pm} in orange, green and red. Clearly, the solver stagnates and does not succeed to reduce the residual over the number of iterations.

Chapter 3

Solvers and preconditioners

The Stokes-Darcy problem from the last chapter results in a sparse, unsymmetric, potentially large and ill-conditioned linear system that needs to be solved. For such systems, the runtime for well-converging iterative solvers often scales better with an increasing problem size than for direct solvers. However, we have seen that the iterative GMRES solver stagnates if it is applied to the Stokes-Darcy problem in its current formulation. This sets the stage for the next two chapters that present the required theory in order to develop and implement tailored solver strategies for the Stokes-Darcy problem.

In Section 3.1 we discuss the two big approaches to solve linear systems of equations: direct and iterative solvers. We introduce concrete methods from both groups that are applicable to the sparse and unsymmetric Stokes-Darcy system and elaborate why we prefer iterative solvers for this type of problems.

A big drawback of iterative methods is their lower robustness compared to direct solvers, which can lead to stagnation or slow convergence. To remedy the stagnation of iterative solvers for the Stokes-Darcy problem is therefore the main goal of this work. *Preconditioning* is a strategy to increase the robustness and efficiency for iterative schemes by transforming the original problem into one that is easier to solve and has better properties with regard to the convergence speed of iterative solvers. By preconditioning the Stokes-Darcy problem we hope to improve the solver convergence and to consequently reduce the number of required iterations in order to achieve a better time to solution than with direct methods.

In the discussion of preconditioning techniques for the Stokes-Darcy system, its special block structure and the properties of the single blocks play a key role. On one hand, the structure of the Stokes-Darcy system poses a challenge to many standard preconditioning techniques or even prevents their application to the whole matrix. On the other hand, we expect to obtain better results if the structural knowledge is directly used to build tailored preconditioning techniques for the Stokes-Darcy system. With this goal in mind, Section 3.2 introduces the general preconditioning concept along with a number of standard methods. Those methods are the ingredients for the development of a preconditioning approach that considers the full Stokes-Darcy system in Chapter 4. There, the introduced methods from Section 3.2 are either applied on a block level or to individual blocks of the Stokes-Darcy system.

Section 3.3 links the discussions about solvers and preconditioners, and explains the necessary reformulations to transform the iterative GMRES method into a preconditioned solver. In its preconditioned form this iterative solver can then be used in conjunction with the Stokes-Darcy preconditioners that are developed in Chapter 4.

3.1 Solvers

To solve the large, unsymmetric system of equations that results from discretizing the coupled Stokes-Darcy problem in the last chapter, we need efficient numerical solvers. This section presents a selection

of methods to compute the solution vector $x \in \mathbb{R}^N$ for the sparse unsymmetric linear system from Equation (2.15), which takes the form

$$Ax = b.$$

Here $A \in \mathbb{R}^{N \times N}$ is the sparse unsymmetric stiffness matrix and $b \in \mathbb{R}^N$ the right hand side. In correspondence to Equation (2.15), N equals the number of degrees of freedom of the coupled Stokes-Darcy system.

To obtain the solution vector x there exists a vast number of *direct* and *iterative* solvers, whose applicability mainly depends on the structure of the coefficient matrix A. For the sparse stiffness matrix, we consider sparse direct solvers as a subclass of direct solvers within this thesis. All those methods compute a sparse decomposition of a potentially reordered, scaled and partitioned matrix A. The main argument for direct methods is their robustness and predictable behavior. On the down side, they often produce a considerable amount of fill-in for sparse matrices, which is inherent to decomposition approaches even though state of the art techniques apply sophisticated reordering and partitioning to the matrix in order to minimize the fill-in [11]. Hence, memory is often an issue; Especially if the system matrix has so many degrees of freedom that storing it in a sparse format already uses most of the available memory. A last point of importance for most practical applications is that their efficient parallel implementation often requires sophisticated routines that go far beyond the often simpler parallelization of matrix-vector products. This is part of the reason that high-performance sparse direct solvers often come in the form of software packages like UMFPACK [19] or PARDISO [47].

The main argument for sparse direct solvers is also the main argument against iterative solvers: Iteratively solving an arbitrary linear system does not have to be efficient [46, Chap. 9.1] and convergence cannot necessarily be guaranteed. This is illustrated by the stagnating iterative solver for the Stokes-Darcy problem in Chapter 2.2. One of the main arguments for the use of iterative methods is that they often require less memory because they only have to store the (changing) solution vector and a handful of vectors for intermediate steps. Also, iterative solvers are often easier to implement on parallel architectures than direct solvers [46, Preface] because they are generally based on matrix-vector products, which is another argument to use them for computation intensive applications like ours.

In the following, we introduce two concrete solver strategies. We first discuss the software package UMFPACK [18, 19] in Section 3.1.1. It implements a state of the art sparse direct solver, which uses an advanced and highly parallel modification to the direct LU decomposition. Because UMFPACK works with the porous media simulation package DUMU$^{\text{X}}$, we regard it as a baseline to solve the Stokes-Darcy stiffness system from Equation (2.15). We then turn to the iterative GMRES solver, which is a Krylov subspace method that approximates the solutions of unsymmetric linear systems of equations. Since the memory requirements for the vanilla GMRES method can be comparable to storing the sparse system matrix A in a dense format, we discuss modifications to the GMRES solver: GMRES(m) and PD-GMRES. Both are successful in limiting the required storage for auxiliary variables. Even though they are not as theoretically well-founded as the unmodified GMRES method, our evaluation in Section 6.1.1 shows them to work well for the Stokes-Darcy system.

3.1.1 UMFPACK - a DUMU$^{\text{X}}$ standard solver

A software package that also supplies a state of the art direct solver is UMFPACK [18, 19], which is part of Matlab's backslash operator and also used as standard solver in many DUMU$^{\text{X}}$ porous media model problems. We discuss the numerics environment DUNE and its porous media simulation package DUMU$^{\text{X}}$ in our implementation Chapter 5.

UMFPACK comprises a set of routines that can be used to solve linear systems with sparse, unsymmetric, square coefficient matrices A, as it is the case for our system. Even though UMFPACK can also be used

to factorize general rectangular matrices, we only outline the solution strategy for unsymmetric square matrices, as this is the relevant case for solving the coupled Stokes-Darcy system in Equation (2.15).

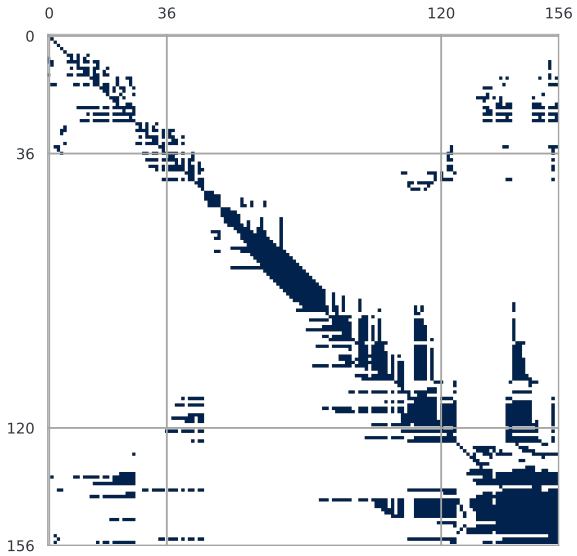
UMFPACK factorizes the square matrix $\bar{A}$ into the product LU with L and U being triangular upper and lower matrices, respectively. The factorized matrix $\bar{A}$ is a potentially reordered and scaled version of the original system matrix A. Reordering and scaling are chosen to minimize the fill-in and therefore reduce the memory requirements for storing the factors. The factorization itself uses the Unsymmetric-pattern MultiFrontal method [20, 21]. Instead of performing an LU decomposition on the whole matrix, this method factorizes a sequence of dense rectangular frontal matrices and obtains the factors L and U from this sequence.

With the triangular factors L and U of A, calculating the solution vector comes down to a backward-forward substitution like in a regular LU decomposition as

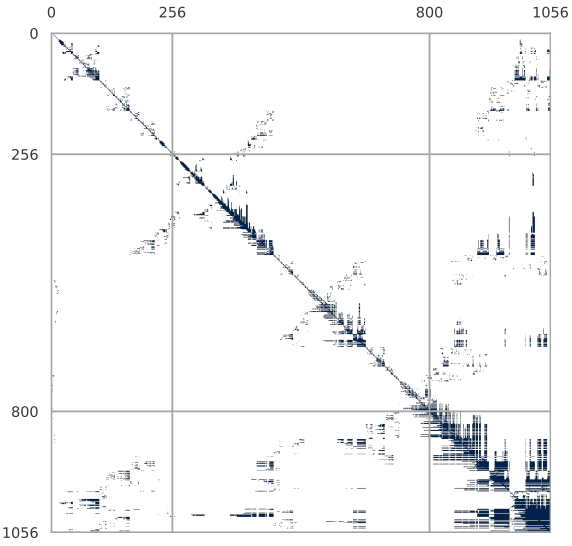
$$\bar{A}\bar{x} = \bar{b} \quad \Leftrightarrow \quad LU\bar{x} = \bar{b} \quad \Leftrightarrow \quad Ly = \bar{b} \text{ and } U\bar{x} = y,$$

where $\bar{x}$ and $\bar{b}$ are solution vector and right hand side, reordered and scaled to agree with $\bar{A}$. For more details on the factorization process, reordering and scaling performed by UMFPACK, we refer to the UMFPACK user guide [19].

A graphical representation of the fill-in that is produced by UMFPACK's LU decomposition is given in Figure 3.1. The figure shows the summed factors L and U for matrices that stem from 6×6 and 16×16 cell discretization of the horizontal flow problem from Chapter 2. The matrix that is decomposed for Figure 3.1a is the same matrix that is also shown in the matrix structure visualization in Figure 2.3 for the horizontal flow.



(a) 6×6 cell discretization



(b) 16×16 cell discretization

Figure 3.1: Fill-in produced by UMFPACK in the sparse factors $L + U$ on horizontal flow matrices.

Visual illustration of the fill-in that is produced in the factors L and U by the application of the sparse direct solver provided by UMFPACK. The factors are summed and shown as $L + U$ for a more compact presentation. All nonzero entries are highlighted. The problems have 156 and 1056 degrees of freedom, which results from a 6×6 and 16×16 cell discretization of the horizontal flow problem. The stiffness matrix that corresponds to the 6×6 cell discretization is shown in Figure 2.3 in Chapter 2.

DoF		**156**	**1056**	**10100**	**102720**	**1001000**
Domain discretization		6×6	16×16	50×50	160×160	500×500
Horizontal flow	Matrix	6.7	7.3	7.6	7.7	7.7
	Decomp.	19.9	38.6	78.6	145.7	241.4
Vertical flow	Matrix	6.6	7.3	7.6	7.7	7.7
	Decomp.	18.5	36.0	76.7	139.7	244.4

Table 3.1: Memory requirements for storing the sparse matrix and its decomposition relative to the degrees of freedom.

The table sets the memory requirements for storing the sparse decomposition generated with UMFPACK in relation to the nonzero entries that are needed to store the sparse matrix. We show the number of nonzero entries relative to the degrees of freedom for problems that are discretized with $\{6^2, 16^2, 50^2, 160^2, 500^2\}$ cells for both domains. The number of nonzero entries in the resulting matrix A and for the sparse LU decomposition for storing all matrices L, U, P, Q and R are given. For the problem with 156 degrees of freedom that results from the 6×6 domain discretization, the value of 6.7 for the horizontal flow matrix means the matrix has $156 \cdot 6.7$ nonzero entries and all matrices that are required for the sparse decomposition have a total of $156 \cdot 19.9$ nonzero entries. It is important to note that we consider the memory requirements for those nonzero entries only, and do not take into consideration the data format that is required to store the location information of those nonzero entries in the matrix.

To represent the original matrix A, UMFPACK has to store the factors L and U, the permutation matrices P and Q and optionally the scaling matrix R. The decomposition of A into L and U is as follows:

$$LU = P(R^{-1}A)Q$$

The matrices P, Q and R all have a number of entries equal to DoF(A), because they permute rows or columns, or scale rows. Therefore, the required memory for storing the sparse LU decomposition mainly depends on the nonzero entries of L and U. On their own, the sparse matrices stemming from the Stokes-Darcy problem have non-zero entries of about seven to eight times the number of degrees of freedom. In the 6×6 cell example, all factors together have 19 times the number of degrees of freedom as number of nonzero entries. For the 16×16 horizontal flow example, UMFPACK needs to store about 38 times more entries than the problem's degrees of freedom and for a 500×500 cell example with 1001000 degrees of freedom roughly 241 times as many entries are produced. For a full overview of the memory requirements relative to the problem's degrees of freedom to store the sparse matrices and sparse decompositions for different matrix sizes, we refer to Table 3.1.

In addition to storing the raw entries, further memory is needed for a data structure that allows to store the entries in a sparse format. The data format must also store the positions of the entries in the original matrix. All this illustrates the large memory requirements of UMFPACK for Stokes-Darcy problems with many degrees of freedom. In Section 6.1.1 we relate the memory requirements for the sparse direct decomposition with UMFPACK to those of iterative restarted GMRES solvers.

3.1.2 Iterative solvers: GMRES and its variants

In this section, we describe the Generalized Minimal Residual method (GMRES); a Krylov subspace method that iteratively approximates the solution to linear, unsymmetric systems of equations [44]. It finds the approximate solution in a suitably constructed Krylov subspace. The approximate solution is optimal in the sense that its residual is orthogonal to all vectors in the Krylov subspace. In every iteration of the algorithm, the Krylov subspace where we seek the approximate solution is extended by one dimension, which leads to a consecutive improvement of the approximation.

It can be shown that GMRES terminates after at most N iterations for a $N \times N$ matrix in exact arithmetics [44]. In our case where N is very large, it is not an option to perform N iterations as this would annihilate our previously stated reason to use iterative methods: their favorable memory requirements for sparse matrices. Because GMRES requires to compute and store all basis vectors of the growing Krylov subspace, the computational effort and needed memory increases with every iteration. After N iterations, the memory requirements for GMRES are are identical to storing the coefficient matrix or its factorization in a dense way because it stores N basis vectors for the Krylov subspace. To render GMRES applicable to our Stokes-Darcy problem, it is clear that we need to limit the number of basis vectors that are calculated and stored.

While ensuring the convergence of the method in the general case, to the best of our knowledge there exists no strategy to limit the computational and memory resources of GMRES below the requirements of a Krylov subspace with the same dimensions as the original matrix. But several heuristic strategies have been proposed to restart GMRES after a certain number of iterations. Even without a strong theoretical foundation, they have shown to work well in practice. After a more comprehensive motivation for restarting GMRES, we introduce two of them: GMRES(m) and PD-GMRES. Both of them can easily be added to an existing GMRES implementation. During the evaluation in Section 6.1.1 we compare both heuristics for our Stokes-Darcy system.

GMRES

We first describe the general working principle and the algorithm, and discuss computational and memory aspects as well as the convergence behavior further below in this section. The *Generalized Minimal Residual method (GMRES)* [44] was developed to iteratively solve the linear system

$$Ax = b, \quad \text{with } A \in \mathbb{R}^{N\times N} \text{ and } x, b \in \mathbb{R}^N$$

for cases where A can be unsymmetric and not positive definite. Because the stiffness matrix of the Stokes-Darcy system is unsymmetric, we apply GMRES as iterative solver throughout this thesis.

GMRES is designed to be mathematically equivalent to the generalized conjugate residual method (GCR) [24, Chap. 5] and ORTHODIR [56], while eliminating the shortcomings of those methods. CGR, for example, potentially breaks down in the iteration process if A is not positive definite. Compared to ORTHODIR, GMRES is designed to be numerically more stable.

As Krylov subspace method, GMRES defines the approximate solution in iteration k as sum of the initial solution x_0 and a vector in the Krylov subspace $\mathcal{K}_k = \text{span}\{r_0, Ar_0, \ldots, A^{k-1}r_0\}$, where $r_0 = b - Ax_0$ is the initial residual. Additionally, the residual $r_k = b - Ax_k$ should be orthogonal to all vectors of the Krylov subspace $\mathcal{K}_k$. This optimality condition is also called Petrov-Galerkin condition. All together, the iterative solution x_k is defined as

$$x_k = x_0 + \mathcal{K}_k \quad \text{s.t.} \quad r_k \perp z,\ \forall z \in \mathcal{K}_k.$$

GMRES minimizes the iterative solution's residual in every iteration, which explains the method's name. This is done by selecting the additive summand z in the k^{th} Krylov subspace, which minimizes the overall residual according to the definition of $x_k = x_0 + z$. Using the definition of $V_k \in \mathbb{R}^{N\times k}$ as the matrix of basis vectors $(v_1, \ldots, v_k)$ for the Krylov subspace with $v_i \in \mathbb{R}^N$ for $i = 1 \ldots N$ and the fact that every element in $\mathcal{K}_k$ can be written as $z = V_k y$ with coefficients $y \in \mathbb{R}^k$, this leads to the following least squares problem for GMRES:

$$\min_{z\in\mathcal{K}_k} \|b - A(x_0 + z)\| = \min_{z\in\mathcal{K}_k} \|r_0 - Az\| \tag{3.1}$$

$$= \min_{y\in\mathbb{R}^k} \big\|\|r_0\|v_1 - AV_k y\big\| \tag{3.2}$$

Algorithm 3.1 Vanilla GMRES

1: **procedure** GMRES(x_0, A, b)
2: $r_0 = b - Ax_0$
3: $v_1 = \frac{r_0}{\|r_0\|}$
4:
5: **for** $i = 1, \ldots, k$ **do**
6: $h_{j,i} = \langle Av_i, v_j \rangle$ **for** $j = 1, \ldots, i$
7: $\hat{v}_{i+1} = Av_i - \sum_{j=1}^{i} h_{j,i} v_j$
8: $h_{i+1,i} = \|\hat{v}_{i+1}\|$
9: $v_{i+1} = \frac{\hat{v}_{i+1}}{h_{i+1,i}}$
10: $y_k = \min_y \left\| \|r_0\| e_1 - \bar{H}_k y \right\|$
11: $x_k = x_0 + V_k y_k$
12: **return** x_k

Algorithm 3.2 Variants of restarted GMRES

1: **procedure** GMRES-RESTART(x_0, A, b, m)
2: $r_0 = b - Ax_0$
3: $v_1 = \frac{r_0}{\|r_0\|}$
4: **while** not converged **do**
5: **for** $i = 1, \ldots, m$ **do**
6: $h_{j,i} = \langle Av_i, v_j \rangle$ **for** $j = 1, \ldots, i$
7: $\hat{v}_{i+1} = Av_i - \sum_{j=1}^{i} h_{j,i} v_j$
8: $h_{i+1,i} = \|\hat{v}_{i+1}\|$
9: $v_{i+1} = \frac{\hat{v}_{i+1}}{h_{i+1,i}}$
10: $y_m = \min_y \left\| \|r_0\| e_1 - \bar{H}_m y \right\|$
11: $x_m = x_0 + V_m y_m$
12: $r_m = b - Ax_m$, test convergence
13: **if** no convergence **then**
14: $r_0 = r_m$, $v_1 = \frac{r_m}{\|r_m\|}$
15: update m (optional)
16: **return** x_m

The first line uses the definition of r_0, and poses the question how we minimize over elements in the Krylov subspace. To do so, GMRES builds a basis for the Krylov subspace, represents each element in the subspace as linear combination of the basis elements and minimizes over the coefficients $y \in \mathbb{R}^k$. This reformulation that takes place in the second line. According to the definition of $\mathcal{K}_k$ the first basis vector is the (normalized) initial residual $v_1 = \frac{r_0}{\|r_0\|}$, which explains the the substitution for r_0 in the second line.

This raises the question how to find a basis for the Krylov subspace. It is computationally desirable to reuse the previous basis vectors V_{k-1} and add a new one to obtain V_k. *Arnoldi's method* [2] does exactly this: it incrementally computes a basis for the Krylov subspace $\mathcal{K}_k$. With this basis we can subsequently solve the least squares problem (3.2), which gives an optimal y to update the initial solution as $x_k = x_0 + V_k y$.

With those ingredients, Algorithm 3.1 outlines the general procedure for GMRES. First, the initial residual is computed and through normalization we obtain v_1, the first and only basis vector of $\mathcal{K}_1$. This is followed by Arnoldi's algorithm in Lines 5-9, which incrementally extends the basis of the Krylov subspace up to a sufficient k. We discuss the question how we identify whether the number of iterations k was sufficient to stop the algorithm below. When Arnoldi's algorithm produced the basis for a sufficiently sized Krylov subspace, the coefficients y_k are computed using a slight reformulation of the least squares problem (3.2) in Line 10 and the iterative solution x_k is updated as $x_k = x_0 + V_k y_k$.

The reformulation is preferred over the the least squares formulation (3.2), because it reuses quantities that were already computed during Arnoldi's algorithm. There, we compute the coefficients $h_{i,j} = \langle Av_j, v_i \rangle$ that form the entries of the upper Hessenberg matrix $H_k \in \mathbb{R}^{k \times k}$ with $H_k = V_k^\top A V_k$. The matrix $\bar{H}_k \in \mathbb{R}^{k+1 \times k}$ is the matrix with entries $h_{i,j}$ generated after k Arnoldi steps, and in comparison to H_k it has an additional row with a single nonzero entry, $h_{k+1,k}$.

As indicated before, from the outlined vanilla-GMRES algorithm it is not immediately clear if and when Arnoldi's algorithm should be stopped. It would be desirable to stop extending the Krylov subspace when the residual for the iterative solution is reasonably small. But to compute the residual we normally need the iterative solution x_k, which is computed only after Arnoldi's algorithm terminated and is therefore not available during the Arnoldi iteration. However, we illustrate below that factorizing $\bar{H}$,

which is necessary to solve the least squares problem, can be done incrementally *while* the basis is build. During the incremental update the value of $\|r_k\|$ is computed as a side product and can be used for the convergence check, without the necessity to compute the updated solution.

With the relation $AV_k = V_{k+1}\bar{H}_k$ we can rewrite the least squares problem from Equation (3.2) to

$$\min_{y \in \mathbb{R}^k} \left\| \|r_0\| e_1 - \bar{H}_k y \right\|, \tag{3.3}$$

which is the formulation used in GMRES. For more details on its derivation, we refer to the original publication of Saad and Schultz [44] or to the books of Meister [38, Chap. 4.3.2.4] and Saad [46, Chap. 6.5]. The important thing to note is that solving Equation (3.3) can be achieved through a QR factorization of $\bar{H}_k$ into $Q_k \in \mathbb{R}^{k+1 \times k+1}$ and $R_k \in \mathbb{R}^{k+1 \times k}$. As shown in [44], the factorization can be executed incrementally while columns of $\bar{H}_k$ are added during Arnoldi iterations. Q_k is unitary, therefore we can reformulate Equation (3.3) using $Q_k \bar{H}_k = R_k$ to

$$\min_{y \in \mathbb{R}^k} \left\| \|r_0\| Q_k e_1 - R_k y \right\| \overset{\tilde{b} = \|r_0\| Q_k e_1}{=} \min_{y \in \mathbb{R}^k} \left\| \tilde{b} - R_k y \right\|. \tag{3.4}$$

Note, that the structure is still equivalent to the initial least squares problem (3.2) where $\tilde{b}$ is the transformed right hand side. By construction, R_k is an upper triangular matrix with an additional zero row in row $k+1$. If we omit the last rows of $\tilde{b}$ and $R_k y$, we can solve the system for y. Therefore, $\tilde{b}$ and $R_k y$ are identical except for the last row, where the difference is $[\tilde{b}]_{k+1} = [\|r_0\| Q_k e_1]_{k+1}$. According to Equation (3.2) the expression to minimize is the residual norm, hence $[\tilde{b}]_{k+1} = \|r_k\|$. If $\bar{H}$ is incrementally factorized during Arnoldi's algorithm, calculating current residual norm $\|r_k\|$ comes at no additional cost because it is simply the last entry in $\tilde{b}$.

In practice, the minimization functional from Equation (3.4) is not solved via a full QR factorization in every step, but implemented more efficiently with Givens rotations. This is based on the insight, that Q_k can be represented as product of k Givens rotations $G_{k+1,k} \dots G_{2,1}$ with $G_{i+1,i} \in \mathbb{R}^{k+1,k+1}$ for $i = 1 \dots k$. The Givens rotations are computed iteratively with the growing Krylov space, using already known quantities only. Details on their computation and more theory are in [44]. The important consequence is that $\tilde{b}$, which uses Q_k, is therefore computable via the k Givens rotations in step k. Because the solution to the minimization problem in Equation (3.3) is the residual norm

$$\|r_k\| = [\tilde{b}]_{k+1} = \Big[\|r_0\| G_{k+1,k} \cdot \ldots \cdot G_{2,1} e_1 \Big]_{k+1},$$

the residual norm can be computed efficiently using the available Givens rotations.

After we outlined GMRES and gave details on the convergence check during Arnoldi's algorithm, we now discuss other matters of importance for any practical application: memory, computational efficiency and the convergence of the method. We begin with the first two points, for which we analyze Arnoldi's algorithm again. From there we see that for every new basis vector v_{i+1} it is necessary to compute a scalar product between Av_i and all previous vectors v_j, with $j = 1, \dots, i$. Therefore, the number of scalar products to compute increases for every additional vector. The cost for each GMRES iteration increases proportional to the increased cost for the Arnoldi iterations. In every iteration i we need to store the vector Av_i, all previous basis vectors v_i and the iterative solution, which gives storage requirements of

$$(i + 2) \times (\text{vector of length } N).$$

This implies that building the Krylov subspace's basis is only efficient, if GMRES does not perform too many iterations.

GMRES is proven to converge to the true solution of a $N \times N$ system in at most N steps [44]. But as we have just seen, N iterations means a storage requirement of $(N + 2)N$, which is as expensive as storing

the matrix in a dense format. Yet it is reasonable to hope for good results with much fewer iterations, depending on the matrix structure. For example, Saad and Schultz prove for a diagonalizable matrix A with $A = XDX^{-1}$ that the residual norm after $m+1$ steps is bounded from above by the initial residual norm as follows:

$$\|r_{m+1}\| \leq \|X\|\|X^{-1}\|\varepsilon^{(m)}\|r_0\|$$

Here, $\varepsilon^{(m)} = \min_{p \in P_m,\, p(0)=1} \max_{\lambda_i \in \sigma} |p(\lambda_i)|$ with σ being the spectrum of A and P_m is the space of polynomials of degree $\leq m$. But despite the good hope for fast convergence, it is important to keep in mind that for a general matrix A it is possible that GMRES only converges in the last step.

Below, we discuss strategies to bound or a least lower the memory requirements of GMRES. An experimental evaluation of those strategies and a discussion about the resulting memory requirements follows during the evaluation in Section 6.1.1. For the Stokes-Darcy problem, we experimentally observe a good convergence for one of these strategies, which is illustrated by the convergence plots throughout our numerical experiments in Section 6.1.

Restarted GMRES

Even though there is no guarantee in the general case that GMRES converges before N iterations were performed, we now discuss heuristics for limiting the number of Arnoldi iterations that have been applied successfully in practice. A simple idea to limit the required resources for GMRES is to bound the maximal number of used basis vectors (and hence the number of Arnoldi iterations) to some value m. If the maximum number of basis vectors is reached, we restart Arnoldi's algorithm and call the basis vector computations between two restarts an *Arnoldi cycle*. In the following, we state ideas related to choosing the maximum number m of Arnoldi iterations per Arnoldi cycle. The general restarted procedure is outlined and compared to vanilla GMRES in Algorithm 3.2.

Before we introduce two concrete methods, GMRES(m) and PD-GMRES, we discuss another theoretical result related to restarting GMRES. From the fact that the initial solution's correction is optimal in the successively growing Krylov subspace, it is intuitively clear that the residual norm is nonincreasing. One of the most relevant results with regard to an early termination of Arnoldi's algorithm comes from Greenbaum, Pták, and Strakoš [29]. They show that for any positive nonincreasing sequence of N numbers, there exists a matrix $A \in \mathbb{R}^{N \times N}$ and a (residual) vector $r_0 \in \mathbb{R}^N$, so that the sequence of $\|r_k\|$ produced by GMRES is equal to this sequence. In other words, there exists a matrix and an initial solution for which GMRES will converge only in the last step. Herein lies the problem when we restart the iteration after a number of steps, as there is no guarantee that the solution is improved after those iterations. If the restart is not chosen well for a given problem, this can lead to stagnating iteration schemes.

For the reason above, only heuristics can be used to find good restart parameters for GMRES, as there is no theoretical guarantee to the convergence of a restarted scheme up to the point of its restart. We briefly cover two strategies for choosing the maximal number of basis vectors: *GMRES(m)*, which uses a fixed restart parameter and *PD-GMRES*, which aims to adapt the parameter to the convergence speed.

GMRES(m)

The procedure for GMRES(m) is straight forward and equivalent to standard GMRES, except that Arnoldi's algorithm is restarted after a fixed number of m iterations. It is shown in Algorithm 3.2, without the m-update in Line 15. After each restart, the updated solution x_m is explicitly calculated, and the residual r_m is used as initial residual for the next Arnoldi-cycle.

GMRES(m) can suffer from two effects: On the one side, it may stagnate if m was chosen too small and convergence cannot be achieved within $\mathcal{K}_m$. This is also due to the fact that the orthogonality of the

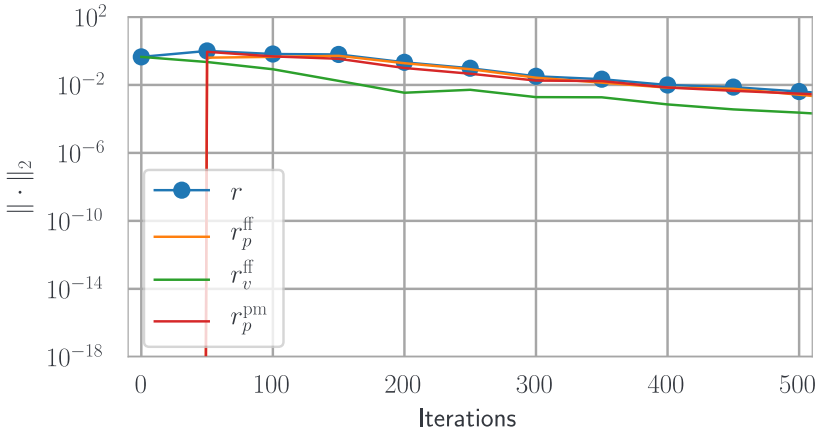


Figure 3.2: Showcasing GMRES(m) restart behavior.

Residual reduction for PD-GRMES(50) over the number of iterations, applied to a preconditioned vertical flow Stokes-Darcy problem. The overall residual is marked in blue, the beginning of every PD-GMRES cycle is shown with a blue circle. The GMRES iteration is restarted after 50 iterations, therefore all blue circles are 50 iterations apart. Because we use a left-preconditioned GMRES(m), the residual is not monotonically decreased. See Section 3.3 for more information as well as Section 6.1 for more details about the convergence plot layout and specifics.

new vectors to the previously computed space is lost after restarting, which slows the convergence. On the other hand, choosing m too big unnecessarily increases the cost for every cycle. The general restart behavior of GMRES(50) for a preconditioned Stokes-Darcy problem is illustrated in Figure 3.2.

PD-GMRES

To overcome some of the shortcomings of a fix restart parameter, a first idea to stop GMRES(m) from stagnation is to increase m whenever the convergence slows. However, this strategy does not necessarily speed up the convergence and may be too expensive [26, 40]. To avoid stagnation and oversolving as in GMRES(m), the *proportional–derivative* strategy PD-GMRES($\mathrm{m_{init}}$, $\mathrm{m_{min}}$, $\mathrm{m_{step}}$, α, β) [40] treats finding the optimal restart parameter m for the next iteration as control problem with m as control variable.

The main motivation for the development of PD-GMRES is the insight that over several GMRES(m) cycles, the resulting residual can point to alternating directions. This in turn worsens the convergence of the whole GMRES(m) iteration. Therefore, modifying m in every iteration is an attempt to avoid this alternating behavior. To choose m such that the convergence of the GMRES(m) iteration is improved, can be regarded as control theory problem.

Therefore, Núñez, Schaerer, and Bhaya [40] reformulate the GMRES(m) procedure into a controlled GMRES(m) algorithm:

$$\begin{cases} r_i = b - Ax_i \\ z_i = \mathrm{GMRES}(A, r_i, m_i) \\ u_i = \psi(r_q, m_i) \\ m_{i+1} = m_i + u_i \\ x_{i+1} = x_i + z_i \end{cases}$$

It is the goal to control the variable x_{i+1} in order to steer the residual r_i to zero. The restarted GMRES algorithm is seen as a controller for the update variable z_i, which is used to update x_i. The convergence and therefore the output of the GMRES(m) procedure depends on the problem matrix A, the current

Algorithm 3.3 Chosing m for PD-GMRES [40]

procedure PDRULE(m_i, m_{init}, m_{min}, m_{step}, $\|r_i\|$, $\|r_{i-1}\|$, $\|r_{i-2}\|$, α, β)
  if $i > 3$ **then**
    $m_{i+1} = m_i + \left\lfloor \alpha \frac{\|r_i\|}{\|r_{i-1}\|} + \beta \frac{\|r_i\| - \|r_{i-2}\|}{2\|r_{i-1}\|} \right\rfloor$
  else if $i > 2$ **then**
    $m_{i+1} = m_i + \left\lfloor \alpha \frac{\|r_i\|}{\|r_{i-1}\|} \right\rfloor$
  else
    $m_{i+1} = m_{\text{init}}$
  if $m_{i+1} < m_{\text{min}}$ **then**
    $m_{\text{init}} = m_{\text{init}} + m_{\text{step}}$
    $m_j = m_{\text{init}}$

residual r_i and the restart parameter m_i. So far, this describes the classical restarted GMRES(m) algorithm. The novelty is the update for m_{i+1}, which uses u_i as control variable for m_{i+1}. This control variable u_i is steered by the controller ψ, which has access to the current restart parameter m_i and the residuals. Because ψ can theoretically access all previous residuals, the used residuals are denoted by r_q and not r_i alone. In this formulation, the question is how to design the controller ψ in order to reduce r_i.

Before we describe the construction of the controller, an important observation can be made from the residuals. As PD-GMRES tries to avoid alternating residuals, this concept should be formalized first. Therefore, we define sequential and skip angles, where the sequential angle $\theta_i = \angle(r_i, r_{i-1})$ is the angle between the two residuals at the end of the GMRES(m) cycles i and $i+1$ and the skip angles are the angles between every other residual [40]. For the sequential angle it holds

$$\cos(\theta_i) = \cos(\angle(r_i, r_{i-1})) = \frac{\langle r_i, r_{i-1} \rangle}{\|r_i\|_2 \|r_{i-1}\|_2} = \frac{\|r_i\|_2}{\|r_{i-1}\|_2}, \tag{3.5}$$

where the last step uses that $\langle r_i, r_{i-1} \rangle = \|r_i\|_2^2$, which follows after a short computation in [40, Sec. 3] from the orthogonality of the residual and the vectors of the Krylov subspace. Based on theory for Lyapunov functions and using the residual and sequential angles, it is possible to relate the stability of the controlled GMRES(m) formulation to the quality of convergence of the GMRES(m) algorithm.

The key insights that are proven in [40] are as follows: Good GMRES(m) convergence implies that $\frac{\|r_i\|_2}{\|r_{i-1}\|_2} \approx 0$, which means we have a skip angle of about $90°$. The residual should be reduced compared to the previous step, $\|r_i\|_2 < \|r_{i-1}\|_2$. In the case of slow convergence or stagnation, the relation between the two residuals is $\frac{\|r_i\|_2}{\|r_{i-1}\|_2} \approx 1$, which implies a skip angle of about $0°$. Also in this case, the improvement of the residual norm over two GMRES(m) cycles is small compared to the magnitude of the residual norm: $\frac{\|r_i\|_2 - \|r_{i-2}\|_2}{\|r_{i-1}\|_2} \approx 0$.

By using those insights in the form of a proportional–derivative (PD) controller, PD-GMRES computes the restart parameter m_{i+1} for the next GMRES(m) cycle $i+1$ according to

$$m_{i+1} = m_i + \left\lfloor \alpha \underbrace{\frac{\|r_i\|}{\|r_{i-1}\|}}_{\text{proportional term}} + \beta \underbrace{\frac{\|r_i\| - \|r_{i-2}\|}{2\|r_{i-1}\|}}_{\text{derivative term}} \right\rfloor,$$

taking into account the residual norms from the current and two previous GMRES(m) cycles, $\|r_i\|$, $\|r_{i-1}\|$ and $\|r_{i-2}\|$. In general, the rule above is targeted to decrease m if the convergence stagnates, which means that the proportional term is close to one and the derivative term goes to zero.

Meanwhile it increases m when the convergence is fast, and the derivative term large. Núñez, Schaerer, and Bhaya point out that leaving the restart parameter unchanged, even if the convergence was fast in the previous cycle, will not necessarily yield fast convergence in the next cycle due to the possibility of alternating residuals. Therefore, the algorithm is designed to constantly change the restart parameter and decrease it whenever possible. From [40] we use $\alpha = -3$ and $\beta = 5$ for all our numerical experiments with PD-GMRES, the values were determined experimentally.

Algorithm 3.3 shows the full behavior of the PD-rule. If the stagnation brings m to fall below a fix threshold m_{min}, the initial value m_{init} is raised by the value m_{step} and directly used as new restart parameter. Therefore, m_{init} keeps increasing over time, while m_{min} marks a fix lower bound for the restarts. The intended restart behavior of a PD-GMRES(5,5,5,-3,5) implementation on a preconditioned problem is illustrated in Figure 3.3.

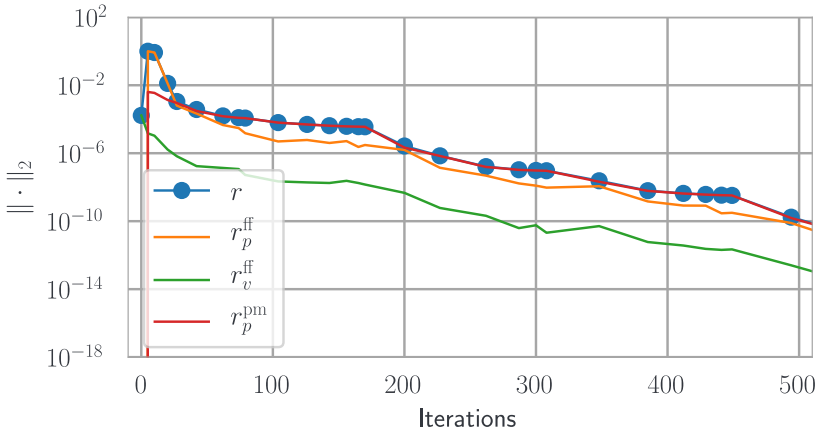


Figure 3.3: Showcasing PD-GMRES restart behavior.

Residual reduction for PD-GRMES(5,5,5,-3,5) over the number of iterations, applied to a preconditioned vertical flow Stokes-Darcy problem. The overall residual is marked in blue, the beginning of every PD-GMRES cycle is shown with a blue circle. The number of iterations between two consecutive circles is the current restart parameter m. The example nicely shows the intended restart behavior: when the convergence slows around iteration 100, PD-GMRES reduces the restart parameter in every cycle until it goes below m_{min} at iteration 175. Then m is raised to its highest value so far to successfully improve convergence in the next cycle. This behavior repeats when convergence slows again around iteration 250 and 400. Because we use a left-preconditioned PD-GMRES, the residual is not monotonically decreased. See Section 3.3 for more information as well as Section 6.1 for more details about the convergence plot layout and specifics.

PD-GMRES can greatly improve the convergence behavior for the restarted GMRES iteration. We compare different configurations of PD-GMRES with the fixed m version GMRES(m) in Section 6.1.1 of the evaluation chapter. For very ill-conditioned problems, however, the variation of m alone may not be sufficient for a good convergence and require the application of preconditioning techniques to the problem [40].

3.2 Preconditioning

In the previous section we covered multiple versions of GMRES as iterative solver for unsymmetric, sparse matrices. This allows us to apply GMRES to the Stokes-Darcy problem from Chapter 2. During the evaluation in Section 6.1 we show that without any modification to the problem all presented variants of GMRES suffer from stagnation when they are used to solve the Stokes-Darcy system from Equation (2.15). To remedy the stagnation of iterative GMRES, we apply preconditioning to our problem.

Preconditioning is a strategy to improve the robustness and efficiency of iterative solvers, by transforming the problem to be solved into one with better spectral properties that is "easier" to solve [46, Chap. 9.1]. The system matrix and the right hand side change during preconditioning and the solution x of the original problem should be cheaply computable from the preconditioned system's solution:

$$\underbrace{Ax = b}_{\text{no/slow convergence}} \xrightarrow{\text{Preconditioning}} \underbrace{\tilde{A}\tilde{x} = \tilde{b}}_{\text{good convergence}} \quad \text{with} \quad \underbrace{x = f(\tilde{x})}_{\text{cheaply computable}}$$

As before, A and $\tilde{A}$ are matrices in $\mathbb{R}^{N\times N}$, where N is the total number of degrees of freedom in the Stokes-Darcy system. The solution vectors x and $\tilde{x}$, as well as the right hand sides b and $\tilde{b}$ are vectors in $\mathbb{R}^N$.

Two cues can help to assess whether a linear system will induce fast solver convergence: The condition number $\kappa_{\|\cdot\|}(A)$ and spectrum $\sigma(A)$ of the system matrix A. The condition number for invertible matrices is defined as

$$\kappa_{\|\cdot\|}(A) = \|A\|\|A^{-1}\|,$$

where the subscript symbolizes the used matrix norm. The condition number is always bigger or equal to one. Matrices with condition numbers close to one are called "well-conditioned", whereas those with $\kappa_{\|\cdot\|}(A) \gg 1$ are called "ill-conditioned". Therefore, the exemplary matrices from the Stokes-Darcy system in Section 2.2 with their condition numbers of 6.0×10^6 and 1.5×10^6 are very ill-conditioned.

Based on the condition number there are several upper bounds that can be derived for the residual of Krylov subspace methods like GMRES [46, Proposition 6.32, Corollary 6.33]. The idea to improve the system's condition also explains the origin of the name as pre*conditioning* methods. However, for Krylov methods in general tighter estimates for residual and error after a certain number of iterations can be obtained by using the eigenvalue distribution (spectrum) of the matrix [15]. For every Krylov subspace method, the following equality holds for the residual after m iterations with the initial residual r_0 [38, Theorem 4.62]:

$$\|r_m\|_2 = \min_{p\in\mathcal{P}_m^1} \|p(A)r_0\|_2$$

Here, $\mathcal{P}_m^1$ are all polynomials p of degree $\leq m$ with $p(0) = I$, where I is an identity matrix. To obtain an estimate for r_m one often chooses a certain, well behaved type of polynomials like Chebyshev polynomials. Such choices are often based on the maximal and minimal eigenvalue, which are closely linked to the condition number. However, it is clear that with more detailed knowledge about the distribution of the eigenvalues, tighter residual bounds can be obtained by, for example, by choosing more specific polynomials. Therefore, the eigenvalue distribution of the matrix is a second and probably better indicator for the solver convergence. Fast convergence is often obtained if the spectrum is clustered away from zero [7].

As the spectral properties for a given problem are fixed, preconditioning intends to change the problem by formally multiplying a nonsingular preconditioning matrix $\mathcal{P}$ from the left or right side to the problem. We call $\mathcal{P}$ a preconditioner and emphasize that a preconditioner does not need a matrix representation. It is sufficient to know how $\mathcal{P}$ can be applied to a vector.

One possible classification for preconditioning techniques is the "direction" in which the preconditioner matrix is multiplied to the problem. In the scope of this thesis, we use only *left preconditioning* [7], where we multiply the linear system from the left with the nonsingular preconditioning matrix to obtain

$$\mathcal{P}Ax = \mathcal{P}b.$$

It is also possible to precondition the system from the right, where we formally multiply a unity matrix $I = \mathcal{P}\mathcal{P}^{-1}$ between the system matrix A and the solution x. *Right preconditioning* requires to solve the system and the substitution for x,

$$A\mathcal{P}y = b \quad \text{and} \quad x = \mathcal{P}y.$$

Right and left preconditioning can also be combined, which is then called *split preconditioning*.

All formulations above require matrix products of the system matrix with the preconditioner. In real applications, those matrix products are not calculated explicitly. Instead of computing the preconditioned system, the used solver is constructed *for* the preconditioned system. We show this for the preconditioned GMRES solver in Section 3.3.

Preconditioned solvers are constructed in such a way that it suffices to know how the preconditioner acts on a vector:

$$y = \mathcal{P}x$$

This also allows matrix free preconditioner formulations, and gives rise to another classification for preconditioners. *Explicit preconditioners* compute an approximation to A^{-1}, that might be expensive during its assembly. But once obtained, applying the preconditioner is only a matrix-vector product. In contrast, *implicit preconditioning* targets finding an approximation to A that is easier to invert. This often does not require many computation for the assembling process, but inverting this matrix is necessary for every application of the preconditioner. Applying an iterative solver for the original matrix as preconditioner is an interesting application of implicit preconditioning, since solving a system acts as multiplication of the right hand side with the inverted system matrix.

To render iterative solvers applicable to our coupled Stokes-Darcy problem, we first introduce several preconditioning techniques that are either commonly used or especially suitable for certain structural parts of the Stokes-Darcy problem. We begin with splitting-based Jacobi and Gauss-Seidel preconditioning. Then, we continue with implicit preconditioners, starting with the incomplete LU preconditioner (ILU) [32, 37]. Following ILU, we cover preconditioners derived from solvers. There, we introduce the Algebraic Multigrid Method (AMG) [13, Chap. 8][43, 50] and the Uzawa iteration [25] that were both developed as solvers but are also capable preconditioners.

3.2.1 Common splitting-based preconditioning techniques: Jacobi and Gauss-Seidel preconditioners

A very straight forward idea to construct an approximation to A^{-1} that can be used for an explicit preconditioner definition is to split the matrix into components and invert only a subset of them [46, Chap. 10.2]. If we split A into its diagonal D, the lower triangular and the upper triangular remainders L and U we have

$$A = \overbrace{D}^{\text{Jacobi}} + \underbrace{L}_{\text{Gauss-Seidel}} + U.$$

Using the diagonal as approximation to A gives the *Jacobi preconditioner*, if we additionally include L we get the *Gauss-Seidel preconditioner*. In both cases, the reduced matrix to approximate A needs to be regular.

Constructing the Jacobi preconditioner $\mathcal{P}_{\text{J}} = D^{-1}$ requires significantly fewer operations than constructing the Gauss-Seidel preconditioner $\mathcal{P}_{\text{GS}} = (D + L)^{-1}$, because we only have to take the inverse of the diagonal entries instead of inverting a triangular matrix. It is not possible to say that inverting more of the original matrix helps per se to obtain a better preconditioner. But as we see during the evaluation in Section 6.1.4, on a block level it has a positive effect on the convergence rate of iterative solvers to precondition them with a Gauss-Seidel preconditioner in contrast to Jacobi preconditioning.

3.2.2 Factorization-based preconditioning techniques: ILU

The *incomplete LU factorization (ILU)* is an implicit preconditioning technique that approximates the LU decomposition of A. A problem of the classic LU factorization for sparse matrices is the significant fill-

Algorithm 3.4 ILU(0) [45]

```
1: procedure ILU(0)(A)
2:     for i = 2, ..., N do
3:         u_{i,*} = A_{i,*}
4:         for k = 1, ..., i − 1 and if (i, j) ∈ D do
5:             l_{i,k} = u_{i,k} / u_{k,k}
6:             for j = k + 1, ..., N and if (i, j) ∈ D do
7:                 u_{i,j} = u_{j,j} − l_{i,k} u_{k,j}
```

in that is produced during the factorization. Also, a complete factorization would solve the problem in a direct way and therefore render preconditioning unnecessary. Additionally, it would come with all drawbacks of direct methods that are discussed in Section 3.1.1 and the solver introduction. In contrast, the incomplete LU factorization cuts memory and computational cost by computing the factors only for "meaningful" matrix entries.

Before we explain how such meaningful entries are determined, we first discuss requirements for the existence of an incomplete LU decomposition. It is important to point out that a regular matrix A, which implies the existence of a full LU factorization, is not enough. Meijerink and Van Der Vorst [37] prove the existence of an incomplete LU factorization for M-matrices. For A to be an M-matrix, all of its off-diagonal entries need to be smaller or equal to zero and all eigenvalues must have nonnegative real parts. We note that the full stiffness matrix for the coupled Stokes-Darcy problem is not an M-matrix, while some sub-blocks of it are. More details on the properties of the Stokes-Darcy sub-matrices are discussed when we develop the block preconditioning strategies in Section 4.2.

We continue to discuss the computation of the incomplete LU decomposition. As the name suggests, the incomplete LU decomposition computes so called incomplete factors $\tilde{L}$ and $\tilde{U}$ that have entries only in specific, predefined positions. We define the incomplete factors via

$$A = \tilde{L}\tilde{U} - R,$$

where R is the remainder of the matrix that is not covered by the incomplete factorization. The entries that are dropped for the incomplete factors $\tilde{L}$ and $\tilde{U}$, are chosen by the *dropping strategy* $\mathcal{D}$,

$$\mathcal{D} \subset \mathcal{D}_N = \{(i,j) | i \neq j,\ 1 \leq i \leq N,\ 1 \leq j \leq N\},$$

which picks entries from the set of off-diagonal entries $\mathcal{D}_N$. In general, only such entries of A that are already zero are added to the dropping strategy. For sparse matrices as in our case, this also yields sparse factors $\tilde{L}$ and $\tilde{U}$. The dropping strategy that selects all off-diagonal zero entries of A is called ILU(0), and illustrated in Algoritm 3.4. The notation $A_{i,*}$ denotes the i^{th} row of A. Here, the entries for $\tilde{L}$ and $\tilde{U}$ are only computed for those entries, where A is nonzero. In any applicable implementation of Algorithm 3.4 the check if the entry $(i,j) \in \mathcal{D}$ would be replaced by a check that is computationally easier to perform. If we use the dropping strategy that keeps only diagonal and non-zero entries, checking $A_{i,j} \neq 0$ suffices.

Yet the dropping strategy is not limited to A's zero entries, and adding more entries to the incomplete factors can improve the quality of the incomplete factors. We subsequently discuss ILU(p), where we give more details on how to choose meaningful zero entries that should not go into the dropping strategy to improve the approximation quality.

ILU(p)

One problem with ILU(0) is that it is often insufficient for obtaining a good convergence behavior for GMRES [32]. An idea to keep more entries in the incomplete factors that are related to the matrix structure, is to formally multiply the incomplete factors together and use the product's nonzero entries for the dropping strategy. The product has a new structure of nonzero entries that is less sparse than the original matrix, and can be used as new structure for the next factors. Performing this once yields the dropping strategy for ILU(1), while repeating the process p times gives ILU(p).

The idea is realized through the so called level of fill-in (LOFI), which is subsequently used to determine whether an entry is kept by the dropping strategy. Depending on matrix structure and chosen p, the preconditioner setup that computes the matrix structure up to p is potentially expensive. Also the cost for computing the structure for ILU(p) is hard to assess in advance because the number of nonzero entries for the product of the incomplete LU factors strongly depends on nonzero pattern of A. There is no general theory except for special cases that a higher p does improve the solver convergence. However experience and our experiments in Section 6.1.3 show that the resulting preconditioners improve solver convergence.

The optimal choice of p is often a balance between the setup cost and expected solver speedup. We use different values for p during our evaluation and discuss this trade-off in Section 6.1.3.

3.2.3 Preconditioning with solvers: AMG and Uzawa

As noted in the preconditioning introduction, to formulate preconditioned solver variants it is enough to know how the preconditioner $\mathcal{P}$ is applied to a vector x, via

$$y = \mathcal{P}x.$$

For that reason, a preconditioner can also be an arbitrary iterative method that is applied to x in order to obtain y. Within this thesis, we apply two sophisticated iterative methods as preconditioners. Both of them are briefly introduced now and described below.

Using an *algebraic multigrid method* (AMG) [13, Chap. 8][43, 50] as preconditioner is, in principle, possible for any matrix. The non-smooth aggregation AMG [9] as implemented in the iterative solver template library ISTL [8], which can be used for Stokes-Darcy models in DUMU$^{\text{X}}$, is especially suitable for unsymmetric matrices or problems with discontinuous coefficients. Algebraic multigrid methods are a very powerful solver class that reduce the computational cost by using a multilevel grid hierarchy. We also consider a method that is tailored for saddle point problems like the Stokes problem: an *Uzawa iteration (Uzawa)* [25] based preconditioner. As this method actively exploits the known saddle point structure of the matrix, it is a capable preconditioner for the free flow components of coupled Stokes-Darcy problems.

AMG

The algebraic multigrid method (AMG) is an extension of the classic multigrid method (MG) solver to cases where the grid structure of the underlying problem is either unknown or irregular. While MG methods explicitly use information about the discretization's geometry, AMG methods construct a "grid hierarchy" only with information from the coefficient matrix $A \in \mathbb{R}^{N \times N}$. This renders them applicable to solving general problems $Ax = b$. We will give a brief overview of AMG methods that is inspired from [13, Chap. 8] and also assume familiarity with the multigrid concept. For more information on AMG methods we refer the reader to the works of Ruge and Stüben [43] and Stüben [50].

Given our typical system of equations $Ax = b$, we can regard the entries of the solution vector $x = (x_1, \ldots, x_N)^\top$ as the N points of the fine grid. Connections between them are defined through

A, which is interpreted as undirected adjacency graph, meaning that every matrix entry $a_{i,j} \neq 0$ signifies a connection between the points x_i and x_j. If the grid geometry is unknown, it is not intuitively clear how we can construct a grid hierarchy, which forms the core of the multigrid idea. While for classic multigrid methods we first choose a coarse grid to represent the smooth functions, relax the smooth error components there, and then choose the inter-grid operators, this order changes for AMG methods. Here, we begin by picking a relaxation scheme, which will define what smoothness (of an error) means. Smoothness is not defined in terms of Fourier modes as for MG methods, but defined algebraically instead, which names the class of algebraic multigrid methods. The smoothness definition is then used to select a subset of the fine grid points for the coarse grid and the interpolation operator. This is used to define the coarse grid representation A_C.

Algebraic smoothness is one of the two key concepts to AMG methods. It formalizes the meaning of a "smooth error" in a setting, where we cannot simply state that smooth error components are those with low frequency in a Fourier sense. We generalize observations from MG methods and define a smooth error as being not significantly reducible by relaxation. This concept is formalized in [13, Chap. 8], which leads to the insights that on average, smooth errors have pointwise small residuals and can be represented as weighted sum of their neighboring errors. The second key concept is *strong dependence*, which formalizes the strength of connection between the "grid points" x_i. Here, we regard the coefficients $A_{i,*}$ associated with the calculation of element i as measure for the importance of point x_j with weight $A_{i,j}$ for calculating point x_i. Point x_i is called strongly dependent on x_j, if $A_{i,j} \approx \max_{k \neq i}\{A_{i,k}\}$. Because strong dependence is directed (x_i strongly dependent *on* x_j), it is sufficient to consider the i^{th} row of A in the definition.

The coarse grid should do three things: it should approximate the smooth error accurately, smooth functions should be interpolated correctly from it and it should have much fewer grid points than the fine grid. To construct it, first an initial partition of points x_i into coarse and fine grid points is constructed. In a second step this initial partition is used to construct an interpolation operator and the partition is adjusted to satisfy heuristic rules that ensure a good interpolation while keeping the set of coarse grid points small. To construct the interpolation operator both key concepts, algebraic smoothness and strong dependence, are used. For more details on the process we refer the interested reader to [13, Chap. 8].

With the grid hierarchy and the operators in place, AMG methods perform the same steps as traditional multigrid to calculate the approximate solution to $Ax = b$. Different cycle types like V- or W-cycles can be implemented. The price for a multigrid scheme with missing grid geometry information is that the computational cost cannot be predicted, as the ratio of fine and coarse grid points is not known a-priory and is determined during the setup phase.

The solver library ISTL [8] that is used when we simulate the coupled Stokes-Darcy problem with DUMU$^{\text{X}}$ provides a preconditioner that uses *AMG based on non-smooth aggregation* [9]. This AMG strategy is especially suited for unsymmetric matrices. The traditional AMG approach as outlined above divides the unknowns in those that exist entirely on the fine level and those that also exist on the coarse level. Based on this splitting, the interpolation operators are defined. AMG based on aggregation regards the fine level unknowns as entities that are clustered into so called aggregates. Each aggregate then represents a coarse grid entity. If the inter level interpolation based on those aggregates uses piecewise linear interpolation, the resulting scheme is called a non-smooth aggregation method. It is also possible to smooth the interpolation, which results in smoothed aggregation methods and increases the size of the coarse level stencils. The aggregation approach comes from a high performance computing perspective, where the coarsening should be performed in parallel and the processors have to find a common splitting of the fine grid variables into fine grid only and coarse grid without too much communication.

The AMG method based on non-smooth aggregation as presented in [9] and implemented in ISTL uses a special greedy heuristic algorithm to build the aggregations using the strength of connection. We refer to the original publication of Blatt, Ippisch, and Bastian [9] for details. As a result, this non-smooth aggregation AMG can build "round" aggregations of almost arbitrary size, which do not include high

contrast coefficient jumps into one aggregation. We use this non-smooth aggregation implementation with Gauss-Seidel smoothing for all numerical experiments during the evaluation in Chapter 6.

Another important point with regard to applying AMG as preconditioner for the coupled Stokes-Darcy system is the applicability of AMG to pure Neumann problems. If only Neumann boundary conditions are defined in a system, its solution is defined up to a constant. In this case, an unmodified AMG scheme might converge to a different solution with every application, and therefore stagnate. Preconditioners that are not iterative solvers, such as ILU, work without this problem. We discuss the effect in greater detail during the evaluation of the horizontal flow Stokes-Darcy problem in Section 6.1.3.

Uzawa iteration

Before we introduce the Uzawa iteration, we recall the saddle point structure of the free flow block A' of the coupled Stokes-Darcy problem. It describes the interaction between the pressure p and the velocity v in the free flow domain with given right hand sides g and f as

$$\underbrace{\begin{pmatrix} 0 & B \\ C & V \end{pmatrix}}_{A'} \underbrace{\begin{pmatrix} p \\ v \end{pmatrix}}_{x'} = \underbrace{\begin{pmatrix} g \\ f \end{pmatrix}}_{b'}. \tag{3.6}$$

The Uzawa solver directly exploits the saddlepoint structure of A', by creating an iterative scheme from the two resulting equations

$$Bv = g \tag{3.7}$$
$$Vv = f - Cp, \tag{3.8}$$

where (3.8) is the reordered second equation from (3.6). To turn those into an iterative scheme, the modified Richardson iteration $x'_{k+1} = x'_k + \omega(b' - A'x'_k)$ with the relaxation parameter $\omega \in \mathbb{R}$ is used to obtain an iterative pressure equation.

For an iterative velocity update, we regard v in iteration $k+1$ as previous iterate plus an error term, and use the residual equation to replace the error. In equations, we begin with $v_{k+1} = v_k + e_k^v$ and the residual equation $Ve^v = r^v$ for Equation (3.8). When we invert the residual equation and plug in the residual $r^v = f - (Cp + Vv)$, this leaves us with the following Uzawa iteration:

$$v_{k+1} = v_k + V^{-1}(f - (Cp_k + Vv_k))$$
$$p_{k+1} = p_k + \omega(g - Bv_{k+1})$$

The scheme above is also called an exact Uzawa iteration, because it contains the exact inverse of V and no approximation to it. Since V in our Stokes-Darcy problem is a matrix of dimensions $\approx 2N_xN_y$, an exact inversion of the matrix should be avoided at all cost. In practice, we use an inexact version of Uzawa [25], which utilizes a suitable iterative scheme like AMG to approximate V^{-1}.

Exact vs. inexact Uzawa iteration

From a theoretical perspective it is important to know what we can expect for an inexact formulation of the Uzawa iteration in terms of convergence rates and error bounds compared to the exact formulation. In [25], Elman and Golub analyze the convergence rates for the exact and a very general formulation of the inexact Uzawa scheme. Below we summarize the key points of their findings.

Our saddle point problem (3.6) exhibits a slightly different structure to the one in [25], which treats the case where the upper left block is nonzero. Additionally, their off-diagonal blocks are transposed

versions of the same matrix, making the overall system symmetric:

$$\begin{pmatrix} -E & B \\ B^\top & V \end{pmatrix} \begin{pmatrix} p \\ v \end{pmatrix} = \begin{pmatrix} 0 \\ f \end{pmatrix} \tag{3.9}$$

Above, the matrix V is symmetric positive definite and the matrix C symmetric positive semidefinite. Using the exact iterations $Vv_{k+1} = f - B^\top p_k$ and $p_{k+1} = p_k + \omega(Bv_{k+1} - Vp_k)$ and plugging v_{k+1} in the iteration for p_{k+1} gives

$$p_{k+1} = p_k + \omega(BV^{-1}f - (BV^{-1}B^\top + E)p_k).$$

From this formulation, the convergence for the *exact* Uzawa algorithm is analyzed. Using the corresponding Richardson iteration and the fact that for a symmetric matrix the spectral radius $\rho(A) = \max\{|\lambda_1|, \dots, |\lambda_N|\}$ is equal to its two-norm, one can derive

$$\|p - p_k\|_2 \leq \Big(\rho(I - \omega(BV^{-1}B^\top + E)\Big)^k \|p - p_0\|_2. \tag{3.10}$$

This shows that the error for p after k exact Uzawa iterations is bounded from above by the initial error, multiplied by a k-dependent factor. If the spectral radius of $(I - \omega(BV^{-1}B^\top + E))$ is smaller than one, the iteration converges. We call this spectral radius expression *convergence factor* and find that it is optimal for $\omega = \frac{2}{\lambda_{\min}+\lambda_{\max}}$.

A similar expression can be shown for the *inexact* Uzawa iteration. It is important to point out that the inexact iteration is analyzed independent of the concrete method used to approximate the inverse of V. For this, consider the inexact iteration

$$\begin{aligned} Vv_{k+1} &= f - B^\top p_k + \delta_k, \\ p_{k+1} &= p_k + \omega(Bv_{k+1} - Ep_k), \end{aligned}$$

where δ_k is used to represent the residual induced by inexactly solving $V^{-1}(f - B^\top p_k)$. The inexact residual can be bound from above by the pressure residual in the first row of (3.9), which is computed in the previous Uzawa iteration:

$$\delta_k \leq \tau\|Bv_k - Ep_{k-1}\|_2. \tag{3.11}$$

The parameter τ is a measure for the accuracy of V^{-1} and therefore also for the accuracy of the inexact Uzawa iteration.

We now define the approximation error vectors for the pressure $e_k^p = p - p_k$ and velocity $e_k^v = v - v_k$, and plug those formulations into the right hand side of Equation (3.11). Because the residual is zero for the exact solution, this leaves the expression $\tau\|Be_k^v - Ee_{k-1}^p\|_2$. It only depends on the errors for pressure and velocity.

In an exact setting, it is possible to derive a bound to this error expression analogous to (3.10) as

$$\|Be_{k+1}^v - Ee_k^p\|_2 \leq \Big(\rho(I - \omega(BV^{-1}B^\top + E))\Big)^k \|Be_1^v - Ee_0^p\|_2. \tag{3.12}$$

A very similar result can be derived for the inexact setting. Here, we only state the result from [25] for the inexact case as

$$\|Be_{k+1}^v - Ee_k^p\|_2 \leq \Big(\rho + \underbrace{\eta + \frac{1}{2}(\rho+\eta)\Big(\sqrt{1 + \frac{4\eta}{(\rho+\eta)^2}} - 1\Big)}_{\xi}\Big)^k c. \tag{3.13}$$

In the above formulation, c is a k-independent constant, $\rho = \rho(I - \omega(BV^{-1}B^\top + E))$ is the spectral radius from Equation (3.12) and $\eta = \tau\|BV^{-1}\|_2$. When we compare the bounds for the exact and inexact

residual in Equations (3.12) and (3.13), we see a difference of ξ in the convergence rates. Therefore, ξ is a bound on the performance degradation for using the inexact Uzawa scheme. As it also depends on the solver accuracy τ, and closer examination reveals that it vanishes for $\tau \to 0$. In combination with Equation (3.11) this indicates that if $V^{-1}(f - B^\top p_k)$ is solved arbitrarily exact, the convergence rate of the inexact scheme approaches the exact scheme's rate with $\mathcal{O}(\tau)$. From the inexact error bound (3.13) it is also possible to derive bounds for the two norms of e_k^v and e_k^p as

$$\begin{aligned}\|e_k^p\|_2 \leq& \rho^{k-2}\|e_2^p\|_2 + \hat{c}(\rho+\xi)^k \quad \text{and}\\ \|e_k^v\|_2 <& \tilde{c}(\rho+\xi)^k.\end{aligned}$$

The quantities $\hat{c}$ and $\tilde{c}$ are independent of k. Also, the errors associated with the solution components p and v converge to zero, and the convergence rate is by ξ "slower" than the convergence rate of the exact Uzawa iteration. However, we can improve the convergence rate by reducing τ, which equals a more accurate solver for V. This leads to an inexact Uzawa scheme that converges with a rate close to the exact Uzawa iteration.

We believe the proof from [25] that is outlined above can also be conducted for unsymmetric matrices like the Stokes-Darcy system in Equation (2.15) with minor modifications. The current proof uses the system's symmetry for the derivation of Equations (3.10) and (3.12) to exchange the two-norm of $I - \omega\,(B\,V^{-1}\,B^\top + E)$ with the spectral radius. For symmetric matrices M_s it holds $\rho(M_s) = \|M_s\|_2$, while for any complex matrix M the spectral radius is a lower bound to induced matrix norms and hence $\rho(M) \leq \|M\|_2$. Because the proof itself treats the spectral radius as convergence rate but does not use other properties of the spectral radius, we believe that a slightly weaker bound can be derived with the two norm as convergence rate instead. Further, for the Stokes-Darcy system we have $E = 0$ and $B := B \neq B^\top =: C$, if we use our notation for the Stokes-Darcy problem from Equation (3.6). The form $E = 0$ simplifies almost all equations above, we especially emphasize that the left hand sides of Equations (3.12) and (3.13) are reduced to $\|Be_{k+1}^v\|_2$. The spectral radius in those equations, which would be replaced by a two norm in the unsymmetric case, reduces to $\|I - \omega\,B\,V^{-1}\,C\,\|_2$.

Also, numerical experiments in Section 6.1.5 indicate that the convergence rate of the inexact Uzawa iteration can be brought close to the convergence rate of the exact one, if the inversion of V is performed with higher accuracy. This supports the presumption that the proof's statement above might be applicable to the unsymmetric case with weaker error bounds.

3.3 Preconditioned GMRES(m) and PD-GMRES

In the previous sections of this chapter we discussed the GMRES method as applicable iterative solver for the Stokes-Darcy problem and a variety of preconditioning techniques. As indicated before, preconditioning the system does not mean to explicitly multiply the preconditioning matrix to the system, but rather to adapt the solver so that only preconditioner applications to vectors are required. This motivated the using the iterative AMG and Uzawa methods as preconditioning techniques. The goal of this last section is to discuss a preconditioned version of GMRES and to interlink the previous discussions about solvers and preconditioners.

Generally when we precondition a solver, the chosen preconditioning technique has to be compatible with the solver. If, for example, the solver only works with symmetric matrices, applying the preconditioner must not compromise the system's symmetry. Fortunately, GMRES is applicable to linear, unsymmetric systems of equations, which allows us to combine it with all previously introduced preconditioning strategies.

It is possible to develop GMRES for all three preconditioning types: left, right and split preconditioning. In the scope of this thesis we use the left preconditioned variant of GMRES. The reason to use a left

Algorithm 3.5 Left preconditioned restarted GMRES [46, Chap. 9.3]

procedure LEFTPRECONDITIONEDGMRES(x_0, A, b, m, $\mathcal{P}$)
  $r_0 = \mathcal{P}(b - Ax_0)$
  $v_1 = \frac{r_0}{\|r_0\|}$
  while not converged **do**
    for $i = 1, \dots, m$ **do**
      $h_{j,i} = \langle \mathcal{P}Av_i, v_j \rangle$ **for** $j = 1, \dots, i$
      $\hat{v}_{i+1} = \mathcal{P}Av_i - \sum_{j=1}^{i} h_{j,i} v_j$
      $h_{i+1,i} = \|\hat{v}_{i+1}\|$
      $v_{i+1} = \frac{\hat{v}_{i+1}}{h_{i+1,i}}$
    $y_m = \min_y \left\| \|r_0\| e_1 - \bar{H}_m y \right\|$
    $x_m = x_0 + V_m y_m$
    $r_m = b - Ax_m$, test convergence
    if no convergence **then**
      $r_0 = \mathcal{P}r_m$, $v_1 = \frac{r_m}{\|r_m\|}$
    update m (only for PD-GMRES)
  return x_m

preconditioned GMRES method is that it exists in the solver library ISTL [8], because we use ISTL to simulate the Stokes-Darcy problem. Parts of ISTL that are relevant for this work are discussed in Chapter 5.

For the left preconditioned version of GMRES [46, Chap. 9.1] as implemented in ISTL, we intend to solve the left preconditioned system

$$\mathcal{P}Ax = \mathcal{P}b.$$

This brings two main modifications to the GMRES algorithm. First, the residual of the preconditioned problem is the preconditioned residual of the original problem

$$r = \mathcal{P}(b - Ax)$$

and second, the Krylov subspace is build with the preconditioned matrix instead of A alone: $\mathcal{K}_k = \text{span}\{r_0, \mathcal{P}Ar_0, \dots, (\mathcal{P}A)^{k-1} r_0\}$, with $r_0 = \mathcal{P}(b - Ax_0)$. Algorithm 3.5 shows those modifications for the restarted GMRES. Blue scripts marks differences to the unpreconditioned, restarted GMRES Algorithm 3.2.

An important observation for the left preconditioned version of the restarted GMRES method is that the residual norm is only available at the end of every cycle of Arnoldi's algorithm. The residual norms that are available during Arnoldi's algorithm as discussed in Section 3.1.2 are *preconditioned* residuals and can exhibit a different convergence behavior compared to the unpreconditioned residual. This implies that convergence checks using the unpreconditioned residual can only be performed after each Arnoldi cycle, which prevents the early interruption of a cycle due to a sufficient residual reduction. We include the unpreconditioned residual control into ISTL's restarted GMRES implementation, more details are given in Section 5.4. To circumvent the convergence control issues described for left and split preconditioning, it would also be possible to implement a right conditioned version of GMRES [46, Chap. 9.3]. In this formulation, Arnoldi's algorithm computes unpreconditioned residual norms. However, a right preconditioned restarted GMRES is not part of ISTL's solver library.

Another side effect of using the preconditioned residual in the Arnoldi iteration is that all theory about a monotonically decreasing residual therefore applies to the preconditioned residual. Hence it is possible to observe an increasing *unpreconditioned* residual during the application of a left preconditioned GMRES

method. This can be seen after the first restarted GMRES cycle in many GMRES convergence plots in the evaluation in Section 6.1.

We have demonstrated how to modify the restarted GMRES algorithm to solve a left preconditioned system. Every preconditioner that was introduced in this chapter can be applied to GMRES in its left preconditioned formulation. In the next chapters we develop preconditioning techniques that are especially designed for the Stokes-Darcy problem and describe their implementation in ISTL, before we continue with results that are obtained by iteratively solving the Stokes-Darcy system with left preconditioned GMRES and different preconditioners in Chapter 6.

Chapter 4

Preconditioning for the Stokes-Darcy system

We now discuss how to use the preconditioning techniques from the last chapter for the Stokes-Darcy problem. For this, we recall the linear system, which originated from discretizing the coupled Stokes-Darcy system in Section 2.2:

$$Ax = b \tag{4.1}$$

$$\begin{pmatrix} A' & B' \\ C' & D' \end{pmatrix} \begin{pmatrix} x^{\mathrm{ff}} \\ x^{\mathrm{pm}} \end{pmatrix} = \begin{pmatrix} b^{\mathrm{ff}} \\ b^{\mathrm{pm}} \end{pmatrix} \tag{4.2}$$

$$\begin{pmatrix} \begin{bmatrix} 0 & B \\ C & V \end{bmatrix} & \begin{bmatrix} 0 \\ B_1' \end{bmatrix} \\ \begin{bmatrix} 0 & C_1' \end{bmatrix} & D' \end{pmatrix} \begin{pmatrix} \begin{bmatrix} p^{\mathrm{ff}} \\ v^{\mathrm{ff}} \end{bmatrix} \\ p^{\mathrm{pm}} \end{pmatrix} = \begin{pmatrix} \begin{bmatrix} h \\ g \end{bmatrix} \\ l \end{pmatrix} \tag{4.3}$$

Depending on the domain discretization, A can become very large and is sparse. In the last chapter we extensively discussed that iterative solvers are the method of choice for computing the solution x in this setup. But as A is ill-conditioned, we need to apply suitable preconditioning in order to obtain an iterative scheme whose convergence is fast enough to compete with direct solvers.

A straight forward approach is to precondition A directly with the previously introduced techniques from Section 3.2. However, for the introduced preconditioners this procedure is ill-advised for two reasons. First, a number of preconditioners cannot be applied directly, because A's zero diagonal block leads to divisions by zero during the application of Jacobi, Gauss-Seidel or ILU preconditioners, to name a few. Second, A's block structure reflects the underlying model, which leads to the saddle point structure of A'. We also know that D' stems from a diffusive process. If the relation to the underlying model is addressed explicitly by the preconditioning technique, it is not far-fetched to expect a better performance in comparison to leaving this additional information aside.

In the remainder of this chapter, we explore a block preconditioning strategy that performs a high-level block Gauss-Seidel preconditioning and leaves the preconditioner for each block to be chosen independently. This idea is inspired from Cai, Mu, and Xu [14].

4.1 Partially decoupled Stokes-Darcy preconditioner

4.1.1 Block Gauss-Seidel preconditioner

To build a preconditioner for A, we seek an approximation to its inverse A^{-1}. Following the idea of the Gauss-Seidel preconditioner, the *block* Gauss-Seidel preconditioner $\mathcal{P}_{\mathrm{BGS}}$ is obtained by inverting

the reduced matrix $\widehat{A}$, which consists of diagonal and lower triangular blocks of A. This can also be seen as a partial decoupling of the system as suggested in [14]. Depending on whether we consider the blocks of A as belonging to the free flow and porous media flow, as in Equation (4.2), or if we split A in the pressures and velocities as in Equation (4.3), this yields slightly different formulations. We call the first case *two-phase* block preconditioning and mark it with $^{\mathrm{tp}}$, and the latter *pressure-velocity* block preconditioning, which is marked with $^{\mathrm{pv}}$.

We start by considering the two-phase case in Equation (4.2) and define $\widehat{A}^{\mathrm{tp}}$ to be the Gauss-Seidel approximation of A:

$$\widehat{A}^{\mathrm{tp}} = \begin{pmatrix} A' & 0 \\ C' & D' \end{pmatrix}$$

An application of the two-phase Gauss-Seidel preconditioner can now be defined as multiplication with the block inverse of $\widehat{A}^{\mathrm{tp}}$, which is

$$(\widehat{A}^{\mathrm{tp}})^{-1} = \begin{pmatrix} A'^{-1} & 0 \\ -D'^{-1}C'A'^{-1} & D' \end{pmatrix}.$$

This formulation requires the square matrices A' and D' to be invertible. A full inversion of A' and D' is computationally expensive. Luckily this is not even necessary, as the preconditioner $\mathcal{P}^{\mathrm{tp}}$ should only improve the condition number when multiplied to A and not be an exact inverse. Therefore, we approximate A'^{-1} and D'^{-1} with preconditioning matrices build for A' and D', and denote those by $\mathcal{P}_{A'}$ and $\mathcal{P}_{D'}$. This replacement gives our formulation of the **two-phase block Gauss-Seidel preconditioner**:

$$\mathcal{P}^{\mathrm{tp}}_{\mathrm{BGS}}(\mathcal{P}_{A'}, \mathcal{P}_{D'}) \coloneqq \begin{pmatrix} \mathcal{P}_{A'} & 0 \\ \mathcal{P}^{\mathrm{tp}}_{1,0} & \mathcal{P}_{D'} \end{pmatrix} \coloneqq \begin{pmatrix} \mathcal{P}_{A'} & 0 \\ -\mathcal{P}_{D'}C'\mathcal{P}_{A'} & \mathcal{P}_{D'} \end{pmatrix} \tag{4.4}$$

The two-phase block Gauss-Seidel preconditioner requires two independent preconditioners for the block matrices A' and D', but not the matrices themselves nor their analytic inverses. If it is clear from the context or a subsequent matrix definition, we omit the used preconditioners in brackets. To reference those sub-preconditioners, we call $\mathcal{P}_{A'}$ the preconditioner for the *free flow block* and $\mathcal{P}_{D'}$ preconditioner for the *porous media (pressure) block*. We can use the word *pressure* as identifier for the sub-preconditioner, because we see below that specifying a preconditioner for the free flow pressure block alone is never required in our block preconditioner definitions.

Applying $\mathcal{P}^{\mathrm{tp}}_{\mathrm{BGS}}$ to a vector $z = (z_1, z_2)^\top$ is straight forward:

$$\mathcal{P}^{\mathrm{tp}}_{\mathrm{BGS}} z = \begin{pmatrix} \mathcal{P}_{A'} & 0 \\ -\mathcal{P}_{D'}C'\mathcal{P}_{A'} & \mathcal{P}_{D'} \end{pmatrix} \begin{pmatrix} z_1 \\ z_2 \end{pmatrix} = \begin{pmatrix} \mathcal{P}_{A'} z_1 \\ \mathcal{P}_{D'} z_2 - \mathcal{P}_{D'}(C'\mathcal{P}_{A'} z_1) \end{pmatrix}$$

Given the special structure of A, it might be useful to treat the free flow matrix block as two blocks because it comprises the sub-blocks for pressure and velocity. This gives rise to our second formulation. The pressure-velocity Gauss-Seidel matrix $\widehat{A}^{\mathrm{pv}}$ is

$$\widehat{A}^{\mathrm{pv}} = \begin{pmatrix} \begin{bmatrix} I & 0 \\ C & V \end{bmatrix} & 0 \\ 0 \quad C_1' & D' \end{pmatrix}.$$

It is important to point out that we replaced the zero diagonal block with an identity matrix. This ensures that resulting preconditioner is nonsingular. After block inverting $\widehat{A}^{\mathrm{pv}}$ and replacing the diagonal block inverses by preconditioner applications, analogous to what we did for $\mathcal{P}^{\mathrm{tp}}_{\mathrm{BGS}}$, we obtain the **pressure-velocity block Gauss-Seidel preconditioner** $\mathcal{P}^{\mathrm{pv}}_{\mathrm{BGS}}$ as

$$\mathcal{P}^{\mathrm{pv}}_{\mathrm{BGS}}(\mathcal{P}_V, \mathcal{P}_{D'}) \coloneqq \begin{pmatrix} \mathrm{I} & 0 & 0 \\ \mathcal{P}^{\mathrm{pv}}_{1,0} & \mathcal{P}_V & 0 \\ \mathcal{P}^{\mathrm{pv}}_{2,0} & \mathcal{P}^{\mathrm{pv}}_{2,1} & \mathcal{P}_{D'} \end{pmatrix} \coloneqq \begin{pmatrix} \mathrm{I} & 0 & 0 \\ -\mathcal{P}_V C\mathrm{I} & \mathcal{P}_V & 0 \\ \mathcal{P}_{D'}C_1'\mathcal{P}_V C\mathrm{I} & -\mathcal{P}_{D'}C_1'\mathcal{P}_V & \mathcal{P}_{D'} \end{pmatrix}. \tag{4.5}$$

We deliberately use the notation I instead of I to emphasize that I is an identity preconditioner and not necessarily an identity matrix. For a more compact notation, we denote the off-diagonal preconditioners by $\mathcal{P}^{\mathrm{pv}}_{1,0}$, $\mathcal{P}^{\mathrm{pv}}_{2,0}$ and $\mathcal{P}^{\mathrm{pv}}_{2,1}$, and neglect the superscript $^{\mathrm{pv}}$ if it becomes clear from context. While $\mathcal{P}_{D'}$ preconditions the same block as in the two-phase block Gauss-Seidel preconditioner, we call $\mathcal{P}_V$ *preconditioner for the free flow velocity block* and emphasize that also in the pressure-velocity case $\mathcal{P}_{D'}$ is the only eligible preconditioner that is applied to a pressure block.

4.1.2 Block Jacobi and partial block Gauss-Seidel preconditioner

For the two-phase and pressure-velocity block Gauss-Seidel strategies we choose two preconditioning techniques, which fully define the off-diagonal block preconditioners $\mathcal{P}^{\mathrm{tp}}_{1,0}$, $\mathcal{P}^{\mathrm{pv}}_{1,0}$, $\mathcal{P}^{\mathrm{pv}}_{2,0}$ and $\mathcal{P}^{\mathrm{pv}}_{2,1}$. If we would set all these preconditioners to adequately shaped zero matrices, our block Gauss-Seidel preconditioner simplifies to a block diagonal preconditioner. This is nothing else but the block version of Jacobi preconditioning. Therefore we define the **two-phase block Jacobi preconditioner** as

$$\mathcal{P}^{\mathrm{tp}}_{\mathrm{BJ}}(\mathcal{P}_{A'}, \mathcal{P}_{D'}) \coloneqq \begin{pmatrix} \mathcal{P}_{A'} & 0 \\ 0 & \mathcal{P}_{D'} \end{pmatrix} \tag{4.6}$$

and the **pressure-velocity block Jacobi preconditioner** as

$$\mathcal{P}^{\mathrm{pv}}_{\mathrm{BJ}}(\mathcal{P}_V, \mathcal{P}_{D'}) \coloneqq \begin{pmatrix} \mathrm{I} & 0 & 0 \\ 0 & \mathcal{P}_V & 0 \\ 0 & 0 & \mathcal{P}_{D'} \end{pmatrix}. \tag{4.7}$$

In the two-phase block scenario we only have one off-diagonal preconditioner, $\mathcal{P}^{\mathrm{tp}}_{1,0}$. If we calculate it according to its definition, we obtain the two-phase block Gauss-Seidel preconditioner and if we set it to zero, we end with the two-phase block Jacobi. But the pressure-velocity case the decision is not binary, as we have three off-diagonal preconditioners that can potentially be set to zero. To extend the choice of possible block preconditioning techniques, we call all pressure-velocity preconditioners with more than zero and fewer than three nonzero off-diagonal preconditioners **partial (pressure-velocity) block Gauss-Seidel preconditioners**. Occasionally we omit the pressure-velocity identifier, as partial block preconditioning is not possible for the two-phase scenario for the reason above. For each partial block preconditioner, we need to specify which of the blocks $\mathcal{P}^{\mathrm{pv}}_{1,0}$, $\mathcal{P}^{\mathrm{pv}}_{2,0}$ and $\mathcal{P}^{\mathrm{pv}}_{2,1}$ is nonzero. With generic diagonal preconditioners $\mathcal{P}_V$ and $\mathcal{P}_{D'}$ we introduce the notations $\mathcal{P}^{\mathrm{pv}}_{\mathrm{BGS}}(\mathcal{P}_V, \mathcal{P}_{D'}, \mathcal{P}_{1,0})$, $\mathcal{P}^{\mathrm{pv}}_{\mathrm{BGS}}(\mathcal{P}_V, \mathcal{P}_{D'}, \mathcal{P}_{2,1})$ and $\mathcal{P}^{\mathrm{pv}}_{\mathrm{BGS}}(\mathcal{P}_V, \mathcal{P}_{D'}, \mathcal{P}_{1,0}, \mathcal{P}_{2,1})$, where the added off-diagonal preconditioners are considered as applied. Neglected off-diagonal preconditioners are treated as zero.

4.2 Choice of preconditioning techniques for flow types

For all block preconditioner types from the previous section it is necessary to specify two sub-preconditioners. Therefore, the obligatory next question is what preconditioners should be used for the diagonal blocks of the original matrix A.

When choosing those sub-block preconditioners, the pressure-velocity formulations offers a greater flexibility for selecting them, because the "critical" zero block A' has already been taken care of by the identity matrix as preconditioner. Both remaining blocks D' and V have only nonzero entries on their diagonals, which prevents an immediate breakdown of Jacobi, Gauss-Seidel, ILU and AMG preconditioners.

The two-phase formulation, on the other hand, treats the free flow block as one, which permits the use of an Uzawa preconditioner on the block's saddle point structure. The use of such a sophisticated

Flow type	Invertible	M-matrix	Symmetric	Positive definite/semi-definite
horizontal flow	$\begin{pmatrix} ✗ & & \\ & ✓ & \\ & & ✗ \end{pmatrix}$	$\begin{pmatrix} ✗ & & \\ & ✗ & \\ & & ✓ \end{pmatrix}$	$\begin{pmatrix} ✓ & & \\ & ✗ & \\ & & ✓ \end{pmatrix}$	$\begin{pmatrix} ✗/✓ & & \\ & ✓/✓ & \\ & & ✗/✓ \end{pmatrix}$
vertical flow	$\begin{pmatrix} ✗ & & \\ & ✓ & \\ & & ✓ \end{pmatrix}$	$\begin{pmatrix} ✗ & & \\ & ✗ & \\ & & ✓ \end{pmatrix}$	$\begin{pmatrix} ✓ & & \\ & ✗ & \\ & & ✓ \end{pmatrix}$	$\begin{pmatrix} ✗/✓ & & \\ & ✓/✓ & \\ & & ✓ \end{pmatrix}$

Table 4.1: Properties for pressure-velocity blocks of A in horizontal and vertical flow setup.

The matrix properties were evaluated on exported matrices coming from problems with about 10000 degrees of freedom from DuMu$^{\text{x}}$ and are verified for several problems with fewer and more degrees of freedom. For relevant matrix blocks or derived matrices, the eigenvalues are computed in Python with NumPy's `linalg.eig` function [41, 51], which uses LAPACK's `dgeev` routine [1]. For the identification of invertible matrices, please refer to the discussion in the paragraphs below. For the M-matrix property we check the definition given below and use NumPy to compute the required eigenvalues. A block is declared symmetric if it is pointwise equal to its transpose. For positive definiteness we check if all eigenvalues of a matrix block are greater than zero and whether they are greater or equal to zero for semi-definiteness.

preconditioner that is inspired by the matrix's structure potentially makes up for the reduced choice of possible preconditioner types that can handle zero entries on the matrix's diagonal.

Before we apply the preconditioners from the last section to the sub-matrices of the Stokes-Darcy problem and test the convergence of preconditioned solvers, we give thought to the question if the conditions for applying the preconditioners are given on the blocks. Therefore we check if the necessary assumptions to apply those preconditioners hold on the sub-matrices. Table 4.1 summarizes important properties for the pressure and velocity blocks of the Stokes-Darcy stiffness matrix A.

The invertibility is either derived directly from the problem or tested numerically. The upper left zero block that exists for both flow types is clearly not invertible. From the discretization of Darcy's law with Neumann boundary conditions for the horizontal flow on the a domain as described in Section 2.2, we obtain a matrix structure that has entries with value 4 on the diagonal and in each row ones for the coupled (lower, upper, left, right) cells. Due to the Neumann boundary conditions, the diagonal entry for boundary cells is reduced by one for every neighboring ghost cell. This yields a matrix where all rows sum up to zero, hence any scaled one-vector is in the matrix kernel. This implies the horizontal flow porous media block has a zero eigenvalue and is not invertible. All other blocks should be invertible given the model assumptions, but we still tested numerically that the determinant is non-zero for those blocks with with NumPy's `linalg.slogdet` function [41, 52], which uses LAPACK's `dgetrf` routine [1].

For a matrix to be an M-matrix, all of its off-diagonal entries must be smaller or equal to zero, the diagonal has to be strictly positive and $\rho(I - D^{-1}A) < 1$, where D is the diagonal of A [46, Theorem 1.31]. A matrix is positive definite if all eigenvalues are strictly greater than zero, or positive semidefinite if they are nonnegative. Hence, the M-matrix property as well as positive (semi) definiteness require information about eigenvalues. Therefore we used NumPy's `linalg.eig` function [41, 51], which uses LAPACK's `dgeev` routine [1] and checked the signs or absolute values of the eigenvalues in order to test both properties. For a detailed discussion about the symmetry of all blocks we refer to Section 2.2, we still validated the symmetry numerically by a pointwise comparison of the blocks to their transposes.

The multiplication with an identity matrix, which we call applying an identity preconditioner, is always possible without any requirements to the sub-matrices. A Jacobi preconditioner application requires all diagonal elements to be nonzero to avoid zero divisions during its application and therefore cannot be applied to the free flow pressure block. Clearly, the same requirement holds for ILU preconditioners. But additionally, the existence of an incomplete LU decomposition is only guaranteed for M-matrices. Both conditions are met on the porous media pressure block, but not on the free flow velocity block. Still we

see in the evaluation that applying ILU preconditioning to the free flow velocity block can be successful despite the nonguaranteed existence of the incomplete decomposition.

The non-smooth aggregation algebraic multigrid method as implemented in ISTL [8] with a Gauss-Seidel iteration as smoother can be applied as preconditioner to invertible, unsymmetric matrices. This applies to the free flow velocity block in both flow scenarios and the porous media pressure block in the vertical flow case, but not to the horizontal flow porous media block.

Last but not least, the Uzawa preconditioner needs a saddle point structure and is therefore applicable the Stokes system's upper two-phase diagonal block. To apply the Uzawa preconditioner we use its algorithmic formulation derived from Equation (3.6). The results about the comparability of the exact and inexact formulations, however, require the free flow velocity block to be symmetric positive definite. Even though it is almost symmetric, the incorporation of the coupling conditions makes certain rows and columns unsymmetric. The free flow pressure block fulfills the assumptions because it is symmetric and positive semidefinite. Hence, applying an Uzawa preconditioner to the free flow block is possible, but due to slight unsymmetries in the matrix caused by the coupling conditions, the discussion about the convergence speed comparison between the exact and inexact Uzawa iteration is not fully applicable.

Chapter 5

Implementation

In the last chapter we have developed block preconditioning concepts for stiffness matrices that stem from discretizing the Stokes-Darcy problem in Chapter 2. One of the contributions of this thesis is to integrate the variable block preconditioning concept into DUNE, a complex and widely used numerics environment for solving PDEs, which is described in this chapter. We start by introducing DUNE, the Distributed Unified Numerics Environment [3, 10] and its module system in Section 5.1. A special focus lies on the DUNE-modules DUMU$^{\text{X}}$ [28, 34] and ISTL [8] that provide means of porous media simulation and a templated library for preconditioning and iterative solving.

After this high-level introduction to the overall numerics environment, we turn to ISTL's preconditioner interface in Section 5.2. Understanding the preconditioner interface lays the foundation for our block preconditioner implementation, which is presented in Section 5.3. We show that our approach integrates with the native ISTL preconditioner interface and provides a bigger flexibility than previously implemented preconditioning algorithms. Last, we briefly describe several modifications to ISTL's GMRES implementation that render the iterative solver better suited for our purposes.

5.1 Porous media flow simulation with DUNE and DUMU$^{\text{X}}$

The *Distributed Unified Numerics Environment*, DUNE [3, 10], provides a modular C++ toolbox to facilitate solving PDEs numerically. It provides a very general framework for grid-based solution techniques such as finite elements, finite volumes and finite differences. A guiding principle for its development is to create a flexible toolbox that can be easily adapted to specialized applications and allows an efficient integration of new libraries by providing slim interfaces. This is inspired by the fact that almost all PDE solvers require a grid, but its structure needs to suit the application [4, 5]. Acknowledging that a general numerics environment cannot provide a "one size fits all" approach to solving PDEs, DUNE implements a component-based architecture. It defines abstract interfaces, which in turn allow multiple implementations with different realizations of a given functionality. An example are general solver or preconditioner interfaces that allow to exchange the underlying algorithm depending on the problem. At the same time this makes it possible to program general routines like preconditioned versions of concrete solvers such as a preconditioned GMRES(m). This idea of having common concepts with exchangeable implementations is realized through C++ template metaprogramming, which provides computational efficiency along with generic interfaces.

To maintain flexibility, DUNE is organized in modules. The base of DUNE consists of five so called *core modules*, which define general DUNE functionalities that are needed for almost all applications. On top of these core modules exists a second layer of interchangeable modules to facilitate more specialized applications or functionalities. A schematic illustration is provided in Figure 5.1. In the following we provide a brief overview of the core modules and a selection of secondary modules, with an emphasis

on the porous media module DuMu^x [28, 34] and the *Iterative Solver Template Library* ISTL [8]. Both modules have special relevance when we implement the block preconditioners from Section 4.1 and apply those preconditioning techniques to the Stokes-Darcy problem from Chapter 2.

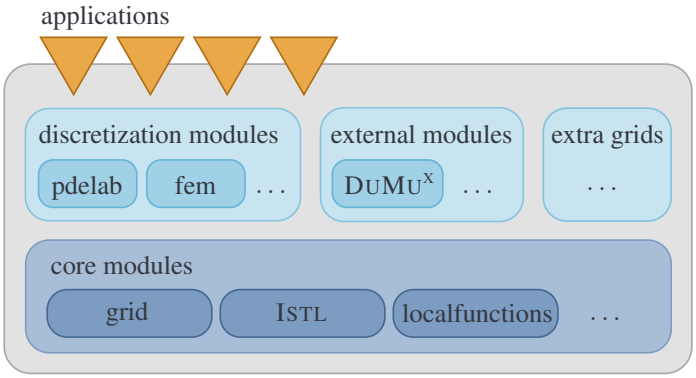


Figure 5.1: Schematic overview of the module system in DUNE.

In order of their appearance on the project website [23], DUNE's five core modules are: dune-common, dune-geometry, dune-grid, dune-istl and dune-localfunctions [3, 10]. The central dune-common module provides general infrastructural classes to orchestrate all other modules. Reference elements for the discretizations, mappings and quadratures are provided by dune-geometry. A generic interface for hierarchical grids and implementations of multiple grid types come from dune-grid [4, 5]. Further grid implementations are optionally provided by additional, non-core modules. For the preconditioner implementation, the *Iterative Solver Template Library* ISTL from dune-istl is the most important module. ISTL provides generic sparse matrix and vector classes, along with a variety of solver and preconditioning routines. Among those are, often applicable both as solver and preconditioner, Krylov subspace methods such as a preconditioned GMRES(m), (block) incomplete factorization ILU(p) and an algebraic multigrid method. PD-GMRES, however, is not a part of dune-istl. We give more details on ISTL's preconditioner interface in the next section and discuss the implementation of a PD-GMRES solver in Section 5.4. The last core module, dune-localfunctions, provides an interface and implementations for the shape functions on reference elements. Further functionalities of the core modules are interfaces for trial and test functions.

Based on these core functionalities build modules that use and extend the common grid interface, for example to support multiple grid managers. Another set of functions on this level provides implementations of the concrete discretization modules for finite elements, finite volumes and finite differences. It is also the place to implement specialized grids or other problem specific core functionalities. External modules also sit on top of the core modules. The most relevant external module for this thesis is DuMu^x [28, 34], which implements DUNE *for Multi-{Phase, Component, Scale, Physics, ...} flow and transport in porous media*. It extends DUNE for porous media simluation, is open source, and the main development currently takes place at the University of Stuttgart. DuMu^x is designed as a framework for research code, and is highly modular. Many models, like the set of equations for the coupled Stokes-Darcy problem in Chapter 2, are pre-implemented in the module. Using an existing model requires the user to specify of two classes: The `Problem` class and the `SpatialParams` class. The `Problem` class is where boundary conditions, initial conditions and volumetric source terms are set, while the `SpatialParams` class defines the spatial distribution of material parameters such as porosity or permeability. Because implementing our block preconditioners only required interaction with DuMu^x to set up the model and very minor changes to the module, we do not give more details here and refer the interested reader to Flemisch et al. [28] and Koch et al. [34].

Listing 5.1: `Preconditioner` class

```
template<class X, class RHS>
class Preconditioner {
public:
  virtual void pre (X& x, RHS& b) = 0;
  virtual void apply (X& v, const RHS& d) = 0;
  virtual void post (X& x) = 0;

  virtual ~Preconditioner () {}
};
```

5.2 Preconditioning with ISTL

After explaining the design principles and introducing a number of modules for DUNE from a high-level perspective, this section gives deeper insight into ISTL's preconditioner interface [8]. Each ISTL preconditioner is derived from a generic `Preconditioner` class that implements a matrix free preconditioner. Preconditioners have to specialize three functions: `pre`, `apply` and `post`. We give details about those functions below. Listing 5.1 illustrates the general structure of the `Preconditioner` base class.

For performance reasons, the `Preconditioner` class is templated and takes two template arguments, `X` and `RHS`. Here, `X` is the update or solution vector type, `RHS` is the type for the right hand side b of the linear system. Of course, derived classes can use more than those two template arguments, but it is mandatory that the signatures of `pre`, `apply` and `post` are not changed.

Applying a preconditioner in ISTL is equal to computing $v = \mathcal{P}d$, where v is some type of update, and $d = b - Ax$ is the current defect. This specification is very general, and does not enforce the update to be implemented as matrix-vector product. Therefore, we do not need an explicit matrix formulation for the preconditioner; A rule how it affects a given vector b is sufficient to calculate v. Within this setting, a Jacobi preconditioner could be implemented in different ways: It can, for example, store a precomputed vector that contains reciprocals of the matrix's diagonal entries and multiply it point wise to the update. Or, it can apply one step of an exiting implementation of Jacobi's method with $x^{(0)} = v = 0$ to obtain the update v as $x^{(1)}$. This example illustrates the flexibility of such a general preconditioner application with ISTL.

A preconditioner can be applied multiple times during a solver's solving routine. For efficiency reasons, `apply` should therefore only perform those steps that are absolutely necessary to compute the update v. All steps related to the setup and destruction of the preconditioner are put in the `pre` and `post` functions. A solver calls the preconditioner's method `pre` once before the first `apply` operation. If the preconditioner needs to calculate auxiliary quantities like incomplete LU decompositions (ILU preconditioner) or grid hierarchies (AMG preconditioner), this should be completed prior to the first preconditioner application. The `pre` method takes the current solution x and the equation's right hand side b. Please note that ISTL does not specify if such pre-computations should be calulated in `pre` or in the constructor. Both ways are possible, it depends on the application if they are equivalent or if one has advantages over the other.

Similar to `pre`, the method `post` is called once when the preconditioner is no longer required. This generally happens when the solver routine has met some type of stopping criteria or broke down. The `post` method is used to safely deallocate memory for auxiliary quantities, and is called with the solution vector x. It remains empty, if no auxiliary quantities are required.

5.3 Stokes-Darcy preconditioner implementation

The biggest challenges we face during our implementation of the block preconditioners are the fully templated classes in ISTL along with the different matrix types. To comply with the existing implementation, our block preconditioners have to be templated and implement the ISTL preconditioner interface. At the same time, we want to use standard matrix types, without the necessity to extend existing matrix definitions.

From our block preconditioner definitions for $\mathcal{P}^{\text{tp}}$ and $\mathcal{P}^{\text{pv}}$ in Equations (4.4) and (4.5) it is immediately clear that our implementation needs to interact with a blocked matrix and block vectors. According to the specification of a preconditioner's `apply` function from the previous section, a block preconditioner application has to compute

$$v = \mathcal{P}b, \tag{5.1}$$

where b and v are block vectors, containing a fix number of sub-vectors v_i and b_i with $i = 1 \dots b$. The preconditioner matrix $\mathcal{P}$ is created for the blocked system matrix A that belongs to the linear system $Ax = b$. Our implementation only covers the case where A is a square matrix and all its diagonal blocks are square. Therefore the number of blocks in b and v are identical, as are the dimensions of the sub-vectors.

ISTL provides a class for explicitly blocked matrices, the `MultiTypeBlockMatrix`. A `MultiTypeBlockMatrix` is structurally a list of lists with different matrix types. As a side effect of this flexibility and their implementation using template meta programming, `MultiTypeBlockMatrices` have no iterators and no random-access operator that work at runtime. This renders it challenging to use them in a setting, where the preconditioner should be adapted during runtime to test different configurations. The second matrix format that is available through ISTL are Block Compressed Row Storage matrices, the `BCRSMatrices`. Because `BCRSMatrices` themselves are build around nested block structures, it is possible to use a `BCRSMatrix` of `BCRSMatrices` to mimic the `MultiTypeBlockMatrix` behavior. This block matrix datatype provides iterators and random access, but does not allow multiple matrix types within a block matrix. In our setup, all matrices are very sparse. Therefore using a `BCRSBlockMatrix` instead of a `MultiTypeBlockMatrix` only differs in the access possibilities of the data format. When the models from Chapter 2 are assembled in DUMU$^{\text{X}}$, the resulting Jacobian is a `MultiTypeBlockMatrix`. That means we must explicitly convert the `MultiTypeBlockMatrix` into a `BCRSBlockMatrix` in our implementation, in order to profit from the easier management options for `BCRSMatrices`. Apart from the required conversion, all solvers and matrix-vector operations work identical on `BCRS-` and `MultiTypeBlockMatrices`. As block matrix compatible block vectors, we use the standard ISTL `BlockVector` and nest them analogous the the `BCRSBlockMatrix` into a `BlockVector` of `BlockVectors`.

Applying a block preconditioner means applying a set of preconditioner objects to the correct vector blocks. Hence the block preconditioner needs access to such a preconditioner set, in addition to the vectors v and b. To supply this set, our preconditioner implementation receives a vector with references to previously initialized preconditioner objects. Those specify the sub-block preconditioners and to which block of the system matrix they are applied. Because these preconditioners for the different sub-blocks all require distinct constructors and setups that depend on the used preconditioners and their required parameters, the preconditioner setup must be performed by the user.

For the block preconditioners $\mathcal{P}^{\text{tp}}$ and $\mathcal{P}^{\text{pv}}$, we add two new general purpose preconditioners to ISTL. The first is a very easy zero preconditioner $\mathcal{P}_0$, which returns a zero vector of size v. It is used on the upper triangular blocks of $\mathcal{P}^{\text{tp}}$ and $\mathcal{P}^{\text{pv}}$. The second preconditioner implements the block inversion for the lower off-diagonal entry of a 2×2 lower triangular block matrix. As arguments, it takes the preconditioners used to approximate the inverses of the matrix's upper and lower diagonal blocks, $\mathcal{P}_{\text{U}}$ and $\mathcal{P}_{\text{L}}$, along with the lower off-diagonal matrix block $A_{1,0}$. Applying the lower diagonal preconditioner $\mathcal{P}_{\text{LD}}$

then implements

$$v = \mathcal{P}_{\mathrm{LD}}\big(\mathcal{P}_{\mathrm{U}}, \mathcal{P}_{\mathrm{L}}, A_{1,0}\big)\, d = -\mathcal{P}_{\mathrm{L}} A_{1,0} \mathcal{P}_{\mathrm{U}}\, d,$$

which simplifies implementing the pressure-velocity block preconditioner from Equation (4.5) to

$$\begin{aligned}\mathcal{P}^{\mathrm{pv}} &:= \begin{pmatrix} \mathrm{I} & 0 & 0 \\ -\mathcal{P}_V C \mathrm{I} & \mathcal{P}_V & 0 \\ \mathcal{P}_{D'} C_1' \mathcal{P}_V C \mathrm{I} & -\mathcal{P}_{D'} C_1' \mathcal{P}_V & \mathcal{P}_{D'} \end{pmatrix} \\ &= \begin{pmatrix} \mathcal{P}_{\mathrm{I}} & \mathcal{P}_0 & \mathcal{P}_0 \\ \mathcal{P}_{\mathrm{LD}}\big(\mathcal{P}_{\mathrm{I}}, \mathcal{P}_V, C\big) & \mathcal{P}_V & \mathcal{P}_0 \\ \mathcal{P}_{\mathrm{LD}}\Big(\mathcal{P}_{\mathrm{LD}}\big(\mathcal{P}_{\mathrm{I}}, \mathcal{P}_V, C\big), \mathcal{P}_{D'}, C'\Big) & \mathcal{P}_{\mathrm{LD}}\big(\mathcal{P}_V, \mathcal{P}_{D'}, C'\big) & \mathcal{P}_{D'} \end{pmatrix}. \end{aligned}$$

Because $\mathcal{P}_{\mathrm{LD}}$ "reuses" existing preconditioner objects, namely those on the diagonal, it is advantageous to use preconditioners that perform preliminary calculations when the constructor is called and not when the `pre` method is executed. If the `pre` method is used to calculate auxiliary quantities, this can lead to re-computations when `pre` is called on the diagonal preconditioners and an instance of $\mathcal{P}_{\mathrm{LD}}$ that uses the diagonal preconditioners. But if this intended behavior is treated accordingly, potentially expensive setups, as in the case of ILU, are performed only once on the diagonal and reused for the off-diagonal preconditioners.

With all necessary general purpose preconditioner classes in place, the apply function is a "matrix-vector product" as defined in (5.1). To obtain a block entry v_i of the block update v, we perform a block matrix-vector product:

$$v_i = \sum_{j=1}^{b} \mathcal{P}_{i,j}\,\texttt{.apply}(v_i^{\text{input}},\ b_j)$$

Here, v_i^{input} is the initial value of v_i when `apply` is called on the block preconditioner $\mathcal{P}$. As one would expect, the block implementation of `apply` calls the specific versions of `apply` on all preconditioners in $\mathcal{P}$. This is not practicable for `pre` and `post`, because the "preconditioner recycling" with $\mathcal{P}_{\mathrm{LD}}$ would lead to multiple initializations. Therefore, the block versions of these two methods call `pre` or `post` only on the diagonal sub-preconditioner objects.

In the current implementation, all possible preconditioners for the diagonal block are specified programmatically for a given matrix structure. Out of those it is possible to select the desired preconditioners via a configuration file and therefore to change the preconditioner configuration for every execution of the program. We plan to remove the problem dependent programmatic specification and replace it with a fully templated concept, which builds the possible diagonal preconditioners implicitly given the block matrix template.

Our implementation should serve as a guidline to implement a block preconditioner, and works reliably if the required sub-preconditioners are specified and initialized prior to initializing the block preconditioner. In future, it would be desirable to parse the sub-preconditioner configurations directly from a parameter file, and initialize the preconditioner objects as part of the block preconditioner initialization. This would allow to precondition `MultiTypeBlockMatrices`, and removes much of the setup overhead for the user. But with an upcoming refactoring of the solver structure in DuMu$^{\mathrm{x}}$, the time to obtain the required in depth knowledge about matrix types in DuMu$^{\mathrm{x}}$ and ISTL that is necessary to satisfactorily implement such a generic solution, is better invested once the new DuMu$^{\mathrm{x}}$ structure is fixed.

All in all, we provide a block preconditioner implementation that is very flexible, because every sub-preconditioner can be specified freely. Therefore, it allows significantly more customization than any existing ISTL preconditioner. It can be applied to block matrices and also allows zero blocks on the diagonal. Because it implements the ISTL preconditioner interface, it integrates with all ISTL solvers, as long as the correct matrix and vector formats are used. Therefore, it may be used with any porous media simulation in DuMu$^{\mathrm{x}}$.

5.4 Modifications to the GMRES implementation

For all experiments within this thesis, left preconditioned restarted GMRES variations are applied. ISTL's solver library provides a left preconditioned GMRES(m) implementation, which we modify in two ways. First, we change the convergence check in order to make use of the unpreconditioned residual instead of its left preconditioned version. Second, we use this adapted GMRES(m) as a starting point to implement the adaptive PD-GMRES from Section 3.1.2.

As discussed in Section 3.3, for a left preconditioned GMRES only the preconditioned residual norm is available during an Arnoldi cycle. By default in ISTL, the residual convergence is is checked after every Arnoldi step and therefore uses the *preconditioned* residual. Also by default, a residual reduction *relative* to the initial (preconditioned) residual is tested. For later evaluations, this is not an optimal choice because often one is interested in reducing the *unpreconditioned* residual below a *fixed* bound. As we discussed before, a reduction of the preconditioned residual does not necessarily imply a reduction of the unpreconditioned residual and a fixed bound simplifies comparisons among solvers. During the evaluation, for example, we are interested in measuring the time to reduce the residual below a given value.

We therefore change ISTL's default residual convergence check to test the residual convergence only after a completed Arnoldi cycle, when the unpreconditioned residual is available. Also, we do not consider a residual reduction relative to the initial residual, but give a target residual and stop the iteration if the unpreconditioned residual is reduced below this target.

Based on this modified left-preconditioned GMRES(m) implementation, we add the proportional–derivative strategy from [40] to adaptively select the restart parameter m after each Arnoldi cycle. Our left-preconditioned PD-GMRES implementation integrates the additional PD-rule from Algorithm 3.3 into the modified GMRES(m) implementation and uses it to select a new value for m before each restart. The capability of the adaptive PD-GMRES implementation in comparison to GMRES(m) is evaluated in Section 6.1.1.

Chapter 6

Results

With the theoretical background and our implementation in place, we now evaluate the performance of preconditioned solvers. For this, we solve the stiffness matrix from Chapter 2 for the horizontal and vertical flow model problems. As iterative solvers, we use different variants of GMRES and compare them against the direct solver that is provided by UMFPACK, all from Chapter 3. The iterative solvers are preconditioned with different configurations of the block preconditioning techniques from Chapter 4 and implemented according to our description in Chapter 5.

Our evaluation targets two core questions: First, we analyze which preconditioner configurations work well for the two flow scenarios and check if the results comply with the theory. For this, we compare the convergence behavior for solvers, preconditioned with different diagonal preconditioner configurations for our block Jacobi preconditioner. Later, we extend this analysis to configurations of the (partial) block Gauss-Seidel preconditioner. The last comparison within this configuration analysis is between exact and inexact Uzawa preconditioning.

The second core question is whether our efforts to construct preconditioning techniques for the GMRES solver were enough to outperform the sparse direct baseline UMFPACK. For this, we compare the runtimes for an increasing matrix size and our best iterative preconditioner configurations to the direct solver.

6.1 Convergence study for block preconditioned GMRES variants

All tests within this section are performed on the two-phase horizontal and vertical flow problems as introduced in Section 2.1, and use the model parameters $K = 10^{-6}$ and $\alpha_{\mathrm{BJ}} = 1.0$. The horizontal flow is induced by a pressure difference of $\delta p = 10^{-9}$ between left and right boundary, while the vertical flow has a pressure of $1.6 \cdot 10^{-4}$ applied to the upper free flow boundary. The free flow and porous media domains are two-dimensional unit squares. Unless otherwise noted, both domains are split into 50×50 cells for solver convergence tests, and 20×20 cells when the eigenvalues of preconditioned matrices are computed. With 50×50 cells per domain, the problem has a total of 10100 degrees of freedom.

For the convergence tests, the solvers are applied for 2000 iterations, without residual check. If an AMG method is used and no other parameters are given, we use one V-cycle of ISTL's AMG based on non-smooth aggregation. As smoother it uses a Gauss-Seidel iteration without relaxation ($\omega = 1$) and performs one iteration for pre- and post-smoothing. Further we use a grid hierarchy with at most three grids and a "coarsentarget"-parameter of 2000, which is only relevant if the data agglomeration happens in parallel [9]. The default Uzawa configuration is an inexact Uzawa that executes one Richardson iteration where the optimal relaxation parameter for the Uzawa iteration ω is determined based on estimates for the biggest and smallest eigenvalue. As inexact solver it also uses one V-cycle of the AMG based on

non-smooth aggregation with one pre- and post-smoothing Gauss-Seidel iteration ($\omega = 1$), a maximum of 15 grids and a "coarsentarget"-parameter of 2000.

Whenever we apply PD-GMRES within this chapter, we use the parameters $\alpha = -3$ and $\beta = 5$, which were determined experimentally in [40]. Instead of identifying each PD-GMRES configuration with the full set of parameters as PD-GMRES(m_{init}, m_{min}, m_{step}, α, β), we omit the fixed parameters α and β and write PD-GMRES(m_{init}, m_{min}, m_{step}).

6.1.1 GMRES(m) vs. PD-GMRES

Instead of directly investigating the suitability of different preconditioning techniques, we postpone the answer to this first core question, and instead start by identifying a good restarting strategy. Finding a good solver is as important as effective preconditioning of the system, because the time to solution ultimately depends on the solver's capacity to converge towards an iterative solution with a good rate of convergence. To identify a capable iterative solver for our preconditioner evaluation, we compare the solver performance of our restarted GMRES variants PD-GMRES and the nonadaptive restarted GMRES(m). We only present evaluation results for our horizontal flow setup with 50×50 cells in free flow and porous media domain because almost identical results are obtained for the vertical flow scenario. All solver configurations use the preconditioner configuration

$$\mathcal{P}_{\text{BJ}}^{\text{pv}}(\text{AMG}, \text{ILU(3)}) = \begin{pmatrix} \text{I} & 0 & 0 \\ 0 & \text{AMG} & 0 \\ 0 & 0 & \text{ILU(3)} \end{pmatrix},$$

in order to achieve convergence and iterate each preconditioner for 2000 iterations. For the solver evaluation, we compare the residual's evolution over number of solver iterations for GMRES(5), GMRES(20), GMRES(50). As parameters for PD-GMRES($m_{\text{init}}, m_{\text{min}}, m_{\text{step}}$) we use $m_{\text{init}} = m_{\text{min}} \in \{3, 5, 7\}$ and $m_{\text{step}} \in \{3, 5, 7\}$.

In the evaluation in Figures 6.1 and 6.2 we plot the logarithm of the residual's two-norm over the number of GMRES iterations. Therefore, the plots also illustrate the restart behavior of GMRES(m) and PD-GMRES, as one full restart cycle needs to be completed before the next residual is available. The number of iterations between two residual measurements is the current restart parameter of PD-GMRES. Along with the two-norm of the overall residual (blue line, datapoints marked as dots) we also plot the residual's components associated with the different blocks. Recall that our solution vector is a block vector $x = (p^{\text{ff}}, v^{\text{ff}}, p^{\text{pm}})^\top$, so we can interpret the residual as blocked residual $r = (r_p^{\text{ff}}, r_v^{\text{ff}}, r_p^{\text{pm}})^\top$. The two-norms of those components are plotted in orange, green, and red, respectively.

(a) GMRES(5) **(b)** GMRES(20) **(c)** GMRES(50)

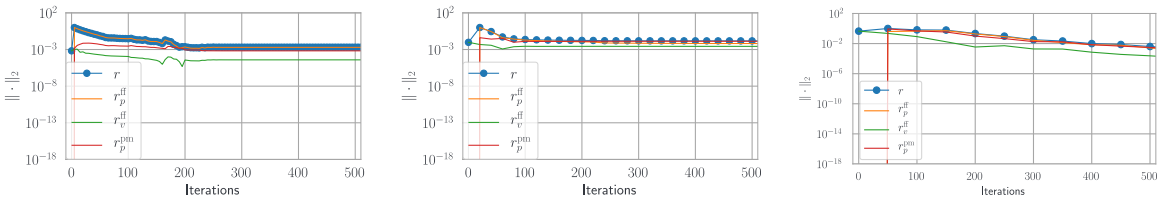


Figure 6.1: GMRES(m) solver comparison on horizontal flow.

The two norm of the residual after each cycle is plotted over the number of iterations for a 50×50 cell discretization on both domains.

For the different restart parameters of GMRES(m) in Figure 6.1 we observe an early stagnation for $m = 5$ and $m = 20$. With a larger choice of $m = 50$ convergence can be achieved, but the convergence rate

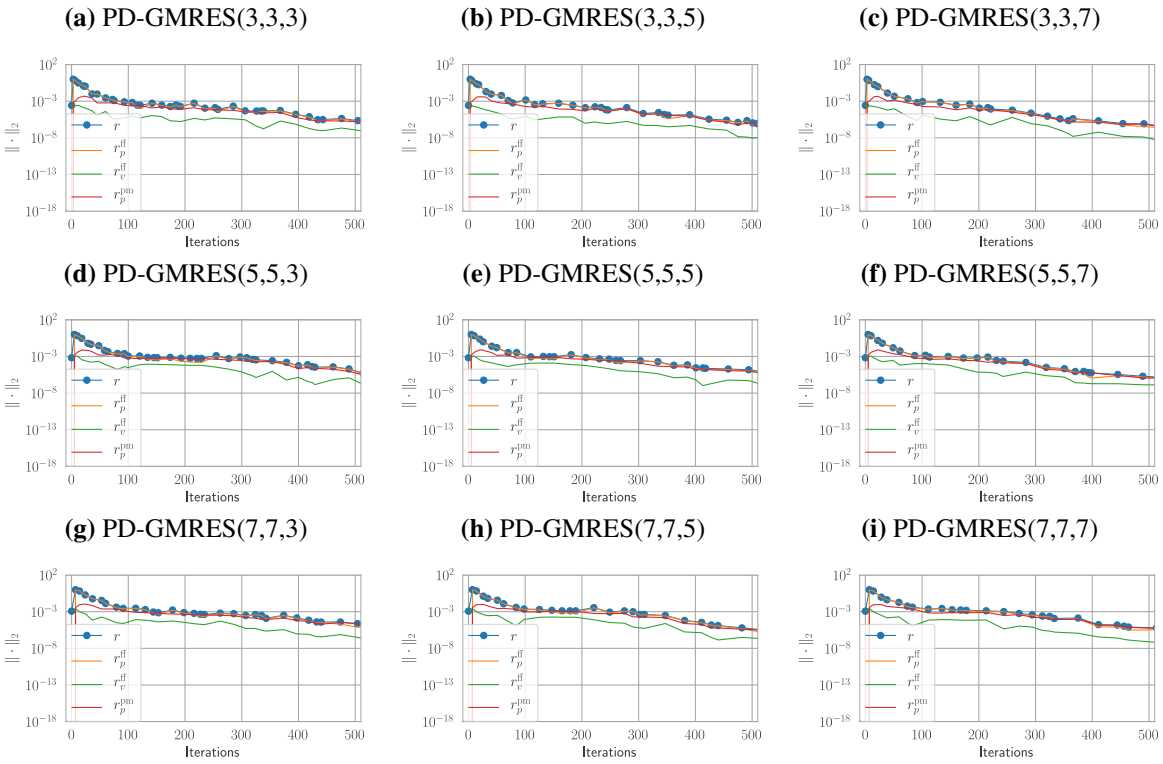


Figure 6.2: PD-GMRES solver comparison on horizontal flow.

Convergence behavior for PD-GMRES($m_{\text{init}}, m_{\text{min}}, m_{\text{step}}$) with the parameters $m_{\text{init}} = m_{\text{min}} \in \{3, 5, 7\}$ and $m_{\text{step}} \in \{3, 5, 7\}$. Again, the residual's two norm is plotted over the number of solver iterations, the discretization has 50×50 cells in each domain.

is low. Also, computing 50 basis vectors per cycle makes every cycle very costly. One reason is that for each new basis vector, scalar products with every previous basis vector have to be calculated. The other reason is that addition to the iterative solution and the matrix-vector product Av_i we need to store all 50 basis vectors by the end of every cycle, where every basis vector has as many entries as the problem's degrees of freedom. As a comparison, for the same problem the sparse direct solver from UMFPACK requires the equivalent of about 78 basis vectors to store all non-zero entries of the matrix decomposition. For a full overview of the required memory for UMFPACK on different problem sizes we remind the reader of Table 3.1, where the memory requirements for the decomposition are analogue to the number of vectors that could be stored by GMRES, until the memory requirements of both solvers are "identical". However, this omits that for GMRES to converge we also have to store or calculate variables for the preconditioners, but also that the values for UMFPACK are those for storing the raw data, which neglects the required data structure to store those sparse entries.

In comparison to PD-GMRES(50), all PD-GMRES configurations in Figure 6.2 yield better convergence rates. The differences between the configurations are small. In general, configurations where $m_{\text{init}} = m_{\text{min}} = 3$ converge faster than configurations with the same m_{step}. For $m_{\text{step}} = 7$ the convergence rate is slightly better than for smaller values. At the same time, we observe restart parameters of about 50 between iterations 200 and 400 for all configurations with $m_{\text{step}} = 7$, which is an undesirable hight number due to the GMRES cost and storage argument above. All PD-GMRES variations with $m_{\text{step}} = 3$ and $m_{\text{step}} = 5$ produce GMRES cycles with fewer than 50 basis vectors. Therefore, choosing a PD-GMRES configuration $m_{\text{step}} = 5$ gives a good balance between moderate restart parameter size with its linked lower memory and computational cost per cycle, and a good convergence rate.

Overall, we observe that PD-GMRES offers advantages over the nonadaptive GMRES(m), which turns it into our solver of choice. Unless otherwise noted we use the solver PD-GMRES(3,3,5) in the following evaluations because it offers a good combination of fast convergence and moderate restart parameter magnitudes.

6.1.2 Notes on comparing convergence on horizontal and vertical flow

In the subsequent sections, we compare different preconditioner setups and their influence on the convergence behavior of PD-GMRES(3,3,5). Often, we show results only for one flow type, horizontal or vertical, as the convergence behavior is very similar. However, there are two important points to note when comparing the scenarios.

First, the scales in the velocity and pressure fields produced by horizontal and vertical flow differ in their extent. We remind the reader of Figure 2.2 for an illustration of the matter. The vertical flow setup produces pressure and velocity fields with a larger order of magnitude than the horizontal flow setup. Because all solvers use $p^{\mathrm{ff}} = v^{\mathrm{ff}} = p^{\mathrm{pm}} = 0$ as initial solution, the initial residual for the horizontal flow setup is always smaller than for the vertical flow case. To account for these differences, we plot relative residuals $\frac{\|r^{(n)}\|}{\|r^{(0)}\|}$ when comparing the residual convergence between horizontal and vertical flow.

A second point to consider is that the vertical flow setup can only be solved iteratively if the inflow is driven by pressure Dirichlet boundary conditions and not by an applied velocity. Setting velocity Dirichlet conditions instead leads to a pure Neumann problem for the pressure in the free flow domain, which leads to nonconverging iterative solvers irrespective of the preconditioning technique. To avoid this problem we use the analogous formulation, which sets the pressure values explicitly on the upper boundary.

6.1.3 Preconditioner comparison for block Jacobi preconditioner

In the first section we have seen convergence results for different GMRES-variants with $\mathcal{P}_{\mathrm{BJ}}^{\mathrm{pv}}(\mathrm{AMG}, \mathrm{ILU(3)})$ as preconditioner. Now, we evaluate different preconditioner choices for the blocks of $\mathcal{P}_{\mathrm{BJ}}^{\mathrm{pv}}$ while taking into consideration the similarities and differences in the solver performance between our two model setups.

All residual convergence figures like Figure 6.3 within this chapter have the same structure: we plot the two norm of the overall residual over the number of iterations in blue with blue markers. Since the exact residual is only available after a PD-GMRES restart, the number of iterations between two blue markers is the current restart parameter for PD-GMRES. Additionally, we plot the two norms for sub-blocks of the blocked residual vector that are associated with p^{ff}, v^{ff} and p^{pm}. Those are marked in orange, green and red without marker. We plot them along with the full residual to see if a prevented or slowing convergence can be linked to a specific block of the Stokes-Darcy problem.

Due to the left preconditioned GMRES implementation in ISTL, any variant of PD-GMRES or GMRES(m) minimizes the *preconditioned* residual [46, Chap. 9.3]. We discussed this effect in Section 3.3. Therefore, a monotonically decreasing unpreconditioned residual cannot be guaranteed, which can often be observed in an increased residual after the first PD-GMRES cycle. Because the true residual for the porous media pressure is almost zero in the beginning (also compare the structure of the matrix system in Equation (2.15) and the matrix structure in Figure 2.3 for an illustration), the porous media pressure residual often seems to come from zero in the plots.

Block Jacobi comparison

Our first comparison explores the suitability of different preconditioner types for the diagonal blocks of a pressure-velocity block preconditioner. For this evaluation, PD-GMRES(3,3,5) is our solver of choice. As baseline, we always compare to solving the unpreconditioned system, which is the same as preconditioning with the identity preconditioner $\mathcal{P}_{\mathrm{BJ}}^{\mathrm{pv}}(\mathrm{I},\mathrm{I}) = \mathcal{P}_{\mathrm{I}}$. Using the generic preconditioners $\mathcal{P}_A$ for the free flow velocity block and $\mathcal{P}_B$ for the porous media pressure block, we always consider the three variations: $\mathcal{P}_{\mathrm{BJ}}^{\mathrm{pv}}(\mathcal{P}_A,\mathrm{I})$, $\mathcal{P}_{\mathrm{BJ}}^{\mathrm{pv}}(\mathrm{I},\mathcal{P}_B)$ and $\mathcal{P}_{\mathrm{BJ}}^{\mathrm{pv}}(\mathcal{P}_A,\mathcal{P}_B)$. For our evaluation, $\mathcal{P}_A$ and $\mathcal{P}_B$ are selected from the preconditioning techniques Jacobi, ILU(3), and AMG. Additionally, we include $\mathcal{P}_{\mathrm{BJ}}^{\mathrm{tp}}(\mathrm{Uzawa},\mathrm{I})$ into the evaluation due to its structural similarity. A full overview over all convergence rates can be found in Table 6.1 that is shown further below in this section. We subsequently discuss selected results for the horizontal flow scenario, which are shown in Figure 6.3.

First, we observe that preconditioning the velocity block in Figures 6.3a, 6.3d and 6.3g has a bigger influence on the overall convergence than preconditioning the porous-media pressure block in Figures 6.3b, 6.3e and 6.3h. If no preconditioning is applied to the system, the solver stagnates as shown in Figure 6.3a. This is also the case if only the porous-media pressure block is preconditioned, irrespective of the preconditioner applied. If, however, the velocity block is preconditioned with a capable technique such as ILU(3) or AMG, the solver reduces the residual. In the case of convergence, both pressure residuals r_p^{ff} and r_p^{pm} generally dominate the velocity residual r_v^{ff}, while r_v^{ff} dominates in the case of stagnation. This shows that preconditioning the free flow velocity block has a special relevance in order to obtain a convergent scheme. A question that is not explored here is whether this is due to its larger dimensions compared to the pressure blocks or due to other structural characteristics.

We also observe that applying a preconditioner that uses more information about the original matrix yields faster solver convergence. While the Jacobi preconditioner $\mathcal{P}_{\mathrm{BJ}}^{\mathrm{pv}}(\mathrm{JAC},\mathrm{I})$, which inverts the matrix's diagonal, cannot overcome the solver stagnation, ILU(3) with its denser approximation leads to solver convergence. ILU(0), which is not shown in this figure, performs better than Jacobi but worse than ILU(3). If the matrix inversion is approximated even better with AMG, the convergence also proceeds faster. Following the general observation that the use of more information about A also leads to faster convergence, it is no surprise that the Uzawa two-phase block preconditioner in Figure 6.3j achieves the best convergence results. In contrast to the pressure-velocity block preconditioners $\mathcal{P}_{\mathrm{BJ}}^{\mathrm{pv}}$, the two-phase block preconditioner approximates the inverse of the full free flow block through Uzawa preconditioning. The coupling blocks between p^{ff} and v^{ff} are included, which is not the case for the other preconditioner setups, and the Uzawa based preconditioner yields a better performance.

Yet the effect of preconditioning the porous media block is not negligible. The combination of velocity and porous media block preconditioning gives faster convergence than for preconditioning the velocity block alone. To further illustrate this point, Figure 6.4 shows the eigenvalue distribution for the horizontal flow stiffness matrix, preconditioned with ILU(3)-variations from the third row in Figure 6.3. Applying ILU(3) preconditioning to the porous media pressure block in Figure 6.4c barely changes the distribution in comparison to the unpreconditioned system in Figure 6.4a. Mostly eigenvalues on the right end of the spectrum are drawn together. Preconditioning the free flow velocity block in Figure 6.4b, however, significantly pushes the spectrum away from zero, closer to one. On this shifted spectrum, the additional porous media block preconditioning has a stronger influence on compacting the overall distribution, which is shown in Figure 6.4d. With the bulk of eigenvalues moved to the right side by the velocity preconditioner, the porous media block preconditioning draws the eigenvalues along the real axis further together. This effect can be observed for Jacobi, ILU(0) and ILU(3) preconditioning. The result for $\mathcal{P}_{\mathrm{BJ}}^{\mathrm{pv}}(\mathrm{AMG},\mathrm{AMG})$ in Figure 6.3 an exception to this rule, because here the combination of both diagonal block preconditioners does not lead to convergence even though AMG on the velocity block alone did. We discuss this observation below.

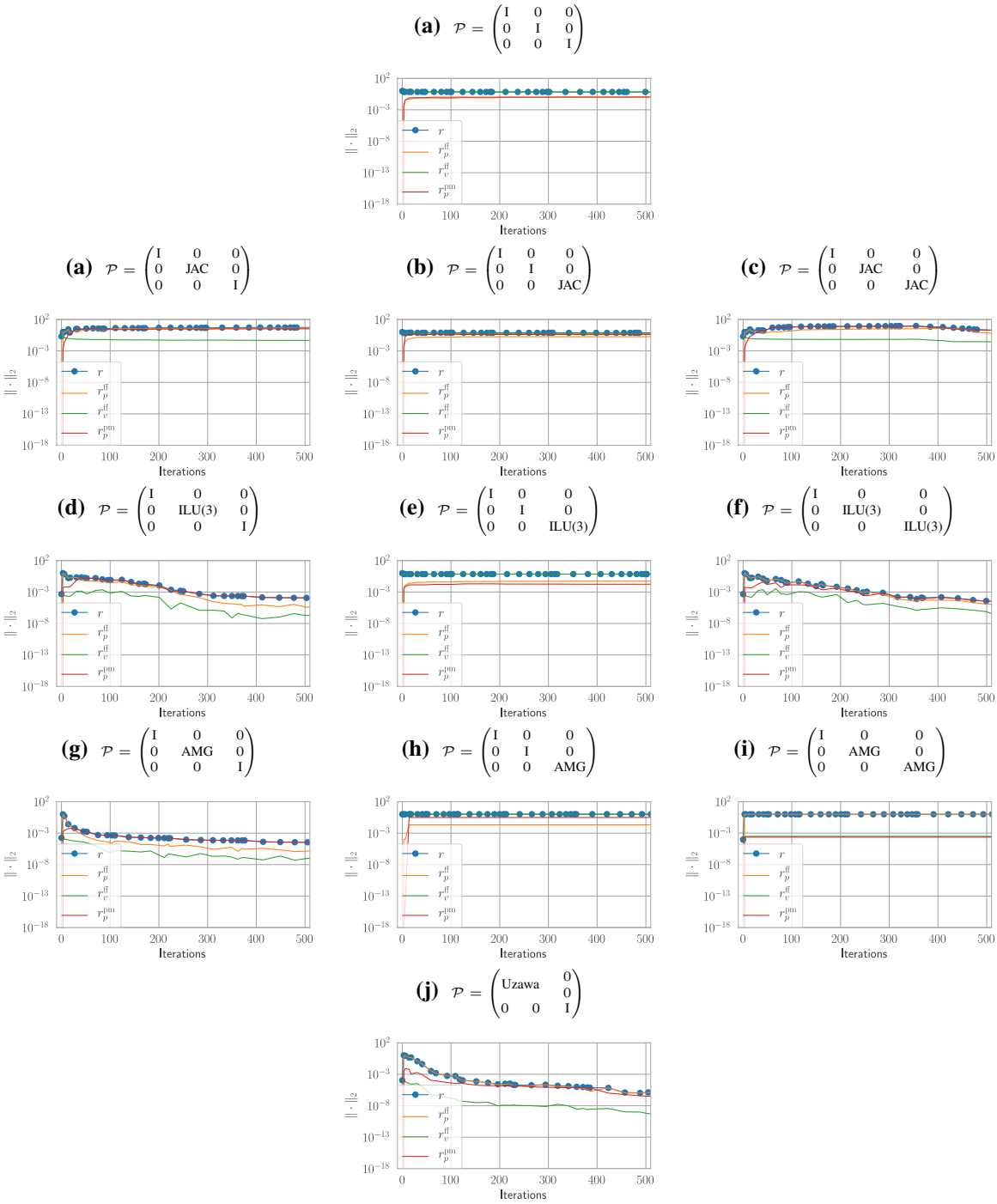


Figure 6.3: Residual convergence for block Jacobi preconditioners on horizontal flow.

We plot the residual norm over the number of preconditioned PD-GMRES(5,5,5) iterations on a 50×50 cell domain discretization. Each row uses a different preconditioner for the matrix blocks, and its position in the block preconditioner is varied per row. From top to bottom: unpreconditioned system, Jacobi preconditioner, ILU(3) preconditioner, AMG preconditioner and Uzawa preconditioner. For Jacobi, ILU(3) and AMG the preconditioners are applied from left to right to the free flow velocity block, porous media pressure block and both of them.

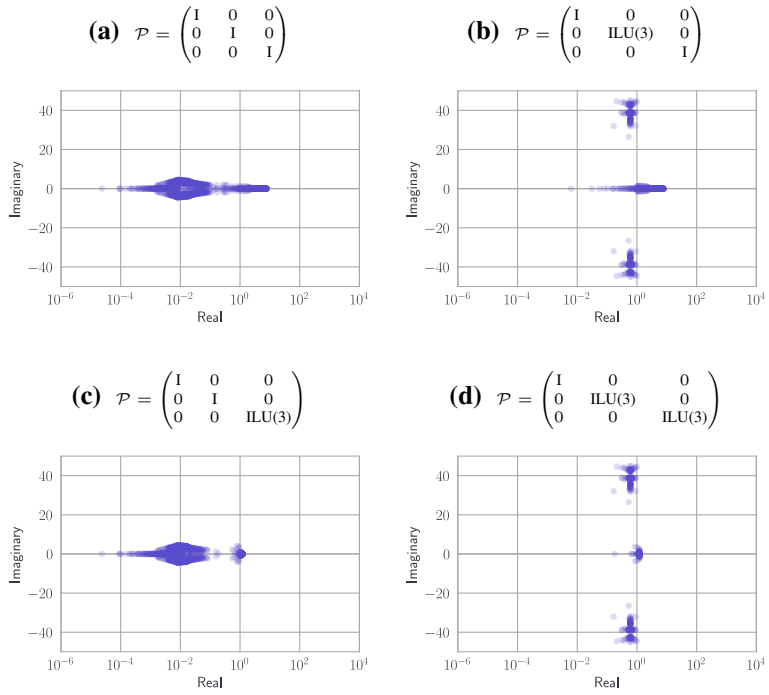


Figure 6.4: Spectrum for ILU(3) block Jacobi preconditioner variations on horizontal flow.

Eigenvalue plot for the preconditioned matrix of a 20×20 cell domain discretization. Note the log scale on the real axis. This figure is the companion figure for the third row of the convergence rates in Figure 6.3.

Approximate convergence rates for all tested block Jacobi preconditioner combinations for the scenarios horizontal and vertical flow are shown in Table 6.1. The convergence rates are calculated between the iterations k_{start} and k_{end}, usually between iterations 1 and 400, as

$$\text{convRate(K)} = \left(\frac{\text{r}^{(\text{k}_{\text{end}})}}{\text{r}^{(\text{k}_{\text{start}})}} \right)^{\frac{1}{k_{\text{end}} - k_{\text{start}}}}.$$

For this evaluation we choose $k_{\text{end}} = 400$ for all block preconditioners that do not utilize Uzawa, as non of them converges until iteration 400 on a 50×50 problem with PD-GMRES(3,3,5). If Uzawa is used, the minimal common number of iterations before the convergence levels out is 200 and therefore chosen for the Uzawa-based block preconditioner. At the end of this section we discuss block Jacobi preconditioning with Uzawa in greater detail, an therefore refer to the results in Figure 6.6 for an illustration of the 200 iteration limit. If the residual is not available at k_{end} because GMRES did not restart in this iteration, the next bigger iteration number with an available residual is chosen. We do the same for k_{start}, because the initial residual r_0 is often much lower than the residual after the first PD-GMRES cycle.

Table 6.2 summarizes the corresponding runtimes to the convergence rates in Table 6.1. The runtimes include the setup time for the respective preconditioners and the time to reduce the residual below the bound that is given in the table explanation. The target residual is 5×10^{-10} for the horizontal flow case and 5×10^{-8} for the vertical flow on a 50×50 cell discretization. All runtimes are measured on a single core of an AMD EPYC 7551P with 2.0 GHz.

We observe that the convergence rates do not directly translate to runtimes. Especially for ILU(3) on the free flow velocity block, the setup cost increases the overall runtimes despite its convergence rate, which is comparable to AMG. Also in the runtime metric, preconditioning the free flow block with the Uzawa

	I	JAC	ILU(0)	ILU(3)	AMG
I	1.00	1.00	1.00	1.00	1.00
JAC	1.00	1.00	1.00	1.00	1.00
ILU(0)	1.00	1.00	0.99		1.00
ILU(3)	0.98	0.98		0.98	1.00
AMG	0.98	0.98	0.98	0.97	1.00
Uzawa	0.97	0.97	0.96	0.92	1.00

	I	JAC	ILU(0)	ILU(3)	AMG
I	1.00	1.00	1.00	1.00	1.00
JAC	1.00	1.00	1.00	1.00	1.00
ILU(0)	0.99	0.99	0.99		0.99
ILU(3)	0.98	0.98		0.97	0.96
AMG	0.98	0.98	0.98	0.97	0.95
Uzawa	0.97	0.96	0.96	0.93	0.92

Table 6.1: Block Jacobi convergence rates for horizontal (left) and vertical flow (right).

Convergence rates until iteration 400 for different pressure-velocity block Jacobi configurations and until iteration 200 for preconditioner setups with Uzawa. The preconditioner for the free flow velocity block is given in the column, the preconditioner for porous media pressure per row. Different parameters for ILU were not cross-tested.

	I	JAC	ILU(0)	ILU(3)	AMG
I	–	–	–	–	–
JAC	29.42	27.04	13.81	18.74	–
ILU(0)	1.98	2.72	1.96		–
ILU(3)	20.15	22.86		19.23	–
AMG	0.66	1.01	0.44	0.46	–
Uzawa	0.37	0.22	0.33	0.42	–

	I	JAC	ILU(0)	ILU(3)	AMG
I	–	–	–	–	–
JAC	71.44	–	24.77	30.15	22.23
ILU(0)	16.59	14.58	5.12		1.16
ILU(3)	35.47	50.38		21.36	18.88
AMG	2.60	4.87	2.11	0.89	0.36
Uzawa	0.95	0.74	0.52	0.49	0.47

Table 6.2: Block Jacobi runtimes for horizontal (left) and vertical flow (right).

Runtimes including preconditioner setup time in seconds on a 50×50 cell discretization for free flow and porous media flow domain on a single core of an AMD EPYC 7551P with 2.0 GHz. Analogous to Table 6.1, the preconditioner for the free flow velocity block is given in the column, the preconditioner for porous media pressure per row. The runtimes are for PD-GMRE(3,3,5) until the residual is reduced below 5×10^{-10} for the horizontal flow case and 5×10^{-8} for the vertical flow. Due to the different target residuals and problems, the runtimes for the horizontal and vertical flow case are not directly comparable. The used residual bounds are below the residuals that are reached if UMFPACK's sparse direct solver is used on the system. Configurations marked with "–" did exceed a maximum runtime of 120 seconds.

preconditioner is better than most other methods. Only in the vertical flow case, $\mathcal{P}_{\mathrm{BJ}}^{\mathrm{pv}}(\mathrm{AMG}, \mathrm{AMG})$ solves the system faster. In the overall comparison, preconditioners of the form $\mathcal{P}_{\mathrm{BJ}}^{\mathrm{pv}}(\mathrm{AMG}, *)$ are often closer to Uzawa-derived preconditioners in terms of runtime than to the other methods. To precondition the free flow velocity block with "weaker" preconditioners like the Jacobi preconditioner does not yield competitive setups, because the slow convergence increases the runtime despite the Jacobi preconditioner's low application costs.

As a last point we note that similar to the convergence rates, the choice of the porous media block preconditioner seems to have the lesser influence on the overall runtime compared to the free flow preconditioner. A good porous media preconditioner, however, can further improve the runtime of a capable free flow preconditioner.

AMG for horizontal and vertical flow

When we compare the exceptional behavior of the AMG preconditioner on the porous media pressure block in the horizontal flow case to the vertical flow scenario, we make an interesting observation. Figure 6.5 compares the three AMG block Jacobi preconditioner variations for our horizontal and vertical flow setups. The convergence behavior for AMG on the velocity block in Figures 6.5a and 6.5b, and AMG on the porous media pressure block in Figures 6.5c and 6.5d is very similar. They are in line with

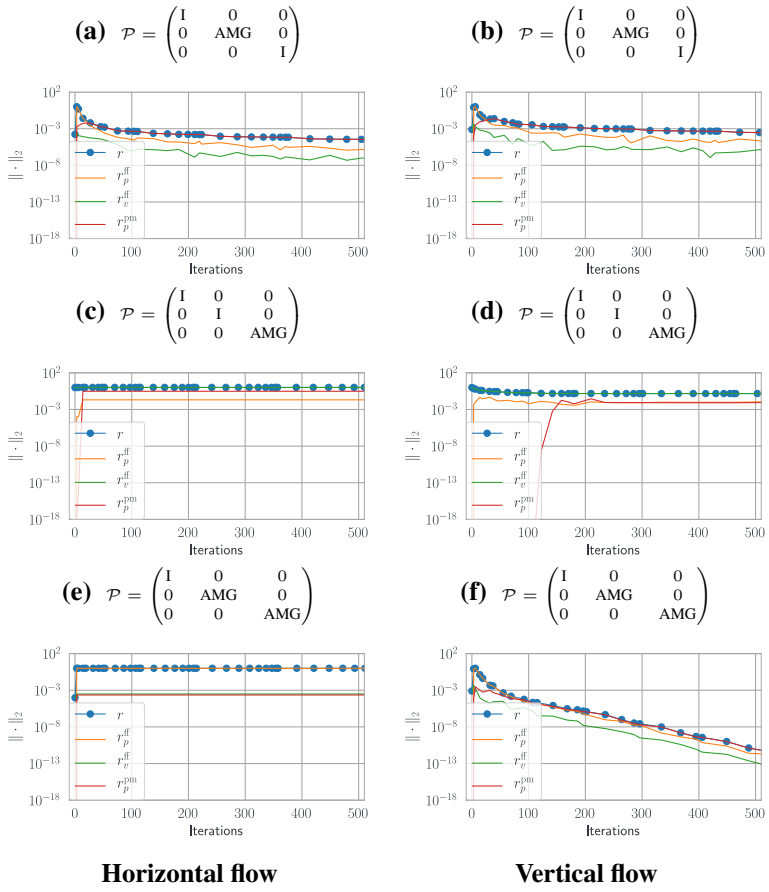


Figure 6.5: AMG preconditioner for horizontal and vertical flow.

Effect of AMG preconditioning on the residual convergence for the horizontal and vertical flow simulation on a 50×50 cell discretization on both domains. The top row compares a block Jacobi preconditioner with the AMG preconditioner applied to the porous media pressure block, the bottom row preconditions both diagonal blocks with AMG. In the horizontal flow case, vanilla AMG is not suitable for for preconditioning the porous media pressure block because it is a pure Neumann problem.

the observation that preconditioning the velocity block has the bigger influence on obtaining a convergent solver.

But for the vertical flow case we also observe that the stagnation for the AMG preconditioned porous media block in Figure 6.5d is not as “rigid” as in the horizontal flow case in Figure 6.5c. Also, preconditioning both blocks with AMG leads to a converging scheme for the vertical flow in Figure 6.5f, while it stagnates for the horizontal flow in Figure 6.5e. This indicates that the reason for the observed differences lies in the AMG preconditioned porous media pressure block.

We recall the boundary conditions in the porous media domain from Figure 2.1. In the horizontal case, we solve a problem with only Neumann boundary conditions, while the vertical case has Dirichlet boundary conditions on the lower boundary. Hence, in the horizontal flow scenario, the porous media pressure solution is defined up to an additional constant.

This poses a problem to vanilla AMG, which might converge to a different solution in every step and therefore prevents successful preconditioning. Other preconditioners like Jacobi or ILU(p) that do not function iteratively but directly instead, still work on this matrix structure.

In vertical flow scenario, the Dirichlet boundary conditions ensure a unique solution for the porous media pressure block alone and make AMG applicable without modification. For the considered preconditioner

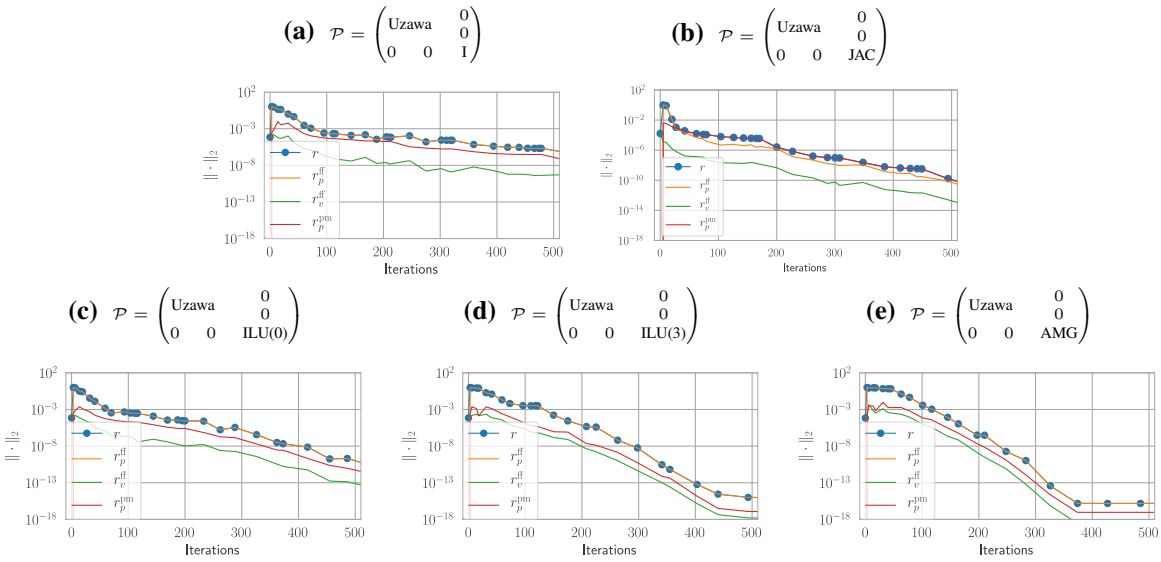


Figure 6.6: Block Jacobi preconditioning with Uzawa for vertical flow.

Comparison of different block Jacobi preconditioner variations using Uzawa on the free flow component. Simulations run with a 50×50 cell discretization for both domains.

setups we can experimentally confirm that AMG preconditioning on the porous media pressure block leads to solver stagnation in the horizontal flow scenario, irrespective of preconditioners applied to the other blocks.

Block Jacobi with Uzawa

As we have seen in our first comparison, applying the two-phase block Jacobi preconditioner $\mathcal{P}_{\text{BJ}}^{\text{tp}}(\text{Uzawa}, \text{I})$ with Uzawa on the free flow block is most effective to reduce the residual for a given number of iterations. In Figure 6.6 we compare Uzawa preconditioning on the free flow block, combined with different preconditioners on the porous media block. All block preconditioners are applied to the vertical flow case.

Here too, we observe a correlation between the amount of information about A that is used by the sub-preconditioners and the solver convergence speed, this time explicitly for the porous media block. Even though we only show illustrations for different preconditioners on the porous media pressure block and a fixed Uzawa preconditioner on the free flow block, we note that the visible trend can be confirmed experimentally also for other preconditioning techniques on the free flow block. For the porous media block, the preconditioners rank from slow to fast convergence in the order identity, Jacobi, ILU(0), ILU(3) and (if the flow permits) AMG. The residual reduction for ILU(3) and AMG stops after about 300 iterations, even though we iterate for 2000 iterations without specified residual stopping criterion. This happens when the computed update to the iterative solution becomes so small in comparison to the solution that the update gets canceled due to floating point rounding errors.

If the porous media block is preconditioned with an identity or Jacobi preconditioner, the residual convergence is dominated by the residual parts linked to both pressures p^{ff} and p^{pm}. However, when we use stronger preconditioner like ILU(p) or AMG, the residual convergence is only determined by the free flow pressure residual.

To conclude the investigation of the block Jacobi preconditioner configurations, we note the following three observations as interim result: Preconditioning the free flow velocity block has the biggest influence on the overall solver convergence and runtime, when we only consider preconditioned diagonal

blocks. However, a capable preconditioner for the porous media pressure block contributes to improving convergence speed in both metrics. For both discussed blocks, we generally observe that preconditioning schemes that use more information or structural cues about the original matrix lead to a quicker residual reduction in terms of iterations and runtimes, as long as the preconditioner's setup time is not too long.

6.1.4 Partial vs. block Gauss-Seidel preconditioning

In the last section we found that sub-block preconditioners that invert more of the original matrix lead to an improved solver convergence. Now we investigate if this observation also holds true on a block matrix level. We recall the pressure-velocity block Gauss-Seidel preconditioner definition as given in Equation (4.5) from Section 4.1:

$$\mathcal{P}_{\mathrm{BGS}}^{\mathrm{pv}} \coloneqq \begin{pmatrix} \mathrm{I} & 0 & 0 \\ -\mathcal{P}_V C\mathrm{I} & \mathcal{P}_V & 0 \\ \mathcal{P}_{D'} C_1' \mathcal{P}_V C\mathrm{I} & -\mathcal{P}_{D'} C_1' \mathcal{P}_V & \mathcal{P}_{D'} \end{pmatrix} = \begin{pmatrix} \mathrm{I} & 0 & 0 \\ \mathcal{P}_{1,0}^{\mathrm{pv}} & \mathcal{P}_V & 0 \\ \mathcal{P}_{2,0}^{\mathrm{pv}} & \mathcal{P}_{2,1}^{\mathrm{pv}} & \mathcal{P}_{D'} \end{pmatrix}$$

It is clear, that our block Gauss-Seidel preconditioner inverts more of the original matrix on a block level than the block Jacobi in the last section. This raises two questions: The obvious question is whether this additional effort to compute $\mathcal{P}_{1,0}^{\mathrm{pv}}$, $\mathcal{P}_{2,1}^{\mathrm{pv}}$ and $\mathcal{P}_{2,1}^{\mathrm{pv}}$ improves the convergence of solvers, when we precondition with block Gauss-Seidel. And if so, is is necessary to compute all off-diagonal preconditioners that form the block Gauss-Seidel formulation? For this we examine if it is possible to set one or several of $\mathcal{P}_{1,0}^{\mathrm{pv}}$, $\mathcal{P}_{2,1}^{\mathrm{pv}}$ and $\mathcal{P}_{2,1}^{\mathrm{pv}}$ to zero without a significant reduction of the convergence rate.

Block Jacobi and block Gauss-Seidel

To answer the first question, we compare a representative selection of block Jacobi preconditioners from the last section to their block Gauss-Seidel counterparts. In Figure 6.7 we compare pressure-velocity block preconditioners that use the same preconditioner types (Jacobi, ILU(3) and AMG) on both diagonal blocks. The block Jacobi versions are in the top row, the corresponding block Gauss-Seidel preconditioners are below. Figure 6.8 has the same structure, this time for the two-phase block preconditioners. Here, Uzawa is always applied to the free flow block, while the porous media block is preconditioned with either an identity matrix, ILU(3) or AMG. Both figures show the convergence for PD-GMRES(3,3,5) on the vertical flow setup.

When we compare block Jacobi to block Gauss-Seidel preconditioning, both figures show that the block Gauss-Seidel preconditioning improves the convergence rate. This is also consistent for preconditioner setups that are not shown here and can be found in Table 6.3, which summarizes convergence rates for all evaluated block preconditioner configurations on the vertical flow setup. The same table for the horizontal flow can be found in the appendix. On average, when using block Gauss-Seidel instead of block Jacobi preconditioning, the number of iterations we need to reduce the residual below a given threshold goes down by one third. Our experiments also show that if a preconditioned solver stagnates for block Jacobi preconditioning, block Gauss-Seidel preconditioning with the same diagonal configuration does not turn it into a convergent solver. This can be seen for the block preconditioners that use Jacobi on the diagonals in Figures 6.7a and 6.7d.

The best tested block Gauss-Seidel preconditioner configuration uses Uzawa and AMG preconditioning in Figure 6.8f. Here, to obtain an iterative solution so that the iterative update goes down to machine precision takes about 250 iterations. For the block Jacobi configuration in Figure 6.8c, this takes about 120 iterations more.

In this specific example, a reduction of iterations also comes with a reduced runtime in order to reduce the systems residual below a fix threshold. Table 6.4 summarizes runtimes for all tested block Gauss-Seidel preconditioner configurations, also including partial block Gauss-Seidel preconditioner that are

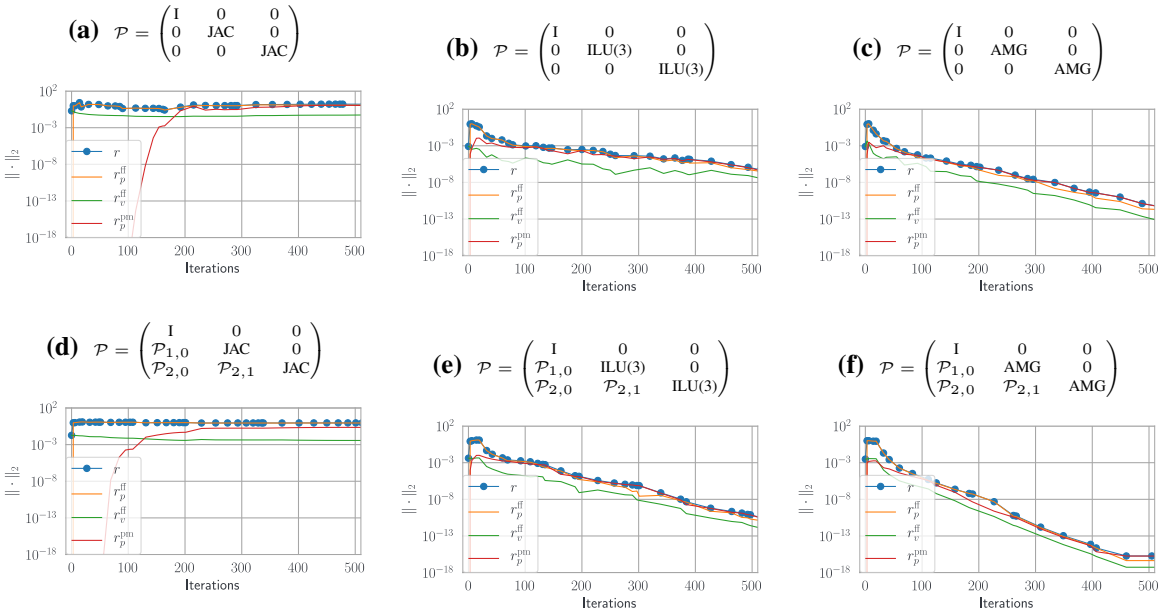


Figure 6.7: Block pressure-velocity Jacobi and Gauss-Seidel for vertical flow.

Comparison of block Jacobi preconditioners (top row) and their block Gauss-Seidel counterparts (bottom row). From left to right, Jacobi preconditioning, ILU(3) and AMG are used as preconditioners on both diagonal blocks. Convergence results are for PD-GMRES(3,3,5) on a discretization with 50×50 cells per domain.

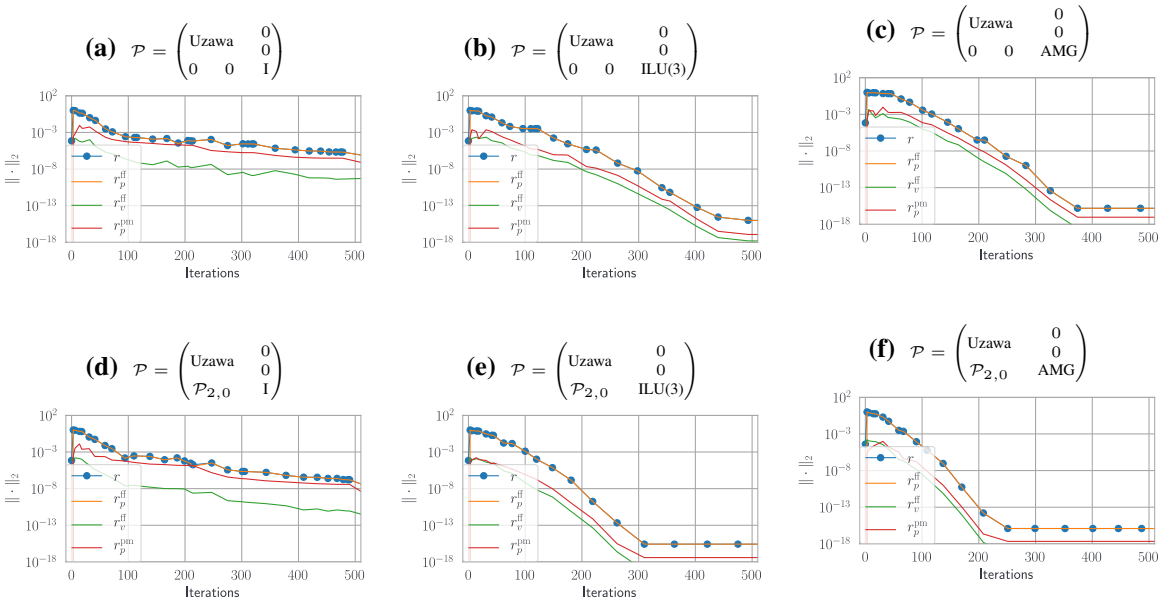


Figure 6.8: Block two-phase Jacobi and Gauss-Seidel with Uzawa on vertical flow.

The effect of off-diagonal preconditioning on two-phase block preconditioning. All convergence plots are generated with PD-GMRES(3,3,5) on a domain discretization with 50×50 cells per domain. Each column shows Uzawa preconditioning for the free flow phase combined with one further preconditioner (identity, ILU(3) and AMG) on the porous media block. The baseline performance plots for block Jacobi are shown in the upper row, results for a full block Gauss-Seidel are depicted below.

A, B	$\begin{pmatrix} \mathrm{I} & 0 & 0 \\ 0 & \mathrm{A} & 0 \\ 0 & 0 & \mathrm{B} \end{pmatrix}$	$\begin{pmatrix} \mathrm{I} & 0 & 0 \\ \mathcal{P}_{1,0} & \mathrm{A} & 0 \\ 0 & 0 & \mathrm{B} \end{pmatrix}$	$\begin{pmatrix} \mathrm{I} & 0 & 0 \\ 0 & \mathrm{A} & 0 \\ 0 & \mathcal{P}_{2,1} & \mathrm{B} \end{pmatrix}$	$\begin{pmatrix} \mathrm{I} & 0 & 0 \\ \mathcal{P}_{1,0} & \mathrm{A} & 0 \\ 0 & \mathcal{P}_{2,1} & \mathrm{B} \end{pmatrix}$	$\begin{pmatrix} \mathrm{I} & 0 & 0 \\ \mathcal{P}_{1,0} & \mathrm{A} & 0 \\ \mathcal{P}_{2,0} & \mathcal{P}_{2,1} & \mathrm{B} \end{pmatrix}$
I, I	1.00	1.00	1.00	1.00	1.00
I, JAC	1.00	1.00	1.00	1.00	1.00
I, ILU(0)	1.00	1.00	1.00	1.00	1.00
I, ILU(3)	1.00	1.00	1.00	1.00	1.00
I, AMG	1.00	1.00	1.00	1.00	1.00
JAC, I	1.00	1.00	1.00	1.00	1.00
JAC, JAC	1.00	1.00	1.00	1.00	1.00
JAC, ILU(0)	1.00	1.00	1.00	1.00	1.00
JAC, ILU(3)	1.00	1.00	1.00	1.00	1.00
JAC, AMG	1.00	1.00	1.00	1.00	1.00
ILU(0), I	0.99	0.99	0.99	0.99	0.99
ILU(0), JAC	0.99	0.99	0.99	0.99	0.99
ILU(0), ILU(0)	0.99	0.99	0.99	0.99	0.99
ILU(0), AMG	0.99	0.99	0.99	0.98	0.98
ILU(3), I	0.98	0.98	0.98	0.98	0.98
ILU(3), JAC	0.98	0.99	0.98	0.99	0.99
ILU(3), ILU(3)	0.97	0.98	0.97	0.95	0.95
ILU(3), AMG	0.96	0.98	0.96	0.93	0.93
AMG, I	0.98	0.98	0.98	0.98	0.98
AMG, JAC	0.98	0.99	0.98	0.98	0.98
AMG, ILU(0)	0.98	0.98	0.98	0.98	0.98
AMG, ILU(3)	0.97	0.98	0.96	0.95	0.95
AMG, AMG	0.95	0.97	0.93	0.92	0.92
Uzawa, I	0.97				0.95
Uzawa, JAC	0.96				0.95
Uzawa, ILU(0)	0.96				0.93
Uzawa, ILU(3)	0.93				0.90
Uzawa, AMG	0.92				0.86

Table 6.3: Convergence rates for block Gauss-Seidel configurations on vertical flow.

Convergence rate calculated over the first 400 iterations for different preconditioner configurations on the diagonal (per row) and all combinations of partial block Gauss-Seidel preconditioners (columns). Configurations with Uzawa use 200 iterations for the convergence rate. All domains were discretized with 50×50 cells.

discussed below. Like the convergence tests the runtime tests are performed on 50×50 cells for both domains, too. Also, the solvers are iterated to a residual bound of 5×10^{-10} for the horizontal flow and to 5×10^{-8} for the vertical flow, which is approximately the residual that is obtained if UMFPACK is used to solve the linear system to allow for a fair runtime comparison. The measured runtime includes the setup time for the preconditioners and the time to reduce the residual. The results for the horizontal flow can be found in the appendix in Table A.2.

From the convergence Table 6.3 we see that the two-phase block Gauss-Seidel preconditioner with Uzawa always has better convergence rates than its block Jacobi counterpart. However, this does not translate directly to the runtime results. There, we generally observe increased runtimes for the block Gauss-Seidel preconditioners that solve the problem in under two second. This is due to the increased amount of computations per iteration, which are required for preconditioning the off-diagonal blocks. The very high runtimes for ILU(0) and ILU(3) are caused by their long setup phases, which are required to compute the incomplete LU decomposition. The interesting question that is answered in Section 6.2 is whether a longer setup phase can pay off for larger problem sizes, where a faster convergence per iteration may amortize the longer setup.

A, B	$\begin{pmatrix} \mathrm{I} & 0 & 0 \\ 0 & \mathrm{A} & 0 \\ 0 & 0 & \mathrm{B} \end{pmatrix}$	$\begin{pmatrix} \mathrm{I} & 0 & 0 \\ \mathcal{P}_{1,0} & \mathrm{A} & 0 \\ 0 & 0 & \mathrm{B} \end{pmatrix}$	$\begin{pmatrix} \mathrm{I} & 0 & 0 \\ 0 & \mathrm{A} & 0 \\ 0 & \mathcal{P}_{2,1} & \mathrm{B} \end{pmatrix}$	$\begin{pmatrix} \mathrm{I} & 0 & 0 \\ \mathcal{P}_{1,0} & \mathrm{A} & 0 \\ 0 & \mathcal{P}_{2,1} & \mathrm{B} \end{pmatrix}$	$\begin{pmatrix} \mathrm{I} & 0 & 0 \\ \mathcal{P}_{1,0} & \mathrm{A} & 0 \\ \mathcal{P}_{2,0} & \mathcal{P}_{2,1} & \mathrm{B} \end{pmatrix}$
I, I	–	–	–	–	–
I, JAC	–	–	–	–	–
I, ILU(0)	–	109.54	–	91.96	115.04
I, ILU(3)	–	48.90	–	51.26	62.24
I, AMG	–	43.62	21.60	43.20	59.30
JAC, I	71.44	30.54	57.04	20.54	28.93
JAC, JAC	–	60.30	120.36	64.10	73.15
JAC, ILU(0)	24.77	16.22	28.41	14.00	18.41
JAC, ILU(3)	30.15	13.85	22.08	7.14	6.16
JAC, AMG	22.23	20.08	19.87	5.15	9.35
ILU(0), I	16.59	4.56	12.91	2.94	3.32
ILU(0), JAC	14.58	6.97	11.39	5.80	5.24
ILU(0), ILU(0)	5.12	3.07	8.36	2.37	3.57
ILU(0), AMG	1.16	1.90	2.33	1.12	1.09
ILU(3), I	35.47	41.27	46.03	52.65	64.89
ILU(3), JAC	50.38	60.16	83.96	69.24	102.21
ILU(3), ILU(3)	21.36	25.62	26.22	23.81	25.57
ILU(3), AMG	18.88	28.16	23.47	20.90	22.71
AMG, I	2.60	4.27	3.91	4.39	4.85
AMG, JAC	4.87	6.76	6.58	6.55	9.44
AMG, ILU(0)	2.11	2.81	2.21	2.36	2.84
AMG, ILU(3)	0.89	1.57	0.95	1.06	1.16
AMG, AMG	0.36	1.14	0.52	0.66	0.57
Uzawa, I	0.95				1.05
Uzawa, JAC	0.74				1.27
Uzawa, ILU(0)	0.52				0.66
Uzawa, ILU(3)	0.49				0.64
Uzawa, AMG	0.47				0.54

Table 6.4: Runtimes for block Gauss-Seidel configurations on vertical flow.

Runtimes in seconds for different block Gauss-Seidel preconditioner configurations on a single core of an AMD EPYC 7551P with 2.0 GHz. Solver times include setup time and time to reduce the residual below 5^{-8} with PD-GMRES(3,3,5) on a 50×50 cell discretization for free flow and porous media domain. Cells marked with "–" exceeded a maximal time of 120 seconds to achieve convergence.

Partial pressure-velocity block Gauss-Seidel

We have seen that block Gauss-Seidel preconditioning improves the solver convergence in comparison to applying preconditioning to the block diagonals, but also leads to higher runtimes on a 50×50 cell discretization. For a two-phase block Gauss-Seidel preconditioner we have to compute one off-diagonal preconditioner, while its pressure-velocity analogue requires three off-diagonal preconditioners, $\mathcal{P}^{\mathrm{pv}}_{1,0}$, $\mathcal{P}^{\mathrm{pv}}_{2,1}$ and $\mathcal{P}^{\mathrm{pv}}_{2,1}$. Clearly, the total number of off-diagonal preconditioners is not a useful metric because the underlying matrices and preconditioner applications have smaller dimensions for the pressure-velocity case. However, with only one off-diagonal preconditioner for two-phase Gauss-Seidel, there is not much room to reduce the number of off-diagonal preconditioners that are used. If we leave the only one out, we end with a two-phase Jacobi preconditioner. Therefore, we are left with discussing the partial options for the pressure-velocity Gauss-Seidel case. Here, we can set some of the off-diagonal preconditioners to zero and explore variations that lie between a block Jacobi and block Gauss-Seidel.

In Figure 6.9 we compare the solver convergence with block Jacobi, partial Gauss-Seidel and Gauss-Seidel preconditioning. All of the block preconditioners use ILU(3) and AMG on the diagonal blocks. We choose this example because it demonstrates the general trends and refer to Table 6.3 for a com-

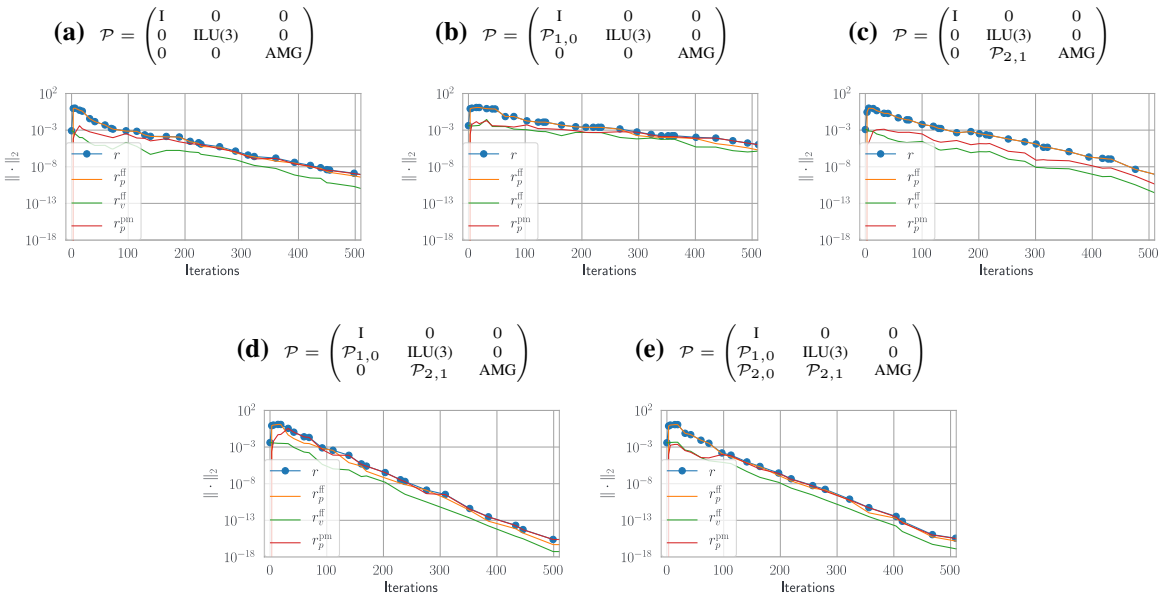


Figure 6.9: (Partial) block Gauss-Seidel with ILU(3) and AMG preconditioning, vertical flow.

PD-GMRES(3,3,5) residual reduction on a 50×50 cell discretization, preconditioned with different block Gauss-Seidel variations. All variations use ILU(3) for the free flow velocity block and AMG for porous media pressure block. The baseline is the corresponding block Jacobi preconditioner in Figure 6.9a. Each depicted preconditioner uses a different combination of included off-diagonal blocks. Using all off-diagonal blocks yields the full block Gauss-Seidel preconditioner in Figure 6.9e.

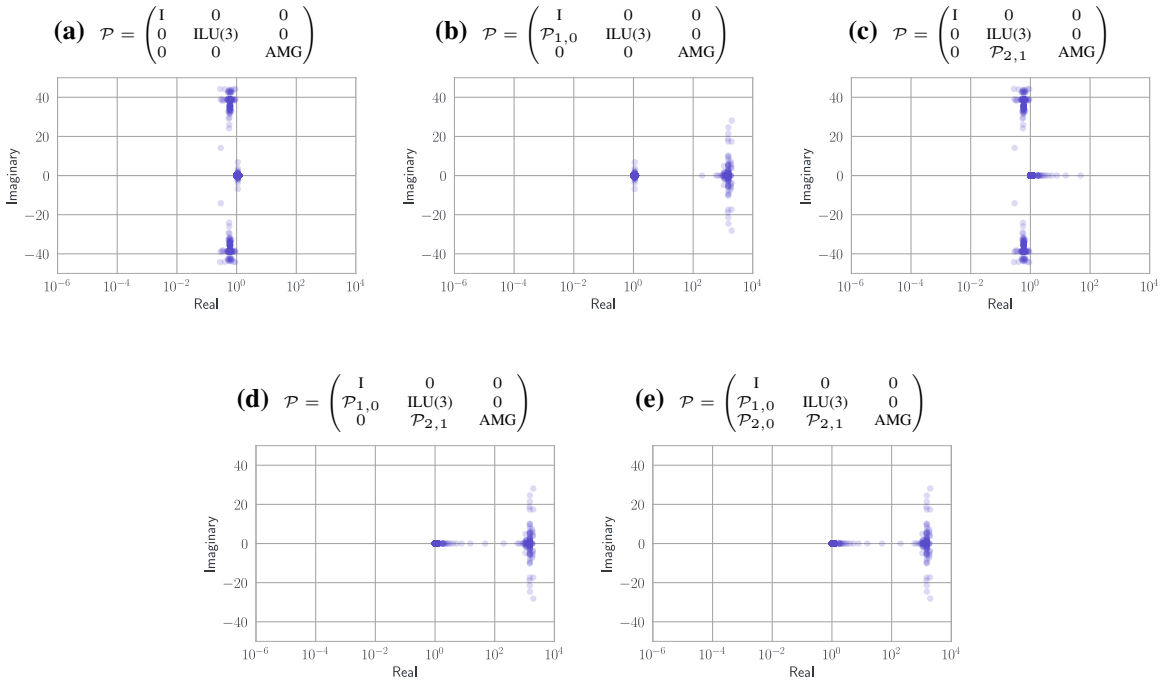


Figure 6.10: Spectrum visualization for Figure 6.9.

Eigenvalue plot for the preconditioned vertical flow stiffness matrix, the free flow and porous media domains are both discretized with 20×20 cells. The applied preconditioners correspond to the ones in Figure 6.9. A log scale is applied to the real axis.

plete list of convergence rates for block Jacobi, block Gauss-Seidel and all corresponding partial block preconditioners. From the top left to bottom right, the used preconditioners are a block Jacobi in Figure 6.9a, the partial block Gauss-Seidel preconditioners with $\mathcal{P}_{1,0}^{\text{pv}} \neq 0$ in Figure 6.9b, with $\mathcal{P}_{2,1}^{\text{pv}} \neq 0$ in Figure 6.9c, with $\mathcal{P}_{1,0}^{\text{pv}} \neq 0$ and $\mathcal{P}_{2,1}^{\text{pv}} \neq 0$ in Figure 6.9d and the full block Gauss-Seidel preconditioner in Figure 6.9e. In the same order, Figure 6.10 visualizes the spectra for the preconditioned matrices from the vertical flow setup with 20×20 instead of 50×50 cells.

In the convergence plot we see that adding $\mathcal{P}_{2,1}^{\text{pv}}$ to the block Jacobi preconditioner in Figure 6.9c slightly improves the rate of convergence, while adding $\mathcal{P}_{1,0}^{\text{pv}}$ in Figure 6.9b worsens it. When we include the spectra for those preconditioners into our analysis, we observe that using $\mathcal{P}_{1,0}^{\text{pv}}$ in Figure 6.10b moves the two clusters of eigenvalues above and below the real axis closer to it, but shifted about three orders of magnitude away from one. In contrast, applying $\mathcal{P}_{2,1}^{\text{pv}}$ in Figure 6.10c stretches the eigenvalue cluster on the real axis away from zero, compared to the baseline block Jacobi preconditioner in Figure 6.10a.

If we now turn to the second row of Figures 6.9 and 6.10, we observe an interesting effect. Judging by the convergence rate, preconditioning with a partial block Gauss-Seidel that uses $\mathcal{P}_{1,0}^{\text{pv}}$ and $\mathcal{P}_{2,1}^{\text{pv}}$ in Figure 6.9d is not worse than applying the full block Gauss-Seidel in Figure 6.9e. This result is fortunate from a computational point of view, because applying $\mathcal{P}_{2,0}^{\text{pv}}$ requires more numerical operations than using $\mathcal{P}_{1,0}^{\text{pv}}$ or $\mathcal{P}_{2,1}^{\text{pv}}$. To apply $\mathcal{P}_{2,0}^{\text{pv}}$ means we must use all three diagonal preconditioners and multiply them with two off-diagonal block matrices, which is the most expensive off-diagonal preconditioner computation. If we can leave out $\mathcal{P}_{2,0}^{\text{pv}}$ and obtain almost the same convergence rate as for a full block Gauss-Seidel, this gives us comparable solver performance with a less costly preconditioner.

Also, the spectra for these preconditioners in Figures 6.10d and 6.10e are visually very similar. This spectral similarity further supports the observation that setting $\mathcal{P}_{2,0}^{\text{pv}}$ to zero does not significantly worsen the condition of the preconditioned matrix compared to applying a block Gauss-Seidel preconditioner. Another point we can observe from the spectrum is that using $\mathcal{P}_{1,0}^{\text{pv}}$ and $\mathcal{P}_{2,1}^{\text{pv}}$ in Figure 6.10d indeed performs both changes to the spectrum that can be observed for separately applying $\mathcal{P}_{1,0}^{\text{pv}}$ in Figure 6.10b and $\mathcal{P}_{2,1}^{\text{pv}}$ in Figure6.10c.

If we consider the runtimes in Table 6.4, for the best performing preconditioner setups certain partial block Gauss-Seidel configurations reduce the runtime compared to a full block Gauss-Seidel preconditioning. Still, block Jacobi preconditioning is mostly faster than any partial block Gauss-Seidel preconditioner configuration for those preconditioners that solve the problem in under two seconds. For configurations that use a Jacobi or ILU(0) preconditioner on the free flow velocity block, using $\mathcal{P}_{1,0}^{\text{pv}}$ and $\mathcal{P}_{2,1}^{\text{pv}}$ indeed improves the runtime compared to the block Jacobi setup. Due to their generally long runtimes however, most of them are not recommended choices.

Summing up our findings, we have seen that applying a block Gauss-Seidel preconditioner to the stiffness matrix of our Stokes-Darcy system indeed leads to a faster solver convergence compared to applying a block Jacobi preconditioner. However, if the combination of preconditioners on the diagonal blocks did not yield a good convergence in a block Jacobi setting, using block Gauss-Seidel will not be as effective as exchanging the diagonal preconditioners by more suitable ones. Additionally, the runtimes for block Gauss-Seidel preconditioners are usually higher than for their block Jacobi analogues. In the case of the pressure-velocity block Gauss-Seidel, we see that setting the off-diagonal preconditioner $\mathcal{P}_{2,0}^{\text{pv}}$ to zero leads to a preconditioned matrix with very similar spectral properties to a matrix after a full block Gauss-Seidel preconditioning. This is a very fortunate finding, as computing $\mathcal{P}_{2,0}^{\text{pv}}$ requires more operations than computing $\mathcal{P}_{1,0}^{\text{pv}}$ or $\mathcal{P}_{2,1}^{\text{pv}}$ alone. It can have a positive effect on the runtime to additionally use those two off-diagonal preconditioners, however the effect is strongest for configurations that have long runtimes. For the best performing configurations, the additional effort to compute the off-diagonal preconditioners does lead to higher runtimes compared to the block Jacobi preconditioner setup.

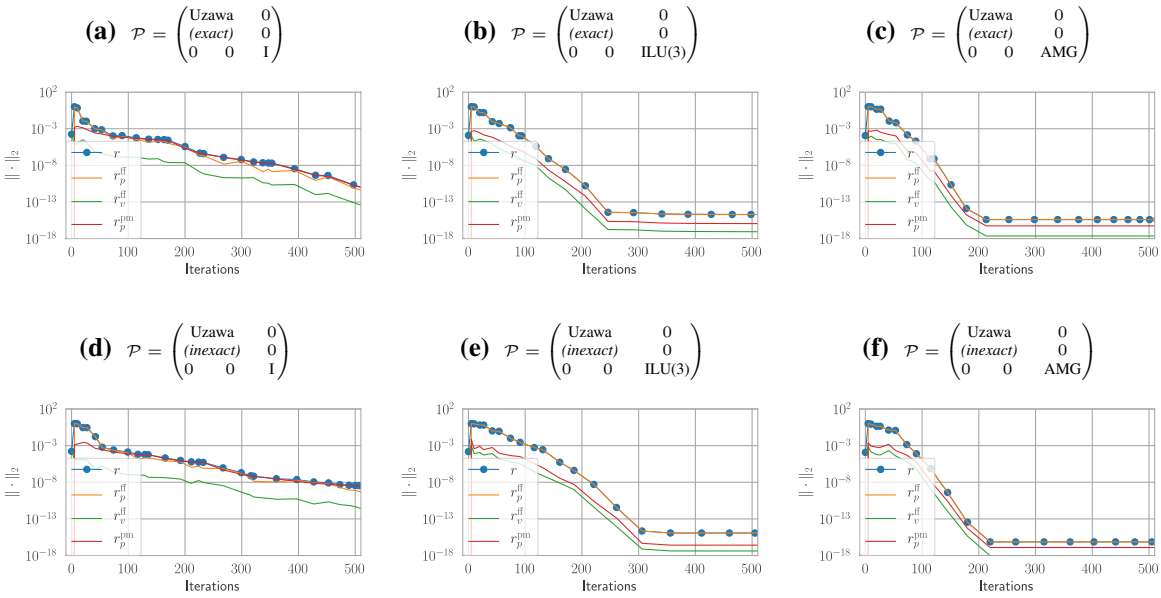


Figure 6.11: Exact and inexact Uzawa for block Jacobi preconditioning on vertical flow.

Two-phase block Jacobi preconditioner with Uzawa preconditioner for free flow block and identity, ILU(3) or AMG preconditioners for the porous media block. We compare PD-GMRES(3,3,5) on a 50×50 cell discretization for an "exact" inversion of V with UMFPACK (top row) and use an AMG for the inexact inversion (bottom row).

6.1.5 Exact and inexact Uzawa preconditioning

After the introduction of the Uzawa iteration in Section 3.2.3, we stated a result by Elman and Golub [25] regarding the behavior of the inexact Uzawa iteration for symmetric matrices. In this case, the inexact Uzawa iteration approaches the exact Uzawa iteration as the approximate inversion of V comes closer to the true inverse. Now, we test if we can qualitatively confirm this result in experiments with our nonsymmetric matrix.

We regard an Uzawa iteration as "exact", if the inversion of V is performed with UMFPACK. For the inexact Uzawa iteration, V^{-1} is approximated with one V-cycle of the ISTL AMG method based on non-smooth aggregation that uses the following set of parameters: one pre- and post-smoothing step of a Gauss-Seidel iteration with $\omega = 1$, a maximum of 15 grids and "coarsentarget" $= 2000$.

Figure 6.11 compares two-phase block Jacobi configurations with the exact or inexact Uzawa iteration for the free flow block and one of the options identity, ILU(3) or AMG as preconditioner for the porous media block. For all three porous media preconditioners we see an improved convergence rate for the exact Uzawa iteration. The difference between the exact and inexact iteration is smallest for the AMG configuration, which is the best performing setup. This shows that increasing the exactness of the Uzawa iteration indeed improves the solver convergence in terms of iterations.

In a next experiment, we increase the approximation quality for the inexact Uzawa iteration and observe the effect on the residual, that is obtained after a fixed number of iterations. Therefore, we consider GMRES(20) in order to get the residual norm after 100 iterations for every test and precondition it with $\mathcal{P}_{\text{BJ}}^{\text{tp}}(\text{Uzawa}, \text{I})$. In the current implementation of the Uzawa iteration in DUMU$^{\text{X}}$, the easiest way to increase the AMG method's approximation quality is to increase the number of pre- and post-smoothing steps. The standard setup uses 1 step for pre- and post-smoothing. We now consider the values 1, 5, 10, 50, 100 and 500, and compare the residual after 100 iteration as well as the runtime for those 100 iterations in Table 6.5 against the inversion with UMFPACK. The runtimes are measured on a single core of an AMD EPYC 7551P with 2.0 GHz.

Pre- and post-smoothing	UMFPACK	1	5	10	50	100	500
Residual ($\times 10^{-5}$)	1.03	8.46	2.54	2.11	0.87	1.89	1.33
Time (s)	5.48	2.33	6.05	10.51	46.03	90.61	447.25

Table 6.5: Runtimes and residuals for increasingly exact Uzawa on vertical flow.

For $\mathcal{P}_{\mathrm{BJ}}^{\mathrm{tp}}(\mathrm{Uzawa}, \mathrm{I})$ we compare the runtime and residual after 100 iterations of GMRES(20) on a 160×160 cell discretization for both vertical flow domains. We regard the inversion of V with UMFPACK as "exact" inversion and use AMG for the inexact inversion. To increase the accuracy of AMG, we gradually raise the number of pre- and post-smoothing steps. All runtimes are measured on a single core of an AMD EPYC 7551P with 2.0 GHz.

We can see that after 100 iterations the residual gradually approaches the exact Uzawa residual with an increasing number of pre- and post-smoothing steps. While more smoothing steps increase the AMG's approximation quality, it is certainly not the most efficient way to produce better approximations. The time for performing 100 iteration scales almost linear with the number of smoothing steps. In this setting we also see a trade off between the residual reduction and runtime when we compare the exact Uzawa iteration to the AMG iteration with one pre- and post-smoothing step. While the inversion with UMFPACK yields a better residual reduction, it takes almost twice as long as computing a single AMG iteration as inversion. Therefore, the best choice depends on the problem and how much accuracy for the solution is required.

6.2 Solver runtime comparison

In the last section we observed that the convergence rate of a preconditioned solver is a rough indicator for the runtime, and measured runtimes only on a 160×160 cell discretization on each flow domain. Now we systematically investigate the influence of the matrix size on the runtimes of block preconditioned PD-GMRES(3,3,5) solvers and compare it against the sparse direct solver provided by UMFPACK.

In Section 6.2.1 we present runtime scaling results for increasing matrix sizes and different block preconditioning techniques. The selected block preconditioners are chosen based on the runtime results on the 160×160 cell domain discretization from the last section. Our main goal is to measure the performance in comparison to UMFPACK. Therefore, we usually solve up to a residual of the same order of UMFPACK and investigate if the runtime trends that were observed on the 160×160 cell domain generalize with regard to block Jacobi and block Gauss-Seidel preconditioning.

Section 6.2.2 shifts the focus away from finding the most runtime efficient preconditioner configuration. We rather consider an aspect that was not deeply discussed within this thesis so far, which is the fact that iterative solvers allow us to set a target residual for the approximate solution. In contrast to (sparse) direct solvers it is therefore possible to adapt the computational resources to the required precision of the approximate solution. In other words, if the approximate solution does not require to be very exact for hypothetical consecutive operations, investing a lot of resources to solve up to a great accuracy might be a waste of time and energy. Hence we consider different target residuals for a specific block preconditioner and observe how the runtimes develop based on the different stopping criteria for an increasing matrix size.

6.2.1 Runtime scaling for selected block preconditioner

In the following we test the runtime scaling of different block preconditioned PD-GMRES(3,3,5,-3,5) solvers if they are iterated up to the residual precision of the solution produced through UMFPACK. Because UMFPACK reaches different residuals for the problem variants horizontal and vertical flow, we iterate to a residual of $5 \cdot 10^{-10}$ for all horizontal flow test cases and to $5 \cdot 10^{-5}$ for the vertical flow. For

all tested problem sizes the UMFPACK residual is between $1 \cdot 10^{-10}$ and $3 \cdot 10^{-10}$ for horizontal flow setups and from $1 \cdot 10^{-5}$ to $4 \cdot 10^{-5}$ for the vertical flow case.

The problem setup and the setups for the preconditioners are as described in the first two paragraphs of Section 6.1, but now with a differing number of cells in the free flow and porous media domain. The relation between the domain discretization and the degrees of freedom is given in Equation (2.16) in Chapter 2.2. We use it to construct discretizations that lead to logarithmically growing degrees of freedom.

All performance tests within this chapter are run on discretizations that use one of $\{5^2, 16^2, 50^2, 160^2, 500^2, 1600^2\}$ cells for both domains, which results in 110, 1065, 10100, 102720, 1001000 and 10243200 degrees of freedom and matrices in $\mathbb{R}^{\mathrm{DoF}\times\mathrm{DoF}}$. The measured runtimes include the time to set up the preconditioners and iterative solvers, and to reduce the residual to the given target. The time to assemble the stiffness matrix is not included in the measurements. For the runtime measurements we use a single core of an AMD EPYC 7551P with 2.0 GHz. As in the previous section, we omit the parameters $\alpha = -3$ and $\beta = 5$ for PD-GMRES as they are the same throughout this evaluation. Therefore, we refer to a PD-GMRES configuration in the shortened form PD-GMRES($\mathrm{m_{init}}$, $\mathrm{m_{min}}$, $\mathrm{m_{step}}$).

General runtime comparison

Figure 6.12 shows the behavior of a selection of block preconditioned solvers compared to UMFPACK for the horizontal flow scenario. Figure 6.13 illustrates the vertical flow. For an overview of all tested configurations we refer to Table 6.6 and Table 6.7. The configurations in both tables are identical, with the exception that the AMG preconditioner is never used on the porous media pressure block in the horizontal flow scenario.

In terms of runtime, the preconditioned solvers in both figures are either comparable or better than our baseline UMFPACK, which is marked in yellow. From the Tables 6.6 and 6.7 we also see that the majority of tested block preconditioned solvers are faster than UMFPACK, independent of the problem size. Especially a $\mathcal{P}^{\mathrm{pv}}_{\mathrm{BJ}}(\mathrm{AMG}, \mathrm{ILU(0)})$-preconditioned PD-GMRES(3,3,5) solver appears to be an efficient choice for problems with many degrees of freedom for both flow setups. Its performance is shown in Figures 6.12 and 6.13. It is comparably fast or faster than UMFPACK for small to medium problem sizes and for large problems it reaches UMFPACK's residual nine times faster than the sparse direct solver.

On the vertical flow setup with a block Jacobi preconditioner and a fixed porous media pressure preconditioner , we see that using AMG as preconditioner on the free flow block instead of Uzawa generally leads to lower runtimes for large problem sizes. This indicates that using an efficient preconditioner like AMG on the free flow velocity block instead of Uzawa on the full free flow system is effective enough to assure good convergence while resulting in a lower runtime per iteration. Therefore, the block preconditioner variations of the form $\mathcal{P}^{\mathrm{pv}}_{*}(\mathrm{AMG}, *)$ can perform more iterations in a given time and compensate for their slower convergence rates compared to Uzawa based preconditioners, which we measured earlier.

Another effect we see is that the use of ILU(1) of ILU(3) preconditioners often leads to increased runtimes, especially for larger problem sizes. It seems that also on larger problems, the better convergence rate that is achieved by ILU(1) or ILU(3) in comparison to ILU(0) does not make up for the increased time to compute the incomplete factorization on large matrices.

For pressure velocity block Jacobi preconditioning on small problem sizes, more complex preconditioners on the porous media pressure block like AMG, ILU(1) or ILU(3) often lead to shorter computation times, while Jacobi or ILU(0) preconditioning can take longer to converge on those problems, especially in the vertical flow scenario. This ordering by runtime, however, is often inverted for large problem sizes.

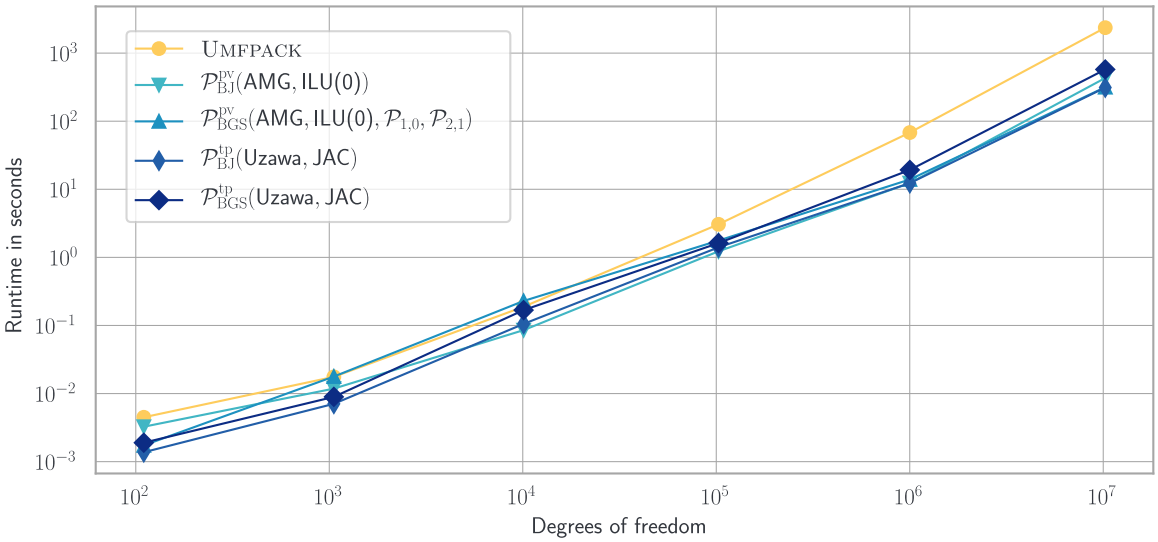


Figure 6.12: Runtime scaling for selected block preconditioners on the horizontal fl setupow.

Runtime comparison for a residual reduction to $5 \cdot 10^{-10}$ for different block preconditioned PD-GMRES(3,3,5) variations on domains with increasingly fine discretizations. The domain sizes of $\{5^2, 16^2, 50^2, 160^2, 500^2, 1600^2\}$ lead to roughly 10^2-10^7 degrees of freedom. Both axis are in log scale. The residual bound was chosen to have the same order of magnitude as the residual produced by UMFPACK.

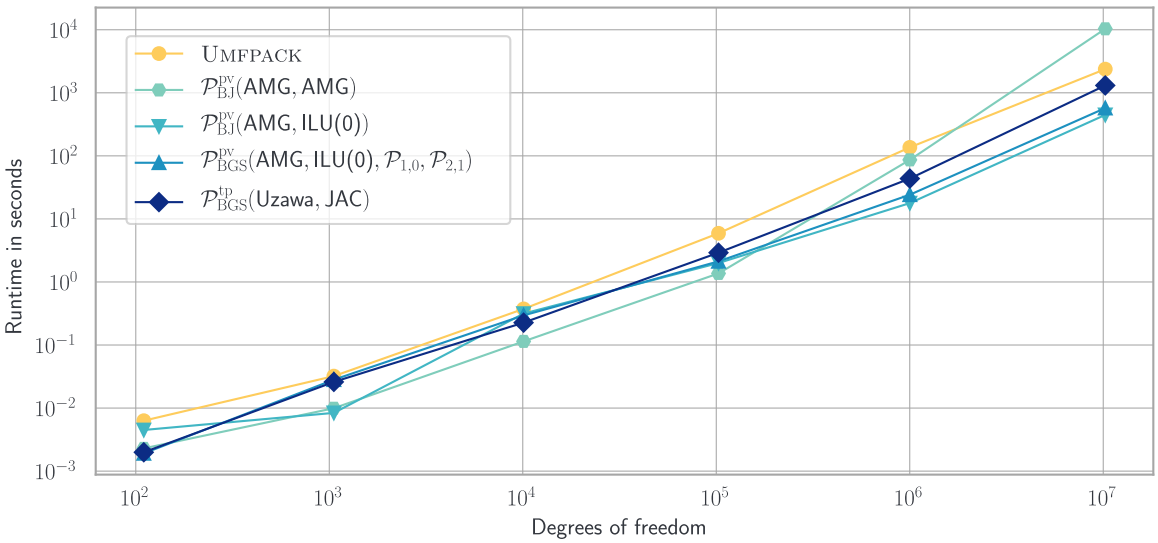


Figure 6.13: Runtime scaling for selected block preconditioners on the vertical flow setup.

Same setup as for Figure 6.12 and with a targeted residual reduction of $5 \cdot 10^{-5}$, which is slightly below the residual produced by UMFPACK but within the same order of magnitude. We present a slightly varied selection of block preconditioners. Compared to Figure 6.12, $\mathcal{P}_{\mathrm{BJ}}^{\mathrm{pv}}(\mathrm{AMG}, \mathrm{AMG})$ is added to the test while $\mathcal{P}_{\mathrm{BJ}}^{\mathrm{tp}}(\mathrm{Uzawa}, \mathrm{JAC})$ is not shown, because $\mathcal{P}_{\mathrm{BJ}}^{\mathrm{pv}}(\mathrm{AMG}, \mathrm{AMG})$ performs better on this problem. Preconditioner configurations that appear in both plots have the same line colors and markers for both plots.

	110	**1056**	**10100**	**102720**	**1001000**	**10243200**
UMFPACK	$4.49 \cdot 10^{-3}$	$1.76 \cdot 10^{-2}$	$1.89 \cdot 10^{-1}$	$3.06 \cdot 10^{0}$	$6.81 \cdot 10^{1}$	$2.36 \cdot 10^{3}$
$\mathcal{P}^{\text{pv}}_{\text{BJ}}$(AMG, I)	$1.68 \cdot 10^{-3}$	$1.20 \cdot 10^{-2}$	$1.27 \cdot 10^{-1}$	$1.67 \cdot 10^{0}$	$1.85 \cdot 10^{1}$	$5.40 \cdot 10^{2}$
$\mathcal{P}^{\text{pv}}_{\text{BJ}}$(AMG, JAC)	$1.80 \cdot 10^{-3}$	$1.41 \cdot 10^{-2}$	$2.39 \cdot 10^{-1}$	$2.71 \cdot 10^{0}$	$2.07 \cdot 10^{1}$	$2.14 \cdot 10^{2}$
$\mathcal{P}^{\text{pv}}_{\text{BJ}}$(AMG, ILU(0))	$3.26 \cdot 10^{-3}$	$1.18 \cdot 10^{-2}$	$8.54 \cdot 10^{-2}$	$1.23 \cdot 10^{0}$	$1.24 \cdot 10^{1}$	$4.31 \cdot 10^{2}$
$\mathcal{P}^{\text{pv}}_{\text{BJ}}$(AMG, ILU(1))	$1.85 \cdot 10^{-3}$	$1.22 \cdot 10^{-2}$	$8.52 \cdot 10^{-2}$	$1.23 \cdot 10^{0}$	$1.26 \cdot 10^{1}$	$6.40 \cdot 10^{2}$
$\mathcal{P}^{\text{pv}}_{\text{BJ}}$(AMG, ILU(3))	$2.66 \cdot 10^{-3}$	$2.05 \cdot 10^{-2}$	$1.38 \cdot 10^{-1}$	$4.47 \cdot 10^{0}$	$2.55 \cdot 10^{2}$	–
$\mathcal{P}^{\text{pv}}_{\text{BJ}}$(ILU(0), ILU(0))	$1.57 \cdot 10^{-3}$	$5.65 \cdot 10^{-2}$	$6.33 \cdot 10^{-1}$	$3.50 \cdot 10^{0}$	$7.17 \cdot 10^{1}$	$1.97 \cdot 10^{3}$
$\mathcal{P}^{\text{pv}}_{\text{BJ}}$(ILU(1), ILU(1))	$1.01 \cdot 10^{-3}$	$2.23 \cdot 10^{-2}$	$7.50 \cdot 10^{-1}$	$1.31 \cdot 10^{1}$	$4.61 \cdot 10^{2}$	–
$\mathcal{P}^{\text{pv}}_{\text{BGS}}$(AMG, ILU(0), $\mathcal{P}_{1,0}$)	$3.11 \cdot 10^{-3}$	$1.26 \cdot 10^{-2}$	$1.74 \cdot 10^{-1}$	$1.42 \cdot 10^{0}$	$8.92 \cdot 10^{0}$	$3.97 \cdot 10^{2}$
$\mathcal{P}^{\text{pv}}_{\text{BGS}}$(AMG, ILU(0), $\mathcal{P}_{1,0}$, $\mathcal{P}_{2,1}$)	$1.74 \cdot 10^{-3}$	$1.78 \cdot 10^{-2}$	$2.29 \cdot 10^{-1}$	$1.77 \cdot 10^{0}$	$1.39 \cdot 10^{1}$	$3.16 \cdot 10^{2}$
$\mathcal{P}^{\text{pv}}_{\text{BGS}}$(AMG, ILU(0))	$1.42 \cdot 10^{-3}$	$1.37 \cdot 10^{-2}$	$2.33 \cdot 10^{-1}$	$1.56 \cdot 10^{0}$	$1.74 \cdot 10^{1}$	$3.94 \cdot 10^{2}$
$\mathcal{P}^{\text{tp}}_{\text{BJ}}$(Uzawa, JAC)	$1.37 \cdot 10^{-3}$	$7.07 \cdot 10^{-3}$	$1.05 \cdot 10^{-1}$	$1.40 \cdot 10^{0}$	$1.23 \cdot 10^{1}$	$3.12 \cdot 10^{2}$
$\mathcal{P}^{\text{tp}}_{\text{BJ}}$(Uzawa, ILU(0))	$1.40 \cdot 10^{-3}$	$1.56 \cdot 10^{-2}$	$2.05 \cdot 10^{-1}$	$1.57 \cdot 10^{0}$	$1.15 \cdot 10^{1}$	$2.83 \cdot 10^{2}$
$\mathcal{P}^{\text{tp}}_{\text{BJ}}$(Uzawa, ILU(1))	$1.43 \cdot 10^{-3}$	$1.36 \cdot 10^{-2}$	$1.51 \cdot 10^{-1}$	$1.55 \cdot 10^{0}$	$1.61 \cdot 10^{1}$	$4.26 \cdot 10^{2}$
$\mathcal{P}^{\text{tp}}_{\text{BGS}}$(Uzawa, JAC)	$1.90 \cdot 10^{-3}$	$8.95 \cdot 10^{-3}$	$1.68 \cdot 10^{-1}$	$1.62 \cdot 10^{0}$	$1.92 \cdot 10^{1}$	$5.76 \cdot 10^{2}$

Table 6.6: Runtimes for different block preconditioner configurations on horizontal flow.

Runtimes in seconds for different block Gauss-Seidel preconditioner configurations on problems with an increasing number of degrees of freedom, from about 100 to 10 million. The measurements are taken on a single core of an AMD EPYC 7551P with 2.0 GHz. The solver times include the setup time for the preconditioner and the time to reduce the residual below 5^{-10} with PD-GMRES(3,3,5). Fields marked with "–" exceeded a maximal computation time of three hours or 1.08×10^4 seconds.

	110	**1056**	**10100**	**102720**	**1001000**	**10243200**
UMFPACK	$6.31 \cdot 10^{-3}$	$3.21 \cdot 10^{-2}$	$3.74 \cdot 10^{-1}$	$5.92 \cdot 10^{0}$	$1.36 \cdot 10^{2}$	$2.37 \cdot 10^{3}$
$\mathcal{P}^{\text{pv}}_{\text{BJ}}$(AMG, I)	$6.73 \cdot 10^{-3}$	$8.18 \cdot 10^{-3}$	$3.00 \cdot 10^{-1}$	$2.81 \cdot 10^{0}$	$3.70 \cdot 10^{1}$	$7.38 \cdot 10^{2}$
$\mathcal{P}^{\text{pv}}_{\text{BJ}}$(AMG, JAC)	$1.43 \cdot 10^{-2}$	$8.40 \cdot 10^{-3}$	$4.87 \cdot 10^{-1}$	$5.21 \cdot 10^{0}$	$3.54 \cdot 10^{1}$	$4.81 \cdot 10^{2}$
$\mathcal{P}^{\text{pv}}_{\text{BJ}}$(AMG, ILU(0))	$4.49 \cdot 10^{-3}$	$8.39 \cdot 10^{-3}$	$3.15 \cdot 10^{-1}$	$2.00 \cdot 10^{0}$	$1.78 \cdot 10^{1}$	$4.47 \cdot 10^{2}$
$\mathcal{P}^{\text{pv}}_{\text{BJ}}$(AMG, ILU(1))	$2.83 \cdot 10^{-3}$	$8.63 \cdot 10^{-3}$	$3.55 \cdot 10^{-1}$	$1.15 \cdot 10^{0}$	$2.16 \cdot 10^{1}$	$6.86 \cdot 10^{2}$
$\mathcal{P}^{\text{pv}}_{\text{BJ}}$(AMG, ILU(3))	$3.40 \cdot 10^{-3}$	$1.12 \cdot 10^{-2}$	$1.85 \cdot 10^{-1}$	$4.33 \cdot 10^{0}$	$2.57 \cdot 10^{2}$	–
$\mathcal{P}^{\text{pv}}_{\text{BJ}}$(AMG, AMG)	$2.29 \cdot 10^{-3}$	$9.94 \cdot 10^{-3}$	$1.14 \cdot 10^{-1}$	$1.37 \cdot 10^{0}$	$8.64 \cdot 10^{1}$	$1.04 \cdot 10^{4}$
$\mathcal{P}^{\text{pv}}_{\text{BJ}}$(ILU(0), ILU(0))	$6.18 \cdot 10^{-3}$	$1.67 \cdot 10^{-2}$	$2.12 \cdot 10^{-1}$	$8.61 \cdot 10^{0}$	$5.81 \cdot 10^{2}$	–
$\mathcal{P}^{\text{pv}}_{\text{BJ}}$(ILU(1), ILU(1))	$2.85 \cdot 10^{-3}$	$5.76 \cdot 10^{-3}$	$1.68 \cdot 10^{-1}$	$1.69 \cdot 10^{1}$	$7.22 \cdot 10^{3}$	–
$\mathcal{P}^{\text{pv}}_{\text{BGS}}$(AMG, ILU(0), $\mathcal{P}_{1,0}$)	$1.33 \cdot 10^{-3}$	$2.51 \cdot 10^{-2}$	$3.32 \cdot 10^{-1}$	$2.57 \cdot 10^{0}$	$1.79 \cdot 10^{1}$	$4.66 \cdot 10^{2}$
$\mathcal{P}^{\text{pv}}_{\text{BGS}}$(AMG, ILU(0), $\mathcal{P}_{1,0}$, $\mathcal{P}_{2,1}$)	$1.91 \cdot 10^{-3}$	$2.82 \cdot 10^{-2}$	$2.95 \cdot 10^{-1}$	$2.12 \cdot 10^{0}$	$2.42 \cdot 10^{1}$	$5.72 \cdot 10^{2}$
$\mathcal{P}^{\text{pv}}_{\text{BGS}}$(AMG, ILU(0))	$1.82 \cdot 10^{-3}$	$3.63 \cdot 10^{-2}$	$3.34 \cdot 10^{-1}$	$2.65 \cdot 10^{0}$	$3.05 \cdot 10^{1}$	$7.27 \cdot 10^{2}$
$\mathcal{P}^{\text{tp}}_{\text{BJ}}$(Uzawa, JAC)	$1.49 \cdot 10^{-3}$	$1.61 \cdot 10^{-2}$	$1.40 \cdot 10^{-1}$	$1.77 \cdot 10^{0}$	$2.70 \cdot 10^{1}$	$6.94 \cdot 10^{2}$
$\mathcal{P}^{\text{tp}}_{\text{BJ}}$(Uzawa, ILU(0))	$1.47 \cdot 10^{-3}$	$1.99 \cdot 10^{-2}$	$2.67 \cdot 10^{-1}$	$1.52 \cdot 10^{0}$	$1.61 \cdot 10^{1}$	$7.31 \cdot 10^{2}$
$\mathcal{P}^{\text{tp}}_{\text{BJ}}$(Uzawa, ILU(1))	$1.32 \cdot 10^{-1}$	$2.67 \cdot 10^{0}$	$3.71 \cdot 10^{1}$	$2.16 \cdot 10^{2}$	$2.98 \cdot 10^{3}$	–
$\mathcal{P}^{\text{tp}}_{\text{BJ}}$(Uzawa, AMG)	$1.62 \cdot 10^{-3}$	$1.93 \cdot 10^{-2}$	$4.32 \cdot 10^{-1}$	$9.37 \cdot 10^{0}$	$2.20 \cdot 10^{2}$	$5.88 \cdot 10^{3}$
$\mathcal{P}^{\text{tp}}_{\text{BGS}}$(Uzawa, JAC)	$1.99 \cdot 10^{-3}$	$2.60 \cdot 10^{-2}$	$2.26 \cdot 10^{-1}$	$2.91 \cdot 10^{0}$	$4.36 \cdot 10^{1}$	$1.31 \cdot 10^{3}$
$\mathcal{P}^{\text{tp}}_{\text{BGS}}$(Uzawa, AMG)	$2.29 \cdot 10^{-3}$	$1.79 \cdot 10^{-2}$	$3.22 \cdot 10^{-1}$	$5.73 \cdot 10^{0}$	$1.56 \cdot 10^{2}$	$3.66 \cdot 10^{1}$

Table 6.7: Runtimes for different block preconditioner configurations on vertical flow.

Runtimes in seconds for different block Gauss-Seidel preconditioner configurations on problems with an increasing number of degrees of freedom, from about 100 to 10 million. The measurements are taken on a single core of an AMD EPYC 7551P with 2.0 GHz. The solver times include the setup time for the preconditioners and the time to reduce the residual below 5^{-5} with PD-GMRES(3,3,5). Fields marked with "–" exceeded a maximal computation time of three hours or 1.08×10^4 seconds.

On large problems, preconditioners that require less time for their assembly and fewer operations for their application – even though the convergence per iteration is slower – tend to lead to an overall reduced time. Here, the time per iteration seems to have a stronger effect on the time to convergence than a strong convergence per iteration.

We therefore note two intermediate results. First, we observe that depending on the configuration, block preconditioned solvers are not only able to compete with UMFPACK in terms of runtime on a number of scales. In fact they often outperform the sparse direct baseline, especially for larger matrices. Among the tested preconditioner configurations, a greater time efficiency can be observed for block Jacobi preconditioners with less computationally intense porous media pressure preconditioners, especially in the vertical flow case.

Block Jacobi vs block Gauss-Seidel preconditioner

Next, we investigate if the trend that block Gauss-Seidel preconditioning is mostly slower than preconditioning with the corresponding block Jacobi preconditioner generalizes from the 160×160 cell discretization in Section 6.1.4 to other problem sizes.

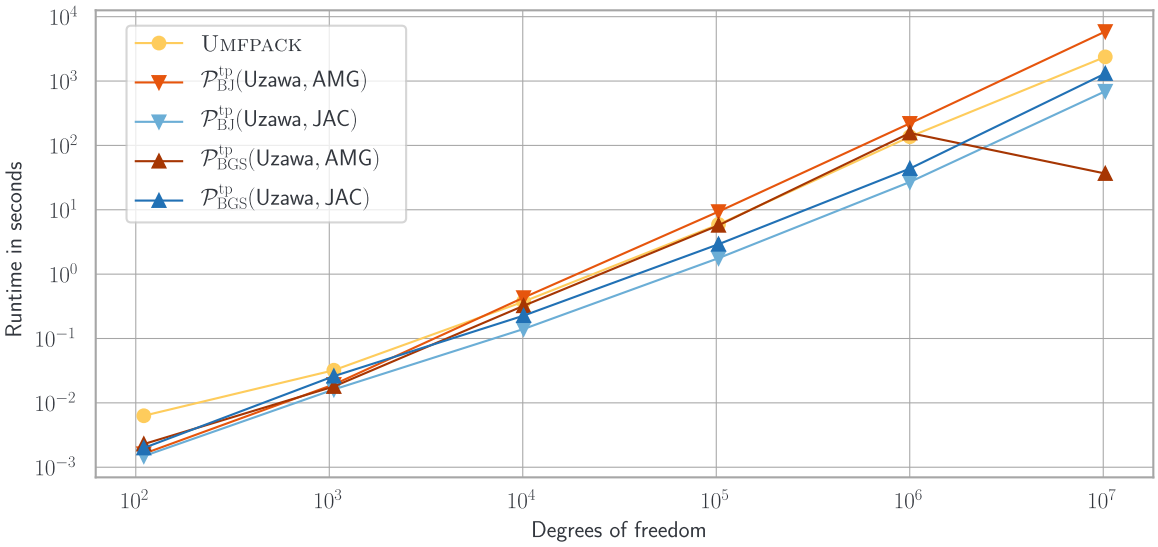


Figure 6.14: Runtime scaling for two phase block preconditioning with Uzawa, vertical flow.

Comparison of two-phase block Jacobi and block Gauss-Seidel preconditioning with Uzawa for the free flow block and AMG or Jacobi preconditioning on the porous media block. Given are times to reduce the residual below 5×10^{-5} for the different block preconditioned PD-GMRES(3,3,5) solvers and UMFPACK. The time measurement for the largest problem with $\mathcal{P}^{\text{tp}}_{\text{BJ}}$(Uzawa, AMG) did not finish within 10.8×10^{3} seconds.

Figure 6.14 shows the runtime scaling behavior for block Jacobi and block Gauss-Seidel preconditioner configurations using Uzawa preconditioning on the free flow and AMG or Jacobi preconditioning on the porous media block. In agreement with the observations on the fixed problem size in Table 6.4, block Gauss-Seidel preconditioning with Uzawa and AMG tends to be slower than block Jacobi preconditioning with the same sub-preconditioners. But for the configurations using Uzawa and Jacobi as sub-preconditioners, we observe an inverted trend. Here, the block Gauss-Seidel configuration is faster than the block Jacobi preconditioner for all problems with more than 10^3 degrees of freedom. The exceptionally low runtime of $\mathcal{P}^{\text{tp}}_{\text{BGS}}$(Uzawa, AMG) on the largest problem size is far off its general runtime scaling and regarded as outlier.

This example demonstrates that it is hard to derive general runtime trends for the two-phase block preconditioner in their versions Jacobi and Gauss-Seidel.

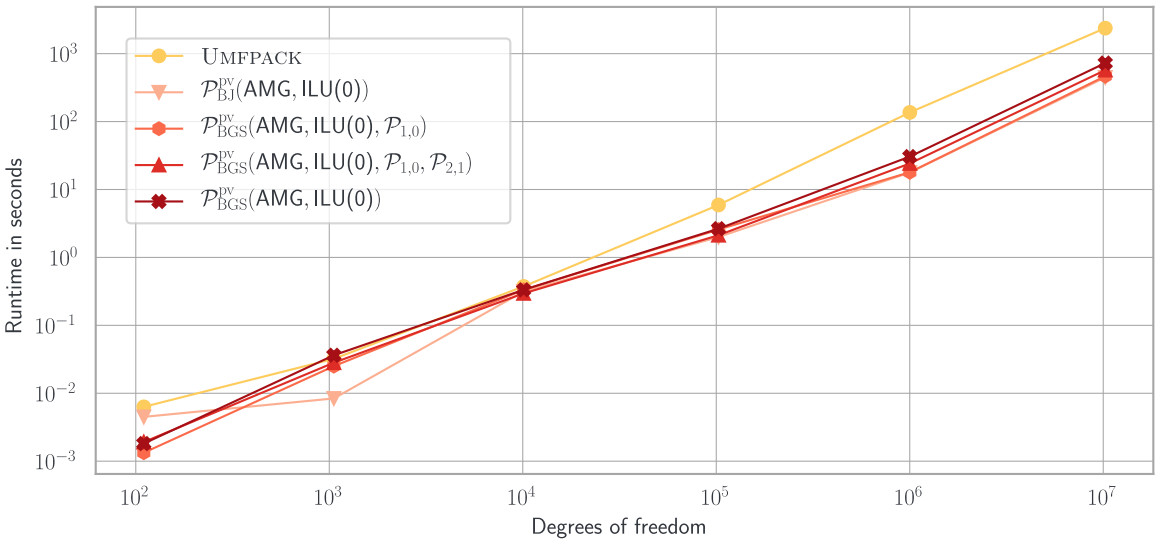


Figure 6.15: Runtime scaling for partial block Gauss-Seidel preconditioning, vertical flow.

Comparison of (partial) pressure-velocity block Gauss-Seidel preconditioning for an AMG preconditioner on the free flow velocity and an ILU(0) preconditioner on the porous media pressure block. The iterative solvers reduce the residual below a bound of 5×10^{-5}.

Next, we investigate the runtime scaling of (partial) block Gauss-Seidel preconditioning for pressure-velocity block preconditioners. Figure 6.15 shows all variations of a pressure-velocity block preconditioner with AMG on the free flow velocity block and ILU(0) for the porous media pressure block. A companion illustration for the horizontal flow can be found in Figure A.1 in the appendix. In Section 6.1.4 on a 50×50 cell discretization, we observed that any additional off-diagonal preconditioner yields an increased runtime in comparison to the block Jacobi preconditioner $\mathcal{P}_{\mathrm{BJ}}^{\mathrm{pv}}(\mathrm{AMG}, \mathrm{ILU(0)})$. This trend can generally be confirmed for multiple problem sizes.

We also observe that adding $\mathcal{P}_{1,0}$ to the block Jacobi preconditioner increases the runtime, but compared to adding two or all off-diagonal preconditioners the runtime difference to a pure block Jacobi preconditioner is smallest. In the horizontal flow case, which is shown in the appendix in Figure A.1, a crossover can be observed for the largest problem size, where the off-diagonal preconditioned solvers are faster than the block Jacobi preconditioner. Another point that confirms our observations from the previous convergence study is that in the shown vertical flow case, the runtimes for a full block Gauss-Seidel preconditioner are worse than those for using a block Jacobi preconditioner with $\mathcal{P}_{1,0}$ and $\mathcal{P}_{2,1}$.

It is hard to draw a rigid conclusion from our evaluation results with respect to the question if one should use block Jacobi, block Gauss-Seidel or partial block Gauss-Seidel preconditioning. Like often, it depends on the problem. As general guideline for large problem sizes we recommend to try block Jacobi preconditioning first and test off-diagonal preconditioners if the involved sub-preconditioners have modest application costs, such as Jacobi or ILU(0) preconditioners.

6.2.2 Runtime scaling to target residual

As last point of this section, we consider a property of iterative methods that was not part of the discussion in this thesis so far: their capability to iterate to a selected residual accuracy. While UMFPACK performs a decomposition of the matrix, which yields a single approximate solution after inversion and application

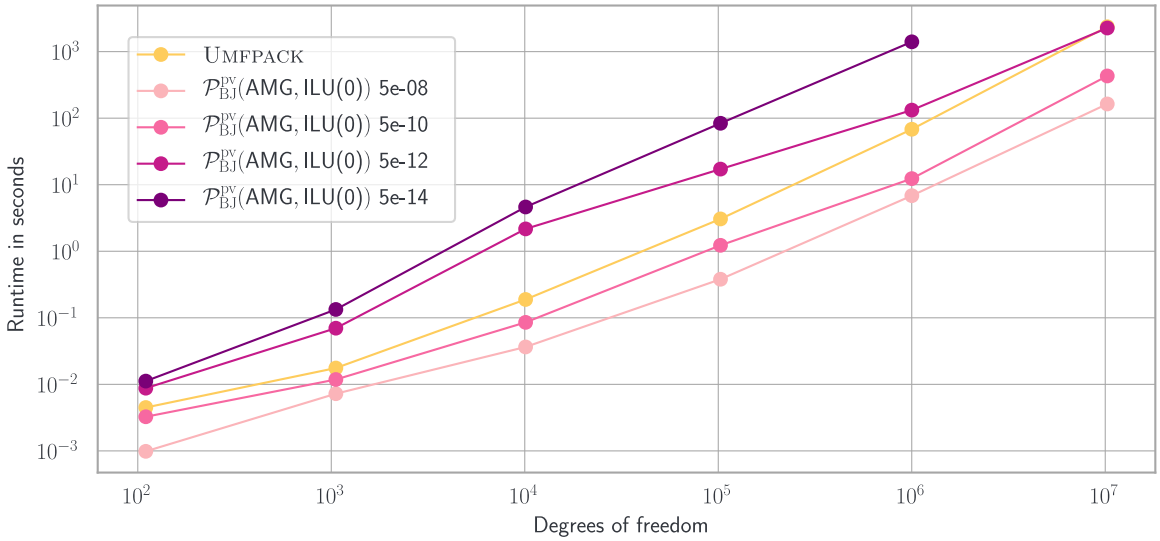


Figure 6.16: Runtime Scaling for $\mathcal{P}_{\mathrm{BJ}}^{\mathrm{pv}}(\mathrm{AMG}, \mathrm{ILU}(0))$ and different target residuals, horizontal flow.

Runtimes for iterating a $\mathcal{P}_{\mathrm{BJ}}^{\mathrm{pv}}(\mathrm{AMG}, \mathrm{ILU}(0))$-preconditioned PD-GMRES(3,3,5) solver to residuals of $5 \cdot 10^{-8}$, $5 \cdot 10^{-10}$, $5 \cdot 10^{-12}$ and $5 \cdot 10^{-14}$. As before, all runtime tests are performed on a single core of an AMD EPYC 7551P with 2.0 GHz.

of the factors, iterative solvers can be used to iterate the approximate solution to an "arbitrary" residual bound. Of course in practice, the minimal reachable residual is determined by the spectral properties of the matrix, the floating point precision and rounding errors.

This is advantageous if a larger residual is acceptable, or if time and memory are not an issue but the final residual should be very small. Iterating up to different residuals is only possible with iterative solvers, because sparse direct methods end in one final decomposition.

To emphasize this advantage of iterative methods, Figure 6.16 shows the runtimes for a $\mathcal{P}_{\mathrm{BJ}}^{\mathrm{pv}}(\mathrm{AMG}, \mathrm{ILU}(0))$-preconditioned PD-GMRES(3,3,5) solver that is iterated to different residual bounds. In the displayed horizontal flow scenario, the residuals produced by UMFPACK are between $1 \cdot 10^{-10}$ and $3 \cdot 10^{-10}$ depending on the degrees of freedom. The iterative solver is iterated to target residuals of $5 \cdot 10^{-8}$, $5 \cdot 10^{-10}$, $5 \cdot 10^{-12}$ and $5 \cdot 10^{-14}$.

We see that the iterative solver runtimes scale more favorably than those of UMFPACK. In the same time as UMFPACK we can compute the solution up to the comparable residual of $5 \cdot 10^{-10}$ for small problem sizes, but can also iterate to a residual of $5 \cdot 10^{-12}$ for the largest problem. Also, we require a lot less runtime if the approximate solution is only required up to a residual of $5 \cdot 10^{-8}$

This property of iterative solvers gives us a much greater flexibility. For a given timeframe, we can compute the best approximation within this time or determine the maximal solvable problem size if the desired residual is given. All those optimizations are a lot harder to perform with direct solvers.

6.3 Concluding remarks

After two relatively distinct aspects of preconditioned solvers have been considered in Section 6.1 and 6.2, we want to summarize the key aspects of this evaluation.

First, we found that PD-GMRES(3,3,5,-3,5) offers the best trade off between a fast convergence and a moderate growth of the restart parameter over the number of iterations, which reduces the computational cost per PD-GMRES cycle.

In the consecutive comparison of block preconditioner setups in Section 6.1.3, we identified the free flow velocity or free flow block (depending on the block preconditioner) to have the biggest influence on the preconditioned solver's convergence rate. The best convergence rates, however, can only be achieved in combination with a capable preconditioner on the porous media pressure block. In terms of convergence rates, the best performing preconditioners are Uzawa, AMG and ILU(3), which also have the biggest impact on the spectrum of the preconditioned matrix.

Further we found that (partial) Gauss-Seidel preconditioning has a positive impact on the convergence rate, which is improved if more off-diagonal preconditioners are added. We also identified that adding only $\mathcal{P}_{1,0}$ to the block Jacobi preconditioner already has a positive effect on the convergence rate, and that the additional use of $\mathcal{P}_{2,1}$ leads to convergence rates that are almost as good as a full block Gauss-Seidel preconditioner. In terms of runtime, however, we found that additional off-diagonal preconditioners mostly lead to increased runtimes. We also see that combinations of AMG or Uzawa for the free flow block with preconditioners like AMG or ILU(0) on the porous media domain lead to the fastest solvers for the considered problem size. With regard to the Uzawa iteration, we experimentally validated that the convergence rate of the inexact Uzawa iteration can be gradually improved as the approximation of V^{-1} becomes better.

Considering the solver runtime over the problem size, we also found that for different problem scales the convergence rate of a preconditioned solver does not directly translate to runtime. The most runtime efficient preconditioners for PD-GMRES are those that combine a fair convergence rate with a medium to low computational cost per iteration. With several preconditioner configurations, we can outperform the sparse direct baseline UMFPACK and achieve a ten times lower runtime on the largest tested problems with 10 million degrees of freedom. However, the best solver configuration is in general hard to predict as it depends on the problem and might therefore require a certain degree of experimental assessment.

With block preconditioning it is now possible to use iterative solvers in DUMU$^{\text{X}}$, which results in lower runtimes for solving the Stokes-Darcy problem from Chapter 2. Apart from the lower runtimes, the iterative approach allows to solve the problem up to specified residual bounds, which was not possible with the previously available sparse direct solver from UMFPACK.

Chapter 7

Conclusion and outlook

7.1 Conclusion

In this thesis, we present a flexible block preconditioning framework that significantly improves the performance of iterative solvers for stationary Stokes-Darcy flows. This shows that block preconditioning is a capable approach to solve those types of ill-conditioned problems.

By using knowledge about the underlying fluid-mechanical model and the structure of its stiffness matrix it is possible to construct an efficient block preconditioner architecture. Because of its general formulation, our approach allows to apply specialized preconditioners to different blocks of the system matrix that can exploit explicit knowledge about the model physics or properties of the sub-matrix. For example, we observe during our evaluation that applying one Uzawa iteration as preconditioner to the saddle point structure of the free flow block yields highly effective preconditioners for the overall system, whose performance can be increased even more through considerate preconditioning of the remaining components. Of course, such an approach profits from – but also depends on – a certain amount of knowledge about the underlying model or matrix structure in order to select the right preconditioning techniques for the components.

Using this structure-based approach, it is possible to precondition matrices that could not be preconditioned with standard approaches. This is also the case for the two-phase Stokes-Darcy model, where the zero block corresponding to the free flow pressure prevents the application of most black box preconditioners. Additionally, our block preconditioning allows to apply or neglect off-diagonal block preconditioners, which can be used to balance computational cost with better performing preconditioners. Depending on the computational resources available, it is possible to choose between block Jacobi, partial block Gauss-Seidel or block Gauss-Seidel preconditioning. As we show in the evaluation, including more parts of the matrix on a block level generally increases the preconditioners convergence, and can have a positive effect on the preconditioner runtime.

Because our block preconditioning concept is implemented in DUNE, future projects can profit from the flexibility of our approach as it is easily adaptable to the preconditioner requirements of other matrix structures. For our Stokes-Darcy problem, block preconditioning makes the application of iterative solvers not only feasible, but outperforms the direct solver UMFPACK in terms of runtime for several preconditioner configurations.

7.2 Outlook

One of the most capable preconditioners in our numerical experiments is the algebraic multigrid method combined with an additional preconditioner on the free flow and porous media blocks. In our experi-

ments, we selected suitable parameters for the non-smooth aggregation AMG method, but did not optimize the parameters for specific blocks. Instead, we use the same parameter settings for all AMG methods on the free flow and porous media blocks, as well as the Uzawa iteration. It is likely that further tuning for the different parameters will yield a better performance. Another point to address is the current inapplicability of AMG preconditioning to the pure Neumann problem that is defined on the horizontal flow porous media block. Because the pure Neumann problem causes the solution to be defined up to a constant, a way to circumvent the convergence to different solutions could be to always select one specific out of the infinite amount of solutions. This can be implemented by scaling the iterative solution after every AMG application such that the integral over the solution on the porous media domain has always the same value, for example zero.

During our evaluation we have seen that a good performance of a preconditioned solvers in terms of iterations does not necessarily imply a fast runtime. Especially when we consider partial block preconditioners we observe that including certain blocks into the preconditioners improves the performance in one or both metrics more than other blocks. In the work of [16] so called *constraint preconditioners* are considered as extension to the approach of [14] that is the basis for the block Jacobi and Gauss-Seidel preconditioner herein. Those constraint preconditioners additionally include upper diagonal blocks, which leads to preconditioning of blocks that are not preconditioned if only lower and diagonal blocks are used in the preconditioner. Therefore, this is an extension worth exploring. Additionally it would be easy to include this approach into the existing block preconditioner interface that we added to ISTL, because it simply modifies the functionality of our lower-diagonal preconditioner implementation.

We see that block preconditioning is very successful for stationary Stokes-Darcy systems. An interesting line of research is to apply this concept to instationary problems, and to investigate if our insights from the Stokes-Darcy generalize to time-dependent problems. To solve the time dependency, a time stepping scheme is required, and the stiffness matrix needs to be inverted for every time step. The main problem is that the stiffness matrix generally changes in every time step due to the time dependency. Even though the general matrix structure remains a blocked matrix with the same blocks (p^{ff}, v^{ff} and p^{pm}) as for the coupled Stokes-Darcy system, preconditioners would need to be recomputed for every time step to account for the changing matrix structure.

A second possible generalization is the extension of our linear Stokes-Darcy problem to the nonlinear Navier-Stokes problem. The additional nonlinear transport term must be linearized, and the nonlinearity intensifies the unsymmetry of the Jacobian matrix. Also, several steps of Newton's method have to be performed in order to resolve the nonlinearity, which leads to changing coefficient matrices like in the time-dependent case. A strategy that is sometimes applied to the stiffness matrix is a matrix recycling over several iterations, which reduces the number of stiffness matrix assemblies but at the same time uses an inaccurate stiffness matrix for the Newton step, which might lead to slower convergence. A key question is whether this strategy is also applicable to the whole or parts of our block preconditioners and whether certain preconditioning techniques are more robust with respect to recycling. The general grid hierarchy, which uses relative entry magnitudes to construct the coarser grids and builds the backbone of AMG methods, might for example be better suited for reuse on several similarly structured matrices than a Jacobi preconditioner, which uses the exact diagonal values of the matrix. Yet, the Jacobian's general block structure remains unchanged as long as no additional transport equations are added to the system, which would result in more blocks. Therefore, the block preconditioning approach remains valid, but the suitability of the sub-preconditioner needs to be re-evaluated depending on the new structure of the blocks. If, for example, additional equations are added in order to model a heat flow, the number of blocks in the Jacobian increases. The current implementation requires only small changes to add more blocks to the block preconditioner, but it still remains the need to evaluate suitable preconditioner configurations for the diagonal blocks associated to those new degrees of freedom.

Another idea we could not implement within the given time is concerned with an extension of the Uzawa iteration. Traditionally, we use the Stokes system with its pressure and velocity blocks to define the

saddle point problem. However, it is possible to treat the coupled free flow velocity and porous medium pressure system as one block, and to still maintain the typical saddle point structure. It would be interesting to know if such an Uzawa preconditioner on the whole matrix performs better than block preconditioning. Extensions of Uzawa iterations to unsymmetric saddle point problems have been discussed in the literature [12]. Also, in [36] an Uzawa iteration that treats both the free flow and porous media flow velocity as velocities in the saddle point problem has been applied as smoother for an AMG solver on the coupled Stokes-Darcy problem. However, to the best of our knowledge using an Uzawa iteration on the coupled Stokes-Darcy system to precondition a GMRES solver has not been studied.

7.3 Acknowledgments

Like most big projects, thesis would not have been possible without the support of a number of people. We want to thank Prof. Dr. Dominik Göddeke for supervising this thesis, his constructive criticism and the opportunity to present this work to a wider circle of researchers, M.Sc. Mirco Altenbernd for the discussions on various topics related to scientific computing, B.Sc. Cedric Rietmüller for the great collaboration during the familiarization with DUNE and DUMU$^{\text{X}}$, M.Sc. Kilian Weishaupt for his support in navigating the depths of DUMU$^{\text{X}}$ and his expertise with modeling flows, and last but not least Prof. Dr. Bernd Flemisch for his Uzawa implementation.

This work is embedded into the project area A “free flow and porous-media flow” of the Sonderforschungsbereich 1313 (SFB1313) “Interface-Driven Multi-Field Processes in Porous Media – Flow, Transport and Deformation”. The authors would further like to thank the SFB1313 for the opportunity to participate in the status seminar 2020.

Appendix A

Further results

A.1 Iterations and runtimes for block preconditioner on horizontal flow

Tables A.1 and A.2 show the convergence rates and runtimes for different partial block Gauss-Seidel preconditioner configurations on a 50×50 horizontal flow setup. Their contents are additional material for Section 6.1.4.

A.2 Runtime scaling for partial block Gauss-Seidel preconditioning

Figure A.1 shows a comparison of different (partial) block Gauss-Seidel configurations for the horizontal flow to complete the vertical flow illustration of Figure 6.15 in Section 6.2.1.

A, B	$\begin{pmatrix} \mathrm{I} & 0 & 0 \\ 0 & \mathrm{A} & 0 \\ 0 & 0 & \mathrm{B} \end{pmatrix}$	$\begin{pmatrix} \mathrm{I} & 0 & 0 \\ \mathcal{P}_{1,0} & \mathrm{A} & 0 \\ 0 & 0 & \mathrm{B} \end{pmatrix}$	$\begin{pmatrix} \mathrm{I} & 0 & 0 \\ 0 & \mathrm{A} & 0 \\ 0 & \mathcal{P}_{2,1} & \mathrm{B} \end{pmatrix}$	$\begin{pmatrix} \mathrm{I} & 0 & 0 \\ \mathcal{P}_{1,0} & \mathrm{A} & 0 \\ 0 & \mathcal{P}_{2,1} & \mathrm{B} \end{pmatrix}$	$\begin{pmatrix} \mathrm{I} & 0 & 0 \\ \mathcal{P}_{1,0} & \mathrm{A} & 0 \\ \mathcal{P}_{2,0} & \mathcal{P}_{2,1} & \mathrm{B} \end{pmatrix}$
I, I	1.00	1.00	1.00	1.00	1.00
I, JAC	1.00	1.00	1.00	1.00	1.00
I, ILU(0)	1.00	1.00	1.00	1.00	1.00
I, ILU(3)	1.00	1.00	1.00	1.00	1.00
I, AMG	1.00	1.00	1.00	1.00	1.00
JAC, I	1.00	1.00	1.00	0.99	1.00
JAC, JAC	1.00	1.00	1.00	1.00	1.00
JAC, ILU(0)	1.00	1.00	1.00	0.99	0.99
JAC, ILU(3)	1.00	1.00	1.00	0.99	0.99
JAC, AMG	1.00	1.00	1.00	1.00	1.00
ILU(0), I	1.00	1.00	1.00	0.99	0.99
ILU(0), JAC	1.00	1.00	0.99	0.99	0.99
ILU(0), ILU(0)	0.99	0.99	1.00	0.99	0.99
ILU(0), AMG	1.00	1.00	1.00	1.00	1.00
ILU(3), I	0.98	0.98	0.98	0.98	0.98
ILU(3), JAC	0.98	0.99	0.98	0.98	0.98
ILU(3), ILU(3)	0.98	0.98	0.97	0.97	0.97
ILU(3), AMG	1.00	1.00	1.00	1.00	1.00
AMG, I	0.98	0.98	0.98	0.98	0.98
AMG, JAC	0.98	0.98	0.98	0.98	0.98
AMG, ILU(0)	0.98	0.98	0.98	0.98	0.98
AMG, ILU(3)	0.97	0.98	0.97	0.97	0.97
AMG, AMG	1.00	1.00	1.00	1.00	1.00
Uzawa, I	0.97				0.94
Uzawa, JAC	0.97				0.95
Uzawa, ILU(0)	0.96				0.94
Uzawa, ILU(3)	0.92				0.89
Uzawa, AMG	1.00				1.00

Table A.1: Convergence rates for block Gauss-Seidel configurations on horizontal flow.

Convergence rate calculated over the first 400 iterations for different preconditioner configurations on the diagonal (per row) and all combinations of partial block Gauss-Seidel (columns). Configurations with Uzawa use 200 iterations for the convergence rate.

A, B	$\begin{pmatrix} \mathrm{I} & 0 & 0 \\ 0 & \mathrm{A} & 0 \\ 0 & 0 & \mathrm{B} \end{pmatrix}$	$\begin{pmatrix} \mathrm{I} & 0 & 0 \\ \mathcal{P}_{1,0} & \mathrm{A} & 0 \\ 0 & 0 & \mathrm{B} \end{pmatrix}$	$\begin{pmatrix} \mathrm{I} & 0 & 0 \\ 0 & \mathrm{A} & 0 \\ 0 & \mathcal{P}_{2,1} & \mathrm{B} \end{pmatrix}$	$\begin{pmatrix} \mathrm{I} & 0 & 0 \\ \mathcal{P}_{1,0} & \mathrm{A} & 0 \\ 0 & \mathcal{P}_{2,1} & \mathrm{B} \end{pmatrix}$	$\begin{pmatrix} \mathrm{I} & 0 & 0 \\ \mathcal{P}_{1,0} & \mathrm{A} & 0 \\ \mathcal{P}_{2,0} & \mathcal{P}_{2,1} & \mathrm{B} \end{pmatrix}$
I, I	–	56.96	–	35.00	46.40
I, JAC	–	11.55	–	8.98	14.26
I, ILU(0)	–	10.33	–	8.80	11.60
I, ILU(3)	–	15.88	–	14.86	18.94
I, AMG	–	–	–	–	–
JAC, I	29.42	9.29	22.31	12.02	8.47
JAC, JAC	27.04	7.61	25.96	7.57	11.88
JAC, ILU(0)	13.81	7.80	27.93	10.06	6.52
JAC, ILU(3)	18.74	4.79	37.14	10.10	1.98
JAC, AMG	–	–	–	–	–
ILU(0), I	1.98	1.39	2.49	0.87	1.06
ILU(0), JAC	2.72	1.37	2.05	0.74	1.33
ILU(0), ILU(0)	1.96	1.71	1.82	1.00	0.65
ILU(0), AMG	–	–	–	–	–
ILU(3), I	20.15	24.57	22.73	24.37	26.18
ILU(3), JAC	22.86	25.62	25.79	29.73	33.79
ILU(3), ILU(3)	19.23	22.36	21.32	21.40	22.64
ILU(3), AMG	–	–	–	–	–
AMG, I	0.66	0.79	0.64	0.96	1.29
AMG, JAC	1.01	1.32	1.13	1.53	1.62
AMG, ILU(0)	0.44	0.72	0.44	0.53	0.69
AMG, ILU(3)	0.46	0.45	0.40	0.57	0.71
AMG, AMG	–				–
Uzawa, I	0.37				0.60
Uzawa, JAC	0.22				0.35
Uzawa, ILU(0)	0.33				0.56
Uzawa, ILU(3)	0.42				0.52
Uzawa, AMG	–				–

Table A.2: Runtimes including preconditioner setup time for block Gauss-Seidel configurations on horizontal flow.

Runtimes in seconds for different block Gauss-Seidel preconditioner configurations. Solver times include setup time and time to reduce the residual below 10^{-10} with PD-GMRES(3,3,5) on a 50×50 cell discretization for free flow and porous media domain. Cells marked with "–" exceeded a maximal time of 120 seconds to achieve convergence. Measurements taken on a single core of an AMD EPYC 7551P with 2.0 GHz.

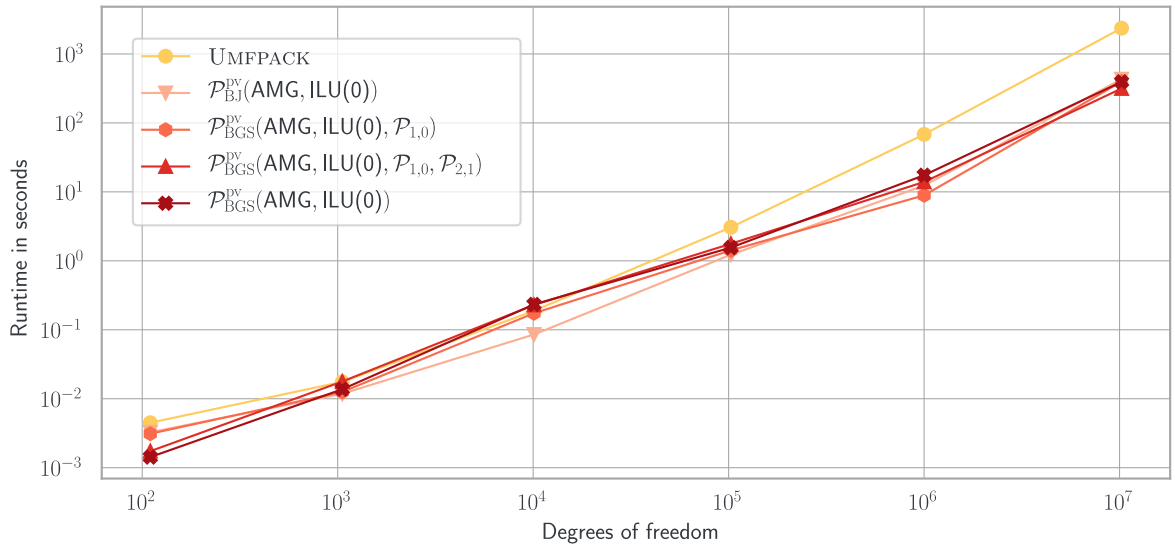


Figure A.1: Runtime scaling for partial block Gauss-Seidel preconditioning, horizontal flow.

Comparison of (partial) pressure-velocity block Gauss-Seidel preconditioning for an AMG preconditioner on the free flow velocity and an ILU(0) preconditioner on the porous media pressure block. The iterative solvers reduce the residual below a bound of 5×10^{-10}. The time measurements are taken on a single core of an AMD EPYC 7551P with 2.0 GHz. The vertical flow setup is shown in Figure 6.15.

Bibliography

[1] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. Third Edition. Philadelphia, PA: SIAM, 1999.

[2] W. E. Arnoldi. "The Principle of Minimized Iterations in the Solution of the Matrix Eigenvalue Problem". In: *Quarterly of Applied Mathematics* 9.1 (1951), pp. 17–29.

[3] P. Bastian, M. Blatt, A. Dedner, C. Engwer, R. Klöfkorn, S. Kuttanikkad, M. Ohlberger, and O. Sander. "The Distributed and Unified Numerics Environment (DUNE)". In: *Proc. of the 19th Symposium on Simulation Technique in Hannover, Sep. 12 - 14*. 2006.

[4] P. Bastian, M. Blatt, A. Dedner, C. Engwer, R. Klöfkorn, R. Kornhuber, M. Ohlberger, and O. Sander. "A Generic Grid Interface for Parallel and Adaptive Scientific Computing. Part II: Implementation and Tests in DUNE". In: *Computing* 82.2-3 (2008), pp. 121–138.

[5] P. Bastian, M. Blatt, A. Dedner, C. Engwer, R. Klöfkorn, M. Ohlberger, and O. Sander. "A Generic Grid Interface for Parallel and Adaptive Scientific Computing. Part I: Abstract Framework". In: *Computing* 82.2-3 (2008), pp. 103–119.

[6] G. K. Batchelor. *An Introduction to Fluid Dynamics*. Cambridge University Press, 2000.

[7] M. Benzi. "Preconditioning Techniques for Large Linear Systems: A Survey". In: *Journal of Computational Physics* 182.2 (2002), pp. 418–477.

[8] M. Blatt and P. Bastian. "The Iterative Solver Template Library". In: *International Workshop on Applied Parallel Computing*. Springer. 2006, pp. 666–675.

[9] M. Blatt, O. Ippisch, and P. Bastian. *A Massively Parallel Algebraic Multigrid Preconditioner Based on Aggregation for Elliptic Problems with Heterogeneous Coefficients*. Tech. rep. arXiv:1209.0960. 2012.

[10] M. Blatt, A. Burchardt, A. Dedner, C. Engwer, J. Fahlke, B. Flemisch, C. Gersbacher, C. Gräser, F. Gruber, C. Grüninger, et al. "The Distributed and Unified Numerics Environment, Version 2.4". In: *Archive of Numerical Software* 4.100 (2016), pp. 13–29.

[11] M. Bollhöfer, O. Schenk, R. Janalík, S. Hamm, and K. Gullapalli. *State-of-the-Art Sparse Direct Solvers*. Tech. rep. arXiv:1907.05309. arXiv, 2019.

[12] J. Bramble, J. Pasciak, and A. Vassilev. "Uzawa Type Algorithms for Nonsymmetric Saddle Point Problems". In: *Mathematics of Computation* 69.230 (2000), pp. 667–689.

[13] W. L. Briggs, V. E. Henson, and S. F. McCormick. *A Multigrid Tutorial*. Second Edition. SIAM, 2000.

[14] M. Cai, M. Mu, and J. Xu. "Preconditioning Techniques for a Mixed Stokes/Darcy Model in Porous Media Applications". In: *Journal of Computational and Applied Mathematics* 233.2 (2009), pp. 346–355.

[15] E. Carson and Z. Strakoš. "On the Cost of Iterative Computations". In: *Philosophical Transactions of the Royal Society A* 378.2166 (2020).

[16] P. Chidyagwai, S. Ladenheim, and D. B. Szyld. “Constraint Preconditioning for the Coupled Stokes–Darcy System”. In: *SIAM Journal on Scientific Computing* 38.2 (2016), A668–A690.

[17] W. Dahmen, T. Gotzen, S. Müller, and M. Rom. “Numerical Simulation of Transpiration Cooling through Porous Material”. In: *International Journal for Numerical Methods in Fluids* 76.6 (2014), pp. 331–365.

[18] T. A. Davis. “Algorithm 832: UMFPACK V4.3—an Unsymmetric-Pattern Multifrontal Method”. In: *ACM Transactions on Mathematical Software (TOMS)* 30.2 (2004), pp. 196–199.

[19] T. A. Davis. “UMFPACK User Guide”. In: *Website: http://www.suitesparse.com* (2018).

[20] T. A. Davis and I. S. Duff. “A Combined Unifrontal/Multifrontal Method for Unsymmetric Sparse Matrices”. In: *ACM Transactions on Mathematical Software (TOMS)* 25.1 (1999), pp. 1–20.

[21] T. A. Davis and I. S. Duff. “An Unsymmetric-Pattern Multifrontal Method for Sparse LU Factorization”. In: *SIAM Journal on Matrix Analysis and Applications* 18.1 (1997), pp. 140–158.

[22] M. Discacciati, A. Quarteroni, et al. “Navier-Stokes/Darcy Coupling: Modeling, Analysis, and Numerical Approximation”. In: *Revista Matemática Complutense* 22.2 (2009), pp. 315–426.

[23] Dune-Project. *DUNE*. 2006. URL: `https://www.dune-project.org/` (visited on 04/30/2020).

[24] H. C. Elman. “Iterative Methods for Large, Sparse, Nonsymmetric Systems of Linear Equations”. PhD thesis. Yale University New Haven, Conn, 1982.

[25] H. C. Elman and G. H. Golub. “Inexact and Preconditioned Uzawa Algorithms for Saddle Point Problems”. In: *SIAM Journal on Numerical Analysis* 31.6 (1994), pp. 1645–1661.

[26] M. Embree. “The Tortoise and the Hare Restart GMRES”. In: *SIAM Review* 45.2 (2003), pp. 259–266.

[27] R. Eymard, T. Gallouët, and R. Herbin. *Finite Volume Methods*. Vol. 7. Elsevier, 2000, pp. 713–1018.

[28] B. Flemisch, M. Darcis, K. Erbertseder, B. Faigle, A. Lauser, K. Mosthaf, S. Müthing, P. Nuske, A. Tatomir, M. Wolff, et al. “DuMux: DUNE for Multi-{Phase, Component, Scale, Physics,...} Flow and Transport in Porous Media”. In: *Advances in Water Resources* 34.9 (2011), pp. 1102–1112.

[29] A. Greenbaum, V. Pták, and Z. Strakoš. “Any Nonincreasing Convergence Curve is Possible for GMRES”. In: *SIAM Journal on Matrix Analysis and Applications* 17.3 (1996), pp. 465–469.

[30] C. Grüninger. *Numerical Coupling of Navier–Stokes and Darcy Flow for Soil-Water Evaporation*. Stuttgart : Eigenverlag des Instituts für Wasser- und Umweltsystemmodellierung der Universität Stuttgart, 2017.

[31] V. Gurau and J. A. Mann Jr. “A Critical Overview of Computational Fluid Dynamics Multiphase Models for Proton Exchange Membrane Fuel Cells”. In: *SIAM Journal on Applied Mathematics* 70.2 (2009), pp. 410–454.

[32] D. Hysom and A. Pothen. “Level-Based Incomplete LU Factorization: Graph Model and Algorithms”. In: *SIAM Journal on Matrix Analysis and Applications* (2002).

[33] T. Koch, K. Heck, N. Schröder, H. Class, and R. Helmig. “A New Simulation Framework for Soil–Root Interaction, Evaporation, Root Growth, and Solute Transport”. In: *Vadose Zone Journal* 17.1 (2018).

[34] T. Koch, D. Gläser, K. Weishaupt, S. Ackermann, M. Beck, B. Becker, S. Burbulla, H. Class, E. Coltman, S. Emmert, et al. “DuMux 3—An Open-Source Simulator for Solving Flow and Transport Problems in Porous Media with a Focus on Model Coupling”. In: *Computers & Mathematics with Applications* (2020).

[35] W. J. Layton, F. Schieweck, and I. Yotov. "Coupling Fluid Flow with Porous Media Flow". In: *SIAM Journal on Numerical Analysis* 40.6 (2002), pp. 2195–2218.

[36] P. Luo, C. Rodrigo, F. J. Gaspar, and C. W. Oosterlee. "Uzawa Smoother in Multigrid for the Coupled Porous Medium and Stokes Flow System". In: *SIAM Journal on Scientific Computing* 39.5 (2017), S633–S661.

[37] J. A. Meijerink and H. A. Van Der Vorst. "An Iterative Solution Method for Linear Systems of Which the Coefficient Matrix Is a Symmetric M-matrix". In: *Mathematics of Computation* 31.137 (1977), pp. 148–162.

[38] A. Meister. *Numerik Linearer Gleichungssysteme*. Vol. 5. Springer, 2015.

[39] K. Mosthaf, K. Baber, B. Flemisch, R. Helmig, A. Leijnse, I. Rybak, and B. Wohlmuth. "A Coupling Concept for Two-phase Compositional Porous-Medium and Single-Phase Compositional Free Flow". In: *Water Resources Research* 47.10 (2011).

[40] R. C. Núñez, C. E. Schaerer, and A. Bhaya. "A Proportional-Derivative Control Strategy for Restarting the GMRES(m) Algorithm". In: *Journal of Computational and Applied Mathematics* 337 (2018), pp. 209–224.

[41] T. Oliphant. *NumPy: A guide to NumPy*. USA: Trelgol Publishing. 2006–2020.

[42] M. Rozhanskaya and I. Levinova. "Statics". In: *Encyclopaedia of the History of Arabic Science* 3 (1996), pp. 614–642.

[43] J. W. Ruge and K. Stüben. "Algebraic Multigrid". In: *Multigrid Methods*. 1987. Chap. 4, pp. 73–130.

[44] Y. Saad and M. H. Schultz. "GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems". In: *SIAM Journal on Scientific and Statistical Computing* 7.3 (1986), pp. 856–869.

[45] Y. Saad. "ILUT: A Dual Threshold Incomplete LU Factorization". In: *Numerical Linear Algebra with Applications* 1.4 (1994), pp. 387–402.

[46] Y. Saad. *Iterative Methods for Sparse Linear Systems*. Second Edition. SIAM, 2003.

[47] O. Schenk, K. Gärtner, W. Fichtner, and A. Stricker. "PARDISO: A High-performance Serial and Parallel Sparse Linear Solver in Semiconductor Device Simulation". In: *Future Generation Computer Systems* 18.1 (2001), pp. 69–78.

[48] M. Schneider, K. Weishaupt, D. Gläser, W. M. Boon, and R. Helmig. "Coupling Staggered-Grid and MPFA Finite Volume Methods for Free Flow/Porous-Medium Flow Problems". In: *Journal of Computational Physics* 401 (2020).

[49] S. M. Shokri-Kuehni, B. Raaijmakers, T. Kurz, D. Or, R. Helmig, and N. Shokri. "Water Table Depth and Soil Salinization: From Pore-Scale Processes to Field-Scale Responses". In: *Water Resources Research* 56.2 (2020).

[50] K. Stüben. "A Review of Algebraic Multigrid". In: *Numerical Analysis: Historical Developments in the 20th Century*. Elsevier, 2001, pp. 331–359.

[51] The SciPy community. *NumPy Linalg Eig*. 2020. URL: `https://numpy.org/doc/stable/reference/generated/numpy.linalg.eig.html` (visited on 05/25/2020).

[52] The SciPy community. *NumPy Linalg Slogdet*. 2020. URL: `https://numpy.org/doc/stable/reference/generated/numpy.linalg.slogdet.html` (visited on 05/25/2020).

[53] J. Vanderborght, T. Fetzer, K. Mosthaf, K. M. Smits, and R. Helmig. "Heat and Water Transport in Soils and across the Soil-Atmosphere Interface: 1. Theory and Different Model Concepts". In: *Water Resources Research* 53.2 (2017), pp. 1057–1079.

[54] P. Verboven, D. Flick, B. Nicolai, and G. Alvarez. “Modelling Transport Phenomena in Refrigerated Food Bulks, Packages and Stacks: Basics and Advances”. In: *International Journal of Refrigeration* 29.6 (2006), pp. 985–997.

[55] E. Vidotto, T. Koch, T. Köppl, R. Helmig, and B. Wohlmuth. “Hybrid Models for Simulating Blood Flow in Microvascular Networks”. In: *Multiscale Modeling & Simulation* 17.3 (2019), pp. 1076–1102.

[56] D. M. Young and K. C. Jea. “Generalized Conjugate-Gradient Acceleration of Nonsymmetrizable Iterative Methods”. In: *Linear Algebra and its Applications* 34 (1980), pp. 159–194.

[57] L. Zhuang, S. M. Hassanizadeh, C. van Duijn, S. Zimmermann, I. Zizina, and R. Helmig. “Experimental and Numerical Studies of Saturation Overshoot during Infiltration into a Dry Soil”. In: *Vadose Zone Journal* 18.1 (2019).

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature