Low-rank update of preconditioners for saddle-point systems in fluid flow problems

Vom Promotionsausschuss der
Technischen Universität Hamburg

zur Erlangung des akademischen Grades

Doktorin der Naturwissenschaften (Dr. rer. nat.)

genehmigte Dissertation

von
Rebekka Salome Beddig

aus
Bayreuth

2024

1. Gutachterin: Prof. Dr. Sabine Le Borne
2. Gutachter: Prof. Dr. Jörn Behrens

Tag der mündlichen Prüfung: 15. März 2024

doi:10.15480/882.13171
ORCiD:  https://orcid.org/0000-0002-3773-8530

Acknowledgments

This thesis was written in the course of my time at the Institute of Mathematics at the Hamburg University of Technology. In this regard, I acknowledge the support by the Deutsche Forschungsgemeinschaft (DFG) within the Research Training Group GRK 2583 "Modeling, Simulation and Optimization of Fluid Dynamic Applications".

First, I thank my supervisor Prof. Dr. Sabine Le Borne for the valuable scientific advice and feedback as well as the patient and supportive guidance. Furthermore, I would like to thank my second supervisor Prof. Dr. Jörn Behrens for the scientific input regarding the test cases and his advice. I am also grateful to Dr. Konrad Simon who provided the test cases and his code.

I further thank all colleagues of the research training group GRK 2583 and the Institute of Mathematics at the Hamburg University of Technology for the friendly and supportive working atmosphere.

Furthermore, I would like to thank my colleagues Jonas Grams and Lina Fesefeldt for proofreading parts of this thesis.

Finally, I thank my friends and family for the emotional support. Thank you for listening, encouraging, and calming me.

Summary

The objective of this thesis is the development and analysis of multiplicative low-rank corrections for preconditioners for saddle-point systems that arise in the numerical solution of incompressible fluid flow problems. We focus on a class of low-rank updates that are based on (randomized) low-rank approximations of the difference between the identity matrix and a preconditioned matrix or Schur complement.

Our application of interest is a model for buoyancy-driven fluid flows. This model is described by the rotating Boussinesq equations that link the rotating incompressible Navier-Stokes equations with an advection-diffusion equation for the temperature field. The numerical simulation requires solving various saddle-point systems for the fluid velocity and pressure as well as solving symmetric positive definite systems for the temperature. The numerical solution of the saddle-point problems consumes the largest part of the solver times for the Boussinesq problems. Thus, we focus on preconditioners for saddle-point systems. After reviewing and comparing common preconditioning techniques, we derive low-rank update schemes for the preconditioners.

We apply the developed update methods to different components of a block preconditioner for the saddle-point systems: to pressure Schur complement preconditioners in the form of inner and outer updates as well as to preconditioners for the velocity-related matrix block. The low-rank updates are applied to the well-known Schur complement preconditioners, the least-squares commutator and the SIMPLE preconditioner. Introducing a relaxation parameter for the possibly updated Schur complement preconditioner update improves the convergence of the iterative solver significantly. The low-rank corrections are constructed with different low-rank approximation methods from (randomized) linear algebra. We compare the approximation strategies in terms of the effectiveness of the resulting preconditioner updates as well as their computational costs. We derive (theoretical) computational complexities of the update construction and application and relate the computational costs to the complexity of the initial preconditioners and the (restarted) GMRes iterations.

An error analysis explains why low-rank corrections may counter-intuitively deteriorate a given preconditioner. Furthermore, we analyze the update methods (numerically) in terms of the spectral properties of the updated preconditioners.

Various numerical experiments illustrate the effectiveness of the update strategies. We investigate the influence of the test systems, initial preconditioners, the relaxation parameter, the update rank, and the low-rank approximation strategy on the effect of the low-rank corrections. Based on the numerical tests, we conclude that the update techniques often reduce iteration counts. Especially Schur complement preconditioners with inner updates turn out to be beneficial since these updates are inexpensive to construct and apply while typically needing only a small update rank for a significant improvement. The accuracy of the underlying low-rank approximation has a significant influence on the quality of the updated preconditioners. It can be beneficial to employ a more accurate but typically also more expensive low-rank approximation for the update construction to reduce iteration counts and thus save solver times. Depending on the update scheme and the corresponding additional construction costs for setting up the low-rank updates, the updates do not only reduce solver times but also total times.

Contents

Chapter 1

Introduction

Various industrial and scientific applications in computational fluid dynamics require the numerical solution of large-scale linear saddle-point systems. In this thesis, we deal with the numerical solution of the Boussinesq equations where the incompressible fluid dynamics are coupled to a varying temperature field. Versions of the Boussinesq equations arise in many geophysical applications such as ocean and atmospheric dynamics [23, 56] or earth mantle convection [37, 44]. We focus on the simulation of buoyancy-driven atmospheric dynamics which are driven by temperature differences. The model equations combine the incompressible rotating Navier-Stokes equations that describe the fluid pressure and velocity and a temperature advection-diffusion equation. The computational bottleneck of the simulation is the numerical solution of the Navier-Stokes equations which requires, after discretization and linearization, the solution of multiple saddle-point problems of the form

$$\begin{pmatrix} A & B^{\mathrm{T}} \\ B & 0 \end{pmatrix} \begin{pmatrix} \mathbf{u} \\ p \end{pmatrix} = \begin{pmatrix} \mathbf{f} \\ g \end{pmatrix}.$$

Especially for simulations in three-dimensional spatial domains, these systems typically have a large number of degrees of freedom. Furthermore, a challenge for numerical solvers is posed by the indefiniteness of saddle-point problems.

An overview of the numerical solution of such systems is for example given in [11] or [26]. Direct solvers are typically not feasible for the numerical solution of large linear saddle-point systems due to time and memory requirements. An alternative is given by iterative (Krylov) subspace solvers which require effective preconditioners to accelerate convergence. Preconditioners transform the linear system into a system with more favorable properties regarding the convergence of the iterative solver. A goal may be to improve spectral properties such as the clustering of eigenvalues of the preconditioned system matrix which is typically beneficial for the convergence of Krylov subspace solvers [26]. The construction and application of a practical preconditioner should be relatively inexpensive while still leading to a preconditioned system that is easy to solve. Furthermore, an "optimal" preconditioner leads to convergence behavior that does not depend on system parameters such as mesh size.

Suitable preconditioners for saddle-point systems typically exploit knowledge of the underlying problem [11]. A class of preconditioners frequently used for fluid flow problems are block preconditioners that are based on a block factorization of the system matrix. Especially the block triangular preconditioner is a common choice in fluid dynamics [26, 28].

Block preconditioners typically require an approximation to the inverse upper left matrix block A and to the inverse (negative) pressure Schur complement $S := BA^{-1}B^{\mathrm{T}}$.

Potential preconditioners for the matrix block A are multigrid methods [18, 64, 68], domain decomposition methods [22, 67], sparse approximate inverses [15, 21], or inexact factorizations [61] which can be further accelerated by inexact iterative solves [44]. Common preconditioners for the pressure Schur complement are commutator-based preconditioners or SIMPLE-type preconditioners. Examples of commutator-based preconditioners are given by the least-squares commutator [25, 26, 27], the commuted BFBt method [53, 72], the weighted BFBt method [59], or the pressure-convection-diffusion preconditioner [27, 43, 65] which are based on approximate commutators and avoid the evaluation of the pressure Schur complement. The SIMPLE preconditioner [35, 55, 58, 70, 71] is a simple approximation where the inverse A^{-1} is approximated by the inverse of the diagonal of the matrix block A. Block preconditioners that are based on a transformed saddle-point system are grad-div preconditioners [29, 30, 47] or augmented Lagrangian preconditioners [12, 13, 14, 36, 54] which shift the difficulty in preconditioning from the Schur complement to the modified upper left matrix block. Another approach is given by rank-structured preconditioners such as hierarchical matrix preconditioners [20, 45]. Alternative techniques that are not based on a block factorization are the nullspace method [11, 46] or coupled multigrid methods [1, 41].

Preconditioning techniques can be (hopefully) further improved by low-rank updates. Low-rank corrections typically aim to improve the spectral properties of the preconditioned matrix. Large eigenvalues or eigenvalues close to the origin may deteriorate the convergence of the iterative solver. Low-rank corrections aim to shift these outliers of the spectrum such that the eigenvalues become more clustered. An updated inexpensive initial preconditioner may be advantageous compared to initially more accurate but more expensive preconditioners since low-rank updates typically only introduce little additional application costs. A survey on low-rank techniques for preconditioners is given in [16]. Low-rank corrections are applied in various preconditioning techniques [3, 4, 16, 31, 33, 39, 48, 49, 73, 74]. In [16, 39], low-rank updates are applied to inexact factorization preconditioners for linear systems. In [72], low-rank updates are applied to saddle-point systems but only to the preconditioner for the upper left matrix block and not to the Schur complement preconditioner. In [4, 73, 74], low-rank corrections are applied to the Schur complement preconditioner that arises in a domain decomposition method.

The objective of this thesis is to develop and analyze low-rank updates for preconditioners for saddle-point problems in fluid flow applications. We focus on a class of low-rank updates that is based on the difference of the identity matrix and the preconditioned matrix as utilized in [74] in a domain decomposition method. A similar approach is used in [39] where it is applied to inexact LU factorization preconditioners. The motivation for this work is to investigate whether block preconditioners in fluid flow problems can be enhanced as well by this type of low-rank corrections.

The construction of low-rank updates typically requires the low-rank approximation of a matrix that is not given explicitly but is defined by its action on a vector. In [72], [73], and [74], an Arnoldi iteration is used to determine suitable update vectors. Methods from randomized numerical linear algebra are also utilized for low-rank corrections [4, 33, 39]. In [4], the Nyström method, a randomized approach suitable for symmetric positive definite matrices, is used. A randomized singular value decomposition and a randomized

interpolative decomposition are employed in [39]. More insight into further randomized low-rank approximation methods is given in the surveys [34] and [51].

The low-rank updates in this thesis are based on a low-rank approximation of the difference between the identity matrix and a preconditioned matrix or Schur complement. This difference is not given explicitly but defined by its action on a vector. Especially for Schur complement preconditioners, it is not feasible to compute an explicit representation since the Schur complement $S = BA^{-1}B^{\mathrm{T}}$ is typically dense and expensive to evaluate due to the large inverse A^{-1}. The difference can be approximated by a low-rank factorization if its singular values decay sufficiently fast. We will observe that this may be satisfied but is not guaranteed for all settings. To construct the low-rank corrections, we exploit a randomized power range finder from [34, 51] and the Arnoldi iteration.

We examine different versions of such low-rank updates. First, we derive a general update scheme for left and right preconditioned linear systems. Then, we introduce a relaxation parameter for the Schur complement preconditioner in the block (triangular) preconditioner. Numerical experiments illustrate that this relaxation parameter has a significant influence on the convergence of the iterative solver. We adapt the general right and left update schemes to relaxed Schur complement preconditioners that are used in a block preconditioner. The updates can reduce iteration counts significantly but introduce additional construction costs. We show that a part of the setup costs can be saved by the repeated application of the low-rank updates. Applying two updates of rank r instead of one update of rank $2r$ to a preconditioner can save construction costs. We furthermore discuss low-rank updates that are applied to inner preconditioners which approximate Poisson-type problems that arise in the considered Schur complement preconditioners, the least-squares commutator and the SIMPLE preconditioner. The setup costs as well as the application costs of these updates are cheap while the updates also reduce the required number of iterations significantly.

A theoretical and numerical error analysis for the considered class of low-rank updates links the approximation error of the low-rank approximation to the approximation error of the updated preconditioner. We furthermore analyze numerically how the low-rank corrections act on the spectrum of the preconditioned matrix or Schur complement. Additionally, we discuss the computational complexity of the construction and application of the derived update schemes.

Extensive numerical analysis shows the influence of various parameters on the effectiveness of the update scheme. We investigate the quality of the low-rank updates not only regarding the underlying low-rank approximation but also for different model problems, initial preconditioners, the relaxation parameter, and the update rank.

As test problems, we choose different saddle-point systems that arise in the numerical solution process of the rotating Boussinesq equations. We consider two different three-dimensional domains, a unit cube and a spherical shell. For both domains, we consider two linearization strategies for the rotating incompressible Navier-Stokes equations: a quasi-Stokes discretization that shifts the nonlinear advection to the right-hand side and a Picard correction that leads to sequences of Oseen-type systems.

The main contributions of this thesis are

- the development of multiplicative error-based low-rank updates for preconditioners and their application to different components of a block preconditioner for saddle-point systems;

- a comparison of different low-rank approximation techniques from (randomized) numerical linear algebra to construct the low-rank updates and an analysis of the resulting updates regarding complexity and effectiveness;
- the improvement of the initial block preconditioner as well as the low-rank updates by a relaxation parameter for the Schur complement preconditioner;
- a complexity analysis of the construction and application of the low-rank updates;
- a theoretical and numerical error analysis that explains why low-rank updates may (counter-intuitively) deteriorate a given preconditioner; and
- an extensive numerical analysis with test problems obtained from the rotating Boussinesq equations to analyze the influence of various parameters on the effectiveness of the updated preconditioners.

The remainder of the thesis is structured as follows: We provide mathematical background on saddle-point systems, iterative solvers and preconditioning in Chapter 2. We furthermore review low-rank approximation techniques for matrices that are not explicitly given. Next, we describe our application of interest, a model for atmospheric dynamics described by the Boussinesq equations in Chapter 3. We furthermore discuss the spatial and temporal discretization of the Boussinesq equations and provide a numerical solution algorithm. We also review block preconditioners for the arising saddle-point systems. The main contributions of this thesis are presented in Chapters 4–5. In Chapter 4, we derive various low-rank correction schemes for preconditioners that are based on a low-rank approximation of the error between the identity matrix and the preconditioned matrix or Schur complement. We analyze the numerical low-rank property of this error for different components of a block preconditioner. The low-rank schemes are applied to a relaxed Schur complement preconditioner as well as to inner Poisson-type problems that arise in the Schur complement preconditioners, the least-squares commutator and the SIMPLE preconditioner. We analyze the complexity of the construction and application of the derived low-rank updates. Furthermore, we provide an error analysis and a numerical spectral analysis. Chapter 5 illustrates the effectiveness of such low-rank updates with various numerical experiments. We numerically analyze the quality of the low-rank updates for different components of a block preconditioner in dependence on the initial preconditioner, the relaxation parameter, the update rank, and the underlying low-rank approximation. Chapter 6 concludes this thesis with a summary of the main results and an outlook on possible future tasks.

Chapter 2

Prerequisites from numerical linear algebra

This chapter gives some mathematical foundations that are required for this thesis. We start with a definition of saddle-point systems in Section 2.1. Then, we discuss iterative solvers for the preconditioned indefinite linear systems in Section 2.2 and provide a brief introduction to preconditioning in Section 2.2.1. In Section 2.3, we review low-rank approximation techniques for matrices that are defined by their action on a vector. Section 2.4 summarizes the notation and abbreviations that are used in this thesis.

2.1 Saddle-point systems

The main part of this thesis focuses on the numerical solution of saddle-point systems. In this section, we introduce linear saddle-point systems and briefly discuss conditions for unique solvability. An overview of the numerical solution of saddle-point problems is given in [11] and with a focus on incompressible fluid dynamics also in [26]. Saddle-point systems have the form

$$\begin{pmatrix} A & B^{\mathrm{T}} \\ C & -D \end{pmatrix} \begin{pmatrix} \mathbf{u} \\ p \end{pmatrix} = \begin{pmatrix} \mathbf{f} \\ g \end{pmatrix}$$

with $A \in \mathbb{R}^{n_u \times n_u}$, B, $C \in \mathbb{R}^{n_p \times n_u}$, $D \in \mathbb{R}^{n_p \times n_p}$, $\mathbf{u}$, $\mathbf{f} \in \mathbb{R}^{n_u}$, p, $g \in \mathbb{R}^{n_p}$, and satisfy at least one of the following properties:

- A is symmetric,
- the symmetric part of A, $M \coloneqq \frac{1}{2}\left(A + A^{\mathrm{T}}\right)$, is positive semidefinite,
- $B = C$,
- D is symmetric positive semidefinite,
- $D = 0$.

We consider two types of saddle-point systems:

1. symmetric systems arising from the discretized (quasi-)Stokes equations with symmetric positive definite matrix block A, $B = C$, and $D = 0$ and

2. asymmetric systems arising from the discretized and Picard-linearized Navier-Stokes equations with nonsingular A, $B = C$, and $D = 0$.

We now discuss conditions for the unique solvability of saddle-point systems with nonsingular block A. If the matrix block A is nonsingular, a block factorization of the system matrix is given by

$$\mathcal{A} = \begin{pmatrix} A & B^{\mathrm{T}} \\ C & D \end{pmatrix} = \begin{pmatrix} \mathrm{I}_{n_u} & 0 \\ CA^{-1} & \mathrm{I}_{n_p} \end{pmatrix} \begin{pmatrix} A & 0 \\ 0 & -S \end{pmatrix} \begin{pmatrix} \mathrm{I}_{n_u} & A^{-1}B^{\mathrm{T}} \\ 0 & \mathrm{I}_{n_p} \end{pmatrix} \tag{2.1}$$

where $S \coloneqq -D + CA^{-1}B^{\mathrm{T}}$ is the (negative) Schur complement of A in $\mathcal{A}$. In the context of fluid flow problems, we refer to S as the pressure Schur complement since it is related to the equation for the fluid pressure.

As can be seen from the block factorization (2.1), the saddle-point matrix $\mathcal{A}$ with nonsingular A is nonsingular if and only if the Schur complement $S = -D + CA^{-1}B^{\mathrm{T}}$ is nonsingular as well [11]. For the invertibility of S, we require restrictions on the matrix blocks B, C, and D. Since we only consider systems with $D = 0$ and $B = C$, we focus on conditions for B.

Symmetric saddle-point systems with symmetric positive definite A, $B = C$ and $D = 0$ are invertible if and only if B^{T} has full column rank since then $S = BA^{-1}B^{\mathrm{T}}$ is symmetric positive definite [11]. For asymmetric systems with $D = 0$, the following theorem from [11] gives a necessary condition for unique solvability.

Theorem 2.1. [11, Theorem 3.3] *If the matrix*

$$\mathcal{A} = \begin{pmatrix} A & B^{\mathrm{T}} \\ C & 0 \end{pmatrix}$$

with $A \in \mathbb{R}^{n_u \times n_u}$ and $B, C \in \mathbb{R}^{n_p \times n_u}$ is nonsingular, then we have $\operatorname{rank}(B) = n_p$ *and* $\operatorname{rank}\left(\begin{smallmatrix} A \\ C \end{smallmatrix}\right) = n_u$.

The condition that $\left(\begin{smallmatrix} A \\ C \end{smallmatrix}\right)$ has full column rank is satisfied for saddle-point systems with nonsingular matrix blocks A. The following theorem from [11] states a sufficient condition for invertibility of the matrix $\mathcal{A}$.

Theorem 2.2. [11, Theorem 3.4] *Assume that M, the symmetric part of A, is positive semi-definite, the matrix blocks $B = C$ have full rank, and D is symmetric positive semidefinite (possibly zero). Then*

1. $\ker(M) \cap \ker(B) = \{0\} \Rightarrow \mathcal{A}$ *is invertible,*
2. $\mathcal{A}$ *is invertible* $\Rightarrow \ker(A) \cap \ker(B) = \{0\}$.

The converses of 1 and 2 do not hold in general.

So, in both cases, unique solvability requires B^{T} to have full column rank. However, in our applications, the matrix block B^{T} is rank deficient by one, i.e. $\operatorname{rank}(B^{\mathrm{T}}) = n_p - 1$ since we consider saddle-point systems from fluid flow problems where the pressure is only defined up to a constant. We discuss in Section 3.5.3 further how we solve the saddle-point systems to obtain a reliable solution.

2.2 Preconditioned iterative solvers

This section is concerned with the iterative solution of linear systems of the form

$$\mathcal{A}x = b.$$

Iterative solvers are typically accelerated by preconditioning methods. We start with a brief summary of the idea of preconditioning and proceed with a specific solver for the preconditioned equations that is suitable for indefinite systems, the generalized minimum residual method (abbreviated as GMRes) and its flexible variant.

2.2.1 Preconditioning

Preconditioners transform a given system such that the transformed system has the same solution but more favorable properties. Preconditioning strategies are typically used to accelerate and improve the robustness of the iterative solver. The transformed system has the form

$$\mathcal{A}\mathcal{P}y = b, \qquad x = \mathcal{P}y$$

with the right-oriented preconditioner $\mathcal{P}$. Alternative options are left-oriented or split preconditioners [61]. We focus on right-oriented preconditioners since the residuals of the right-preconditioned system equate the residuals of the original system during the iterative solve which is not the case for left-oriented or split preconditioners. The preconditioner should improve the spectral properties of the system matrix as reducing its condition number. The preconditioner is not required in explicit form for the considered iterative solvers. It is sufficient to define its action on a vector. Ideally, the preconditioner is relatively cheap to construct and apply. An "optimal" preconditioner leads to convergence behavior that does not depend on the mesh size or physical parameters such as the Reynolds number (considering the Navier-Stokes equations).

2.2.2 Preconditioned GMRes and its flexible variant

In this subsection, we briefly discuss iterative solvers for (right preconditioned) linear equations of the form

$$\mathcal{A}\mathcal{P}y = b, \qquad x = \mathcal{P}y,$$

where $x \in \mathbb{R}^n$ is the solution, $b \in \mathbb{R}^n$ is the right-hand side, the system matrix $\mathcal{A} \in \mathbb{R}^{n\times n}$ is nonsingular and (possibly) indefinite, and $\mathcal{P}$ is a preconditioner.

Starting from an initial solution guess x_0, Krylov subspace methods search an iterate in the affine subspace $x_0 + \mathcal{K}_k(\mathcal{A}\mathcal{P}, v_0)$ by enforcing a Petrov-Galerkin condition $r_k \coloneqq b - \mathcal{A}x_k \perp \mathcal{L}_k$ on the residual r_k where $\mathcal{L}_k$ is a subspace of dimension k. The Krylov subspace $\mathcal{K}_k(\mathcal{A}\mathcal{P}, v_0)$ is defined as

$$\mathcal{K}_k(\mathcal{A}\mathcal{P}, v_0) = \left\{v_0, \mathcal{A}\mathcal{P}v_0, \ldots, (\mathcal{A}\mathcal{P})^{k-1}v_0\right\}$$

for a vector v_0.

We consider two Krylov subspace methods, the generalized minimum residual method and its flexible variant.

The GMRes method developed by Saad and Schultz [62] minimizes the (preconditioned) residual in the Euclidean norm at each iteration. Here, we search for the kth iterate in $x_o + \mathcal{K}_k$ with $\mathcal{K}_k = \mathcal{K}_k(\mathcal{AP}, r_0/\|r_0\|_2)$ where r_0 is the initial residual. We force the residual r_k to be orthogonal to the subspace $\mathcal{L}_k = \mathcal{AP}\mathcal{K}_k$. The iterates are determined by solving the minimization problem

$$x_k = x_0 + \arg\min_{z \in \mathcal{K}_k} \|b - \mathcal{A}(x_0 + z)\|_2. \tag{2.2}$$

GMRes is guaranteed to converge within at most n iterations where n is the system size [61]. However, with increasing the number of iterations the computational complexity and memory requirements increase as well. Especially for large systems, this may become unfeasible. A variant that avoids this problem is given by the restarted GMRes method. The restarted version limits the size of the Krylov subspace by restarting the iteration every k iterations. Restarted GMRes is not guaranteed to converge and may stagnate for indefinite systems. However, suitable preconditioning strategies may overcome this problem. In the remainder of this thesis, we will consider only the restarted version.

Algorithm 2.1 describes the restarted right-preconditioned GMRes method. The Arnoldi process from line 2 to line 10 computes an orthogonal basis of the Krylov subspace $\mathcal{K}_j(\mathcal{AP}, r_0/\|r_0\|_2)$ with the modified Gram-Schmidt method. The approximate solution is then determined by solving the minimization problem (2.2) and updating the solution vector in line 11. If the pre-defined stopping criterion is not met, we restart the iteration with the new iterate. The iteration is stopped if the relative residual drops below a given tolerance. Note that no explicit computation of the residual is required when the minimization problem (2.2) is solved with a QR factorization of $\bar{H}_k$. The residual norm of the residual associated with the approximate solution x_k is then given by the $(k+1)$st component of the vector $Q_k\|r_0\|_2 e_1$ where Q_k comes from the QR factorization of $\bar{H}_k$ [62, Proposition 1].

The Krylov subspace methods discussed so far are designed for linear preconditioners that do not change between iterations. We now discuss a further variant of GMRes which can deal with varying preconditioners (that utilize, for example, inner iterative solvers). This flexible variant was introduced in [60]. In the flexible GMRes method, the vectors $z_j \coloneqq \mathcal{P}_j v_j$ are stored as a matrix such that the approximate solution can be recovered as $x_k = x_0 + Z_k y_k$ where $Z_k = [z_1, \ldots, z_k]$. The main bottleneck of the flexible GMRes method is its memory consumption which is approximately twice as large as that of GMRes.

2.3 Low-rank approximation

For the computation of low-rank updates for preconditioners, we require a low-rank approximation of a given matrix $M \in \mathbb{R}^{n \times n}$ that is not given explicitly but is defined by its action on a vector. We desire to find thin rectangular matrices N_1, $N_2 \in \mathbb{R}^{n \times r}$ such that

$$M \approx N_1 N_2^{\mathrm{T}}$$

where $r \ll n$. This can provide a good approximation if the matrix M has well-separated and fast-decaying singular values. Then, a small rank r may be sufficient to approximate the action of this matrix to a vector with satisfying accuracy.

Algorithm 2.1: Restarted right-preconditioned GMRes [61].

Input: Right-hand side b, system matrix $\mathcal{A}$, initial guess x_0, right preconditioner $\mathcal{P}$, a stopping criterion, restart parameter k.
Output: Approximate solution x_k.

1 Compute the initial residual $r_0 = b - \mathcal{A}x_0$ and set $v_1 = r_0/\|r_0\|_2$.
2 **for** $j = 1, 2, \ldots, k$ **do**
3 Apply the preconditioner $z_j = \mathcal{P}v_j$ and the system matrix $w = \mathcal{A}z_j$.
4 **for** $i = 1, \ldots, j$ **do**
5 Compute $h_{ij} = w_j^{\mathrm{T}} v_i$.
6 Compute $w = w - h_{ij}v_i$.
7 **end**
8 Compute $h_{j+1,j} = \|w\|_2$ and $v_{j+1} = w/h_{j+1,j}$.
9 **end**
10 Define $V_k = [v_1, v_2, \ldots, v_k]$ and $\bar{H}_k = \{h_{ij}\}_{1 \le i \le k+1, 1 \le j \le k}$.
11 Compute the minimizer $y_k = \arg\min \|\|r_0\|_2 e_1 - \bar{H}_k y\|$, $e_1 = [1, 0, \ldots, 0]^{\mathrm{T}}$ and the approximate solution $x_k = x_0 + \mathcal{P}V_k y_k$.
12 If x_k does not satisfy the stopping criterion, set $x_0 = x_k$ and restart in line 1.

In the following, we discuss two techniques: a randomized approach based on a power range finder and the Arnoldi iteration. The randomized approach is based on sampling the range of the given matrix M with a few random test vectors. The approximation quality can be enhanced with power iterations and oversampling. In Section 2.3.1, we discuss two algorithms for randomized low-rank approximations, one for a given rank and one for a given target precision. In Section 2.3.2, we discuss the Arnoldi iteration that yields an orthonormal basis of a Krylov subspace and may also serve to determine the desired matrices N_1, N_2. Furthermore, we discuss approximation quality and computational costs for the algorithms.

2.3.1 Randomized low-rank approximation

Computing a low-rank approximation of a given matrix $M \in \mathbb{R}^{n \times n}$ with a randomized technique consists of two tasks:

1. The *range finder problem*: To approximate the range of a given matrix M with a low-dimensional subspace, we compute a matrix $Q \in \mathbb{R}^{n \times r}$, $r \ll n$, with orthonormal columns that satisfies
$$M \approx QQ^{\mathrm{T}}M.$$
2. *Factorization*: With the solution Q of the range finder problem, we compute an approximate low-rank factorization of the matrix $M \approx QN^{\mathrm{T}}$ with $Q, N \in \mathbb{R}^{n \times r}$ and $N \coloneqq M^{\mathrm{T}}Q$.

The factorization of step 2 can be employed to compute an approximate singular value or QR decomposition. However, we only require any low-rank approximation of a given matrix M to construct preconditioner updates and hence a low-rank approximation as given in step 2 is sufficient for our application.

We start with an intuitive description of the randomized range finder and then introduce two low-rank approximation algorithms: one for a fixed rank r and one for a fixed tolerance ζ such that

$$\|M - QQ^{\mathrm{T}}M\| = \|\left(\mathrm{I}_n - QQ^{\mathrm{T}}\right) M\| \leq \zeta$$

is satisfied for some matrix norm $\|\cdot\|$.

Starting with a random test matrix $G \in \mathbb{R}^{n\times r}$, it is likely that its columns are linearly independent and that no linear combination of the column vectors of G is in the kernel of a rank-r matrix M [34]. Then, the column vectors of $Y = MG \in \mathbb{R}^{n\times r}$ are also linearly independent and thus span the range of M. We find an orthonormal basis of this subspace by orthonormalizing the columns of Y. If the rank of M is larger than r, there is a greater chance that the sample vectors span an r-dimensional subspace of the range of M if we use more than r, i.e. $\ell_r = r + k_r$, $k_r > 0$, sample vectors.

For a practical application of this approach, we require a suitable subspace dimension ℓ_r and update rank r. For the following approach, we assume that both are given. Then, we discuss a low-rank approximation method that estimates the required rank for a prescribed tolerance.

Fixed rank

We now describe a randomized low-rank approximation method for a given rank. As stated in [34, 51], applying a few power iterations and orthogonalization steps can improve the accuracy and robustness of the range finder problem. The action of the matrix $M \in \mathbb{R}^{n\times n}$ is now sampled by applying $q \in \mathbb{N}$ power iterations

$$Y = \left(MM^{\mathrm{T}}\right)^q MG = QR$$

where $G \in \mathbb{R}^{n\times \ell}$. Orthonormalizing the columns of Y with a QR factorization yields the matrix Q. To improve the quality of Q, we orthonormalize the result after each multiplication with the given matrix M or its transpose. For the low-rank approximation of M, we may only choose r columns of Q which can be for example selected with a pivoted QR factorization. The calculation steps of the low-rank approximation with the power range finder are summarized in Algorithm 2.2. For the sampling, different randomized embeddings are suitable. Gaussian embeddings yield a comparably simple and precise theoretical analysis. We hence choose the test matrix G as a standard Gaussian matrix whose entries are independent standard normal variables with mean zero and variance one. Results for Gaussian test matrices can be found in [34]. The following corollary from [34] analyzes the expected spectral low-rank approximation error.

Corollary 2.3. [34, Corollary 10.10] *Let $M \in \mathbb{R}^{n\times n}$ be a given matrix with singular values $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_n$ and let $G \in \mathbb{R}^{n\times \ell_r}$ be a standard Gaussian test matrix with $\ell_r = r + k_r \leq n$, target rank $r \geq 2$, and oversampling parameter $k_r \geq 2$. Construct the*

sample matrix $Y = \left(MM^{\mathrm{T}}\right)^q MG$ *for a nonnegative integer* q*. Then*

$$\begin{aligned}\mathbb{E}\left(\left\|\left(\mathrm{I} - YY^{\mathrm{T}}\right)M\right\|_2\right) &\leq \left[\left(1 + \sqrt{\frac{r}{k_r - 1}}\right)\sigma_{r+1}^{2q+1} + \frac{\mathrm{e}\sqrt{\ell_r}}{k_r}\left(\sum_{j>r}\sigma_j^{2(2q+1)}\right)^{\frac{1}{2}}\right]^{\frac{1}{2q+1}} \\ &\leq \left[1 + \sqrt{\frac{r}{k_r - 1}} + \frac{\mathrm{e}\sqrt{\ell_r}}{k_r}\sqrt{n - r}\right]^{\frac{1}{2q+1}} \sigma_{r+1}\end{aligned}$$

where $\mathbb{E}(\cdot)$ *is the expected value of the argument and the number* e *is Euler's number.*

Proof. For the proof, we refer to [34, Corollary 10.10] and the following conclusions in [34]. □

The corollary states that the power scheme drives the approximation error exponentially fast to σ_{r+1} with increasing the number of power iterations q. In practice, a small number of power iterations ($q \leq 3$) usually yields satisfying accuracy [51]. Oversampling, i.e. using more sample vectors than the desired rank r, can further improve the accuracy of the solution to the range finder problem. According to [34], a small oversampling parameter of $k_r = 5$ or $k_r = 10$ is usually sufficient for Gaussian test matrices and a larger oversampling parameter with $k_r > r$ typically does not improve the approximation further.

Algorithm 2.2: Randomized low-rank approximation with the power range finder from [34].

Input: Input matrix $M \in \mathbb{R}^{n\times n}$, rank r, subspace dimension $\ell \geq r$, number of power iterations q.

Output: Low-rank approximation $QN^{\mathrm{T}} \approx M$ with $Q \in \mathbb{R}^{n\times r}$ such that $Q^{\mathrm{T}}Q = \mathrm{I}_r$, $N \in \mathbb{R}^{n\times r}$.

`// Approximate the range of` M`.`

Draw a random matrix $G \in \mathbb{R}^{n\times\ell}$.
Compute $Y_0 = MG$ and compute its QR factorization $Y_0 = Q_0R_0$.
for $j = 1, \ldots, q$ **do**
  Form $\widetilde{Y}_j = M^{\mathrm{T}}Q_{j-1}$ and compute its QR factorization $\widetilde{Y}_j = \widetilde{Q}_j\widetilde{R}_j$.
  Form $Y_j = M\widetilde{Q}_j$ and compute its QR factorization $Y_j = Q_jR_j$.
end
`// Compute the low-rank approximation.`
Set $Q = Q_q$ and truncate Q to keep only r columns.
Compute $N = M^{\mathrm{T}}Q$.

Fixed tolerance

Another algorithm for the randomized range finder problem from [51] determines a suitable subspace dimension for a predefined tolerance ζ. We construct the subspace incrementally and test after each iteration whether we have reached a given tolerance. For this, we need a criterion to estimate the approximation error $\|(\mathrm{I}_n - QQ^{\mathrm{T}})M\|$ in some matrix norm $\|\cdot\|$.

For the incremental construction of the subspace, we apply the power range finder from the preceding section to a block of r_0 random test vectors and obtain an orthonormal matrix $Q_i \in \mathbb{R}^{n\times r_0}$. Then, we orthonormalize the columns of Q_i against the orthonormal matrices $Q_j \in \mathbb{R}^{n\times r_0}$, $j = 0, \ldots, i-1$, of the previous iterations and estimate the quality of the result with a randomized norm estimator. If the approximation error drops below a given tolerance ζ, we stop the iteration and gather the results obtained in the previous iterations in the matrix Q. Otherwise, we compute a further block Q_{i+1} of orthonormal vectors with the power range finder until we have reached the desired tolerance (or a given maximum rank). This method is summarized in Algorithm 2.3. Note that we can estimate the approximation error $\|(\mathrm{I} - QQ^{\mathrm{T}})M\|$ by estimating the norm of the matrix Y since Y holds the sample

$$Y = \left(\mathrm{I} - QQ^{\mathrm{T}}\right) MG$$

after executing line 9 in Algorithm 2.3.

To estimate the quality of a subspace, spanned by the columns of Q, we sample the approximation error in the Frobenius norm in a few random directions. The Gaussian test matrix $\Phi \in \mathbb{R}^{n\times s}$ holds the random sampling directions. We obtain the (unbiased) error estimate

$$\|\left(\mathrm{I} - QQ^{\mathrm{T}}\right) M\|_F^2 \approx \frac{1}{s}\|\left(\mathrm{I} - QQ^{\mathrm{T}}\right) M\Phi\|_F^2 =: X_s \tag{2.3}$$

as the arithmetic mean of the samples [51, Sections 4.8 and 12.1]. According to [51], the expected value $\mathbb{E}(\cdot)$ and the variance $\mathrm{Var}(\cdot)$ of the random variable X_s are given by

$$\mathbb{E}\left(X_s\right) = \|\left(\mathrm{I} - QQ^{\mathrm{T}}\right) M\|_F^2, \qquad \mathrm{Var}\left(X_s\right) = \frac{1}{s}\|\left(\mathrm{I} - QQ^{\mathrm{T}}\right) M\|_4^4.$$

After rescaling, the corresponding sample variance furthermore provides an unbiased error estimate in the fourth power of the Schatten-4 norm $\|\cdot\|_4$ [51, Section 4.8] where the Schatten-4 norm of a matrix $X \in \mathbb{R}^{m\times n}$ is given by the ℓ_4-norm of the singular values σ_i of X, i.e.

$$\|X\|_4^4 = \sum_{i=1}^{\min\{n,m\}} \sigma_i^4.$$

2.3.2 Arnoldi iteration

In this section, we describe the computation of a low-rank approximation with the Arnoldi iteration. In each iteration, we apply the given matrix M to a vector and orthogonalize the result against the results of the previous iterations. After r steps, we find a low-rank approximation of the given matrix M of the form

$$M \approx V_r H_r V_r^{\mathrm{T}}$$

where $V_r \in \mathbb{R}^{n\times r}$ has orthonormal columns and $H_r \in \mathbb{R}^{r\times r}$ is an upper Hessenberg matrix. The columns of V_r form an orthonormal basis of the Krylov subspace

$$\mathcal{V} = \mathrm{span}\left\{v_1, Mv_1, \ldots, M^{r-1}v_1\right\}$$

Algorithm 2.3: Incremental range finder [51].

Input: Input matrix $M \in \mathbb{R}^{n \times n}$, tolerance ζ, block size r_0.
Output: Orthonormal matrix $Q \in \mathbb{R}^{n \times r_0 i}$ such that $\|M - QQ^{\mathrm{T}}M\| \leq \zeta$ with high probability.

Draw a random matrix $G \in \mathbb{R}^{n \times r_0}$.
Form $Y = MG$ and compute its QR factorization $Y = Q_0 R_0$.
Estimate $\eta \approx \|Y\|_F$ with Equation (2.3).
Set $i = 0$.
while $\eta > \zeta$ **do**
    Set $i = i + 1$.
    Draw a random matrix $G \in \mathbb{R}^{n \times r_0}$.
    Form $Y = MG$.
    Compute $Y = Y - \sum_{j=0}^{i-1} Q_j \left(Q_j^{\mathrm{T}} Y \right)$.
    Compute the QR factorization $Y = Q_i R_i$.
    Estimate $\eta \approx \|Y\|_F$ with Equation (2.3).
end
Set $Q = \begin{bmatrix} Q_0 & Q_1 & \dots & Q_i \end{bmatrix}$.
Compute $N = M^{\mathrm{T}} Q$.

where $v_1 \in \mathbb{R}^n$ is a random starting vector. With increasing the rank r, the low-rank approximation error $\|M - V_r H_r V_r^{\mathrm{T}}\|_F$ decreases monotonically [61]. We employ a Gram-Schmidt procedure from [61] for the orthogonalization steps. If we lose orthogonality in the Gram-Schmidt method, we re-orthogonalize the vectors following an approach in [40]. To detect the loss of orthogonality, we compare the norm of a vector y to its norm before starting the orthogonalization. If the norm gets very small, it indicates that the vector is almost in the span of the previous vectors which leads to a loss of precision. We re-orthogonalize if

$$\|y\|_2 \leq \xi \|M v_j\|_2$$

where ξ denotes a tolerance, Mv_j is the vector before the orthogonalization, and v_j is the jth Arnoldi vector, i.e. the jth column of V_r. Algorithm 2.4 summarizes the required steps for the low-rank approximation with the Arnoldi method.

In the Arnoldi iteration, we do not need the transposed application of the matrix as opposed to the randomized methods from Section 2.3.1. Furthermore, we only require r instead of $2\ell(q+1)$ applications of the matrix or its transpose to a vector. However, the r applications need to be performed sequentially whereas the applications of the error matrix can be performed to blocks of vectors for the randomized low-rank approximation.

2.4 Notation

In this section, we summarize the notation and abbreviations that we use in this thesis.

Algorithm 2.4: Arnoldi iteration with re-orthogonalization.

Input: Input matrix $M \in \mathbb{R}^{n\times n}$, rank r, tolerance ξ for re-orthogonalization.
Output: Low-rank approximation $V_r H_r V_r^{\mathrm{T}} \approx M$ with $V_r \in \mathbb{R}^{n\times r}$ such that $V_r^{\mathrm{T}} V_r = \mathrm{I}_r$ and $H_r \in \mathbb{R}^{r\times r}$.

```
Draw a random initial vector v_1 of size n with Euclidean norm 1.
for j = 1, ..., r do
    Compute y_j = M v_j.
    for i = 1, ..., j do
        Compute H_{r,i,j} = y_j^T v_i.
        Compute y_j = y_j - H_{r,i,j} v_i.
    end
    Compute H_{r,j+1,j} = ||y_j||_2.
    if ||y_j||_2 > ξ ||M v_j||_2 then
        Compute v_{j+1} = y_j / H_{r,j+1,j}.
    else
        // Re-orthogonalize.
        for i = 1, ..., j do
            Compute h = y_j^T v_i.
            Compute H_{r,i,j} += h.
            Compute y_j = y_j - h v_i.
        end
        Compute h = ||y_j||_2.
        Compute H_{r,j+1,j} += h.
        Compute v_{j+1} = y_j / h.
    end
end
Set V_r = [v_1, v_2, ..., v_r].
```

Function spaces

We now define function spaces, inner products, and norms used in this thesis. For the spatial discretization with the finite element method we will require the following function spaces: The space of square integrable functions is defined as

$$L^2(\Omega) \coloneqq \left\{ v : \Omega \to \mathbb{R} : \int_\Omega |v|^2 \mathrm{dx} < \infty \right\}.$$

This space is a Hilbert space equipped with the L^2-inner product

$$\langle u, v \rangle_\Omega \coloneqq \int_\Omega u \cdot v \mathrm{dx}$$

and the norm

$$\|u\|_{L^2(\Omega)} \coloneqq \int_\Omega |u|^2 \mathrm{dx}$$

for $u, v \in L^2(\Omega)$. We furthermore require the more regular spaces

$$H^1(\Omega) \coloneqq \left\{ v : \Omega \to \mathbb{R} : v \in L^2(\Omega) \text{ and } \nabla v \in L^2(\Omega)^d \right\}$$

and

$$H_0^1(\Omega) \coloneqq \left\{ v : \Omega \to \mathbb{R} : v \in H^1(\Omega) \text{ and } v|_\Gamma = 0 \right\}.$$

Here, $\cdot|_\Gamma$ denotes the restriction to the boundary of the domain Ω. We define the norm

$$\|u\|_{H^1(\Omega)}^2 \coloneqq \|u\|_{L^2(\Omega)} + \|\nabla u\|_{L^2(\Omega)}$$

and the inner product

$$\langle u, v \rangle_{H^1(\Omega)} \coloneqq \langle u, v \rangle_\Omega + \langle \nabla u, \nabla v \rangle_\Omega$$

for $u, v \in H^1(\Omega)$. The spaces $H^1(\Omega)$, $H_0^1(\Omega)$ equipped with the norm $\|\cdot\|_{H^1(\Omega)}^2$ and the inner product $\langle \cdot, \cdot \rangle_{H^1(\Omega)}$ are Hilbert spaces.

Symbols and abbreviations

Small italic letters denote scalar variables and small letters in boldface denote vector-valued variables. Large letters denote matrices. A list of symbols and acronyms is given in the appendix of this thesis.

Chapter 3

A model for atmospheric dynamics

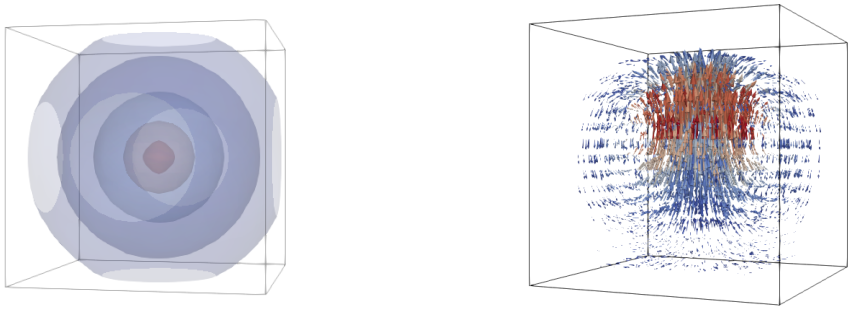
In this chapter, we describe our application of interest, a model for atmospheric dynamics. The dynamics are driven by thermal effects and are described by the rotating Boussinesq equations.

We start in Section 3.1 with a brief description of the Boussinesq approximation. In Section 3.2, we introduce the model equations. Here, we discuss boundary and initial conditions, derive a dimensionless formulation of the rotating Boussinesq equations, and discuss the considered forcings. We conclude the chapter with the problem statement in Section 3.3.

3.1 The Boussinesq approximation

The Boussinesq approximation models buoyancy-driven fluid flows. The Boussinesq model describes dynamics that are driven by temperature-dependant density variations. The physical background of the Boussinesq approximation is for example explained in the books [23] or [24]. The Boussinesq approximation consists traditionally of two main assumptions: neglecting the influence of density variations on the mass balance and neglecting density variations in the momentum balance in all inertial terms. In the Boussinesq approximation, we assume that density variations are significantly smaller than the mean reference density. This assumption can be used for atmospheric dynamics since the main part of density variations are due to hydrostatic pressure effects. So, the temperature-dependant density variations that drive the dynamics are comparatively small. For geophysical flows, relative changes in density are typically smaller than relative changes in velocity. Using this and using that relative changes in density are small, we can replace the mass continuity equation for compressible flows with the volume continuity equation for incompressible flows. This approximation filters out (fast) sound waves that have no direct influence on the dynamics. In the momentum equation, density variations are neglected except for the term that is scaled with gravity which describes the buoyancy forces that induce the dynamics.

In the following, we discuss the model equations for our model for atmospheric dynamics: the rotating Boussinesq equations. Figure 3.1 shows an example velocity field obtained with the rotating Boussinesq equations for a heated ball in the initial temperature field.



(A) Initial temperature field. (B) Velocity field induced by a heated ball.

FIGURE 3.1: Initial temperature field (left) and velocity field (right) for the rotating Boussinesq equations obtained with DEAL.II [6, 5]. The plots are created with `ParaView` [2].

3.2 Model equations

The fluid dynamics, described by the fluid pressure p and the velocity field $\mathbf{u}$, are modeled by the rotating incompressible Navier-Stokes equations with temperature forcing. The rotation is modeled with the Coriolis term $\rho_{\text{ref}}\boldsymbol{\omega} \times \mathbf{u}$ that describes the influence of the earth's rotation on the velocity field. The temperature field T that induces the fluid dynamics is modeled by an advection-diffusion equation. This gives us the following set of partial differential equations,

$$\rho_{\text{ref}}\partial_t \mathbf{u} - \nabla \cdot [2\nu\boldsymbol{\varepsilon}(\mathbf{u})] + \nabla p = \rho(T)\mathbf{g} - \rho_{\text{ref}} (\mathbf{u} \cdot \nabla)\, \mathbf{u} - \rho_{\text{ref}}\boldsymbol{\omega} \times \mathbf{u}, \tag{3.1a}$$

$$\nabla \cdot \mathbf{u} = 0, \tag{3.1b}$$

$$\partial_t T + \nabla \cdot (\kappa \nabla T) = \gamma - (\mathbf{u} \cdot \nabla) T. \tag{3.2}$$

Here, ν denotes the dynamic viscosity of the air, $\boldsymbol{\omega}$ denotes the angular velocity of the earth, γ describes the external heat sources, and κ is the heat diffusivity. The strain-rate tensor $\boldsymbol{\varepsilon}(\cdot)$ is given by $\boldsymbol{\varepsilon}(\mathbf{u}) = \frac{1}{2}\left(\nabla\mathbf{u} + (\nabla\mathbf{u})^{\mathrm{T}}\right)$. We use a linear relation to describe the temperature-dependent density

$$\rho(T) = \rho_{\text{ref}} \left(1 - \beta \left(T - T_{\text{ref}}\right)\right).$$

This is the simplest approximation which is based on the assumption that density variations are small compared to the reference density. The constant ρ_{ref} denotes the density at the reference temperature T_{ref} and β is the heat expansion coefficient.

Often the symmetric operator $\nabla \cdot [2\nu\boldsymbol{\varepsilon}(\mathbf{u})]$ is replaced by the Laplace operator $-\nu\triangle\mathbf{u}$. Both terms are equivalent if the viscosity ν is constant, the second partial derivative of the velocity $\mathbf{u}$ exists and is continuous, and if the fluid is incompressible, i.e. $\nabla \cdot \mathbf{u} = 0$. With these assumptions, we obtain

$$-\nabla \cdot [2\nu\boldsymbol{\varepsilon}(\mathbf{u})] = -\nu\triangle\mathbf{u} - \nu\nabla \cdot (\nabla\mathbf{u})^{\mathrm{T}} = -\nu\triangle\mathbf{u}.$$

The last equality holds since the ith entry of $\nabla \cdot (\nabla\mathbf{u})^{\mathrm{T}}$ is given by

$$\left[\nabla \cdot (\nabla\mathbf{u})^{\mathrm{T}}\right]_i = \sum_j \partial_{x_j} \left[(\nabla\mathbf{u})^{\mathrm{T}}\right]_{i,j} = \sum_j \partial_{x_j}\partial_{x_i}\mathbf{u}_j = \partial_{x_i}\nabla \cdot \mathbf{u} = 0$$

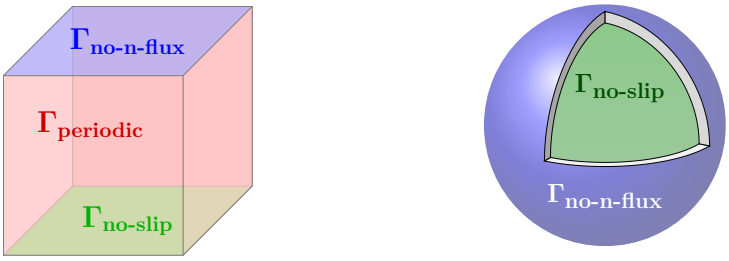


FIGURE 3.2: Splitting of the boundary.

with incompressible conditions. We use the formulation (3.1a) with the symmetric gradient since it yields a physically more accurate formulation. The symmetric gradient includes the coupling of the velocity directions which are neglected in the Laplace operator.

3.2.1 Boundary and initial conditions

In the following, we discuss the boundary and initial conditions for the Boussinesq problem. We start with the Navier-Stokes equations (3.1) and then discuss the conditions for the temperature equation (3.2). We use different boundary conditions for different parts of the boundary Γ of the domain $\Omega \subseteq \mathbb{R}^3$. We consider two domains: a three-dimensional cube and a spherical shell. For the cubic domain, we split the boundary Γ into three parts

$$\Gamma = \Gamma_{\text{no-slip}} \cup \Gamma_{\text{no-n-flux}} \cup \Gamma_{\text{periodic}}.$$

Here, $\Gamma_{\text{no-slip}}$ is the bottom face, $\Gamma_{\text{no-n-flux}}$ the top face, and Γ_{periodic} the side faces of the cube. For the shell, we split the boundary into two parts

$$\Gamma = \Gamma_{\text{no-slip}} \cup \Gamma_{\text{no-n-flux}}$$

where $\Gamma_{\text{no-slip}}$ denotes the inner boundary and $\Gamma_{\text{no-n-flux}}$ denotes the outer boundary of the shell.

Figure 3.2 illustrates the splitting of the boundary for the cube and the spherical shell.

Boundary and initial conditions for the Navier-Stokes equations

We now discuss possible types of boundary conditions that we use for the Navier-Stokes equations.

No-slip condition The no-slip boundary condition describes a solid boundary where the fluid velocity relative to the boundary is zero. This condition is a homogeneous Dirichlet condition for the velocity, i.e.

$$\mathbf{u} = 0 \quad \text{on } \Gamma_{\text{no-slip}}$$

where $\Gamma_{\text{no-slip}}$ denotes the no-slip boundary.

No-normal-flux condition The no-normal-flux condition describes boundaries with only tangential flow, i.e.

$$\mathbf{n} \cdot \mathbf{u} = 0 \qquad \text{on } \Gamma_{\text{no-n-flux}}$$

where $\mathbf{n}$ denotes a unit vector that is normal to the boundary $\Gamma_{\text{no-n-flux}}$. This boundary condition is a Dirichlet condition for the normal part of the velocity. Here, we need a further condition for the tangential part of the traction where we use the condition

$$(\mathrm{I} - \mathbf{n} \otimes \mathbf{n}) \left(\mathbf{n} \cdot \left[p\mathrm{I} - \frac{2}{\mathrm{Re}} \boldsymbol{\varepsilon}(\mathbf{u}) \right] \right) = 0.$$

This type of condition arises in the weak formulation of the Navier-Stokes equations and is applied in weak form. This condition is further discussed in the derivation of the weak formulation in Section 3.4.2.1.

Periodic conditions Periodic boundary conditions describe the periodicity of the variables and are used if we simulate the dynamics on a domain that can be seen as a repeating part of a larger domain. Here, we consider periodic boundary conditions for the velocity and the pressure when we simulate the dynamics on a cube

$$\begin{aligned} \mathbf{u}|_{\Gamma_1} &= \mathbf{u}|_{\Gamma_2}, & \mathbf{u}|_{\Gamma_3} &= \mathbf{u}|_{\Gamma_4}, \\ p|_{\Gamma_1} &= p|_{\Gamma_2}, & p|_{\Gamma_3} &= p|_{\Gamma_4} \end{aligned}$$

where $\Gamma_1 \cup \Gamma_2 \cup \Gamma_3 \cup \Gamma_4 = \Gamma_{\text{periodic}}$ and Γ_1, Γ_2 denote opposite faces of the cube as well as Γ_3, Γ_4. The underlying idea is that the cube describes a part of the spherical shell. The periodic boundary conditions mimic the shell-like geometry of the earth's atmosphere. Note that the periodic conditions are the only pressure boundary conditions that occur in the model. For the Navier-Stokes equations, we further need an initial condition for the fluid velocity. We choose a zero initial velocity on the whole domain

$$\mathbf{u}(0, \cdot) = 0 \quad \text{ on } \Omega.$$

3.2.2 Boundary and initial conditions for the temperature

Now, we discuss the type of temperature boundary and initial conditions of the temperature field in our model (3.1), (3.2). We start with the boundary conditions.

Insulated conditions On an insulated boundary, we do not have any thermal flux. We use an insulated boundary condition on the no-normal-flux boundary

$$\mathbf{n} \cdot \nabla T = 0 \qquad \text{on } \Gamma_{\text{no-n-flux}}.$$

Dirichlet conditions We can prescribe a temperature on the boundary with Dirichlet conditions. We use homogeneous conditions

$$T = 0 \qquad \text{on } \Gamma_{\text{no-slip}}$$

on the no-slip boundary.

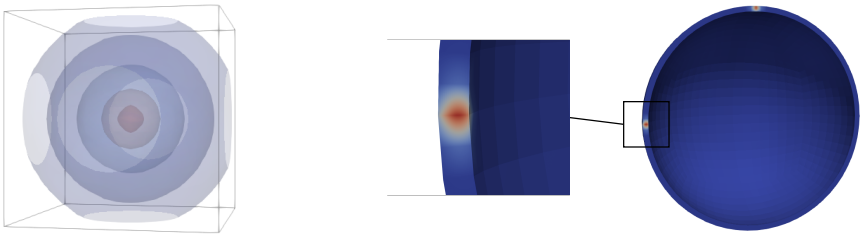


FIGURE 3.3: Initial temperature for the cube and the shell. The plots are created with `ParaView` [2].

Periodic conditions The periodic boundary conditions describe a periodicity in the temperature. As for the velocity and the pressure in the Navier-Stokes equations, we also prescribe periodic conditions for the temperature when we simulate on the cubic domain.

Initial condition The initial temperature field describes warm bubble(s) distributed in the domain. It is inspired by rising bubble tests that are, for example, used in [44]. Figure 3.3 shows qualitatively the initial temperature fields for the cube and the shell. The warm bubbles are modeled as

$$T(\mathbf{x}) = c_0 \sum_{i=0}^{j} \exp\left(-\tfrac{1}{2}(\mathbf{x}-\mathbf{c}_j)^{\mathrm{T}} C (\mathbf{x}-\mathbf{c}_j)\right)$$

where $c_0 > 0$ is a constant, $C \in \mathbb{R}^{d\times d}$ is a matrix, and $\mathbf{c}_j \in \mathbb{R}^d$ are the centers of the warm balls. The matrix C is given as

$$C = \begin{cases} 10/(R_0 - R_1)\mathrm{I}_d & \text{for the shell,} \\ 1/(0.1 d_{\mathrm{cube}})^2 \mathrm{I}_d & \text{for the cube} \end{cases}$$

where d_{cube} is the diameter of the cube and the scalars R_0 and R_1 are the inner and outer radius of the shell. The constant c_0 is given as

$$c_0 = \begin{cases} \sqrt{\det(C)/(2\pi)^d} & \text{for the shell,} \\ \sqrt{\det(C)}/(2\sqrt{2}\pi) & \text{for the cube.} \end{cases}$$

For the cube, we only use one warm ball that is centered at the midpoint of the cube. For the shell, we use two warm balls that are centered at

$$\mathbf{c}_1 = (R_0 + 0.35(R_1 - R_0))\,\mathbf{e}_1, \qquad \mathbf{c}_2 = (R_0 + 0.65(R_1 - R_0))\,\mathbf{e}_2$$

where $\mathbf{e}_i$, $i \in \{1, 2\}$, denotes the d-dimensional unit vector with only zero entries except for the ith entry which is one.

3.2.3 Nondimensionalization

For the nondimensionalization of the Navier-Stokes equations we introduce the scales v_{c}, l_{c} for the velocity $\mathbf{u}$ and the spatial displacement $\mathbf{x}$, i.e.

$$\mathbf{u} = v_{\mathrm{c}}\mathbf{u}', \qquad \mathbf{x} = l_{\mathrm{c}}\mathbf{x}',$$

with the dimensionless velocity $\mathbf{u}'$ and the dimensionless displacement $\mathbf{x}'$. An appropriate scale for the time is then given by

$$t = t_c t' = \frac{l_c}{v_c} t'.$$

Inserting these scales into the Navier-Stokes equations (3.1), we obtain

$$\begin{aligned} \rho_{\text{ref}} \frac{v_c^2}{l_c} \partial_{t'} \mathbf{u}' - \frac{v_c}{l_c^2} \nabla' \cdot \left[2\nu \boldsymbol{\varepsilon}(\mathbf{u}') \right] + \frac{1}{l_c} \nabla' p \\ = \rho(T)\mathbf{g} - \frac{v_c^2}{l_c} \rho_{\text{ref}} \left(\mathbf{u}' \cdot \nabla' \right) \mathbf{u}' - \rho_{\text{ref}} v_c \boldsymbol{\omega} \times \mathbf{u}', \end{aligned} \tag{3.3}$$

$$\frac{v_c}{l_c} \nabla' \cdot \mathbf{u}' = 0. \tag{3.4}$$

Multiplying Equation (3.3) by $l_c/(\rho_{\text{ref}} v_c^2)$ and Equation (3.4) by v_c/l_c and introducing the scales $p_c := \rho_{\text{ref}} v_c^2$ for the pressure p and $g_c := v_c^2/l_c$ for the gravity $\mathbf{g}$, we obtain the dimensionless Navier-Stokes equations

$$\partial_{t'} \mathbf{u}' - \nabla' \cdot \left[\frac{2}{\text{Re}} \boldsymbol{\varepsilon}'(\mathbf{u}') \right] + \nabla' p' = \rho'(T)\mathbf{g}' - \left(\mathbf{u}' \cdot \nabla' \right) \mathbf{u}' - \frac{1}{\text{Ro}} \boldsymbol{\omega}' \times \mathbf{u}', \tag{3.5}$$

$$\nabla' \cdot \mathbf{u}' = 0. \tag{3.6}$$

Here, $\text{Re} := (v_c l_c \rho_{\text{ref}})/\nu$ is the dimensionless Reynolds number, $\text{Ro} := v_c/(l_c \|\boldsymbol{\omega}\|)$ denotes the dimensionless Rossby number and $\boldsymbol{\omega}' = \boldsymbol{\omega}/\|\boldsymbol{\omega}\|$ is the dimensionless earth's angular velocity. The temperature dependant parameter $\rho'(T) := (1 - \beta(T - T_{\text{ref}}))$ denotes the scaled density.

For the nondimensionalization, we furthermore introduce the temperature scaling T_c and obtain

$$\frac{v_c T_c}{l_c} \partial_{t'} T' + \frac{T_c}{l_c^2} \left(\nabla' \cdot \left(\kappa \nabla' T' \right) \right) = \gamma - \frac{v_c T_c}{l_c} \left(\mathbf{u}' \cdot \nabla' \right) T'. \tag{3.7}$$

We multiply Equation (3.7) by $l_c/(v_c T_c)$, define the dimensionless Péclet number $\text{Pe} = (v_c l_c)/\kappa$, and obtain

$$\partial_{t'} T' + \left(\nabla' \cdot \left(\frac{1}{\text{Pe}} \nabla' T' \right) \right) = \gamma' - \left(\mathbf{u}' \cdot \nabla' \right) T' \tag{3.8}$$

where $\gamma' := l_c (v_c T_c)^{-1} \gamma$ is the scaled heat source. Combining Equations (3.5) and (3.8) we obtain the dimensionless set of equations for the Boussinesq approximation

$$\begin{aligned} \partial_{t'} \mathbf{u}' - \nabla' \cdot \left[\frac{2}{\text{Re}} \boldsymbol{\varepsilon}'(\mathbf{u}') \right] + \nabla' p' = \widetilde{\rho}(T)\mathbf{g}' - \left(\mathbf{u}' \cdot \nabla' \right) \mathbf{u}' - \frac{1}{\text{Ro}} \boldsymbol{\omega} \times \mathbf{u}', \\ \nabla' \cdot \mathbf{u}' = 0, \\ \partial_{t'} T' + \left(\nabla' \cdot \left(\frac{1}{\text{Pe}} \nabla' T' \right) \right) = \gamma'(\mathbf{x}') - \left(\mathbf{u}' \cdot \nabla' \right) T'. \end{aligned}$$

3.2.4 Modelling of the forcing

We now discuss the modeling of the dimensionless parameters for the forcings, the gravity vector, the Coriolis term, and the external heat sources.

Gravity The gravity vector shows towards the barycenter of the earth. For the cuboid geometry, we define the dimensionless gravity vector as

$$\mathbf{g}' = \frac{1}{g_c}\mathbf{g} = -\frac{g l_c}{v_c^2}\mathbf{e}_z$$

where $\mathbf{e}_z$ is the upward-showing unit vector in vertical (z-) direction.

For the shell geometry, the gravity vector is computed as

$$\mathbf{g}(\mathbf{x}) = -\frac{g l_c}{v_c^2}\mathbf{x}'/\|\mathbf{x}'\|$$

assuming that the origin of the Cartesian grid coincides with the midpoint of the shell.

Coriolis term We define the dimensionless angular velocity of the earth as

$$\boldsymbol{\omega}' = \frac{\omega l_c}{v_c}\mathbf{e}_z$$

where $\omega = \|\boldsymbol{\omega}\|$ is the angular frequency of the earth's rotation and $\mathbf{e}_z$ describes the direction of the axis of rotation.

Heat sources The external heat sources in the test model are set to zero, i.e. the buoyancy is enforced by the initial temperature differences.

In the remainder of this thesis, we will omit the primes that denote the dimensionless variables for better readability.

3.3 Problem statement

In this section, we summarize the strong formulation of the dimensionless model equations with initial and boundary conditions for the cube and the shell domain.

Problem 3.1 (Rotating Boussinesq model)**.** For a final simulation time $t_f > 0$, the set of model equations for the rotating Boussinesq model is given by:

Find the velocity $\mathbf{u} : [0, t_f] \times \overline{\Omega} \to \mathbb{R}^d$, the pressure $p : [0, t_f] \times \overline{\Omega} \to \mathbb{R}$, and the temperature $T : [0, t_f] \times \overline{\Omega} \to \mathbb{R}$ such that

$$\begin{aligned}
\partial_t \mathbf{u} + (\mathbf{u}\cdot\nabla)\,\mathbf{u} - \nabla\cdot\left[\frac{2}{\mathrm{Re}}\boldsymbol{\varepsilon}(\mathbf{u})\right] + \nabla p &= \widetilde{\rho}(T)\mathbf{g} - (\mathbf{u}\cdot\nabla)\,\mathbf{u} - \frac{1}{\mathrm{Ro}}\boldsymbol{\omega}\times\mathbf{u} && \text{in } (0,t_f]\times\Omega,\\
\nabla\cdot\mathbf{u} &= 0 && \text{in } (0,t_f]\times\Omega,\\
\partial_t T + \left(\nabla\cdot\left(\frac{1}{\mathrm{Pe}}\nabla T\right)\right) &= \gamma - (u\cdot\nabla)\,T && \text{in } (0,t_f]\times\Omega,
\end{aligned}$$

with

$$\begin{aligned}
\mathbf{u}(0,\cdot) &= \mathbf{u}_0 && \text{in } \Omega,\\
\mathbf{u} &= 0 && \text{on } (0,t_f]\times\Gamma_{\text{no-slip}},\\
\mathbf{n}\cdot\mathbf{u} &= 0 && \text{on } (0,t_f]\times\Gamma_{\text{no-n-flux}},\\
(\mathrm{I}-\mathbf{n}\otimes\mathbf{n})\left(\mathbf{n}\cdot\left[p\mathrm{I}-\frac{2}{\mathrm{Re}}\boldsymbol{\varepsilon}(\mathbf{u})\right]\right) &= 0 && \text{on } (0,t_f]\times\Gamma_{\text{no-n-flux}},\\
T(0,\cdot) &= T_0 && \text{in } \Omega,\\
T &= g_T && \text{on } (0,t_f]\times\Gamma_{\text{no-slip}},\\
\mathbf{n}\cdot\nabla T &= 0 && \text{on } (0,t_f]\times\Gamma_{\text{no-n-flux}},
\end{aligned}$$

periodic boundary conditions for $\mathbf{u}$, p, T on Γ_{periodic} for $t\in(0,t_f]$.

The initial velocity field $\mathbf{u}_0$ and the initial temperature field T_0 have to be conforming with the boundary conditions. Furthermore, the initial velocity field $\mathbf{u}_0$ has to be divergence-free. The dimensionless forcings are given by

$$\begin{aligned}
\rho(T) &= 1-\beta(T-T_{\text{ref}}),\\
\boldsymbol{\omega} &= \mathbf{e}_z,\\
\mathbf{g} &= \frac{gl_{\text{c}}}{v_{\text{c}}^2}\begin{cases}\mathbf{e}_z & \text{on a cube,}\\ \mathbf{x}/\|\mathbf{x}\| & \text{on a spherical shell,}\end{cases}
\end{aligned}$$

where $\mathbf{e}_z$ is the unit vector in the vertical (z) direction.

3.4 A solver for the Boussinesq model

Solving the rotating Boussinesq model defined in Problem 3.1 numerically is a demanding task. An efficient solver requires methods for time-stepping, spatial discretization, and linear and nonlinear solvers. These tasks are interconnected and hence cannot be considered completely separately.

In this section, we discuss a numerical solver for the Boussinesq model defined in Problem 3.1. We start in Section 3.4.1 with the temporal discretization of the Boussinesq model using a semi-implicit scheme. We proceed with the spatial discretization utilizing the finite element method in Section 3.4.2. For this, we derive a weak formulation of the model equations, review the nonlinear Picard iteration, and discuss our choice of elements. The fully discrete system combines (sequences of) linear saddle-point system(s) for the velocity and the pressure and a linear symmetric system for the temperature. Then, we discuss adaptive time-stepping related to the Courant-Friedrichs-Lewy condition in Section 3.4.3. In Section 3.4.4, we describe our Boussinesq solver. In the solution scheme, we subsequently solve the discrete Navier-Stokes equations and the temperature advection-diffusion equation in each time step.

3.4.1 Time discretization

We start with the time discretization of the rotating Boussinesq model. For the temporal discretization, we utilize a semi-implicit Euler scheme since it is easy to implement and

typically more stable but less accurate than higher-order schemes. The explicit discretization of some of the terms introduces a Courant-Friedrichs-Lewy (CFL) condition that constrains the time step sizes but offers easier-to-solve systems. In the time discretization we treat the terms that couple the velocity and temperature explicitly. This decouples the Boussinesq equations into two simpler systems, the Navier-Stokes equations and the temperature advection-diffusion equation, such that we can solve both systems separately in each time step. The remaining terms are discretized as follows: We treat the buoyancy forcing $\rho(T)\mathbf{g}$ and the Coriolis term $\boldsymbol{\omega} \times \mathbf{u}$ in the momentum equation and the advection $\mathbf{u} \cdot \nabla T$ in the temperature equation explicitly and the diffusive terms $\nabla \cdot \left[\frac{2}{\mathrm{Re}}\boldsymbol{\varepsilon}(\mathbf{u})\right]$ and $\nabla \cdot \left(\frac{1}{\mathrm{Pe}}\nabla T\right)$ implicitly. For the advection term in the momentum equation, we consider two options: an explicit time discretization which leads to a quasi-Stokes system and an implicit time discretization which leads to a nonlinear Navier-Stokes system. We start with the explicit time discretization of the advection and obtain the time-discrete system

$$\begin{aligned}
\frac{1}{k_n}\left(\mathbf{u}^{n+1} - \mathbf{u}^n\right) - \nabla \cdot \left[\frac{2}{\mathrm{Re}}\boldsymbol{\varepsilon}\left(\mathbf{u}^{n+1}\right)\right] - \nabla p^{n+1} &= \rho(T^n)\mathbf{g} - \left(\mathbf{u}^n \cdot \nabla\right)\mathbf{u}^n - \frac{1}{\mathrm{Ro}}\boldsymbol{\omega} \times \mathbf{u}^n, \\
\nabla \cdot \mathbf{u}^{n+1} &= 0, \\
\frac{1}{k_n}\left(T^{n+1} - T^n\right) - \nabla \cdot \left(\frac{1}{\mathrm{Pe}}\nabla T^{n+1}\right) &= \gamma - \left(\mathbf{u}^n \cdot \nabla\right)T^n,
\end{aligned}$$

where $k_n \coloneqq t_{n+1} - t_n$ is the nth time step size, and the superscripts of $\mathbf{u}$, p, T denote the time step number of the semi-discrete solutions for the velocity, pressure, and temperature. Rearranging leads to

$$\mathbf{u}^{n+1} - \nabla \cdot \left[\frac{2k_n}{\mathrm{Re}}\boldsymbol{\varepsilon}\left(\mathbf{u}^{n+1}\right)\right] - k_n \nabla p^{n+1} = f(\mathbf{u}^n, T^n), \tag{3.9}$$

$$\nabla \cdot \mathbf{u}^{n+1} = 0, \tag{3.10}$$

$$T^{n+1} - \nabla \cdot \left(\frac{k_n}{\mathrm{Pe}}\nabla T^{n+1}\right) = g(\mathbf{u}^n, T^n)$$

where

$$\begin{aligned}
\mathbf{f}_{\mathrm{Stokes}}(\mathbf{u}^n, T^n) &= k_n \rho(T^n)\mathbf{g} - k_n\left(\mathbf{u}^n \cdot \nabla\right)\mathbf{u}^n + \mathbf{u}^n - \frac{k_n}{\mathrm{Ro}}\boldsymbol{\omega} \times \mathbf{u}^n, \\
g(\mathbf{u}^n, T^n) &= k_n \gamma + T^n - k_n\left(\mathbf{u}^n \cdot \nabla\right)T^n.
\end{aligned}$$

An alternative approach is to treat the advection term in the momentum equation implicitly. This may reduce the required time step size constrained by the CFL condition. We obtain the following nonlinear system of equations

$$\mathbf{u}^{n+1} + k_n\left(\mathbf{u}^{n+1} \cdot \nabla\right)\mathbf{u}^{n+1} - \nabla \cdot \left[\frac{2k_n}{\mathrm{Re}}\boldsymbol{\varepsilon}\left(\mathbf{u}^{n+1}\right)\right] - k_n \nabla p^{n+1} = \mathbf{f}_{\mathrm{NSE}}(\mathbf{u}^n, T^n), \tag{3.11}$$

$$\nabla \cdot \mathbf{u}^{n+1} = 0, \tag{3.12}$$

$$T^{n+1} - \nabla \cdot \left(\frac{k_n}{\mathrm{Pe}}\nabla T^{n+1}\right) = g(\mathbf{u}^n, T^n) \tag{3.13}$$

where

$$\mathbf{f}_{\mathrm{NSE}}(\mathbf{u}^n, T^n) = k_n \rho(T^n)\mathbf{g} + \mathbf{u}^n - \frac{k_n}{\mathrm{Ro}}\boldsymbol{\omega} \times \mathbf{u}^n.$$

We linearize the system (3.11)–(3.12) with a nonlinear iteration that we discuss in Section 3.4.2.2.

3.4.2 Spatial discretization with the finite element method

Now, we discuss the spatial discretization with the finite element method. We start with deriving the weak formulation of the Boussinesq model. Then, we discuss its spatial discretization and our choice of finite elements.

3.4.2.1 Weak formulation

Since the semi-implicit time discretization has decoupled the (Navier-)Stokes equations (3.9)–(3.10) or (3.11)–(3.12) and the temperature equation (3.13), we now derive the weak formulations of both problems separately.

We start from the time-discrete Navier-Stokes equations with explicitly treated advection

$$\begin{aligned} \mathbf{u}^{n+1} - \nabla \cdot \left[\frac{2k_n}{\mathrm{Re}} \boldsymbol{\varepsilon}\big(\mathbf{u}^{n+1}\big) \right] - k_n \nabla p^{n+1} &= \mathbf{f}_{\mathrm{Stokes}}(\mathbf{u}^n, T^n), \\ \nabla \cdot \mathbf{u}^{n+1} &= 0. \end{aligned}$$

To derive the weak formulation of the Navier-Stokes equations, we multiply the model equations by test functions

$$\mathbf{v} \in V_u \coloneqq \left\{ v \in H^1(\Omega)^d : v|_{\Gamma_{\text{no-slip}}} = 0, \mathbf{n} \cdot v|_{\Gamma_{\text{no-n-flux}}} = 0 \right\}, \tag{3.14}$$

$$q \in V_p \coloneqq L^2(\Omega), \tag{3.15}$$

and integrate over the domain Ω. The boundary of the domain Ω is split in different parts as described in Section 3.2.1: a no-slip boundary $\Gamma_{\text{no-slip}}$, a no-normal-flux boundary $\Gamma_{\text{no-n-flux}}$, and optionally a periodic boundary Γ_{periodic}. We find

$$\begin{aligned} \big\langle \mathbf{v}, \mathbf{u}^{n+1} \big\rangle - \left\langle \mathbf{v}, \nabla \cdot \left[\frac{2k_n}{\mathrm{Re}} \boldsymbol{\varepsilon}\big(\mathbf{u}^{n+1}\big) \right] \right\rangle - \big\langle \mathbf{v}, k_n \nabla p^{n+1} \big\rangle &= \langle \mathbf{v}, \mathbf{f}_{\mathrm{Stokes}}(\mathbf{u}^n, T^n) \rangle, \\ \big\langle q, \nabla \cdot \mathbf{u}^{n+1} \big\rangle &= 0 \end{aligned}$$

where $\langle \cdot, \cdot \rangle \coloneqq \langle \cdot, \cdot \rangle_\Omega$ denotes the $L^2(\Omega)$-inner product. Since the product of a symmetric and an antisymmetric tensor is zero, we find that

$$\big\langle \nabla \mathbf{v}, \boldsymbol{\varepsilon}\big(\mathbf{u}^{n+1}\big) \big\rangle = \big\langle \boldsymbol{\varepsilon}(\mathbf{v}) + \tfrac{1}{2} \left(\nabla \mathbf{v} - (\nabla \mathbf{v})^{\mathrm{T}} \right), \boldsymbol{\varepsilon}\big(\mathbf{u}^{n+1}\big) \big\rangle = \big\langle \boldsymbol{\varepsilon}(\mathbf{v}), \boldsymbol{\varepsilon}\big(\mathbf{u}^{n+1}\big) \big\rangle .$$

With integration by parts and by using the above identity, we obtain the weak formulation: Find $\mathbf{u}^{n+1} \in V_u$ and $p^{n+1} \in V_p$ such that

$$\begin{aligned} \big\langle \mathbf{v}, \mathbf{u}^{n+1} \big\rangle - \left\langle \mathbf{v}, \nabla \cdot \left[\frac{2k_n}{\mathrm{Re}} \boldsymbol{\varepsilon}\big(\mathbf{u}^{n+1}\big) \right] \right\rangle - \big\langle \mathbf{v}, k_n \nabla p^{n+1} \big\rangle & \\ - \big\langle \mathbf{n} \otimes \mathbf{v}, 2\boldsymbol{\varepsilon}\big(\mathbf{u}^{n+1}\big) \big\rangle_\Gamma + \big\langle \mathbf{n} \cdot \mathbf{v}, p^{n+1} \big\rangle_\Gamma &= \langle \mathbf{v}, \mathbf{f}_{\mathrm{Stokes}}(\mathbf{u}^n, T^n) \rangle, \\ \big\langle q, \nabla \cdot \mathbf{u}^{n+1} \big\rangle &= 0 \end{aligned}$$

for all test functions $\mathbf{v} \in V_u$, $q \in V_p$ and for $t \in [0, t_f]$. The symbol $\otimes$ denotes the outer product and $\langle \cdot, \cdot \rangle_\Gamma$ denotes the $L^2(\Gamma)$-inner product, i.e. an integral over the boundary Γ. To examine the boundary terms, we rearrange them and obtain

$$\begin{aligned}
\left\langle \mathbf{n} \otimes \mathbf{v}, \frac{2}{\mathrm{Re}} \boldsymbol{\varepsilon}\big(\mathbf{u}^{n+1}\big) \right\rangle_\Gamma - \left\langle \mathbf{n} \cdot \mathbf{v}, p^{n+1} \right\rangle_\Gamma &= \sum_{i,j=1}^{d} \left[\left\langle n_i v_j, \frac{2}{\mathrm{Re}} \left[\boldsymbol{\varepsilon}\big(\mathbf{u}^{n+1}\big) \right]_{i,j} \right\rangle - \left\langle n_i v_j, p^{n+1} \delta_{i,j} \right\rangle \right] \\
&= \sum_{i,j=1}^{d} \left[\left\langle n_i v_j, \frac{2}{\mathrm{Re}} \left[\boldsymbol{\varepsilon}\big(\mathbf{u}^{n+1}\big) \right]_{i,j} - p^{n+1} \delta_{i,j} \right\rangle \right] \\
&= \left\langle \mathbf{n} \otimes \mathbf{v}, \frac{2}{\mathrm{Re}} \boldsymbol{\varepsilon}\big(\mathbf{u}^{n+1}\big) - p^{n+1} \mathrm{I} \right\rangle_\Gamma .
\end{aligned}$$

The boundary terms hence vanish for no-slip and no-normal-flux boundary conditions. The weak formulation for the time-discrete Navier-Stokes equations with implicitly treated advection (3.11)–(3.12) is derived analogously. We multiply by test function $v \in V_u$, $q \in V_p$ and integrate over the domain Ω and obtain the weak problem:

Find $\mathbf{u}^{n+1} \in V_u$ and $p^{n+1} \in V_p$ such that

$$\begin{aligned}
\left\langle \mathbf{v}, \mathbf{u}^{n+1} \right\rangle + \left\langle \mathbf{v}, k_n \left(\mathbf{u}^{n+1} \cdot \nabla \right) \mathbf{u}^{n+1} \right\rangle - \left\langle \mathbf{v}, \nabla \cdot \left[\frac{2k_n}{\mathrm{Re}} \boldsymbol{\varepsilon}\big(\mathbf{u}^{n+1}\big) \right] \right\rangle - \left\langle \mathbf{v}, k_n \nabla p^{n+1} \right\rangle & \\
- \left\langle \mathbf{n} \otimes \mathbf{v}, 2\boldsymbol{\varepsilon}\big(\mathbf{u}^{n+1}\big) \right\rangle_\Gamma + \left\langle \mathbf{n} \cdot \mathbf{v}, p^{n+1} \right\rangle_\Gamma & \\
= \left\langle \mathbf{v}, \mathbf{f}_{\mathrm{NSE}}(\mathbf{u}^n, T^n) \right\rangle , & \qquad (3.17a) \\
\left\langle q, \nabla \cdot \mathbf{u}^{n+1} \right\rangle = 0 & \qquad (3.17b)
\end{aligned}$$

for all test functions $\mathbf{v} \in V_u$, $q \in V_p$ and for $t \in [0, t_f]$. The boundary terms vanish as for the quasi-Stokes equations. Note that this system is nonlinear in $\mathbf{u}^{n+1}$. We discuss the treatment of the nonlinear term $\left\langle \mathbf{v}, k_n \left(\mathbf{u}^{n+1} \cdot \nabla \right) \mathbf{u}^{n+1} \right\rangle$ in Section 3.4.2.2.

To derive a weak formulation of the temperature equation, we multiply Equation (3.13) by a test function

$$\tau \in V_T \coloneqq \left\{ v \in H^1(\Omega) : v|_{\Gamma_{\text{no-slip}}} = 0 \right\}, \qquad (3.18)$$

integrate over the domain Ω, and obtain

$$\left\langle \tau, T^{n+1} \right\rangle - \left\langle \tau, \nabla \cdot \left(\frac{k_n}{\mathrm{Pe}} \nabla T^{n+1} \right) \right\rangle = \left\langle \tau, g(\mathbf{u}^n, T^n) \right\rangle .$$

Integration by parts and rearranging lead to the weak formulation:

Find $T^{n+1} \in V_{T,g} \coloneqq \left\{ v \in H^1(\Omega) : v|_{\Gamma_{\text{no-slip}}} = g \right\}$ such that

$$\left\langle \tau, T^{n+1} \right\rangle - \left\langle \tau, \nabla \cdot \left(\frac{k_n}{\mathrm{Pe}} \nabla T^{n+1} \right) \right\rangle - \left\langle \tau, \mathbf{n} \cdot \left(\kappa \nabla T^{n+1} \right) \right\rangle_\Gamma = \left\langle \tau, g(\mathbf{u}^n, T^n) \right\rangle . \qquad (3.19)$$

The boundary term vanishes for insulated boundary conditions $\mathbf{n} \cdot (\kappa \nabla T) = 0$ as well as for homogeneous Dirichlet conditions.

3.4.2.2 Nonlinear iteration

The numerical simulation of the nonlinear Navier-Stokes equations (3.17) requires a linearization of the nonlinear advection term $(\mathbf{u}^{n+1} \cdot \nabla)\, \mathbf{u}^{n+1}$. Typical linearization approaches are Newton's method or Picard-type fixed point iterations. We employ a Picard correction iteration for the linearization that provides an easy-to-assess stopping criterion. Compared to Newton's method, the Picard (correction) iteration has a larger radius of convergence [26] but provides slower convergence in terms of iteration numbers [42]. However, for small time step sizes the initial guess is typically close to the solution of the nonlinear iterations, so typically only a few Picard iterations are sufficient until convergence is achieved [41]. To linearize the Navier-Stokes equations, we start with a Picard-type fixed point iteration and then reformulate the equations to obtain the Picard correction iteration. The Picard iteration starts with the results for the velocity and the pressure of the previous time step $\mathbf{u}_0^{n+1} = \mathbf{u}^n$, $p_0^{n+1} = p^n$. We determine the $(m+1)$th Picard iterate by solving

$$\begin{aligned}
&\left\langle \mathbf{v}, \mathbf{u}_{m+1}^{n+1} \right\rangle + \left\langle \mathbf{v}, k_n \left(\mathbf{u}_m^{n+1} \cdot \nabla \right) \mathbf{u}_{m+1}^{n+1} \right\rangle \\
&\quad - \left\langle \mathbf{v}, \nabla \cdot \left[\frac{2k_n}{\mathrm{Re}} \boldsymbol{\varepsilon}\big(\mathbf{u}_{m+1}^{n+1}\big) \right] \right\rangle - \left\langle \mathbf{v}, k_n \nabla p_{m+1}^{n+1} \right\rangle = \left\langle \mathbf{v}, \mathbf{f}_{\mathrm{NSE}}(\mathbf{u}^n, T^n) \right\rangle, \qquad (3.20a) \\
&\left\langle q, \nabla \cdot \mathbf{u}_{m+1}^{n+1} \right\rangle = 0. \qquad (3.20b)
\end{aligned}$$

Since the boundary terms vanish for our problem, we omit them here. We now split the velocity and pressure iterate into the solution of the previous iteration and a correction

$$\begin{aligned}
\mathbf{u}_{m+1}^{n+1} &= \mathbf{u}_m^{n+1} + \delta \mathbf{u}_m^{n+1}, \qquad (3.21a) \\
p_{m+1}^{n+1} &= p_m^{n+1} + \delta p_m^{n+1} \qquad (3.21b)
\end{aligned}$$

where the subscript m denotes the number of the current nonlinear iteration. To find the Picard correction, we insert the expressions (3.21) for the velocity and the pressure into the Picard-linearized Navier-Stokes equations (3.20), and obtain

$$\begin{aligned}
&\left\langle \mathbf{v}, \mathbf{u}_m^{n+1} + \delta \mathbf{u}_m^{n+1} \right\rangle + \left\langle \mathbf{v}, k_n \left(\mathbf{u}_m^{n+1} \cdot \nabla \right) \left(\mathbf{u}_m^{n+1} + \delta \mathbf{u}_m^{n+1} \right) \right\rangle \\
&\quad - \left\langle \mathbf{v}, \nabla \cdot \left[\frac{2k_n}{\mathrm{Re}} \boldsymbol{\varepsilon}\big(\mathbf{u}_m^{n+1} + \delta \mathbf{u}_m^{n+1}\big) \right] \right\rangle \\
&\qquad - \left\langle \mathbf{v}, k_n \nabla \left(p_m^{n+1} + \delta p_m^{n+1} \right) \right\rangle = \left\langle \mathbf{v}, \mathbf{f}_{\mathrm{NSE}}(\mathbf{u}^n, T^n) \right\rangle, \\
&\qquad \left\langle q, \nabla \cdot \mathbf{u}_m^{n+1} + \delta \mathbf{u}_m^{n+1} \right\rangle = 0.
\end{aligned}$$

The iterates $\mathbf{u}_m^{n+1}$ and p_m^{n+1} are known from the previous iteration. We rearrange all known terms to the right-hand side and obtain

$$\begin{aligned}
&\left\langle \mathbf{v}, \delta \mathbf{u}_m^{n+1} \right\rangle + \left\langle \mathbf{v}, k_n \left(\mathbf{u}_m^{n+1} \cdot \nabla \right) \delta \mathbf{u}_m^{n+1} \right\rangle \\
&\quad - \left\langle \mathbf{v}, \nabla \cdot \left[\frac{2k_n}{\mathrm{Re}} \boldsymbol{\varepsilon}\big(\delta \mathbf{u}_m^{n+1}\big) \right] \right\rangle - \left\langle \mathbf{v}, k_n \nabla \delta \mathbf{u}_m^{n+1} \right\rangle = \left\langle \mathbf{v}, \mathbf{r}_u(\mathbf{u}^n, \mathbf{u}_m^{n+1}, T^n, T_m^{n+1}) \right\rangle, \\
&\qquad \left\langle q, \nabla \cdot \mathbf{u}_m^{n+1} + \delta \mathbf{u}_m^{n+1} \right\rangle = \left\langle q, r_p(\mathbf{u}_m^{n+1}) \right\rangle.
\end{aligned}$$

where the right-hand sides are given by

$$\begin{aligned}\mathbf{r}_u(\mathbf{u}^n, \mathbf{u}_m^{n+1}, T^n, T_m^{n+1}) = \left\langle \mathbf{v}, \mathbf{f}_{\mathrm{NSE}}(\mathbf{u}^n, T^n)\right\rangle - \left\langle \mathbf{v}, \mathbf{u}_m^{n+1}\right\rangle - \left\langle \mathbf{v}, k_n \left(\mathbf{u}_m^{n+1}\cdot\nabla\right)\mathbf{u}_m^{n+1}\right\rangle \\ + \left\langle \mathbf{v}, \nabla\cdot\left[\frac{2k_n}{\mathrm{Re}}\boldsymbol{\varepsilon}\left(\mathbf{u}_m^{n+1}\right)\right]\right\rangle + \left\langle \mathbf{v}, k_n\left(\nabla p_m^{n+1}\right)\right\rangle,\\ r_p(\mathbf{u}_m^{n+1}) = -\left\langle q, \nabla\cdot\mathbf{u}_m^{n+1}\right\rangle.\end{aligned}$$

The right-hand sides $\mathbf{r}_u$ and r_p are the residuals of the nonlinear iteration and can be used to define a stopping criterion of the iteration. We find the Picard corrections $\delta\mathbf{u}_m^{n+1}$, δp_m^{n+1} by solving

$$\begin{aligned}\left\langle \mathbf{v}, \delta\mathbf{u}_m^{n+1}\right\rangle + \left\langle \mathbf{v}, k_n\left(\mathbf{u}_m^{n+1}\cdot\nabla\right)\delta\mathbf{u}_m^{n+1}\right\rangle - \left\langle \mathbf{v}, \nabla\cdot\left[\tfrac{2k_n}{\mathrm{Re}}\boldsymbol{\varepsilon}\left(\delta\mathbf{u}_m^{n+1}\right)\right]\right\rangle - \left\langle \mathbf{v}, k_n\delta p_m^{n+1}\right\rangle \\ = \left\langle \mathbf{v}, \mathbf{r}_u(\mathbf{u}^n, \mathbf{u}_m^{n+1}, T^n, T_m^{n+1})\right\rangle,\\ \left\langle q, \nabla\cdot\delta\mathbf{u}_m^{n+1}\right\rangle = \left\langle q, r_p(\mathbf{u}_m^{n+1})\right\rangle.\end{aligned}$$

This leads to a sequence of linear saddle-point systems per time step.

3.4.2.3 Spatial discretization

For the spatial discretization with the finite element method, we replace the domain Ω with a computational domain $\Omega_h \subset \Omega$. We then choose a suitable mesh $\mathcal{T}_h \subset \Omega_h$. Furthermore, we restrict the solution and test spaces defined in (3.14), (3.15), and (3.18) to the finite-dimensional spaces $V_{u,h} \subset V_u$, $V_{p,h} \subset V_p$, $V_{T,h} \subset V_T$. We choose the inf-sup stable Taylor-Hood elements [17] for the Navier-Stokes equations, i.e. continuous piecewise polynomials of degree $q_p \geq 1$ for the pressure and continuous piecewise polynomials of degree $q_p + 1$ for the velocity. The inf-sup stability is required for the well-posedness of the discretized Navier-Stokes equations. We refer to [42] for more background on finite element methods for incompressible fluid flows including results on conditions for the well-posedness. For the temperature, we choose the same polynomial degree as for the velocity. This choice of basis functions is also used in a model for earth mantle convection [44] that is based on a similar set of equations. The test problems for our numerical experiments are obtained with Taylor-Hood elements of the lowest order, i.e. we set $q_p = 1$. The finite-dimensional spaces for Taylor-Hood elements are given by

$$\begin{aligned}V_{u,h} &= \left\{\mathbf{v} \in V_u : \mathbf{v}|_K \in [\mathbb{Q}_{q_p+1}]^d \text{ for all } K \in \mathcal{T}_h\right\},\\ V_{p,h} &= \left\{q \in V_p : q|_K \in \mathbb{Q}_{q_p} \text{ for all } K \in \mathcal{T}_h\right\},\end{aligned}$$

where $\mathbb{Q}_{q_p}$ is the continuous piecewise q_pth order polynomial function space and K is a cell of the grid $\mathcal{T}_h$.

We discretize the temperature with continuous piecewise polynomials of degree $q_T = q_p + 1$. The space $V_{T,h}$ is given by

$$V_{T,h} = \left\{\tau \in V_T : \tau|_K \in \mathbb{Q}_{q_T} \text{ for all } K \in \mathcal{T}_h\right\}.$$

With these finite element spaces, we obtain the fully discrete formulation of the quasi-Stokes equations (3.16) as:

Find $\mathbf{u}_h^{n+1} \in V_{u,h}$ and $p_h^{n+1} \in V_{p,h}$ such that

$$\begin{aligned}\left\langle \mathbf{v}_h, \mathbf{u}_h^{n+1} \right\rangle - \left\langle \mathbf{v}_h, \nabla \cdot \left[\tfrac{2k_n}{\mathrm{Re}} \boldsymbol{\varepsilon}\left(\mathbf{u}_h^{n+1}\right) \right] \right\rangle - \left\langle \mathbf{v}_h, k_n \nabla p^{n+1} \right\rangle \\ - \left\langle \mathbf{n} \otimes \mathbf{v}, 2\boldsymbol{\varepsilon}\left(\mathbf{u}_h^{n+1}\right) \right\rangle_\Gamma + \left\langle \mathbf{n} \cdot \mathbf{v}_h, p_h^{n+1} \right\rangle_\Gamma = \left\langle \mathbf{v}_h, \mathbf{f}_{\mathrm{Stokes}}(\mathbf{u}_h^n, T_h^n) \right\rangle, \qquad (3.22a)\end{aligned}$$

$$\left\langle q_h, \nabla \cdot \mathbf{u}_h^{n+1} \right\rangle = 0 \qquad (3.22b)$$

for all $\mathbf{v}_h \in V_{u,h}$ and all $q_h \in V_{p,h}$.

The fully discrete Navier-Stokes equations in the $(m+1)$th Picard iteration read:
Find $\mathbf{u}_{h,m+1}^{n+1} \in V_{u,h}$ and $p_{h,m+1}^{n+1} \in V_{p,h}$ such that

$$\begin{aligned}\left\langle \mathbf{v}_h, \mathbf{u}_{h,m+1}^{n+1} \right\rangle + \left\langle \mathbf{v}_h, k_n \left(\mathbf{u}_{h,m}^{n+1} \cdot \nabla \right) \mathbf{u}_{h,m+1}^{n+1} \right\rangle - \left\langle \mathbf{v}_h, \nabla \cdot \left[\tfrac{2k_n}{\mathrm{Re}} \boldsymbol{\varepsilon}\left(\mathbf{u}_{h,m+1}^{n+1} \right) \right] \right\rangle - \left\langle \mathbf{v}_h, k_n \nabla p_h^{n+1} \right\rangle \\ - \left\langle \mathbf{n} \otimes \mathbf{v}_h, 2\boldsymbol{\varepsilon}\left(\mathbf{u}_h^{n+1}\right) \right\rangle_\Gamma + \left\langle \mathbf{n} \cdot \mathbf{v}_h, p_h^{n+1} \right\rangle_\Gamma = \left\langle \mathbf{v}_h, \mathbf{f}_{\mathrm{NSE}}(\mathbf{u}_h^n, T_h^n) \right\rangle, \\ \left\langle q_h, \nabla \cdot \mathbf{u}_h^{n+1} \right\rangle = 0\end{aligned}$$

for all $\mathbf{v}_h \in V_{u,h}$ and all $q_h \in V_{p,h}$. We insert the discrete functions in the weak formulation of the temperature equation (3.19) and obtain

$$\left\langle \tau_h, T_h^{n+1} \right\rangle - \left\langle \tau_h, \nabla \cdot \left(\frac{k_n}{\mathrm{Pe}} \nabla T_h^{n+1} \right) \right\rangle - \left\langle \tau_h, \mathbf{n} \cdot \left(\kappa \nabla T_h^{n+1} \right) \right\rangle_\Gamma = \left\langle \tau_h, g(\mathbf{u}_h, T_h^n) \right\rangle. \qquad (3.24)$$

We construct the discrete solutions at each time step as linear combinations of the basis functions of the finite-dimensional spaces

$$\begin{aligned}\mathbf{u}_h^{n+1}(\mathbf{x}) &= \sum_{i=1}^{n_u} U_i^{n+1} \boldsymbol{\phi}_i^u(\mathbf{x}), \\ p_h^{n+1}(\mathbf{x}) &= \sum_{i=1}^{n_p} P_i^{n+1} \phi_i^p(\mathbf{x}), \\ T_h^{n+1}(\mathbf{x}) &= \sum_{i=1}^{n_T} T_i^{n+1} \phi_i^{\mathrm{T}}(\mathbf{x})\end{aligned}$$

where $\boldsymbol{\phi}_i^u \in V_{u,h}$, $\phi_i^p \in V_{p,h}$, $\phi_i^{\mathrm{T}} \in V_{T,h}$ are the basis functions for the velocity space $V_{u,h}$, the pressure space $V_{p,h}$, and the temperature space $V_{T,h}$, respectively. The superscripts of the coefficients U_i^{n+1}, P_i^{n+1}, T_i^{n+1} correspond to the time step number. The numbers n_u, n_p, and n_T denote the number of degrees of freedom for the velocity, the pressure, and the temperature.

Inserting the basis representations of the discrete velocity, pressure, and temperature in the discrete weak quasi-Stokes equations (3.22), we find the following saddle-point problem

$$\begin{pmatrix} A_{\mathrm{Stokes}} & B^{\mathrm{T}} \\ B & 0 \end{pmatrix} \begin{pmatrix} \mathbf{u}^{n+1} \\ \widetilde{p}^{n+1} \end{pmatrix} = \begin{pmatrix} \mathbf{f}_{\mathrm{Stokes}} \\ g \end{pmatrix} \qquad (3.25)$$

with the scaled pressure $\widetilde{p}_h^{n+1} \coloneqq k_n p_h^{n+1}$. The coefficient vectors $\mathbf{u}^{n+1}$ and $\widetilde{p}^{n+1}$ hold the coefficients U_i^{n+1} and $(k_n P_i^{n+1})$. The matrix blocks are given as

$$A_{\mathrm{Stokes}} = M_u + \tfrac{2k_n}{Re} M_{\mathrm{diff}} \qquad (3.26)$$

with

$$\begin{aligned}[M_u]_{ij} &= \left\langle \boldsymbol{\phi}_i^u, \boldsymbol{\phi}_j^u \right\rangle, \\ [M_{\text{diff}}]_{ij} &= \left\langle \boldsymbol{\varepsilon}(\boldsymbol{\phi}_i^u), \boldsymbol{\varepsilon}(\boldsymbol{\phi}_j^u) \right\rangle, \\ [B^{\text{T}}]_{ij} &= \left\langle \nabla \cdot \boldsymbol{\phi}_i^u, \phi_j^p \right\rangle.\end{aligned}$$

The right-hand side is calculated with

$$[\mathbf{f}_{\text{Stokes}}]_i = \left\langle \boldsymbol{\phi}_i^u, k_n \rho(T_h^n)\mathbf{g} + \mathbf{u}_h^n - k_n \left(\mathbf{u}_h^n \cdot \nabla\right) \mathbf{u}_h^n - \frac{k_n}{\text{Ro}} \boldsymbol{\omega} \times \mathbf{u}_h^n \right\rangle.$$

With our choice of boundary conditions, we furthermore have

$$g = 0.$$

For the linearized Navier-Stokes equations (3.23), we obtain a sequence of linear saddle-point systems per time step. The saddle-point system for the $(m+1)$th Picard iterate in the $(n+1)$th time step is given by

$$\begin{pmatrix} A_{\text{NSE}}^m & B^{\text{T}} \\ B & 0 \end{pmatrix} \begin{pmatrix} \mathbf{u}_{m+1}^{n+1} \\ \widetilde{p}_{m+1}^{n+1} \end{pmatrix} = \begin{pmatrix} \mathbf{f}_{\text{NSE}} \\ g \end{pmatrix} \tag{3.27}$$

where the matrix block A_{NSE}^m is determined by

$$A_{\text{NSE}}^m = M_u + \tfrac{2k_n}{Re} M_{\text{diff}} + k_n M_{\text{adv}}^m \tag{3.28}$$

with

$$[M_{\text{adv}}^m]_{ij} = \left\langle \boldsymbol{\phi}_i^u, (\mathbf{u}_{h,m}^{n+1} \cdot \nabla) \boldsymbol{\phi}_j^u \right\rangle.$$

The matrices M_u and M_{diff} are defined as in Equation (3.26) for the quasi-Stokes equations. The right-hand side is calculated with

$$[\mathbf{f}_{\text{NSE}}]_i = \left\langle \boldsymbol{\phi}_i^u, k_n \rho(T_h^n)\mathbf{g} + \mathbf{u}_h^n - \frac{k_n}{\text{Ro}} \boldsymbol{\omega} \times \mathbf{u}_h^n \right\rangle$$

and $g = 0$ as above.

In the following chapters, we will refer to the quasi-Stokes systems of the form (3.25) as Stokes systems and to the Picard-linearized systems of the form (3.27) as Oseen systems.

Since the pressure is only defined up to a constant, the matrix blocks $B \in \mathbb{R}^{n_p \times n_u}$ and $B^{\text{T}} \in \mathbb{R}^{n_u \times n_p}$ have rank $n_p - 1$ and hence the saddle-point matrices have a non-trivial kernel. We discuss in Section 3.5.3 how we adapt the systems or solvers to deal with the non-trivial kernel.

Inserting the basis functions of the space V_h^{T} as test functions leads to the linear system

$$MT^{n+1} = f \tag{3.29}$$

with

$$M = M_T + \frac{k_n}{\text{Pe}} D_T.$$

The coefficient vector T^{n+1} holds the n_T coefficients T_i^{n+1}. The matrices M_T and D_T are calculated with

$$[M_T]_{ij} = \left\langle \phi_i^{\mathrm{T}}, \phi_j^{\mathrm{T}} \right\rangle,$$
$$[D_T]_{ij} = \left\langle \phi_i^{\mathrm{T}}, \nabla \cdot \left(\nabla \phi_j^{\mathrm{T}} \right) \right\rangle.$$

The right-hand side is determined with

$$[f]_i = \left\langle \boldsymbol{\phi}_i^{\mathrm{T}}, k_n \gamma + T_h^n - k_n \left(\mathbf{u}_h^n \cdot \nabla \right) T_h^n \right\rangle.$$

The system matrix M is symmetric positive definite and we thus expect that the temperature system can be solved efficiently.

3.4.3 CFL condition for adaptive time-stepping

The allowed time step sizes in the semi-implicit time discretization are limited by a Courant-Friedrichs-Lewy (CFL) condition. The time step size k_n, the cell diameter h_K, and the maximum local velocities should satisfy the CFL condition

$$\frac{\|\mathbf{u}^n\|_{L_\infty(K)} k_n}{h_K} \leq C$$

for an appropriate constant C on each cell $K \in \mathcal{T}_h$. The constant C is related to the time-stepping method and satisfies $C \leq 1$ if we use explicit time discretization for some terms of the model equation(s). An appropriate time step size k_n should thus satisfy

$$k_n \leq \frac{C h_K}{\|\mathbf{u}\|_{L_\infty(K)}}$$

for all cells $K \in \mathcal{T}_h$. A time step size that satisfies this condition on all cells is given by

$$k_n = C \min_{K \in \mathcal{T}_h} \frac{h_K}{\|\mathbf{u}\|_{L_\infty(K)}}.$$

We motivate now an expression for the unknown constant C which is an adapted version of the approach used in [7]. The time step sizes are not restricted by the cell diameters h_K but by the distance of grid points which is given by $h_K/(\sqrt{d}q)$ where d is the spatial dimension of the domain and q is the maximum polynomial degree of basis functions. Here, q is the maximum of the polynomial degrees q_u, q_T of the basis functions used in the discretization of the velocity and temperature. Figure 3.4 shows this difference exemplarily for quadratic basis functions on a square two-dimensional cell. We hence replace the formula for the time step size by

$$k_n = \widetilde{C} \min_{K \in \mathcal{T}} \frac{h_K}{\sqrt{d} \max\{q_T, q_u\} \|\mathbf{u}\|_{L_\infty(K)}}. \tag{3.30}$$

Furthermore, the explicitly treated terms affect the constant $\widetilde{C}$. We set the constant $\widetilde{C}$ to

$$\widetilde{C} = \frac{s}{2.1d}$$

where s is a constant chosen as 1 for two dimensions and as 0.25 for three dimensions, and d is the spatial dimension of the domain. This choice for the constant $\widetilde{C}$ is taken from

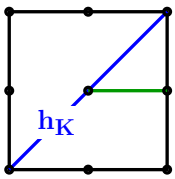


FIGURE 3.4: Cell diameter h_K (blue) and distance of grid points (green) for quadratic basis functions on a square two-dimensional cell. The circles denote support points of the basis functions.

[7] and resulted from numerical tests for a model for earth mantle convection. The time step size might be adapted to the time-stepping scheme and the set of equations that we use. However, the adaptation of time step sizes for the semi-implicit Euler scheme for the rotating Boussinesq equations is not the subject of this thesis and is hence not further discussed.

3.4.4 The Boussinesq solver

In each time step, we obtain a linear temperature system (3.29) and a linear system or a sequence of linear systems for the Navier-Stokes equations (3.25) or (3.27). In each time step, we assemble the Navier-Stokes system and the temperature system and then solve the Navier-Stokes equations, followed by the temperature equation. The numerical solution of the saddle-point system(s) for the discrete Navier-Stokes equations is demanding. We use an iterative solver accelerated with a block preconditioner. Suitable preconditioners are described in the following Section 3.5. The temperature system matrix is symmetric positive definite. Solving the system with the conjugate gradient method and using the inverse diagonal of the system matrix as a preconditioner is already sufficiently efficient. Algorithm 3.1 summarizes the required steps for solving the Boussinesq model numerically.

Table 3.1 gives an overview of how the solver time is distributed on the different solving steps of the Boussinesq solver exemplarily for the Picard-linearized Navier-Stokes equations. The equations are solved on a unit cube ($n_u = 107811$, $n_p = 4913$, $n_T = 35937$) with adaptive time stepping. The (dimensionless) initial time step size is $k_0 = 1\text{e}{-4}$. The shown solver times are obtained for a final time of $k_f = 1\text{e}{-2}$. A simulation of this period involves 116 time steps and requires the solution to 235 saddle-point problems where we stop the Picard iterations when the relative residual drops below 1e−8. Except for the first time step, the Picard-type iteration converges typically after two iterations in this simulation. We solve the Oseen-type systems with restarted GMRes with a restart length of 40 preconditioned with a block triangular preconditioner (3.35) with a block triangular incomplete LU preconditioner (3.38) for the upper left matrix block and the SIMPLE preconditioner (3.44) approximated with an incomplete Cholesky factorization for the Schur complement. The incomplete factorizations are computed without fill-in. We reuse the SIMPLE preconditioner constructed based on the first saddle-point system, i.e. a Stokes system, for preconditioning the remaining saddle-point systems. A further discussion of this approach is given in Sections 3.5.3 and 5.2. More details about the preconditioners are given in the following Section 3.5. The temperature equation is solved with the conjugate gradient method with the inverse diagonal of the system matrix as a preconditioner. We observe that the numerical solution of the Navier-Stokes equations clearly dominates the

solver times. The assembly is another significant part of the simulation which we omit here since it is not optimized in the implementation. We hence focus on preconditioners for saddle-point systems as obtained for the quasi-Stokes system or the Navier-Stokes system in the remainder of this thesis. In the following Section 3.5, we review common block preconditioners for saddle-point systems in fluid flow problems that can be applied to the discrete quasi-Stokes and Oseen-type systems.

TABLE 3.1: Distribution of the solver time with initial time step size $k_0 = 1\text{e}{-4}$ and a dimensionless simulation time of $k_\text{f} = 1\text{e}{-2}$ with adaptive time stepping (116 time steps) on a cube ($n_u = 107811$, $n_p = 4913$, $n_T = 35937$).

task	time in seconds	percentage
Solve Navier-Stokes equations	437	98.7
Solve temperature equation	5.71	1.3

3.5 Block preconditioners for saddle-point problems

We now review standard block preconditioners for saddle-point systems of the form

$$\begin{pmatrix} A & B^\text{T} \\ B & 0 \end{pmatrix} \begin{pmatrix} \mathbf{u} \\ p \end{pmatrix} = \begin{pmatrix} \mathbf{f} \\ g \end{pmatrix}. \tag{3.31}$$

These saddle-point systems arise in the discretization of the linearized Navier-Stokes equations. The block preconditioners are based on (approximate) block factorizations of the system matrix of the saddle-point problem.

The system matrix can be factorized as

$$\begin{pmatrix} A & B^\text{T} \\ B & 0 \end{pmatrix} = \begin{pmatrix} A & 0 \\ B & -S \end{pmatrix} \begin{pmatrix} \text{I}_{n_u} & A^{-1}B^\text{T} \\ 0 & \text{I}_{n_p} \end{pmatrix} \tag{3.32}$$

$$= \begin{pmatrix} \text{I}_{n_u} & 0 \\ BA^{-1} & \text{I}_{n_p} \end{pmatrix} \begin{pmatrix} A & B^\text{T} \\ 0 & -S \end{pmatrix} \tag{3.33}$$

$$= \begin{pmatrix} \text{I}_{n_u} & 0 \\ BA^{-1} & \text{I}_{n_p} \end{pmatrix} \begin{pmatrix} A & 0 \\ 0 & -S \end{pmatrix} \begin{pmatrix} \text{I}_{n_u} & A^{-1}B^\text{T} \\ 0 & \text{I}_{n_p} \end{pmatrix} \tag{3.34}$$

where $S = BA^{-1}B^\text{T}$ is the (negative) pressure Schur complement. With the factorizations (3.32) and (3.33), we find the ideal lower and upper triangular preconditioners

$$P_\text{L,ideal} = \begin{pmatrix} A^{-1} & 0 \\ S^{-1}BA^{-1} & -S^{-1} \end{pmatrix},$$

$$P_\text{U,ideal} = \begin{pmatrix} A^{-1} & A^{-1}B^\text{T}S^{-1} \\ 0 & -S^{-1} \end{pmatrix}$$

and with the factorization (3.34) the block diagonal preconditioner

$$P_\text{D,ideal} = \begin{pmatrix} A^{-1} & 0 \\ 0 & S^{-1} \end{pmatrix}.$$

With the sign choice for the bottom right block of the block diagonal preconditioner $P_{\mathrm{D,ideal}}$, we obtain that the nonsingular preconditioned matrix has at most three distinct real eigenvalues [52]. One can show that GMRes converges within at most two iterations with the ideal block triangular preconditioners [66] and within at most three iterations with the ideal block diagonal preconditioner [11, 52].

The ideal block preconditioners are not practical since they require the inverses of the upper left matrix block A and the pressure Schur complement $S = BA^{-1}B^{\mathrm{T}}$. To obtain a more practical preconditioner, we replace the inverses by approximations $\widehat{A}^{-1} \approx A^{-1}$ and $\widehat{S}^{-1} \approx S^{-1}$. We obtain the block triangular preconditioners

$$P_{\mathrm{L}} = \begin{pmatrix} \widehat{A}^{-1} & 0 \\ \widehat{S}^{-1}B\widehat{A}^{-1} & -\widehat{S}^{-1} \end{pmatrix}, \qquad P_{\mathrm{U}} = \begin{pmatrix} \widehat{A}^{-1} & \widehat{A}^{-1}B^{\mathrm{T}}\widehat{S}^{-1} \\ 0 & -\widehat{S}^{-1} \end{pmatrix} \tag{3.35}$$

and the block diagonal preconditioner

$$P_{\mathrm{D}} = \begin{pmatrix} \widehat{A}^{-1} & 0 \\ 0 & \widehat{S}^{-1} \end{pmatrix}.$$

The application of the block preconditioners P_{L}, P_{U}, P_{D} to a vector requires the application of the preconditioners $\widehat{A}^{-1}$ and $\widehat{S}^{-1}$. The block triangular preconditioners additionally require matrix-vector products with the matrix blocks B or B^{T}. One can show that GMRes preconditioned with a block triangular preconditioner requires about half as many iterations for certain starting vectors as GMRes preconditioned with the block diagonal preconditioner [26]. This is also observed in numerical experiments for general starting vectors [26]. Since the application costs of both preconditioners are similar (applying B or B^{T} to a vector is expected to be significantly cheaper than the application of the preconditioners $\widehat{A}^{-1}$ and $\widehat{S}^{-1}$), using a block triangular preconditioner is expected to be advantageous compared to using the block diagonal preconditioner.

In the remainder of this chapter, we discuss preconditioning techniques for the upper left matrix block in Section 3.5.1 and the Schur complement in Section 3.5.2. We consider Schur complement approximations from the least-squares commutator (LSC) [25] and the so-called semi-implicit method for pressure-linked equations (SIMPLE) [55, 71].

3.5.1 Preconditioning the upper left matrix block

The upper left matrix block A in our application is related to a diffusion-convection-reaction operator for the Oseen problems (3.27) or to a diffusion-reaction operator for the quasi-Stokes problems (3.25). We first recall the compositions of the upper left matrix block given in Equations (3.26) and (3.28) as

$$A_{\mathrm{Stokes}} = M_u + \tfrac{2k_n}{Re} M_{\mathrm{diff}}$$

or

$$A_{\mathrm{NSE}} = M_u + \tfrac{2k_n}{Re} M_{\mathrm{diff}} + k_n M_{\mathrm{adv}}$$

with

$$
\begin{aligned}
[M_u]_{ij} &= \left\langle \boldsymbol{\phi}_i^u, \boldsymbol{\phi}_j^u \right\rangle, \\
[M_{\text{diff}}]_{ij} &= \left\langle \boldsymbol{\varepsilon}(\boldsymbol{\phi}_i^u), \boldsymbol{\varepsilon}\big(\boldsymbol{\phi}_j^u\big) \right\rangle, \\
[M_{\text{adv}}]_{ij} &= \left\langle \boldsymbol{\phi}_i^u, (\mathbf{u}_{h,m}^{n+1} \cdot \nabla)\boldsymbol{\phi}_j^u \right\rangle.
\end{aligned}
$$

We discuss various algebraic preconditioners for $A = A_{\text{NSE}}$ or $A = A_{\text{Stokes}}$. We consider two approaches to constructing a preconditioner for A: (a) preconditioners that are built based on the whole matrix A or an approximation to it and (b) preconditioners that exploit a block structure of A.

We start with approach (a) for which we consider approximations of the inverses of (i) the velocity mass matrix, (ii) the matrix block A, and (iii) an almost block diagonal matrix $\widetilde{A} \approx A$.

We first discuss approach (i). For small time step sizes k_n, the upper left matrix block A is dominated by the velocity mass matrix M_u since all other contributions to A are multiplied by the time step size. In this case, we can use an approximation of the inverse velocity mass matrix as a preconditioner. The velocity mass matrix is block diagonal and thus typically sparser than the matrix block A which leads to lower setup costs for the preconditioner. Furthermore, it does not change for different time steps and hence can be reused if the time step sizes remain sufficiently small. We employ an algebraic multigrid method with smoothed aggregation or an inexact LU factorization to approximate the inverse.

If approach (i) does not lead to satisfying convergence we can set up the preconditioner based on the matrix block A. However, approach (ii) is expected to be expensive since the velocity components are coupled.

This motivates approach (iii). The off-diagonal matrix blocks of A have non-zero entries due to the considered formulation of the diffusion, see Equation (3.1a), and the velocity boundary conditions that couple the velocity components. The symmetric diffusion given by $2\left\langle \boldsymbol{\varepsilon}(\boldsymbol{\phi}_i^u), \boldsymbol{\varepsilon}\big(\boldsymbol{\phi}_j^u\big) \right\rangle$ leads to a coupling of velocity components and thus non-zero off-diagonal blocks in A. To find a preconditioner for A, we consider a replacement matrix $\widetilde{A} \approx A$ that has significantly fewer entries in the off-diagonal blocks. For this, we follow the approach from [44] where it is utilized in a solver for a model for earth mantle convection described by another version of the Boussinesq equations that does not include advection in the momentum balance. For setting up the preconditioner, we replace the symmetric diffusion by the Laplace operator $\left\langle \nabla\boldsymbol{\phi}_i^u, \nabla\boldsymbol{\phi}_j^u \right\rangle$ and define

$$
\widetilde{A}_{\text{NSE}} \coloneqq M_u + \tfrac{2k_n}{Re} L + k_n M_{\text{adv}}, \tag{3.36a}
$$

$$
\widetilde{A}_{\text{Stokes}} \coloneqq M_u + \tfrac{k_n}{Re} L \tag{3.36b}
$$

where

$$
[L]_{ij} = \left\langle \nabla\boldsymbol{\phi}_i^u, \nabla\boldsymbol{\phi}_j^u \right\rangle.
$$

This leads to block diagonal matrices of the form

$$
\widetilde{A} = \begin{pmatrix} A_1 & 0 & 0 \\ 0 & A_2 & 0 \\ 0 & 0 & A_3 \end{pmatrix}
$$

for three-dimensional problems. Applying boundary conditions to the new matrix $\widetilde{A}$ can introduce entries in off-diagonal blocks but only on the boundaries. The replacement matrix hence is still sparser than the matrix block A. As done in [44], we use the same velocity boundary conditions for $\widetilde{A}$ as for the system matrix. To approximate the inverses of A and $\widetilde{A}$, we utilize an algebraic multigrid method with smoothed aggregation or an inexact LU factorization.

We now discuss the block-based approaches (b). For three-dimensional domains, the matrix blocks A_{NSE} and A_{Stokes} have the form

$$A = \begin{pmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{pmatrix}.$$

with block sizes $n_u/3 \times n_u/3$. We assume that the degrees of freedom are sorted such that each diagonal block corresponds to a velocity direction. We consider two approximations to the inverse of A: the block diagonal preconditioner

$$\begin{pmatrix} A_{11}^{-1} & 0 & 0 \\ 0 & A_{22}^{-1} & 0 \\ 0 & 0 & A_{33}^{-1} \end{pmatrix} \approx \begin{pmatrix} \widehat{A}_{11}^{-1} & 0 & 0 \\ 0 & \widehat{A}_{22}^{-1} & 0 \\ 0 & 0 & \widehat{A}_{33}^{-1} \end{pmatrix} =: \widehat{A}_{\mathrm{diag}}^{-1} \tag{3.37}$$

and the upper block triangular preconditioner

$$\begin{aligned} \begin{pmatrix} A_{11} & A_{12} & A_{13} \\ 0 & A_{22} & A_{23} \\ 0 & 0 & A_{33} \end{pmatrix}^{-1} &= \begin{pmatrix} A_{11}^{-1} & -A_{11}A_{12}A_{22}^{-1} & A_{11}^{-1}A_{12}A_{22}^{-1}A_{23}A_{33}^{-1} - A_{11}^{-1}A_{13}A_{33}^{-1} \\ 0 & A_{22}^{-1} & -A_{22}^{-1}A_{23}A_{33}^{-1} \\ 0 & 0 & A_{33}^{-1} \end{pmatrix} \\ &\approx \begin{pmatrix} \widehat{A}_{11}^{-1} & -\widehat{A}_{11}^{-1}A_{12}\widehat{A}_{22}^{-1} & \widehat{A}_{11}^{-1}A_{12}\widehat{A}_{22}^{-1}A_{23}\widehat{A}_{33}^{-1} - \widehat{A}_{11}^{-1}A_{13}\widehat{A}_{33}^{-1} \\ 0 & \widehat{A}_{22}^{-1} & -\widehat{A}_{22}^{-1}A_{23}\widehat{A}_{33}^{-1} \\ 0 & 0 & \widehat{A}_{33}^{-1} \end{pmatrix} \\ &=: \widehat{A}_{\mathrm{tri}}^{-1} \end{aligned} \tag{3.38}$$

with the approximations $\widehat{A}_{ii}^{-1} \approx A_{ii}^{-1}$, $i \in \{1, 2, 3\}$. To approximate the inverses of the diagonal matrix blocks, we use an algebraic multigrid method with smoothed aggregation or an inexact LU factorization. Note that the matrix blocks A_{11}, A_{22}, and A_{33} are not equal after applying the boundary conditions and when the matrix blocks include advection. Both preconditioners require the setup and application of three preconditioners each of size $n_u/3 \times n_u/3$ which is typically faster than the setup and application of one preconditioner of size $n_u \times n_u$. The application of the block triangular preconditioner requires additional matrix-vector multiplications with off-diagonal subblocks of A. This is slightly more expensive than the application of the block diagonal preconditioner but includes a part of the coupling between velocity components and hence yields a more accurate approximation to A^{-1}. A further advantage of the block preconditioners for A is that the construction and application of the three preconditioners $\widehat{A}_{11}$, $\widehat{A}_{22}$, $\widehat{A}_{33}$ can be performed in parallel.

We compare iteration counts, setup and solver times obtained with the discussed preconditioning techniques in Section 5.2.

3.5.2 Schur complement preconditioners

Preconditioning the Schur complement $S = BA^{-1}B^{\mathrm{T}}$ is typically a more demanding task than preconditioning the upper left block of the system matrix since the Schur complement is not available explicitly as it depends on the large-scale inverse A^{-1}. We now discuss two preconditioning techniques, the least-squares commutator and the Schur complement approximation from the SIMPLE method.

3.5.2.1 Least-squares commutator

In the least-squares commutator method [25, 26], the inverse Schur complement is approximated based on an algebraic commutator problem.

Algebraic commutator-based preconditioners are based on (approximately) commuting the large-scale inverse A^{-1} with one of the rectangular matrices B, B^{T} to avoid the evaluation of the inverse upper left block A^{-1}. We consider the approximation

$$S = BA^{-1}B^{\mathrm{T}} \approx BB^{\mathrm{T}}F^{-1}$$

where $F \in \mathbb{R}^{n_p \times n_p}$ acts on the pressure space and is hence smaller than $A \in \mathbb{R}^{n_u \times n_u}$. Assuming that the matrix $F \in \mathbb{R}^{n_p \times n_p}$ satisfies

$$B^{\mathrm{T}}F \approx AB^{\mathrm{T}}, \tag{3.39}$$

we find the approximation of the inverse Schur complement

$$S^{-1} = \left(BA^{-1}B^{\mathrm{T}}\right)^{-1} \approx F\left(BB^{\mathrm{T}}\right)^{-1}, \tag{3.40}$$

assuming that the matrix product BB^{T} is invertible. The matrix product BB^{T} is invertible if B has full row rank, i.e. $\operatorname{rank}(B) = n_p$. This is usually not the case since the pressure is typically only defined up to a constant. We discuss in Section 3.5.3 how we can apply the preconditioner for a rank-deficient matrix B. The approximation (3.40) of the inverse Schur complement does not require the application of the large-scale inverse A^{-1}.

The problem (3.39) is overdetermined since $B^{\mathrm{T}} \in \mathbb{R}^{n_u \times n_p}$ is a rectangular matrix with $\operatorname{rank}(B^{\mathrm{T}}) \leq n_p$ and has only an exact solution if $\operatorname{range}\left(B^{\mathrm{T}}\right) \subset \operatorname{range}\left(A^{-1}B^{\mathrm{T}}\right)$. The problem (3.39) can hence only be solved in a minimizing sense

$$\min_F \|B^{\mathrm{T}}F - AB^{\mathrm{T}}\|$$

for some matrix norm $\|\cdot\|$. Considering the underlying differential operators of the matrix blocks, one could interpret the matrix F as a discrete differential operator acting on the pressure space. This motivates the introduction of a scaling with (diagonal approximations of) the velocity and pressure mass matrices $M_u \in \mathbb{R}^{n_u \times n_u}$ and $M_p \in \mathbb{R}^{n_p \times n_p}$ to include the choice of finite element. This means that we replace the commutator problem (3.39) by

$$M_u^{-1}B^{\mathrm{T}}M_p^{-1}F \approx M_u^{-1}AM_u^{-1}B^{\mathrm{T}}.$$

Rearranging this commutator problem leads to the Schur complement approximation

$$S = BA^{-1}B^{\mathrm{T}} \approx \left(BM_u^{-1}B^{\mathrm{T}}\right)F^{-1}M_p. \tag{3.41}$$

We determine the matrix F by solving a least-squares problem for each column of F

$$\min \| M_u^{-1} B^{\mathrm{T}} M_p^{-1} [F]_j - \left[M_u^{-1} A M_u^{-1} B^{\mathrm{T}} \right]_j \|_{M_u} \tag{3.42}$$

where $\|\cdot\|_{M_u}$ is the associated vector norm weighted by M_u given by $\|v\|_{M_u}^2 = v^{\mathrm{T}} M_u v$ for a vector $v \in \mathbb{R}^{n_u}$. The solution to this least-squares problem is given by

$$F = M_p \left(B M_u^{-1} B^{\mathrm{T}} \right)^{-1} \left(B M_u^{-1} A M_u^{-1} B^{\mathrm{T}} \right)$$

Substituting the expression for F into Equation (3.41) yields the approximation to the inverse Schur complement

$$S_{\mathrm{LSC}}^{-1} = \left(B M_u^{-1} B^{\mathrm{T}} \right)^{-1} \left(B M_u^{-1} A M_u^{-1} B^{\mathrm{T}} \right) \left(B M_u^{-1} B^{\mathrm{T}} \right)^{-1}.$$

The subscript LSC abbreviates the least-squares commutator method. We obtain a practical preconditioner by replacing the velocity mass matrix with a diagonal approximation $D_u \approx M_u$, in our case $D_u = \mathrm{diag}(M_u)$. We additionally replace the Poisson-type problems with approximate problems $\widehat{M}_{\mathrm{BMB^T}} \approx M_{\mathrm{BMB^T}} := B D_u^{-1} B^{\mathrm{T}}$, where the subscript $\mathrm{BMB^T}$ represents the matrices B, $\mathrm{diag}(\underline{M}_u)$ and B^{T}. The approximation $\widehat{M}_{\mathrm{BMB^T}}$ is either given explicitly or its inverse is defined by an inner iterative solver. We obtain the preconditioner

$$\widehat{S}_{\mathrm{LSC}}^{-1} = \widehat{M}_{\mathrm{BMB^T}}^{-1} B D_u^{-1} A D_u^{-1} B^{\mathrm{T}} \widehat{M}_{\mathrm{BMB^T}}^{-1}. \tag{3.43}$$

This preconditioner requires two (approximate) Poisson-type solves per application but avoids the evaluation of the (dense) Schur complement.

The quality of the least-squares preconditioner depends on the error that we make in the approximate commutator problem (3.39). Especially at the boundaries, the error in the commutator problem (3.39) may become large as the preconditioner (3.43) does not take boundary effects into account. It has been shown in [27] and [26] that damping certain boundary-related degrees of freedom can improve the LSC preconditioner significantly. The LSC preconditioner without boundary correction tends to show grid dependant convergence which is avoided by including boundary corrections. In [27, 26], it is proposed to damp tangential velocities that are connected to Dirichlet boundaries. The damping is used to weight the least-squares minimization problems (3.42). This approach is also applied in [1] where very similar results for LSC with and without boundary correction are observed.

However, we do not observe grid-dependant convergence for our test systems if sufficiently accurate approximations to the inner Poisson-type problems are used. This is underlined with numerical experiments in Section 5.2. We hence omit the boundary corrections and use the classical version of the LSC preconditioner.

3.5.2.2 The SIMPLE preconditioner

The semi-implicit method for pressure-linked equations (SIMPLE) is an iterative technique for solving the Navier-Stokes equations and was originally introduced in [55]. The SIMPLE preconditioner is defined as one iteration of this method. One iteration of the SIMPLE algorithm for a saddle-point system as given in (3.31) reads:

1. Solve the momentum equation for $\widetilde{u}$ with an estimate for the pressure p_k

$$A\widetilde{\mathbf{u}} = \mathbf{f} - B^{\mathrm{T}} p_k.$$

2. Compute the pressure correction δp by solving

$$-(B\,\mathrm{diag}(A)^{-1}B^{\mathrm{T}})\delta p = -B\widetilde{\mathbf{u}} + g.$$

3. Compute the velocity correction $\delta\mathbf{u}$ as

$$\delta\mathbf{u} = -\,\mathrm{diag}(A)^{-1}B^{\mathrm{T}}\delta p.$$

4. Update the pressure $p_{k+1} = p_k + \delta p$ and the velocity $\mathbf{u}_{k+1} = \widetilde{\mathbf{u}} + \delta\mathbf{u}$.

The pressure and velocity updates can be scaled with suitable weights $\alpha_1 > 0$, $\alpha_2 > 0$ to accelerate convergence [26, 57]. We then compute the new iterates as

$$\begin{aligned} p_{k+1} &= p_k + \alpha_1\delta p, \\ \mathbf{u}_{k+1} &= \widetilde{\mathbf{u}} + \alpha_2\delta\mathbf{u}. \end{aligned}$$

The SIMPLE iteration in matrix form is given by

$$\begin{pmatrix}\mathbf{u}_{k+1}\\ p_{k+1}\end{pmatrix} = \begin{pmatrix}\mathbf{u}_k\\ p_k\end{pmatrix} + P_{\mathrm{SIMPLE}}\left(\begin{pmatrix}\mathbf{f}\\ g\end{pmatrix} - \begin{pmatrix}A & B^{\mathrm{T}}\\ B & 0\end{pmatrix}\begin{pmatrix}\mathbf{u}_k\\ p_k\end{pmatrix}\right)$$

with the block preconditioner

$$\begin{aligned} P_{\mathrm{SIMPLE}} &= \begin{pmatrix}\alpha_1\mathrm{I}_{n_u} & 0\\ 0 & \alpha_2\mathrm{I}_{n_p}\end{pmatrix}\begin{pmatrix}\mathrm{I}_{n_u} & \mathrm{diag}(A)^{-1}B^{\mathrm{T}}\\ 0 & \mathrm{I}_{n_p}\end{pmatrix}^{-1}\begin{pmatrix}A & 0\\ B & -\widetilde{S}_{\mathrm{SIMPLE}}\end{pmatrix}^{-1} \\ &= \begin{pmatrix}\alpha_1\mathrm{I}_{n_u} & 0\\ 0 & \alpha_2\mathrm{I}_{n_p}\end{pmatrix}\begin{pmatrix}\mathrm{I}_{n_u} & -\,\mathrm{diag}(A)^{-1}B^{\mathrm{T}}\\ 0 & \mathrm{I}_{n_p}\end{pmatrix}\begin{pmatrix}A^{-1} & 0\\ \widetilde{S}^{-1}BA^{-1} & -\widetilde{S}^{-1}_{\mathrm{SIMPLE}}\end{pmatrix} \end{aligned}$$

using the Schur complement approximation $\widetilde{S}_{\mathrm{SIMPLE}} \coloneqq B\,\mathrm{diag}(A)^{-1}B^{\mathrm{T}}$. From this formulation, we find the SIMPLE-type Schur complement preconditioner

$$S^{-1}_{\mathrm{SIMPLE}} = (B\,\mathrm{diag}(A)^{-1}B^{\mathrm{T}})^{-1}.$$

We assume that $B\,\mathrm{diag}(A)^{-1}B^{\mathrm{T}}$ is invertible which is satisfied if B has full row rank. In the practical application, we replace the approximate Schur complement by an approximation $\widehat{M}_{\mathrm{BDB^T}} \approx M_{\mathrm{BDB^T}} \coloneqq B\,\mathrm{diag}(A)^{-1}B^{\mathrm{T}}$ where the subscript denotes the matrices B, $\underline{\mathrm{d}}\mathrm{iag}(A)^{-1}$ and B^{T}. We obtain the preconditioner

$$\widehat{S}^{-1}_{\mathrm{SIMPLE}} = \widehat{M}^{-1}_{\mathrm{BDB^T}}. \tag{3.44}$$

Alternatively, we can use the exact matrix product and replace the inverse with an approximate iterative solver. The SIMPLE-type Schur complement preconditioner is easy to implement and requires only one (approximate) solution to a Poisson-type problem. It is effective for diagonal dominant A since then $\mathrm{diag}(A)^{-1} \approx A^{-1}$, but deteriorates for advection-dominated problems.

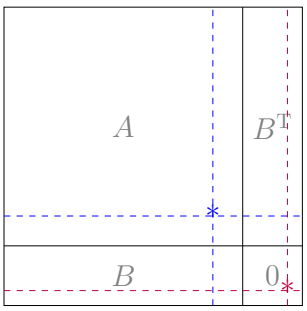


FIGURE 3.5: Illustration of the zero rows and columns in the saddle-point matrix due to elimination of constraints. The blue asterisk refers to a constrained velocity dof and the purple asterisk refers to a constrained pressure dof. The dashed lines denote zero rows and columns. The asterisks denote artificial nonzero diagonal entries.

3.5.3 Implementation details

This section is concerned with implementation details regarding the rank deficiency of the saddle-point matrices in (3.25) or (3.27) which arises since the pressure is only defined up to a constant and recycling of preconditioners. We outline briefly how constrained degrees of freedom, for our test systems due to boundary conditions, are treated. Then, we discuss options to deal with the rank deficiency of the saddle-point matrix. Furthermore, we discuss the recycling of approximations to the inner Poisson-type problems of the Schur complement preconditioners.

Constraints for degrees of freedom arise when we enforce boundary conditions. In the considered test model, constraints come from essential boundary conditions.

Our model is implemented with the library DEAL.II . In DEAL.II , all degrees of freedom (abbreviated as dofs), also those on the boundaries, are kept in the linear systems. The handling of constraints is described in [8]. Constrained degrees of freedom are eliminated by changing the respective matrix rows and columns as well as the respective entries of the right-hand side. The rows and columns of the system matrix that correspond to eliminated degrees of freedom are then set to zero. The corresponding diagonal matrix element is then set to a nonzero value to avoid zero rows (or columns) in the system matrix. Before solving the linear system, the corresponding elements of the initial guess for the solution and of the right-hand side are set to zero. The constrained degrees of freedom are hence "ignored" by the iterative solver. After solving the linear system, the constrained elements of the solution vector are set to the right value. For the upper left matrix block A, this method ensures that A stays invertible. However, the matrix blocks B (B^{T}) will have zero rows (columns) if pressure degrees of freedom are constrained. This is the case if we have for example periodic boundary conditions for the pressure. Figure 3.5 illustrates this.

In both Schur complement preconditioners, LSC and SIMPLE, we need to (approximately) solve linear systems with a system matrix of the type BDB^{T} where D is a diagonal matrix. If B has zero rows, this matrix has zero rows (and columns) as well and is hence not invertible. We add a positive value to the respective diagonal elements of the matrix BDB^{T} to enforce invertibility and positive definiteness. This value does not influence the calculated solution since the respective elements of the solution and right-hand side are zero during the iterative solution process as described above.

Furthermore, B and B^{T} are typically rank deficient since the pressure is only defined up to a constant. There are different approaches to deal with the rank deficiency. We use a method that is often used in practice: We constrain one pressure degree of freedom by setting it to a pre-defined value (such as zero). Although this is not well-defined from an analytical point of view since the pressure is in the space $L_2(\Omega)$ and a point value is thus not defined, this approach typically works in practice. This method has the further advantage that it does not lead to fill-in in the system matrix as opposed to adding a mean value constraint for the pressure. Imposing a mean value constraint on the pressure couples the corresponding degrees of freedom and is hence computationally not efficient.

Now, we discuss how setup costs can be reduced by recycling approximations to the inner Poisson-type problems of both Schur complement preconditioners, the LSC and the SIMPLE preconditioner. For both preconditioners, we require approximate solutions to problems of the type $BD^{-1}B^{\mathrm{T}}x = y$ where D is a diagonal matrix given by $D = \mathrm{diag}(A)$ for the SIMPLE preconditioner and by $D = \mathrm{diag}(M_u)$ for the LSC preconditioner. The matrices B and M_u do not change for different Picard-type iterations or time steps except for possibly arising boundary effects. Therefore, the matrix $B\,\mathrm{diag}(M_u)B^{\mathrm{T}}$ does not change for different Picard-type iterations except for potential boundary effects. For our test setting, we do not observe any change in B. Thus, we can precompute an approximation as given by an algebraic multigrid method or an inexact factorization to the matrix $B\,\mathrm{diag}(M_u)B^{\mathrm{T}}$ or its inverse and reuse it for the following saddle-point problems in different Picard-type iterations or time steps. For the SIMPLE preconditioner, this looks different. The diagonal matrix $D = \mathrm{diag}(A)$ does change for different Picard-type iterations or time steps. However, these changes are small since the matrix A is dominated by the constant velocity mass matrix M_u and the remaining contributions to A (the diffusion and advection) are scaled by (small) time step sizes. Hence, reusing an approximation to the matrix $B\,\mathrm{diag}(A)^{-1}B^{\mathrm{T}}$ is a reasonable approach to save construction times required for precomputing the matrix $B\,\mathrm{diag}(A)^{-1}B^{\mathrm{T}}$ and an approximation to it. A brief comparison of both approaches based on numerical experiments is given in Section 5.2.

Algorithm 3.1: Solution algorithm for the Boussinesq approximation

Input: Temperature initial condition T_0, velocity initial condition $\mathbf{u}_0$, boundary conditions for $\mathbf{u}$, T, simulation time t_f, initial time step size k_0, flag `adapt_timestep`.

Output: Discrete solutions $\{\mathbf{u}_h^n\}$, $\{p_h^n\}$, $\{T_h^n\}$ for the velocity, the pressure, and the temperature at discrete time points in $(0, t_f]$.

Project $\mathbf{u}_0$ onto the discrete initial velocity $\mathbf{u}_h^0$.
Project T_0 onto the discrete initial temperature T_h^0.
Set $t = 0$, $n = 0$.
while $t < t_f$ **do**
  if `adapt_timestep` and $n > 0$ **then**
    Compute time step size k_n with Equation (3.30).
  else
    Set $k_n = k_0$.
  end
  Assemble the Navier-Stokes system, apply the boundary conditions for $\mathbf{u}$, and build the block preconditioner.
  Assemble the temperature system, apply the boundary conditions for T, and build the temperature preconditioner.
  Solve the quasi-Stokes system

$$\begin{pmatrix} A_{\text{Stokes}} & B^{\mathrm{T}} \\ B & 0 \end{pmatrix} \begin{pmatrix} \mathbf{u}_h^{n+1} \\ \widetilde{p}_h^{n+1} \end{pmatrix} = \begin{pmatrix} \mathbf{f}_{\text{Stokes}} \\ g \end{pmatrix}$$

  or solve the Navier-Stokes system with the Picard correction:
  for $m = 1, 2, \ldots$ until convergence **do**
    Solve

$$\begin{pmatrix} A_{\text{NSE}}^m & B^{\mathrm{T}} \\ B & 0 \end{pmatrix} \begin{pmatrix} \delta\mathbf{u}_{h,m}^{n+1} \\ \delta\widetilde{p}_{h,m}^{n+1} \end{pmatrix} = \begin{pmatrix} \mathbf{f}_{\text{NSE}} \\ g \end{pmatrix}.$$

    Update the solutions $\mathbf{u}_{h,m+1}^{n+1} = \mathbf{u}_{h,m}^{n+1} + \delta\mathbf{u}_{h,m}^{n+1}$, $\widetilde{p}_{h,m+1}^{n+1} = \widetilde{p}_{h,m}^{n+1} + \delta\widetilde{p}_{h,m}^{n+1}$.
  end
  Solve the temperature equation

$$MT_h^{n+1} = f.$$

  Set $t \mathrel{+}= k_n$.
  Set $n \mathrel{+}= 1$.
end

Chapter 4

Low-rank updates for preconditioners

Low-rank updates for preconditioners aim to accelerate an iterative solver while introducing low additional costs. In this chapter, we derive several preconditioner updates that are based on a low-rank approximation of the error between the identity matrix and the preconditioned matrix or pressure Schur complement. Such an approach can be utilized to adjust any given preconditioner.

We start with the derivation of a left and a right preconditioner update in Section 4.1. In Section 4.2, we analyze numerically whether the error between the identity matrix and the preconditioned matrix or Schur complement can be approximated with a low-rank factorization. Then in Section 4.3, we introduce a relaxation parameter for the initial Schur complement preconditioner before computing the low-rank approximation. The relaxation parameter has been found to have a significant impact on the convergence behavior of the iterative solver. In Section 4.4, we derive a repeated update scheme that can be (repeatedly) applied to updated preconditioners. Additionally to the outer preconditioner updates, we derive Schur complement preconditioners with inner low-rank corrections in Section 4.5. In Section 4.6, we discuss the practical construction and application of the update schemes and the corresponding computational complexities. We furthermore discuss how certain vectors can be reused to reduce the setup times of the updated preconditioners. In Section 4.7, we proceed with an error analysis for the update scheme. We conclude this chapter with a (numerical) spectral analysis for the derived update techniques in Section 4.8.

This chapter extends our work in [10] and [9]. The notation, structure, and wording of this chapter were influenced by both articles. In [10], we presented an error-based low-rank update for right Schur complement preconditioners using a randomized singular value decomposition and the repeated application of multiplicative low-rank updates. In [9], we extended the derivation for left preconditioners and modified the low-rank update by relaxing the initial Schur complement preconditioner. The updates were constructed with a randomized power range finder and an Arnoldi iteration.

4.1 Error-based update for preconditioners

In this section, we derive a left and a right preconditioner update following our work in [9] but now applied to general matrices. The derivation is based on the ideas from [74] and [39]. In [74], the update scheme is utilized in a power-Schur complement low-rank preconditioner which is applied to the Schur complement arising in a domain decomposition method. In

[39], a similar approach is applied to inexact LU factorization preconditioners. Here, we first derive the update scheme for general matrices and then apply the ideas to the pressure Schur complement emerging in fluid flow problems in the following sections in this chapter.

We start with the linear equation

$$Mx = b$$

with $M \in \mathbb{R}^{n\times n}$, x, $b \in \mathbb{R}^n$ which can be preconditioned with the preconditioner $\widehat{M}^{-1} \approx M^{-1}$.

We aim to utilize a low-rank update to modify the given preconditioner $\widehat{M}^{-1}$. The low-rank correction should provide a more accurate approximation of the inverse M^{-1} and hence accelerate the iterative solver. To derive a suitable update, we start from the error between the inverse matrix M^{-1} and the initial preconditioner $\widehat{M}^{-1}$

$$\widehat{R} = M^{-1} - \widehat{M}^{-1}. \tag{4.1}$$

We multiply the matrix $\widehat{R}$ by the matrix M from the left or the right, respectively, and find

$$E_{\mathrm{L}} \coloneqq \mathrm{I}_n - \widehat{M}^{-1}M, \qquad\qquad E_{\mathrm{R}} \coloneqq \mathrm{I}_n - M\widehat{M}^{-1}. \tag{4.2}$$

The matrices E_{L} and E_{R} represent the error between the identity matrix and the left (index L) and right (index R) preconditioned matrix, respectively. We solve for the matrix M and invert to find the following alternative expressions for the inverse system matrix

$$M^{-1} = (\mathrm{I}_n - E_{\mathrm{L}})^{-1}\widehat{M}^{-1}, \qquad\qquad M^{-1} = \widehat{M}^{-1}(\mathrm{I}_n - E_{\mathrm{R}})^{-1}. \tag{4.3}$$

Note that $\mathrm{I}_n - E_{\mathrm{L}} = \widehat{M}^{-1}M$ and $\mathrm{I}_n - E_{\mathrm{R}} = M\widehat{M}^{-1}$ are invertible. The formulas (4.3) are computationally not feasible since the matrices E_{L}, E_{R} are not given explicitly. Furthermore, both equations require the numerical solution of an additional system of size $n \times n$. Although the error matrices E_{L}, E_{R} are not available explicitly, we can apply them to vectors allowing us to compute low-rank approximations $E_{\mathrm{L}} \approx U_{\mathrm{L},r}V_{\mathrm{L},r}^{\mathrm{T}}$, $E_{\mathrm{R}} \approx U_{\mathrm{R},r}V_{\mathrm{R},r}^{\mathrm{T}}$, with $U_{\mathrm{L},r}, V_{\mathrm{L},r}, U_{\mathrm{R},r}, V_{\mathrm{R},r} \in \mathbb{R}^{n\times r}$, $r \ll n$. For an accurate approximation with an approximate low-rank factorization, the matrices E_{L}, E_{R} are required to have a small numerical rank. The low-rank property of the error matrices is discussed in Section 4.2. We insert the low-rank approximations in (4.3) and obtain the approximations $M^{-1} \approx M_{\mathrm{L}}^{-1}$ and $M^{-1} \approx M_{\mathrm{R}}^{-1}$ with

$$M_{\mathrm{L}}^{-1} = \big(\mathrm{I}_n - U_{\mathrm{L},r}V_{\mathrm{L},r}^{\mathrm{T}}\big)^{-1}\widehat{M}^{-1}, \qquad\qquad M_{\mathrm{R}}^{-1} = \widehat{M}^{-1}\big(\mathrm{I}_n - U_{\mathrm{R},r}V_{\mathrm{R},r}^{\mathrm{T}}\big)^{-1}$$

assuming that the matrices $\mathrm{I}_n - U_{\mathrm{L},r}V_{\mathrm{L},r}^{\mathrm{T}}$ and $\mathrm{I}_n - U_{\mathrm{R},r}V_{\mathrm{R},r}^{\mathrm{T}}$ are still nonsingular. The resulting $n \times n$-system may be solved efficiently by applying the Sherman-Morrison-Woodbury formula. We find

$$\begin{aligned} M_{\mathrm{L}}^{-1} &= \Big(\mathrm{I}_n + U_{\mathrm{L},r}\big(\mathrm{I}_r - V_{\mathrm{L},r}^{\mathrm{T}}U_{\mathrm{L},r}\big)^{-1}V_{\mathrm{L},r}^{\mathrm{T}}\Big)\widehat{M}^{-1}, \\ M_{\mathrm{R}}^{-1} &= \widehat{M}^{-1}\Big(\mathrm{I}_n + U_{\mathrm{R},r}\big(\mathrm{I}_r - V_{\mathrm{R},r}^{\mathrm{T}}U_{\mathrm{R},r}\big)^{-1}V_{\mathrm{R},r}^{\mathrm{T}}\Big). \end{aligned}$$

Here, we only require the additional solution of an $r \times r$-system, $r \ll n$, instead of the solution of an $n \times n$-system.

For symmetric systems with symmetric initial preconditioner $\widehat{M}^{-1}$, the error matrix E_{L} for left preconditioners is the transpose of the error matrix E_{R} for right preconditioners since

$$(E_{\mathrm{L}})^{\mathrm{T}} = \left(\mathrm{I}_n - \widehat{M}^{-1}M\right)^{\mathrm{T}} = \mathrm{I}_n - M\widehat{M}^{-1} = E_{\mathrm{R}}. \tag{4.4}$$

Therefore, a low-rank approximation of E_{L} also yields a low-rank approximation of E_{R} and vice versa.

We derived basic update schemes for left and right preconditioners. In the next section, we numerically analyze the low-rank property of the error matrices (4.2) for initial preconditioners $\widehat{A}^{-1}$ and $\widehat{S}^{-1}$.

4.2 Low-rank property of the error matrix

Whether a low-rank factorization can accurately approximate the error matrix determines the quality of the derived low-rank update strategies. This depends on the decay of singular values. Therefore, we now investigate numerically the low-rank property of the error matrices for the test systems of medium dimensions described in Section 5.1. As initial preconditioners, we consider the LSC (3.43) and SIMPLE (3.44) preconditioner for $\widehat{S}^{-1}$ where the inner Poisson-type problems are approximated by incomplete Cholesky factorizations without fill-in (abbreviated as `IC(0)`). For both preconditioners, we reuse the incomplete Cholesky factorizations based on the first saddle-point system, i.e. the Stokes system, in the following Picard-type iterations, see Sections 3.5.3 and 5.2 for more details. For $\widehat{A}^{-1}$, we consider the block triangular ILU preconditioner (3.38) and the incomplete LU factorization of the velocity mass matrix M_u. We will observe that the block triangular ILU preconditioner leads to the smallest computational times for the majority of the considered test systems. The preconditioner based on M_u is interesting since it does not change for the different saddle-point systems and thus needs to be constructed only once. However, its quality may deteriorate significantly with developing advection and larger time step sizes. We are hence interested to see whether a low-rank update can improve this preconditioner.

We focus on the error matrices with right preconditioners and compute the largest 100 approximate singular values with a randomized singular value decomposition from [50]. We start from the randomized low-rank approximation

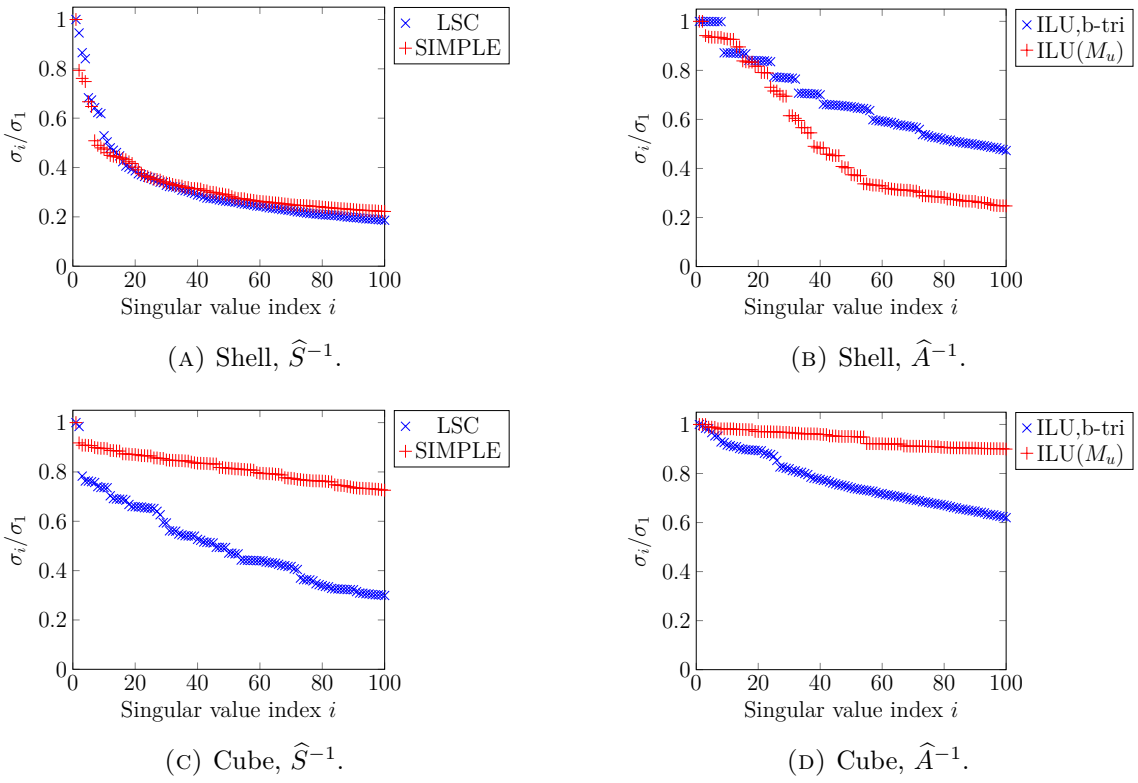
$$E_{\mathrm{R}} = \left((E_{\mathrm{R},\alpha})^{\mathrm{T}}\right)^{\mathrm{T}} \approx \left(QN^{\mathrm{T}}\right)^{\mathrm{T}} = NQ^{\mathrm{T}}$$

obtained with Algorithm 2.2 as

$$E_{\mathrm{R}} \approx U\Sigma V^{\mathrm{T}}Q^{\mathrm{T}} = U\Sigma\widehat{V}^{\mathrm{T}}$$

by computing the singular value decomposition of $N = U\Sigma V^{\mathrm{T}}$ and by setting $\widehat{V} \coloneqq QV$ where $\Sigma, V \in \mathbb{R}^{r\times r}$ and $Q, N, U \in \mathbb{R}^{n\times r}$ with $n = n_p$ for $\widehat{S}^{-1}$ and $n = n_u$ for $\widehat{A}^{-1}$. The diagonal matrix Σ holds the r largest approximate singular values of E_{R}.

For the Schur complement preconditioners, we replace the exact Schur complement $S = BA^{-1}B^{\mathrm{T}}$ by a Schur complement approximation $\widetilde{S} = B\widehat{A}^{-1}B^{\mathrm{T}}$ in the error matrix to compute the approximate singular values. We discuss this further in Section 4.3. Figure 4.1



(A) Shell, $\widehat{S}^{-1}$.

(B) Shell, $\widehat{A}^{-1}$.

(C) Cube, $\widehat{S}^{-1}$.

(D) Cube, $\widehat{A}^{-1}$.

FIGURE 4.1: Decay of singular values of $\widetilde{E}_{\mathrm{R}} = \mathrm{I}_{n_p} - \widetilde{S}\widehat{S}^{-1}$ for the preconditioners $\widehat{S}^{-1}$ and $E_{\mathrm{R}} = \mathrm{I}_{n_u} - A\widehat{A}^{-1}$ for the preconditioners $\widehat{A}^{-1}$ for the Oseen systems (shell with $n_u = 78438$, $n_p = 3474$, $k_n = 5\mathrm{e}{-4}$, cube with $n_u = 107811$, $n_p = 4913$, $k_n = 5\mathrm{e}{-3}$).

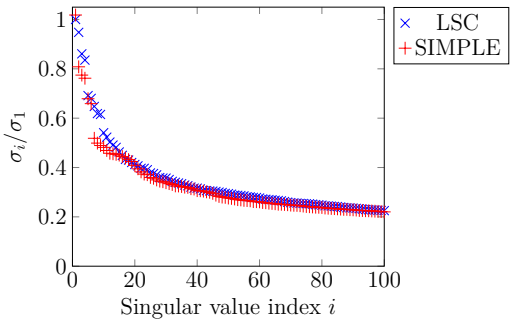
shows the decay of singular values for the error matrix $\widetilde{E}_{\mathrm{R}} = \mathrm{I}_{n_p} - \widetilde{S}\widehat{S}^{-1}$ for preconditioners $\widehat{S}^{-1}$ as well as for the matrix $E_{\mathrm{R}} = \mathrm{I}_{n_u} - A\widehat{A}^{-1}$ for preconditioners $\widehat{A}^{-1}$ for Oseen systems on a cube and a spherical shell domain, see Section 5.1 for more information on the systems. We observe that the singular values decay faster for the Schur complement preconditioners than for the preconditioners $\widehat{A}^{-1}$. Exceptions are the SIMPLE preconditioner for the Oseen system on the cube and the ILU preconditioner based on the velocity mass matrix M_u for the Oseen system on the spherical shell. We furthermore observe that the singular values decay faster for the systems on the spherical shell than for the systems on the cube.

Figure 4.2 shows the decay of singular values of $\widetilde{E}_{\mathrm{R}}$ and E_{R} for the Stokes systems. The singular values decay faster for the Schur complement preconditioners than for the preconditioners $\widehat{A}^{-1}$.

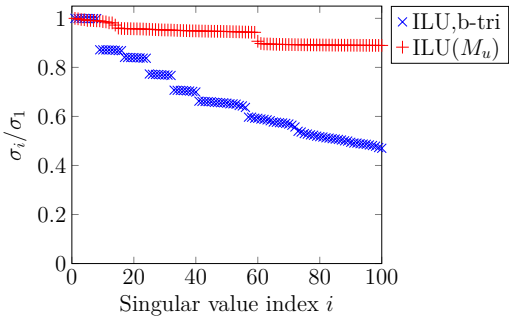
In the following, we therefore focus on low-rank updates for Schur complement preconditioners.

4.3 Update with relaxed initial Schur complement preconditioner

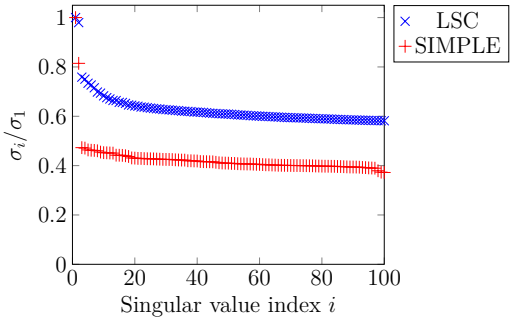
We now apply and adapt the update scheme from Section 4.1 for general preconditioners to Schur complement preconditioners. In this section, we adjust the right and left update by



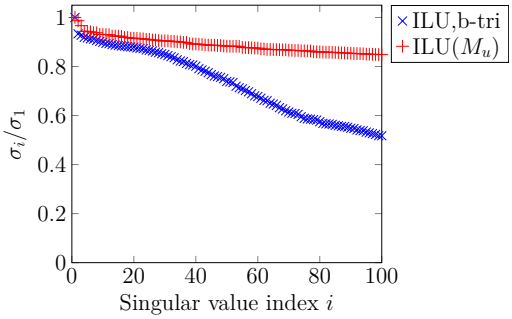
(A) Shell, $\widehat{S}^{-1}$.



(B) Shell, $\widehat{A}^{-1}$.



(C) Cube, $\widehat{S}^{-1}$.



(D) Cube, $\widehat{A}^{-1}$.

FIGURE 4.2: Decay of singular values of $\widetilde{E}_{\mathrm{R}} = \mathrm{I}_{n_p} - \widetilde{S}\widehat{S}^{-1}$ for the preconditioners $\widehat{S}^{-1}$ and $E_{\mathrm{R}} = \mathrm{I}_{n_u} - A\widehat{A}^{-1}$ for the preconditioners $\widehat{A}^{-1}$ for the Stokes systems (shell with $n_u = 78438$, $n_p = 3474$, $k_n = 5\mathrm{e}{-4}$, cube with $n_u = 107811$, $n_p = 4913$, $k_n = 5\mathrm{e}{-3}$).

introducing a relaxation parameter that scales the initial Schur complement preconditioner. Modifying this update scheme with a relaxation parameter can significantly improve the performance of the update as observed in [9]. Furthermore, we approximate the Schur complement in the error matrices to obtain a practical update scheme. The update scheme is based on a low-rank approximation of the approximate error matrices obtained with the Arnoldi iteration or a randomized power range finder.

To improve the spectral properties of the preconditioner, we introduce a relaxation parameter $\alpha > 0$ that scales the initial Schur complement preconditioner. Applied to the upper block triangular preconditioner, this leads to

$$P_{\mathrm{U},\alpha} = \begin{pmatrix} \widehat{A}^{-1} & \alpha\widehat{A}^{-1}B^{\mathrm{T}}\widehat{S}^{-1} \\ 0 & -\alpha\widehat{S}^{-1} \end{pmatrix}$$

as initial block preconditioner. A similar scaling is used in the SIMPLE method, where the pressure update is damped [26, 57]. There, it is observed that damping the pressure update, i.e. $\alpha \in (0, 1]$, can accelerate the SIMPLE solver.

To derive an update with a relaxed initial Schur complement preconditioner, we scale the preconditioner $\widehat{S}^{-1}$ with the relaxation parameter α and replace the error matrices E_{L}, E_{R} with their relaxed versions

$$E_{\mathrm{L},\alpha} = \mathrm{I}_{n_p} - \alpha\widehat{S}^{-1}S, \qquad E_{\mathrm{R},\alpha} = \mathrm{I}_{n_p} - \alpha S\widehat{S}^{-1}. \tag{4.5}$$

We then replace the matrices $E_{\mathrm{L},\alpha}$, $E_{\mathrm{R},\alpha}$ with the low-rank approximations

$$E_{\mathrm{L},\alpha} \approx U_{\mathrm{L},\alpha,r} V_{\mathrm{L},\alpha,r}^{\mathrm{T}}, \qquad E_{\mathrm{R},\alpha} \approx U_{\mathrm{R},\alpha,r} V_{\mathrm{R},\alpha,r}^{\mathrm{T}},$$

where $U_{\mathrm{L},\alpha,r}, V_{\mathrm{L},\alpha,r}, U_{\mathrm{R},\alpha,r}, V_{\mathrm{R},\alpha,r} \in \mathbb{R}^{n_p \times r}$. The low-rank approximations may be obtained by using one of the algorithms described in Section 2.3. We hence find the relaxed update formulas

$$S_{\mathrm{L},\alpha,r}^{-1} = \alpha\Big(\mathrm{I}_{n_p} + U_{\mathrm{L},\alpha,r}\big(\mathrm{I}_r - V_{\mathrm{L},\alpha,r}^{\mathrm{T}} U_{\mathrm{L},\alpha,r}\big)^{-1} V_{\mathrm{L},\alpha,r}^{\mathrm{T}}\Big)\widehat{S}^{-1}, \tag{4.6}$$

$$S_{\mathrm{R},\alpha,r}^{-1} = \alpha\widehat{S}^{-1}\Big(\mathrm{I}_{n_p} + U_{\mathrm{R},\alpha,r}\big(\mathrm{I}_r - V_{\mathrm{R},\alpha,r}^{\mathrm{T}} U_{\mathrm{R},\alpha,r}\big)^{-1} V_{\mathrm{R},\alpha,r}^{\mathrm{T}}\Big) \tag{4.7}$$

by utilizing the Sherman-Morrison-Woodbury formula.

For the computation of low-rank approximations of the matrices $E_{\mathrm{L},\alpha}$ and $E_{\mathrm{R},\alpha}$, we do not require an explicit representation of $E_{\mathrm{L},\alpha}$ and $E_{\mathrm{R},\alpha}$ but need to evaluate their action on a vector. Since both matrices depend on the Schur complement $S = BA^{-1}B^{\mathrm{T}}$ and hence on the large-scale inverse A^{-1}, evaluating matrix-vector products with $E_{\mathrm{L},\alpha}$ or $E_{\mathrm{R},\alpha}$ is computationally not feasible. To obtain a practical update scheme, we approximate the Schur complement S by replacing the inverse A^{-1} by the preconditioner $\widehat{A}^{-1}$ as done in Section 4.2. We obtain the Schur complement approximation

$$\widetilde{S} = B\widehat{A}^{-1}B^{\mathrm{T}}. \tag{4.8}$$

Note that the preconditioner $\widehat{A}^{-1}$ is already available since we need it in the initial block preconditioner for the saddle-point system (3.31).

We find the approximate error matrices

$$\widetilde{E}_{\mathrm{L},\alpha} = \mathrm{I} - \alpha\widehat{S}^{-1}\widetilde{S}, \qquad \widetilde{E}_{\mathrm{R},\alpha} = \mathrm{I} - \alpha\widetilde{S}\widehat{S}^{-1}. \tag{4.9}$$

Solving for $\widetilde{S}$ and inverting yields the approximations

$$S^{-1} \approx \widetilde{S}^{-1} = \alpha\Big(\mathrm{I}_{n_p} - \widetilde{E}_{\mathrm{L},\alpha}\Big)^{-1}\widehat{S}^{-1}, \tag{4.10a}$$

$$S^{-1} \approx \widetilde{S}^{-1} = \alpha\widehat{S}^{-1}\Big(\mathrm{I}_{n_p} - \widetilde{E}_{\mathrm{R},\alpha}\Big)^{-1}. \tag{4.10b}$$

We obtain the practical updated preconditioners by replacing the matrices $\widetilde{E}_{\mathrm{L},\alpha}$, $\widetilde{E}_{\mathrm{R},\alpha}$ with low-rank approximations, computed with one of the algorithms in Section 2.3, and applying the Sherman-Morrison-Woodbury formula.

We consider different methods to obtain low-rank approximations of $\widetilde{E}_{\mathrm{L},\alpha}$ and $\widetilde{E}_{\mathrm{R},\alpha}$: two versions based on a randomized power range finder described in Algorithm 2.2 and the Arnoldi iteration from Algorithm 2.4 possibly followed by projection.

We first derive the update scheme based on the randomized low-rank approximation. The randomized low-rank factorization method from Algorithm 2.2 yields the approximations $\widetilde{E}_{\mathrm{L},\alpha} \approx Q_{\mathrm{L},\alpha,r} N_{\mathrm{L},\alpha,r}^{\mathrm{T}}$, $\widetilde{E}_{\mathrm{R},\alpha} \approx Q_{\mathrm{R},\alpha,r} N_{\mathrm{R},\alpha,r}^{\mathrm{T}}$, with $Q_{\mathrm{L},\alpha,r}, N_{\mathrm{L},\mathrm{r}_\alpha}, Q_{\mathrm{R},\alpha,r}, N_{\mathrm{R},\mathrm{r}_\alpha} \in \mathbb{R}^{n \times r}$. We insert the low-rank approximations in Equation (4.7) and obtain the preconditioners

$$\widehat{S}_{\mathrm{random,L},\alpha,r}^{-1} = \alpha\Big(\mathrm{I}_{n_p} + Q_{\mathrm{L},\alpha,r}\big(\mathrm{I}_r - N_{\mathrm{L},\alpha,r}^{\mathrm{T}} Q_{\mathrm{L},\alpha,r}\big)^{-1} N_{\mathrm{L},\alpha,r}^{\mathrm{T}}\Big)\widehat{S}^{-1}, \tag{4.11a}$$

$$\widehat{S}_{\mathrm{random,R},\alpha,r}^{-1} = \alpha\widehat{S}^{-1}\Big(\mathrm{I}_{n_p} + Q_{\mathrm{R},\alpha,r}\big(\mathrm{I}_r - N_{\mathrm{R},\alpha,r}^{\mathrm{T}} Q_{\mathrm{R},\alpha,r}\big)^{-1} N_{\mathrm{R},\alpha,r}^{\mathrm{T}}\Big). \tag{4.11b}$$

An analogous update scheme can be defined by applying the power range finder to the transposed error matrix. For right updates based on the matrix $\widetilde{E}_{\mathrm{R},\alpha,r}$, this is advantageous since sample vectors can then be recycled to construct (possibly) following updates. We will discuss the recycling of sample vectors further in Section 4.6.3. The preconditioner update based on the approximation

$$\widetilde{E}_{\mathrm{R},\alpha} = \left(\widetilde{E}_{\mathrm{R},\alpha}^{\mathrm{T}}\right)^{\mathrm{T}} \approx \left(\widetilde{Q}_{\mathrm{R},\alpha,r}\widetilde{N}_{\mathrm{R},\alpha,r}^{\mathrm{T}}\right)^{\mathrm{T}} = \widetilde{N}_{\mathrm{R},\alpha,r}\widetilde{Q}_{\mathrm{R},\alpha,r}^{\mathrm{T}}$$

is then defined by setting $Q_{\mathrm{R},\alpha,r} \coloneqq \widetilde{N}_{\mathrm{R},\alpha,r}$ and $N_{\mathrm{R},\alpha,r} \coloneqq \widetilde{Q}_{\mathrm{R},\alpha,r}$ in Equation (4.11b). This approach is justified by the fact that the error matrix $\widetilde{E}_{\mathrm{R},\alpha,r}$ and its transpose have the same singular values. Hence, the theoretical accuracy of both approaches given in Corollary (2.3) is the same if we choose the same update rank, oversampling parameter ($k_r \geq 2$), and number of power iterations. The approximation approach based on the transposed matrix is for example also utilized in [50] for computing a randomized singular value decomposition. Note that we also need the transposed application of the preconditioners $\widehat{A}^{-1}$ and $\widehat{S}^{-1}$ in Algorithm 2.2 to approximate $E_{\mathrm{L},\alpha}$ and $E_{\mathrm{R},\alpha}$ for both randomized approaches.

The Arnoldi iteration from Algorithm 2.4 provides an alternative that does not require the transposed application of $\widehat{A}^{-1}$ or $\widehat{S}^{-1}$. Here, we obtain the low-rank approximations $\widetilde{E}_{\mathrm{L},\alpha} \approx V_{\mathrm{L},\alpha,r}H_{\mathrm{L},\alpha,r}V_{\mathrm{L},\alpha,r}^{\mathrm{T}}$, $\widetilde{E}_{\mathrm{R},\alpha} \approx V_{\mathrm{R},\alpha,r}H_{\mathrm{R},\alpha,r}V_{\mathrm{R},\alpha,r}^{\mathrm{T}}$ with $V_{\mathrm{L},\alpha,r}, V_{\mathrm{R},\alpha,r} \in \mathbb{R}^{n\times r}$ and $H_{\mathrm{L},\alpha,r}, H_{\mathrm{R},\alpha,r} \in \mathbb{R}^{r\times r}$.

For the left and the right update obtained with the Arnoldi iteration, we require an inverse of the type $\left(\mathrm{I}_{n_p} - VHV^{\mathrm{T}}\right)^{-1}$ with $V \in \mathbb{R}^{n_p\times r}$, $V^{\mathrm{T}}V = \mathrm{I}_r$, and an invertible Hessenberg matrix $H \in \mathbb{R}^{r\times r}$. We now rearrange this inverse such that we only require the solution of an $r \times r$-system instead of an $n_p \times n_p$-system. By utilizing the Sherman-Morrison-Woodbury formula we find

$$\left(\mathrm{I}_{n_p} - VHV^{\mathrm{T}}\right)^{-1} = \mathrm{I}_{n_p} + V\left(H^{-1} - V^{\mathrm{T}}V\right)^{-1}V^{\mathrm{T}}.$$

We rearrange following the lines in [74] to avoid solving two nested systems of size $r \times r$ and obtain

$$\begin{aligned}
\mathrm{I}_{n_p} + V\left(H^{-1} - V^{\mathrm{T}}V\right)^{-1}V^{\mathrm{T}} &= \mathrm{I}_{n_p} + V\left(H^{-1} - \mathrm{I}_r\right)^{-1}V^{\mathrm{T}} \\
&= \mathrm{I}_{n_p} + V(\mathrm{I}_r - H)^{-1}HV^{\mathrm{T}} \\
&= \mathrm{I}_{n_p} + V(\mathrm{I}_r - H)^{-1}(\mathrm{I}_r - \mathrm{I}_r + H)V^{\mathrm{T}} \\
&= \mathrm{I}_{n_p} + V\Big((\mathrm{I}_r - H)^{-1} - \mathrm{I}_r\Big)V^{\mathrm{T}}.
\end{aligned}$$

Here, we only have to solve one $r\times r$-system instead of an $n_p\times n_p$ system or two nested $r\times r$-systems. We insert the approximations obtained with the Arnoldi iteration in Equation (4.10), rearrange as above, and obtain the preconditioners

$$\widehat{S}^{-1}_{\mathrm{Arnoldi,L},\alpha,r} = \alpha\Big(\mathrm{I}_{n_p} + V_{\mathrm{L},\alpha,r}\Big((\mathrm{I}_r - H_{\mathrm{L},\alpha,r})^{-1} - \mathrm{I}_r\Big)V_{\mathrm{L},\alpha,r}^{\mathrm{T}}\Big)\widehat{S}^{-1}, \tag{4.12a}$$

$$\widehat{S}^{-1}_{\mathrm{Arnoldi,R},\alpha,r} = \alpha\widehat{S}^{-1}\Big(\mathrm{I}_{n_p} + V_{\mathrm{R},\alpha,r}\Big((\mathrm{I}_r - H_{\mathrm{R},\alpha,r})^{-1} - \mathrm{I}_r\Big)V_{\mathrm{R},\alpha,r}^{\mathrm{T}}\Big). \tag{4.12b}$$

The following approximation strategy combines ideas of the previous two low-rank approximation techniques. First, we approximate the range of the error matrix $\widetilde{E}_{\mathrm{R},\alpha}$ or

$\widetilde{E}_{\mathrm{L},\alpha}$ via the Arnoldi iteration and then obtain the right update vectors with projection. With $V_{\mathrm{L},\alpha,r}$, $V_{\mathrm{R},\alpha,r}$ holding the respective Arnoldi vectors as columns, we find the low-rank approximations

$$\widetilde{E}_{\mathrm{L},\alpha} \approx V_{\mathrm{L},\alpha,r} V_{\mathrm{L},\alpha,r}^{\mathrm{T}} \widetilde{E}_{\mathrm{L},\alpha} = V_{\mathrm{L},\alpha,r} W_{\mathrm{L},\alpha,r}^{\mathrm{T}}$$
$$\widetilde{E}_{\mathrm{R},\alpha} \approx V_{\mathrm{R},\alpha,r} V_{\mathrm{R},\alpha,r}^{\mathrm{T}} \widetilde{E}_{\mathrm{R},\alpha} = V_{\mathrm{R},\alpha,r} W_{\mathrm{R},\alpha,r}^{\mathrm{T}}$$

with $V_{\mathrm{L},\alpha,r}$, $V_{\mathrm{R},\alpha,r} \in \mathbb{R}^{n_p \times r}$ and $W_{\mathrm{L},\alpha,r} \coloneqq E_{\mathrm{L},\alpha}^{\mathrm{T}} V_{\mathrm{L},\alpha,r}$, $W_{\mathrm{R},\alpha,r} \coloneqq E_{\mathrm{R},\alpha}^{\mathrm{T}} V_{\mathrm{R},\alpha,r} \in \mathbb{R}^{n_p \times r}$. The left and right low-rank updates are then constructed as for those based on the randomized range finder in Equation (4.11) by setting $Q_{\mathrm{L},\alpha,r} = V_{\mathrm{L},\alpha,r}$, $N_{\mathrm{L},\alpha,r} = W_{\mathrm{L},\alpha,r}$ or $Q_{\mathrm{R},\alpha,r} = V_{\mathrm{R},\alpha,r}$, $N_{\mathrm{R},\alpha,r} = W_{\mathrm{R},\alpha,r}$. We obtain the updated preconditioners

$$\widehat{S}^{-1}_{\mathrm{ArnoldiP,L},\alpha,r} = \alpha\Big(\mathrm{I}_{n_p} + V_{\mathrm{L},\alpha,r}\big(\mathrm{I}_r - W_{\mathrm{L},\alpha,r}^{\mathrm{T}} V_{\mathrm{L},\alpha,r}\big)^{-1} W_{\mathrm{L},\alpha,r}^{\mathrm{T}}\Big)\widehat{S}^{-1}, \tag{4.13a}$$
$$\widehat{S}^{-1}_{\mathrm{ArnoldiP,R},\alpha,r} = \alpha\widehat{S}^{-1}\Big(\mathrm{I}_{n_p} + V_{\mathrm{R},\alpha,r}\big(\mathrm{I}_r - W_{\mathrm{R},\alpha,r}^{\mathrm{T}} V_{\mathrm{R},\alpha,r}\big)^{-1} W_{\mathrm{R},\alpha,r}^{\mathrm{T}}\Big). \tag{4.13b}$$

For small update ranks, the derived preconditioner updates in (4.11), (4.13), and (4.12) only introduce small additional application costs compared to the initial preconditioner. Besides the application of the initial preconditioner $\widehat{S}^{-1}$, we only require multiplication with thin rectangular $r \times n_p$ and $n_p \times r$ matrices and solving an $r \times r$ system. Additionally, the relaxation parameter introduces the costs of scaling a vector of size n_p which is negligible compared to the application of $\widehat{S}^{-1}$ but may improve the (updated) preconditioners significantly.

The derived low-rank updates are based on low-rank approximations which require prior knowledge of a suitable update rank. It may hence happen that the chosen update rank does not lead to satisfying convergence behavior while using the updated preconditioner in the iterative solver. We might overcome this problem by applying a further low-rank correction to the updated preconditioner. With this approach, we can reuse the already computed update vectors and hopefully require only a small rank for the additional low-rank correction. Furthermore, reusing precomputed vectors from the first update might save computational time in the setup of the second update as we discuss in Section 4.6.3.

In the next section, we derive a repeated update scheme that leads to preconditioners that can be adjusted repeatedly to improve the performance of the preconditioner in the iterative solver. With this scheme, we can reuse the same initial preconditioner and adapt it to similar saddle-point systems.

4.4 Repeated updates

We first derive the formula for a second update, given one of the updated preconditioners in (4.11) or (4.12). Then, we generalize the update formula to arbitrary many updates.

To derive the repeated update scheme, we replace the initial preconditioner $\widehat{S}^{-1}$ in Equation (4.9) by the updated preconditioner $\widehat{S}^{-1}_{\mathrm{L},\alpha,r}$ or $\widehat{S}^{-1}_{\mathrm{R},\alpha,r}$, respectively, and obtain

$$\widetilde{E}_{\mathrm{L},2} = \mathrm{I} - \alpha_2 \widehat{S}^{-1}_{\mathrm{L},\alpha,r}\widetilde{S}, \qquad \widetilde{E}_{\mathrm{R},2} = \mathrm{I} - \alpha_2 \widetilde{S}\widehat{S}^{-1}_{\mathrm{R},\alpha,r}. \tag{4.14}$$

Solving for the approximate Schur complement $\widetilde{S}$ and inverting yields

$$S^{-1} \approx \widetilde{S}^{-1} = \left(\mathrm{I}_{n_p} - \widetilde{E}_{\mathrm{L},2}\right)^{-1} \widehat{S}^{-1}_{\mathrm{L},\alpha,r}, \tag{4.15}$$

$$S^{-1} \approx \widetilde{S}^{-1} = \widehat{S}^{-1}_{\mathrm{R},\alpha,r}\left(\mathrm{I}_{n_p} - \widetilde{E}_{\mathrm{R},2}\right)^{-1}. \tag{4.16}$$

We replace $\widetilde{E}_{\mathrm{L},2}$ and $\widetilde{E}_{\mathrm{R},2}$ with low-rank approximations and rearrange to find the new preconditioners.

To construct the repeated update schemes, we exploit the same approximation techniques as in the previous section: the randomized low-rank approximation with Algorithm 2.2 and the Arnoldi iteration given in Algorithm 2.4. In the following, we assume that we use the same low-rank approximation method in both updates to make the derivation more readable but this is not required in practice.

We start with the randomized low-rank factorization in Algorithm 2.2 and obtain the approximations $\widetilde{E}_{\mathrm{L},2} \approx Q_{\mathrm{L},\alpha_2,r_2} N^{\mathrm{T}}_{\mathrm{L},\alpha_2,r_2}$ and $\widetilde{E}_{\mathrm{R},2} \approx Q_{\mathrm{R},\alpha_2,r_2} N^{\mathrm{T}}_{\mathrm{R},\alpha_2,r_2}$ with $Q_{\mathrm{L},\alpha_2,r_2}$, $N_{\mathrm{L},\alpha_2,r_2}$, $Q_{\mathrm{R},\alpha_2,r_2}$, $N_{\mathrm{R},\alpha_2,r_2} \in \mathbb{R}^{n_p \times r_2}$. Inserting these low-rank approximations in Equation (4.16) and applying the Sherman-Morrison-Woodbury formula leads to the preconditioners

$$\begin{aligned}
\widehat{S}^{-1}_{\mathrm{random,L},2} = \alpha\alpha_2 &\Big(\mathrm{I}_{n_p} + Q_{\mathrm{L},\alpha_2,r_2}\big(\mathrm{I}_{r_2} - N^{\mathrm{T}}_{\mathrm{L},\alpha_2,r_2} Q_{\mathrm{L},\alpha_2,r_2}\big)^{-1} N^{\mathrm{T}}_{\mathrm{L},\alpha_2,r_2}\Big) \\
&\Big(\mathrm{I}_{n_p} + Q_{\mathrm{L},\alpha,r}\big(\mathrm{I}_{r} - N^{\mathrm{T}}_{\mathrm{L},\alpha,r} Q_{\mathrm{L},\alpha,r}\big)^{-1} N^{\mathrm{T}}_{\mathrm{L},\alpha,r}\Big)\widehat{S}^{-1}, \\
\widehat{S}^{-1}_{\mathrm{random,R},2} = \alpha\alpha_2 \widehat{S}^{-1} &\Big(\mathrm{I}_{n_p} + Q_{\mathrm{R},\alpha,r}\big(\mathrm{I}_{r} - N^{\mathrm{T}}_{\mathrm{R},\alpha,r} Q_{\mathrm{R},\alpha,r}\big)^{-1} N^{\mathrm{T}}_{\mathrm{R},\alpha,r}\Big) \\
&\Big(\mathrm{I}_{n_p} + Q_{\mathrm{R},\alpha_2,r_2}\big(\mathrm{I}_{r_2} - N^{\mathrm{T}}_{\mathrm{R},\alpha_2,r_2} Q_{\mathrm{R},\alpha_2,r_2}\big)^{-1} N^{\mathrm{T}}_{\mathrm{R},\alpha_2,r_2}\Big).
\end{aligned}$$

A repeated update scheme based on the Arnoldi Algorithm 2.4 followed by projection can be derived analogously. From the low-rank approximations $\widetilde{E}_{\mathrm{L},2} \approx V_{\mathrm{L},\alpha_2,r_2} W^{\mathrm{T}}_{\mathrm{L},\alpha_2,r_2}$ and $\widetilde{E}_{\mathrm{R},2} \approx V_{\mathrm{R},\alpha_2,r_2} W^{\mathrm{T}}_{\mathrm{R},\alpha_2,r_2}$ with $V_{\mathrm{L},\alpha_2,r_2}$, $W_{\mathrm{L},\alpha_2,r_2}$, $V_{\mathrm{R},\alpha_2,r_2}$, $W_{\mathrm{R},\alpha_2,r_2} \in \mathbb{R}^{n_p \times r_2}$, we find the second update by replacing the matrices $Q_{\mathrm{L},\alpha_2,r_2}$, $Q_{\mathrm{R},\alpha_2,r_2}$ by the matrices $V_{\mathrm{L},\alpha_2,r_2}$, $V_{\mathrm{R},\alpha_2,r_2}$ and by replacing the matrices $N_{\mathrm{L},\alpha_2,r_2}$, $N_{\mathrm{R},\alpha_2,r_2}$ by the matrices $W_{\mathrm{L},\alpha_2,r_2}$, $W_{\mathrm{R},\alpha_2,r_2}$ in Equation (4.17).

Now, we consider a repeated update that utilizes the Arnoldi iteration from Algorithm 2.4 to compute the low-rank approximations $\widetilde{E}_{\mathrm{L},\alpha_2,2} \approx V_{\mathrm{L},\alpha_2,2} H_{\mathrm{L},\alpha_2,2} V^{\mathrm{T}}_{\mathrm{L},\alpha_2,2}$ and $\widetilde{E}_{\mathrm{R},\alpha_2,\mathrm{upd}} \approx V_{\mathrm{R},\alpha_2,2} H_{\mathrm{R},\alpha_2,2} V^{\mathrm{T}}_{\mathrm{R},\alpha_2,2}$ with $V_{\mathrm{L},\alpha_2,2}, V_{\mathrm{R},\alpha_2,2} \in \mathbb{R}^{n_p \times r}$ and $H_{\mathrm{L},\alpha_2,2}, H_{\mathrm{R},\alpha_2,2} \in \mathbb{R}^{r \times r}$. We insert the approximations in Equation (4.16), rearrange and obtain the preconditioners

$$\begin{aligned}
\widehat{S}^{-1}_{\mathrm{Arnoldi,L},2} = \alpha\alpha_2 &\Big(\mathrm{I}_{n_p} + V_{\mathrm{L},\alpha_2,r_2}\Big((\mathrm{I}_{r_2} - H_{\mathrm{L},\alpha_2,r_2})^{-1} - \mathrm{I}_{r_2}\Big) V^{\mathrm{T}}_{\mathrm{L},\alpha_2,r_2}\Big) \\
&\Big(\mathrm{I}_{n_p} + V_{\mathrm{L},\alpha,r}\Big((\mathrm{I}_{r} - H_{\mathrm{L},\alpha,r})^{-1} - \mathrm{I}_{r}\Big) V^{\mathrm{T}}_{\mathrm{L},\alpha,r}\Big)\widehat{S}^{-1},
\end{aligned} \tag{4.18a}$$

$$\begin{aligned}
\widehat{S}^{-1}_{\mathrm{Arnoldi,R},2} = \alpha\alpha_2 \widehat{S}^{-1} &\Big(\mathrm{I}_{n_p} + V_{\mathrm{R},\alpha,r}\Big((\mathrm{I}_{r} - H_{\mathrm{R},\alpha,r})^{-1} - \mathrm{I}_{r}\Big) V^{\mathrm{T}}_{\mathrm{R},\alpha,r}\Big) \\
&\Big(\mathrm{I}_{n_p} + V_{\mathrm{R},\alpha_2,r_2}\Big((\mathrm{I}_{r_2} - H_{\mathrm{R},\alpha_2,r_2})^{-1} - \mathrm{I}_{r_2}\Big) V^{\mathrm{T}}_{\mathrm{R},\alpha_2,r_2}\Big).
\end{aligned} \tag{4.18b}$$

We can repeat the update schemes arbitrarily often which leads to a flexibly updated preconditioner. In each update step, we can choose a new rank r_i, relaxation parameter α_i, and vary the low-rank approximation method.

We introduce the following definitions to obtain a readable update formula:

$$X_{\text{random,L},i} \coloneqq Q_{\text{L},\alpha_i,r_i}\big(\mathrm{I}_{r_i} - N_{\text{L},\alpha_i,r_i}^{\mathrm{T}} Q_{\text{L},\alpha_i,r_i}\big)^{-1} N_{\text{L},\alpha_i,r_i}^{\mathrm{T}},$$
$$X_{\text{Arnoldi,L},i} \coloneqq V_{\text{L},\alpha_i,r_i}\Big((\mathrm{I}_{r_i} - H_{\text{L},\alpha_i,r_i})^{-1} - \mathrm{I}_{r_i}\Big) V_{\text{L},\alpha_i,r_i}^{\mathrm{T}},$$

and

$$X_{\text{random,R},i} \coloneqq Q_{\text{R},\alpha_i,r_i}\big(\mathrm{I}_{r_i} - N_{\text{R},\alpha_i,r_i}^{\mathrm{T}} Q_{\text{R},\alpha_\mathrm{i},\mathrm{r_i}}\big)^{-1} N_{\text{R},\alpha_i,r_i}^{\mathrm{T}},$$
$$X_{\text{Arnoldi,R},i} \coloneqq V_{\text{R},\alpha_i,r_i}\Big((\mathrm{I}_{r_i} - H_{\text{R},\alpha_i,r_i})^{-1} - \mathrm{I}_{r_i}\Big) V_{\text{R},\alpha_i,r_i}^{\mathrm{T}}.$$

Inserting these definitions into Equations (4.17) or (4.18) and generalizing to N updates, we obtain the preconditioners

$$\widehat{S}_{\text{L},N}^{-1} = \left(\prod_{i=1}^{N} \alpha_i\big(\mathrm{I}_{n_p} + X_{(\cdot),\text{L},i}\big)\right)\widehat{S}^{-1}, \tag{4.19a}$$

$$\widehat{S}_{\text{R},N}^{-1} = \widehat{S}^{-1}\left(\prod_{i=1}^{N} \alpha_i\big(\mathrm{I}_{n_p} + X_{(\cdot),\text{R},i}\big)\right). \tag{4.19b}$$

where r_i is the ith update rank, α_i the ith relaxation parameter, and $X_{(\cdot),\text{L},N}$, $X_{(\cdot),\text{R},N}$ describe the ith update for $i = 1, 2, \ldots, N$. The index $(\cdot)$ indicates the utilized low-rank approximation method.

4.5 Schur complement preconditioners with inner updates

In this section, we discuss inner low-rank updates applied to the inner Poisson-type problems that arise in the SIMPLE (3.44) and the LSC preconditioner (3.43). We will observe in Section 5.2 that the accuracy of the (approximate) solution of the inner Poisson-type problems has a significant influence on the convergence behavior of the outer iterative solver. However, accurate inner solvers are computationally expensive, either in the setup as for direct solvers or in the application as for (inexact) iterative solvers. We hope that a low-rank correction applied to a cheap initial approximation can reduce the number of outer iterations with less computational costs. The construction of the update technique for the inner Poisson-type problems is significantly cheaper than the setup of the outer updates for the Schur complement preconditioners. We hence expect that the low-rank correction not only reduces iteration counts but also solver times.

Before we start the derivation of the update scheme, we recall the LSC preconditioner (3.43)

$$\widehat{S}_{\text{LSC}}^{-1} = \widehat{M}_{\text{BMB}^{\mathrm{T}}}^{-1} B D_u^{-1} A D_u^{-1} B^{\mathrm{T}} \widehat{M}_{\text{BMB}^{\mathrm{T}}}^{-1}$$

with $\widehat{M}_{\text{BMB}^{\mathrm{T}}} \approx B D_u^{-1} B^{\mathrm{T}}$, $D_u = \operatorname{diag}(M_u)$, and the SIMPLE preconditioner (3.44)

$$\widehat{S}_{\text{SIMPLE}}^{-1} = \widehat{M}_{\text{BDB}^{\mathrm{T}}}^{-1}$$

with $\widehat{M}_{\text{BDB}^{\mathrm{T}}} \approx B \operatorname{diag}(A)^{-1} B^{\mathrm{T}}$. Both Schur complement preconditioners require the (approximate) solution of inner Poisson-type problems of the type

$$B D^{-1} B^{\mathrm{T}} x = y$$

where $D \in \mathbb{R}^{n_u \times n_u}$ is a diagonal matrix. In the following, we denote the system matrix of the Poisson-type problems by

$$M \coloneqq BD^{-1}B^{\mathrm{T}}$$

and assume that we have an approximation to its inverse, i.e. a preconditioner

$$\widehat{M}^{-1} \approx M^{-1}.$$

To improve the accuracy of the approximation $\widehat{M}^{-1}$, we construct a low-rank correction as discussed in Section 4.1. We then replace the approximations of the inner Poisson-type problems with the corrected versions. To compute the low-rank approximations, we consider the randomized Algorithm 2.2 and the Arnoldi Algorithm 2.4.

For the LSC preconditioner, we find the low-rank approximation

$$E_{M,\mathrm{L}} = \mathrm{I}_{n_p} - \widehat{M}^{-1}_{\mathrm{BMB^T}} BD_u^{-1}B^{\mathrm{T}} \approx Q_{M,\mathrm{L},r} N_{M,\mathrm{L},r}^{\mathrm{T}}, \tag{4.20a}$$

$$E_{M,\mathrm{R}} = \mathrm{I}_{n_p} - BD_u^{-1}B^{\mathrm{T}} \widehat{M}^{-1}_{\mathrm{BMB^T}} \approx Q_{M,\mathrm{R},r} N_{M,\mathrm{R},r}^{\mathrm{T}} \tag{4.20b}$$

with the randomized Algorithm 2.2 with $Q_{M,\mathrm{L},r}$, $N_{M,\mathrm{L},r}$, $Q_{M,\mathrm{R},r}$, $N_{M,\mathrm{R},r} \in \mathbb{R}^{n_p \times r}$. The index M refers to the index B$\underline{\mathrm{M}}$B$^{\mathrm{T}}$. We obtain the corrected LSC-type preconditioners

$$\widehat{S}^{-1}_{M,\mathrm{LSC,random,L}} = \big(\mathrm{I}_{n_p} + X_{M,\mathrm{L},r}\big)\widehat{M}^{-1}_{\mathrm{BMB^T}} BD_u^{-1}AD_u^{-1}B^{\mathrm{T}}\big(\mathrm{I}_{n_p} + X_{M,\mathrm{L},r}\big)\widehat{M}^{-1}_{\mathrm{BMB^T}} \tag{4.21}$$

with $X_{M,\mathrm{L},r} \coloneqq Q_{M,\mathrm{L},r}\Big(\mathrm{I}_r - N_{M,\mathrm{L},r}^{\mathrm{T}} Q_{M,\mathrm{L},r}\Big)N_{M,\mathrm{L},r}^{\mathrm{T}}$ and

$$\widehat{S}^{-1}_{M,\mathrm{LSC,random,R}} = \widehat{M}^{-1}_{\mathrm{BMB^T}}\big(\mathrm{I}_{n_p} + X_{M,\mathrm{R},r}\big)BD_u^{-1}AD_u^{-1}B^{\mathrm{T}}\widehat{M}^{-1}_{\mathrm{BMB^T}}\big(\mathrm{I}_{n_p} + X_{M,\mathrm{R},r}\big) \tag{4.22}$$

with $X_{M,\mathrm{R},r} \coloneqq Q_{M,\mathrm{R},r}\Big(\mathrm{I}_r - N_{M,\mathrm{R},r}^{\mathrm{T}} Q_{M,\mathrm{R},r}\Big)N_{M,\mathrm{R},r}^{\mathrm{T}}$. Similar update formulas can also be found by the Arnoldi Algorithm 2.4 followed by projection.

With the Arnoldi iteration from Algorithm 2.4, we find the low-rank approximation

$$E_{M,\mathrm{L}} = \mathrm{I}_{n_p} - \widehat{M}^{-1}_{\mathrm{BMB^T}} BD_u^{-1}B^{\mathrm{T}} \approx V_{M,\mathrm{L},r} H_{M,\mathrm{L},r} V_{M,\mathrm{L},r}^{\mathrm{T}},$$
$$E_{M,\mathrm{R}} = \mathrm{I}_{n_p} - BD_u^{-1}B^{\mathrm{T}} \widehat{M}^{-1}_{\mathrm{BMB^T}} \approx V_{M,\mathrm{R},r} H_{M,\mathrm{L},r} V_{M,\mathrm{R},r}^{\mathrm{T}}.$$

We obtain the LSC-type preconditioners

$$\widehat{S}^{-1}_{M,\mathrm{LSC,Arnoldi,L}} = \big(\mathrm{I}_{n_p} + Y_{M,\mathrm{L},r}\big)\widehat{M}^{-1}_{\mathrm{BMB^T}} BD_u^{-1}AD_u^{-1}B^{\mathrm{T}}\big(\mathrm{I}_{n_p} + Y_{M,\mathrm{L},r}\big)\widehat{M}^{-1}_{\mathrm{BMB^T}}. \tag{4.23}$$

with $Y_{M,\mathrm{L},r} \coloneqq V_{M,\mathrm{L},r}\Big(\big(\mathrm{I}_r - H_{M,\mathrm{L},r}\big)^{-1} - \mathrm{I}_r\Big)V_{M,\mathrm{L},r}^{\mathrm{T}}$ and

$$\widehat{S}^{-1}_{M,\mathrm{LSC,Arnoldi,R}} = \widehat{M}^{-1}_{\mathrm{BMB^T}}\big(\mathrm{I}_{n_p} + Y_{M,\mathrm{R},r}\big)BD_u^{-1}AD_u^{-1}B^{\mathrm{T}}\widehat{M}^{-1}_{\mathrm{BMB^T}}\big(\mathrm{I}_{n_p} + Y_{M,\mathrm{R},r}\big) \tag{4.24}$$

with $Y_{M,\mathrm{R},r} \coloneqq V_{M,\mathrm{R},r}\Big(\big(\mathrm{I}_r - H_{M,\mathrm{R},r}\big)^{-1} - \mathrm{I}_r\Big)V_{M,\mathrm{R},r}^{\mathrm{T}}$.

For the SIMPLE preconditioner, the randomized Algorithm 2.2 yields the low-rank approximation

$$E_{D,\mathrm{L}} = \mathrm{I}_{n_p} - \widehat{M}^{-1}_{\mathrm{BDB^T}} BD_u^{-1}B^{\mathrm{T}} \approx Q_{D,\mathrm{L},r} N_{D,\mathrm{L},r}^{\mathrm{T}}, \tag{4.25a}$$

$$E_{D,\mathrm{R}} = \mathrm{I}_{n_p} - BD_u^{-1}B^{\mathrm{T}} \widehat{M}^{-1}_{\mathrm{BDB^T}} \approx Q_{D,\mathrm{R},r} N_{D,\mathrm{R},r}^{\mathrm{T}}, \tag{4.25b}$$

with $Q_{D,\mathrm{L},r}$, $N_{D,\mathrm{L},r}$, $Q_{D,\mathrm{R},r}$, $N_{D,\mathrm{R},r} \in \mathbb{R}^{n_p \times r}$. Here, the index D refers to the index $\mathrm{B}\underline{\mathrm{D}}\mathrm{B}^{\mathrm{T}}$. We obtain the left- and right-updated SIMPLE-type preconditioners

$$\widehat{S}^{-1}_{D,\mathrm{SIMPLE,random,L}} = \left(\mathrm{I}_{n_p} + X_{D,\mathrm{L},r}\right)\widehat{M}^{-1}_{\mathrm{BDB^T}} \tag{4.26}$$

with $X_{D,\mathrm{L},r} \coloneqq Q_{D,\mathrm{L},r}\Big(\mathrm{I}_r - N^{\mathrm{T}}_{D,\mathrm{L},r}Q_{D,\mathrm{L},r}\Big)N^{\mathrm{T}}_{D,\mathrm{L},r}$ and

$$\widehat{S}^{-1}_{D,\mathrm{SIMPLE,random,R}} = \widehat{M}^{-1}_{\mathrm{BDB^T}}\left(\mathrm{I}_{n_p} + X_{D,\mathrm{R},r}\right) \tag{4.27}$$

with $X_{D,\mathrm{R},r} \coloneqq Q_{D,\mathrm{R},r}\Big(\mathrm{I}_r - N^{\mathrm{T}}_{D,\mathrm{R},r}Q_{D,\mathrm{R},r}\Big)N^{\mathrm{T}}_{D,\mathrm{R},r}$. The Arnoldi iteration from Algorithm 2.4 yields the low-rank approximation

$$\begin{aligned} E_{D,\mathrm{L}} &= \mathrm{I}_{n_p} - \widehat{M}^{-1}_{\mathrm{BDB^T}}BD_u^{-1}B^{\mathrm{T}} \approx V_{D,\mathrm{L},r}H_{D,\mathrm{L},r}V^{\mathrm{T}}_{D,\mathrm{L},r}, \\ E_{D,\mathrm{R}} &= \mathrm{I}_{n_p} - BD_u^{-1}B^{\mathrm{T}}\widehat{M}^{-1}_{\mathrm{BDB^T}} \approx V_{D,\mathrm{R},r}H_{D,\mathrm{L},r}V^{\mathrm{T}}_{D,\mathrm{R},r}. \end{aligned}$$

We obtain the updated SIMPLE-type preconditioners

$$\widehat{S}^{-1}_{\mathrm{D,Arnoldi,SIMPLE}} = \left(\mathrm{I}_{n_p} + Y_{D,\mathrm{L},r}\right)\widehat{M}^{-1}_{\mathrm{BDB^T}} \tag{4.28}$$

with $Y_{D,\mathrm{L},r} \coloneqq V_{D,\mathrm{L},r}\Big(\left(\mathrm{I}_r - H_{D,\mathrm{L},r}\right)^{-1} - \mathrm{I}_r\Big)V^{\mathrm{T}}_{D,\mathrm{L},r}$ and

$$\widehat{S}^{-1}_{\mathrm{D,Arnoldi,SIMPLE}} = \widehat{M}^{-1}_{\mathrm{BDB^T}}\left(\mathrm{I}_{n_p} + Y_{D,\mathrm{R},r}\right) \tag{4.29}$$

with $Y_{D,\mathrm{R},r} \coloneqq V_{D,\mathrm{R},r}\Big(\left(\mathrm{I}_r - H_{D,\mathrm{R},r}\right)^{-1} - \mathrm{I}_r\Big)V^{\mathrm{T}}_{D,\mathrm{R},r}$. Update variations such as the repeated update scheme from the previous Section 4.4 can straightforwardly be applied to inner updates.

4.6 Construction and application

Now, we discuss the practical construction and application of the update schemes as well as the corresponding computational complexities. In this section, we only consider right updates since the complexities are the same for left updates and the derivations for the left updates are analogous. We restrict the complexity analysis to updates based on the Arnoldi iteration tending to be the cheapest approach and the randomized range finder with powering ($q > 0$) tending to be the most expensive approach of the considered low-rank approximation methods. We discuss the construction and application of the updated preconditioners $\widehat{S}^{-1}_{\mathrm{random,R},\alpha,r}$ (4.11) and $\widehat{S}^{-1}_{\mathrm{Arnoldi,R},\alpha,r}$ (4.12) obtained with the randomized low-rank approximation and with the Arnoldi iteration, respectively. Section 4.6.1 is concerned with the construction along with setup costs of updated preconditioners for different low-rank approximation strategies. In Section 4.6.2, we consider the practical application of the update schemes and corresponding computational costs. We then discuss the construction and application of repeated updates in Section 4.6.3. Here, we study the recycling of sample vectors in the setup with the randomized low-rank approximation. We conclude this section with a complexity analysis for inner updates for Schur complement preconditioners in Section 4.6.4.

We now define the setting and some parameters that we will need to determine the computational complexities. We consider the right-preconditioned linear saddle-point problem

$$\mathcal{A}P_{\mathrm{U}}\begin{pmatrix}\mathbf{x}\\ y\end{pmatrix}=\begin{pmatrix}\mathbf{f}\\ g\end{pmatrix}, \qquad \begin{pmatrix}\mathbf{u}\\ p\end{pmatrix}=P_{\mathrm{U}}\begin{pmatrix}\mathbf{x}\\ y\end{pmatrix} \tag{4.30}$$

with the system matrix $\mathcal{A} \coloneqq \left(\begin{smallmatrix} A & B^{\mathrm{T}} \\ B & 0 \end{smallmatrix}\right)$, and the block preconditioner $P_{\mathrm{U}} = \left(\begin{smallmatrix} \widehat{A}^{-1} & \widehat{A}^{-1}B^{\mathrm{T}}\widehat{S}^{-1} \\ 0 & -\widehat{S}^{-1} \end{smallmatrix}\right)$ from (3.35). The matrix blocks $A \in \mathbb{R}^{n_u \times n_u}$, $B \in \mathbb{R}^{n_p \times n_u}$ are sparse and $\widehat{A}^{-1} \approx A^{-1}$, $\widehat{S}^{-1} \approx S^{-1} = (BA^{-1}B^{\mathrm{T}})^{-1}$ are the initial preconditioners. We furthermore have the vectors $\mathbf{x}, \mathbf{u}, \mathbf{f} \in \mathbb{R}^{n_u}$ and $y, p, g \in \mathbb{R}^{n_p}$.

Definition 4.1. We define the parameters c_A, c_B, $c_{B^{\mathrm{T}}}$, c_M, and c_u via the following equations. The number of nonzero entries of A, B, and B^{T} are given by $\mathrm{nnz}(A) = c_A n_u$, $\mathrm{nnz}(B) = c_B n_p$, and $\mathrm{nnz}(B^{\mathrm{T}}) = c_{B^{\mathrm{T}}} n_u$, respectively, where $\mathrm{nnz}(\cdot)$ denotes the number of nonzero entries of the argument. The dimensions n_u and n_p satisfy the relation $n_u = c_u n_p$. Note that $c_{B^{\mathrm{T}}} = \frac{c_B}{c_u}$ is satisfied since

$$c_B \frac{n_u}{c_u} = c_B n_p = \mathrm{nnz}(B) = \mathrm{nnz}(B^{\mathrm{T}}) = c_{B^{\mathrm{T}}} n_u. \tag{4.31}$$

We furthermore require the sparsity factor c_M that describes the ratio of the number of nonzero entries of the matrix $M \coloneqq BD^{-1}B^{\mathrm{T}}$ with the diagonal matrix $D \in \mathbb{R}^{n_u \times n_u}$ to the number of rows of M, i.e. it is given by $\mathrm{nnz}(M) = c_M n_p$.

4.6.1 Construction

We start with the construction of the preconditioner $\widehat{S}^{-1}_{\mathrm{random,R},\alpha,r}$ (4.11) that employs the randomized low-rank approximation in Algorithm 2.2. We then discuss the setup of the updated preconditioner $\widehat{S}^{-1}_{\mathrm{Arnoldi,R},\alpha,r}$ (4.12) based on the Arnoldi iteration in Algorithm 2.4.

For the setup of the preconditioner $\widehat{S}^{-1}_{\mathrm{random,R},\alpha,r}$ (4.11), we compute the low-rank approximation as $\widetilde{E}^{\mathrm{T}}_{\mathrm{R},\alpha} \approx \widetilde{Q}_{\mathrm{R},\alpha,r}\widetilde{N}^{\mathrm{T}}_{\mathrm{R},\alpha,r}$ with $Q_{\mathrm{R},\alpha,r} \coloneqq \widetilde{N}_{\mathrm{R},\alpha,r}$, $N_{\mathrm{R},\alpha,r} \coloneqq \widetilde{Q}_{\mathrm{R},\alpha,r}$ or as $\widetilde{E}_{\mathrm{R},\alpha} \approx Q_{\mathrm{R},\alpha,r}N^{\mathrm{T}}_{\mathrm{R},\alpha,r}$ by utilizing Algorithm 2.2. Then, we precompute the small $r \times r$ matrix $H_r \coloneqq \mathrm{I}_r - N^{\mathrm{T}}_{\mathrm{R},\alpha,r}Q_{\mathrm{R},\alpha,r}$ and its (exact) LU factorization $H_r = L_r U_r$. We summarize the required setup steps in Algorithm 4.1.

We now discuss the setup costs of the update. To compute the low-rank approximation, we require an $n_p \times \ell_r$ random test matrix. The number of columns $\ell_r = r + k_r$ is given by the sum of the update rank r and the number of oversampling vectors k_r. Filling this matrix with random numbers costs $N_{\mathrm{fillG}} = \mathcal{O}(n_p \ell_r)$ operations. In the power iteration of the randomized range finder, we then multiply $2(q+1)$ times with the error matrix $\widetilde{E}_{\mathrm{R},\alpha} = \mathrm{I}_{n_p} - \alpha\widetilde{S}\widehat{S}^{-1}$ or its transpose. Each multiplication of $\widetilde{E}_{\mathrm{R},\alpha}$ with a vector requires applying the preconditioner $\widehat{S}^{-1}$ and the approximate Schur complement $\widetilde{S}$. Furthermore, we scale a vector of size n_p with the relaxation parameter and subtract a vector of size n_p. Applying $\widetilde{E}_{\mathrm{R},\alpha}$ to a vector hence costs

$$N_{\mathrm{mult}\widetilde{\mathrm{E}}_\alpha} = N_{\mathrm{mult}\widetilde{\mathrm{S}}} + N_{\mathrm{mult}\widehat{\mathrm{S}}^{-1}} + 2n_p \tag{4.32}$$

operations where $N_{\mathrm{mult}\widetilde{\mathrm{S}}}$ denotes the costs of applying the approximate Schur complement $\widetilde{S}$ and $N_{\mathrm{mult}\widehat{\mathrm{S}}^{-1}}$ denotes the costs of applying the preconditioner $\widehat{S}^{-1}$. Applying the matrix

Algorithm 4.1: Construction of the update with the randomized power range finder.

Input: System matrix $\mathcal{A} = \left(\begin{smallmatrix} A & B^{\mathrm{T}} \\ B & 0 \end{smallmatrix}\right)$, preconditioner $\widehat{A}^{-1}$, initial Schur complement preconditioner $\widehat{S}^{-1}$, rank r, relaxation parameter α, number of power iterations q, oversampling parameter k_r, flag `approx_transposed`.

Output: Matrices $Q_{\mathrm{R},\alpha,r}$, $N_{\mathrm{R},\alpha,r}$, L_r, U_r defining the updated preconditioner $\widehat{S}^{-1}_{\mathrm{random,R},\alpha,r}$ (4.11).

Define the approximate Schur complement $\widetilde{S} \coloneqq B\widehat{A}^{-1}B^{\mathrm{T}}$.
if `approx_transposed` **then**
  Compute a low-rank approximation of $\widetilde{E}^{\mathrm{T}}_{\mathrm{R},\alpha} \coloneqq \mathrm{I}_{n_p} - \alpha\widehat{S}^{-\mathrm{T}}\widetilde{S}^{\mathrm{T}} \approx \widetilde{Q}_{\mathrm{R},\alpha,r}\widetilde{N}^{\mathrm{T}}_{\mathrm{R},\alpha,r}$ with Algorithm 2.2.
  Set $Q_{\mathrm{R},\alpha,r} \coloneqq \widetilde{N}_{\mathrm{R},\alpha,r}$ and $N_{\mathrm{R},\alpha,r} \coloneqq \widetilde{Q}_{\mathrm{R},\alpha,r}$.
else
  Compute a low-rank approximation of $\widetilde{E}_{\mathrm{R},\alpha} \coloneqq \mathrm{I}_{n_p} - \alpha\widetilde{S}\widehat{S}^{-1} \approx Q_{\mathrm{R},\alpha,r}N^{\mathrm{T}}_{\mathrm{R},\alpha,r}$ with Algorithm 2.2.
end
Compute $H_r = \mathrm{I}_r - N^{\mathrm{T}}_{\mathrm{R},\alpha,r}Q_{\mathrm{R},\alpha,r}$.
Compute an LU factorization of $H_r = L_rU_r$.
Define the preconditioner $\widehat{S}^{-1}_{\mathrm{random,R},\alpha,r} = \alpha\widehat{S}^{-1}\Big(\mathrm{I}_{n_p} + Q_{\mathrm{R},\alpha,r}(L_rU_r)^{-1}N^{\mathrm{T}}_{\mathrm{R},\alpha,r}\Big)$.

$\widetilde{S} = B\widehat{A}^{-1}B^{\mathrm{T}}$ to a vector involves multiplications with the matrix blocks $B \in \mathbb{R}^{n_p \times n_u}$ and $B^{\mathrm{T}} \in \mathbb{R}^{n_u \times n_p}$ and an application of the preconditioner $\widehat{A}^{-1}$ to a vector of size n_u. We obtain

$$\begin{aligned} N_{\mathrm{mult}\widetilde{\mathrm{S}}} &= N_{\mathrm{multB}} + N_{\mathrm{mult}\widehat{\mathrm{A}}^{-1}} + N_{\mathrm{multB}^{\mathrm{T}}} \\ &= n_p(2c_B - 1) + N_{\mathrm{mult}\widehat{\mathrm{A}}^{-1}} + n_u(2c_{B^{\mathrm{T}}} - 1) \\ &= N_{\mathrm{mult}\widehat{\mathrm{A}}^{-1}} + (4c_B - c_u - 1)n_p \end{aligned}$$

with Equation (4.31).

After applying $\widetilde{E}_{\mathrm{R},\alpha}$ or its transpose to a block of ℓ_r vectors we orthogonalize and optionally re-orthogonalize the results with the modified Gram-Schmidt method. If we re-orthogonalize in each orthogonalization step, we require $N_{\mathrm{orth}} = \mathcal{O}(4n_p\ell_r(\ell_r + 1))$ floating point operations per call of the modified Gram-Schmidt procedure.

Precomputing the matrix $H_r = \mathrm{I}_r - N^{\mathrm{T}}_{\mathrm{R},\alpha,r}Q_{\mathrm{R},\alpha,r}$ involves a matrix-matrix multiplication of an $r \times n_p$ and an $n_p \times r$ matrix. Furthermore, we subtract the result from the $r \times r$ identity matrix. The costs for explicitly computing H_r add up to $N_{H_r} = 2n_pr^2 + r = \mathcal{O}(2n_pr^2)$ operations. The LU factorization of H_r costs additional $\mathcal{O}(r^3)$ operations.

Altogether, we obtain the following setup costs $N_{\mathrm{setup,random}}$ for the low-rank updates

Algorithm 4.2: Construction of the update with the Arnoldi iteration followed by projection.

Input: System matrix $\mathcal{A} = \left(\begin{smallmatrix} A & B^{\mathrm{T}} \\ B & 0 \end{smallmatrix}\right)$, preconditioner $\widehat{A}^{-1}$, initial Schur complement preconditioner $\widehat{S}^{-1}$, rank r, relaxation parameter α.

Output: Matrices $V_{\mathrm{R},\alpha,r}$, $W_{\mathrm{R},\alpha,r}$, L_r, U_r defining the updated preconditioner $\widehat{S}^{-1}_{\mathrm{ArnoldiP,R}}$.

1 Define the approximate Schur complement $\widetilde{S} \coloneqq B\widehat{A}^{-1}B^{\mathrm{T}}$.

2 Apply the Arnoldi Algorithm 2.4 to $\widetilde{E}_{\mathrm{R},\alpha} \coloneqq \mathrm{I} - \alpha\widetilde{S}\widehat{S}^{-1}$ and store the Arnoldi vectors in $V_{\mathrm{R},\alpha,r}$.

3 Compute $W_{\mathrm{R},\alpha,r} = \widetilde{E}^{\mathrm{T}}_{\mathrm{R},\alpha}V_{\mathrm{R},\alpha,r}$ to obtain the low-rank approximation $\widetilde{E}_{\mathrm{R},\alpha} \approx V_{\mathrm{R},\alpha,r}W^{\mathrm{T}}_{\mathrm{R},\alpha,r}$.

4 Compute $H_r = \mathrm{I}_r - W^{\mathrm{T}}_{\mathrm{R},\alpha,r}V_{\mathrm{R},\alpha,r}$.

5 Compute an LU factorization of $H_r = L_rU_r$.

6 Define the preconditioner $\widehat{S}^{-1}_{\mathrm{ArnoldiP,R},\alpha,r} = \alpha\widehat{S}^{-1}\Big(\mathrm{I}_{n_p} + V_{\mathrm{R},\alpha,r}(L_rU_r)^{-1}W^{\mathrm{T}}_{\mathrm{R},\alpha,r}\Big)$.

for the preconditioner (4.11)

$$\begin{aligned} N_{\mathrm{setup,random}} &= N_{\mathrm{fillG}} + 2(q+1)\ell_r N_{\mathrm{mult}\widetilde{\mathrm{E}}_\alpha} + 2(q+1)N_{\mathrm{orth}} + N_{H_r} + N_{\mathrm{LU}} \\ &= n_p\ell_r + 2(q+1)\ell_r\big(N_{\mathrm{mult}\widehat{\mathrm{A}}^{-1}} + N_{\mathrm{mult}\widehat{\mathrm{S}}^{-1}} + (4c_B - c_u + 1)n_p\big) \\ &\qquad + 2(q+1)\mathcal{O}(4n_p\ell_r(\ell_r+1)) + \mathcal{O}(2n_pr^2) + \mathcal{O}(r^3) \\ &= 2(q+1)\ell_r\big(N_{\mathrm{mult}\widehat{\mathrm{A}}^{-1}} + N_{\mathrm{mult}\widehat{\mathrm{S}}^{-1}}\big) + \mathcal{O}(c_{\mathrm{setup,random}}n_p) \end{aligned} \tag{4.33}$$

where $c_{\mathrm{setup,random}} = 2(q+1)\ell_r(4c_B - c_u + 5) + 8(q+1)\ell_r^2 + 2r^2 + \ell_r$.

We now discuss the update construction for the two approaches that exploit the Arnoldi iteration.

For the setup of the preconditioner based on the Arnoldi iteration followed by projection, we compute the low-rank approximation $\widetilde{E}_{\mathrm{R},\alpha} \approx V_{\mathrm{R},\alpha,r}W^{\mathrm{T}}_{\mathrm{R},\alpha,r}$ with Algorithm 2.4 followed by projection with the transposed error matrix. Then, we precompute the small matrix $H_r \coloneqq \mathrm{I}_r - W^{\mathrm{T}}_{\mathrm{R},\alpha,r}V_{\mathrm{R},\alpha,r} \in \mathbb{R}^{r\times r}$ and its (exact) LU factorization $H_r = L_rU_r$. We summarize the required setup steps in Algorithm 4.2.

For the preconditioner (4.12), we compute the low-rank approximation of $\widetilde{E}_{\mathrm{R},\alpha}$ as $\widetilde{E}_{\mathrm{R},\alpha} \approx V_{\mathrm{R},\alpha,r}H_{\mathrm{R},\alpha,r}V^{\mathrm{T}}_{\mathrm{R},\alpha,r}$ with Algorithm 2.4. We then precompute the small matrix $M_r \coloneqq \mathrm{I}_r - H_{\mathrm{R},\alpha,r} \in \mathbb{R}^{r\times r}$ and its exact LU factorization $M_r = L_rU_r$. We summarize the construction steps in Algorithm 4.3. We now discuss the computation costs required for the update construction. We start with the computational costs for computing the low-rank approximation of $\widetilde{E}_{\mathrm{R},\alpha}$ with the Arnoldi iteration. In the Arnoldi iteration, we require a random start vector of size n_p. Filling this vector requires $N_{\mathrm{fillv0}} = n_p$ floating point operations. The Arnoldi iteration involves r products of the matrix $\widetilde{E}_{\mathrm{R},\alpha}$ with a vector. As discussed above, each multiplication of $\widetilde{E}_{\mathrm{R},\alpha}$ with a vector costs

$$N_{\mathrm{mult}\widetilde{\mathrm{E}}_\alpha} = N_{\mathrm{mult}\widehat{\mathrm{A}}^{-1}} + N_{\mathrm{mult}\widehat{\mathrm{S}}^{-1}} + (4c_B - c_u + 1)n_p.$$

After each multiplication with $\widetilde{E}_{\mathrm{R},\alpha}$, we orthogonalize the resulting vector against the previously computed vectors with the modified Gram-Schmidt method. Furthermore, we

Algorithm 4.3: Construction of the update with the Arnoldi iteration.

Input: System matrix $\mathcal{A} = \left(\begin{smallmatrix} A & B^{\mathrm{T}} \\ B & 0 \end{smallmatrix}\right)$, preconditioner $\widehat{A}^{-1}$, initial Schur complement preconditioner $\widehat{S}^{-1}$, rank r, relaxation parameter α.

Output: Matrices $V_{\mathrm{R},\alpha,r}$, $H_{\mathrm{R},\alpha,r}$, L_r, U_r defining the updated preconditioner $\widehat{S}^{-1}_{\mathrm{Arnoldi,R},\alpha,r}$ (4.12).

1 Define the approximate Schur complement $\widetilde{S} \coloneqq B\widehat{A}^{-1}B^{\mathrm{T}}$.

2 Compute a low-rank approximation of $\widetilde{E}_{\mathrm{R},\alpha} \coloneqq \mathrm{I} - \alpha\widetilde{S}\widehat{S}^{-1} \approx V_{\mathrm{R},\alpha,r}H_{\mathrm{R},\alpha,r}V_{\mathrm{R},\alpha,r}^{\mathrm{T}}$ with Algorithm 2.4.

3 Compute $M_r = \mathrm{I}_r - H_{\mathrm{R},\alpha,r}$.

4 Compute an LU factorization of $M_r = L_rU_r$.

5 Define the preconditioner $\widehat{S}^{-1}_{\mathrm{Arnoldi,R},\alpha,r} = \alpha\widehat{S}^{-1}\Big(\mathrm{I}_{n_p} + V_{\mathrm{R},\alpha,r}\Big((L_rU_r)^{-1} - \mathrm{I}_r\Big)V_{\mathrm{R},\alpha,r}^{\mathrm{T}}\Big)$.

re-orthogonalize if necessary. This involves $N_{\mathrm{orth}} = \mathcal{O}(4n_pr(r+1))$ floating point operations assuming that we re-orthogonalize in each step.

Precomputing the small matrix $M_r = \mathrm{I}_r - H_{\mathrm{R},\alpha,r}$ costs $r^2 + r = \mathcal{O}(r^2)$ operations. The LU factorization of M_r requires additional $\mathcal{O}(r^3)$ floating point operations.

The setup costs of the preconditioners in (4.12) hence total to

$$\begin{aligned} N_{\mathrm{setup,Arnoldi}} &= N_{\mathrm{fillv0}} + rN_{\mathrm{mult}\widetilde{\mathrm{E}}_\alpha} + N_{\mathrm{orth}} + N_{M_r} + N_{\mathrm{LU}} \\ &= n_p + r\big(N_{\mathrm{mult}\widehat{\mathrm{A}}^{-1}} + N_{\mathrm{mult}\widehat{\mathrm{S}}^{-1}} + \mathcal{O}((4c_B - c_u + 1)n_p)\big) \\ &\qquad + \mathcal{O}(4n_pr(r+1)) + \mathcal{O}(r^2) + \mathcal{O}(r^3) \\ &= r\big(N_{\mathrm{mult}\widehat{\mathrm{A}}^{-1}} + N_{\mathrm{mult}\widehat{\mathrm{S}}^{-1}}\big) + \mathcal{O}(c_{\mathrm{setup,Arnoldi}}n_p) \end{aligned} \tag{4.34}$$

operations with $c_{\mathrm{setup,Arnoldi}} = r(4c_B - c_u) + 4r^2 + 5r + 1$. The setup costs of the preconditioners $\widehat{S}^{-1}_{\mathrm{random,R},\alpha,r}$ (4.11) and $\widehat{S}^{-1}_{\mathrm{Arnoldi,R},\alpha,r}$ (4.12) are dominated by the multiplications with the approximate error matrix $\widetilde{E}_{\mathrm{R},\alpha}$. The randomized method requires $2(q+1)\ell_r = 2(q+1)(r+k_r)$ applications of the matrix $\widetilde{E}_{\mathrm{R},\alpha}$ or its transpose whereas the Arnoldi iteration only requires r applications of $\widetilde{E}_{\mathrm{R},\alpha}$. Thus, the setup with the Arnoldi iteration is cheaper in terms of floating point operations. However, the r applications of $\widetilde{E}_{\mathrm{R},\alpha}$ in the Arnoldi iteration have to be computed in serial whereas we can apply $\widetilde{E}_{\mathrm{R},\alpha}$ to blocks of $\ell_r = r + k_r$ vectors in the randomized low-rank approximation. We thus can parallelize the applications of $\widetilde{E}_{\mathrm{R},\alpha}$ in the randomized method.

4.6.2 Application

We now discuss the application of the updated preconditioners $\widehat{S}^{-1}_{\mathrm{random,R},\alpha,r}$ (4.11) and $\widehat{S}^{-1}_{\mathrm{ArnoldiP,R},\alpha,r}$ (4.13) as well as $\widehat{S}^{-1}_{\mathrm{Arnoldi,R},\alpha,r}$ (4.12). The application of the preconditioner $\widehat{S}^{-1}_{\mathrm{random,R},\alpha,r}$ is summarized in Algorithm 4.4. The preconditioner $\widehat{S}^{-1}_{\mathrm{ArnoldiP,R},\alpha,r}$ is applied as described in Algorithm 4.4 by setting $Q_{\mathrm{R},\alpha,r} = V_{\mathrm{R},\alpha,r}$ and $N_{\mathrm{R},\alpha,r} = W_{\mathrm{R},\alpha,r}$. Algorithm 4.5 outlines the application of $\widehat{S}^{-1}_{\mathrm{Arnoldi,R},\alpha,r}$.

In addition to the application of the initial preconditioner, the updated preconditioners require matrix-vector products with an $r \times n_p$ matrix and an $n_p \times r$ matrix. Furthermore, we

Algorithm 4.4: Application of the updated preconditioner $\widehat{S}^{-1}_{\text{random,R},\alpha,r}$ or $\widehat{S}^{-1}_{\text{ArnoldiP,R},\alpha,r}$.

Input: Vector $v \in \mathbb{R}^{n_p}$, relaxation parameter α, matrices $Q_{\text{R},\alpha,r}$, $N_{\text{R},\alpha,r}$, L_r, U_r defining the updated preconditioner $\widehat{S}^{-1}_{\text{random,R},\alpha,r}$ (4.11) or $\widehat{S}^{-1}_{\text{ArnoldiP,R},\alpha,r}$ (4.13).

Output: Vector $y = \widehat{S}^{-1}_{\text{random,R},\alpha,r} v \in \mathbb{R}^{n_p}$.

1 Compute $z_r = N^{\text{T}}_{\text{R},\alpha,r} v$.
2 Solve $L_r U_r w_r = z_r$.
3 Compute $x = Q_{\text{R},\alpha,r} w_r$.
4 Compute $x \mathrel{+}= v$.
5 Apply the initial preconditioner $y = \widehat{S}^{-1} x$.
6 Scale with the relaxation parameter $y \mathrel{*}= \alpha$.

Algorithm 4.5: Application of the updated preconditioner $\widehat{S}^{-1}_{\text{Arnoldi,R},\alpha,r}$.

Input: Vector $v \in \mathbb{R}^{n_p}$, relaxation parameter α, matrices $V_{\text{R},\alpha,r}$, L_r, U_r defining the updated preconditioner $\widehat{S}^{-1}_{\text{Arnoldi,R},\alpha,r}$ (4.12).

Output: Vector $y = \widehat{S}^{-1}_{\text{Arnoldi,R},\alpha,r} v \in \mathbb{R}^{n_p}$.

1 Compute $z_r = V^{\text{T}}_{\text{R},\alpha,r} v$.
2 Solve $L_r U_r w_r = z_r$.
3 Compute $w_r \mathrel{-}= z_r$.
4 Compute $x = V_{\text{R},\alpha,r} w_r$.
5 Compute $x \mathrel{+}= v$.
6 Apply the initial preconditioner $y = \widehat{S}^{-1} x$.
7 Scale with the relaxation parameter $y \mathrel{*}= \alpha$.

solve two $r \times r$ triangular systems and add and scale an n_p-dimensional vector. The update based on the Arnoldi iteration (without projection) furthermore requires the addition of small vectors of size r.

The application costs hence sum up to

$$\begin{aligned} N_{\text{apply,random}} &= N_{\text{multQ}} + N_{\text{multN}} + N_{\text{solveLU}} + N_{\text{vecadd}} + N_{\text{vecscale}} + N_{\text{mult}\widehat{\text{S}}^{-1}} \\ &= n_p(2r-1) + r(2n_p-1) + 2(2r^2-r) + n_p + n_p + N_{\text{mult}\widehat{\text{S}}^{-1}} \\ &= \mathcal{O}(c_r n_p) + N_{\text{mult}\widehat{\text{S}}^{-1}} \end{aligned} \tag{4.35}$$

operations with

$$c_r = 4r + 1. \tag{4.36}$$

for the preconditioner $\widehat{S}^{-1}_{\text{random,R},\alpha,r}$.

The application costs of $\widehat{S}^{-1}_{\text{Arnoldi,R},\alpha,r}$ total to

$$\begin{aligned} N_{\text{apply,Arnoldi}} &= N_{\text{multV}} + N_{\text{multV}^{\text{T}}} + N_{\text{solveLU}} + N_{\text{vecadd}} + N_{\text{vecscale}} + N_{\text{mult}\widehat{\text{S}}^{-1}} \\ &= n_p(2r-1) + r(2n_p-1) + 2(2r^2-r) + n_p + r + n_p + N_{\text{mult}\widehat{\text{S}}^{-1}} \\ &= \mathcal{O}(c_r n_p) + N_{\text{mult}\widehat{\text{S}}^{-1}} \end{aligned} \tag{4.37}$$

operations with $c_r = 4r + 1$. The parameter c_r can be split in contributions by the low-rank update and by the relaxation parameter. The term $4r$ refers to the application of the low-rank update and the term 1 refers to the scaling with the relaxation parameter α. The difference in application costs of the preconditioners $\widehat{S}^{-1}_{\text{random,R},\alpha,r}$, $\widehat{S}^{-1}_{\text{ArnoldiP,R},\alpha,r}$, and $\widehat{S}^{-1}_{\text{Arnoldi,R},\alpha,r}$ compared to the initial preconditioner $\widehat{S}^{-1}$ is linear in the number of pressure degrees of freedom n_p since we typically have $r \ll n_p$.

The percentage of additional application costs of a low-rank update for a Schur complement preconditioner is of the order

$$r_{\text{apply}} = \frac{N_{\text{apply}} - N_{\text{mult}\widehat{\text{S}}^{-1}}}{N_{\text{mult}\widehat{\text{S}}^{-1}}} = \frac{\mathcal{O}(c_r n_p)}{N_{\text{mult}\widehat{\text{S}}^{-1}}}.$$

To quantify the ratio r_{apply}, we now discuss the application costs for specific initial preconditioners $\widehat{S}^{-1}$ in the following two propositions. We choose the preconditioners that we also use for the majority of the numerical experiments in Chapter 5. For the initial Schur complement preconditioner $\widehat{S}^{-1}$, we consider the LSC preconditioner and the SIMPLE preconditioner given in the Equations (3.43) and (3.44). Both preconditioners require the approximate solution to Poisson-type problems. For the analysis, we assume that we use an incomplete Cholesky decomposition without fill-in to approximate the Poisson-type problems.

Proposition 4.2. *We consider the LSC preconditioner* (3.43) *where the matrix* $\widehat{M}_{\text{BMB}^{\text{T}}}$ *is approximated by an incomplete Cholesky decomposition without fill-in. One application of the LSC preconditioner to a vector of size* n_p *requires*

$$N_{\text{LSC}} = \mathcal{O}(c_{\text{LSC}} n_p)$$

floating point operations with $c_{\text{LSC}} = 4c_{\text{M}} + 4c_B + 2c_u c_A + 3$ *where the parameters* c_M*,* c_A*,* c_B*,* c_u *are defined as in Definition 4.1.*

Proof. The LSC preconditioner requires two approximate solutions to Poisson-type problems with the system matrix $\widehat{M}_{\text{BMB}^{\text{T}}}$ in addition to the computation of matrix-vector products with the matrix blocks A, B, and B^{T}. Furthermore, we multiply twice with a (precomputed) inverse diagonal matrix. Its application hence adds up to

$$\begin{aligned} N_{\text{LSC}} &= 2N_{\text{solveIC}} + N_{\text{multB}} + N_{\text{mult}}A + N_{\text{multB}^{\text{T}}} + 2N_{\text{multD}^{-1}} \\ &= 4(c_M + 1)n_p + n_p(2c_B - 1) + n_u(2c_A - 1) + n_u(2c_{B^{\text{T}}} - 1) + 2n_u \\ &= 4c_M n_p + 4c_B n_p + 2c_A c_u n_p + 3n_p \\ &= \mathcal{O}(c_{\text{LSC}} n_p) \end{aligned}$$

operations with $c_{\text{LSC}} = 4c_M + 4c_B + 2c_u c_A + 3$ where we used Equation (4.31). □

Proposition 4.3. *We consider the SIMPLE preconditioner* (3.44) *where the matrix* $\widehat{M}_{\text{BDB}^{\text{T}}}$ *is approximated by an incomplete Cholesky decomposition without fill-in. One application of the SIMPLE preconditioner to a vector of size* n_p *requires*

$$N_{\text{SIMPLE}} = \mathcal{O}(c_{\text{SIMPLE}} n_p)$$

floating point operations with $c_{\text{SIMPLE}} = 2(c_M + 1)$ *where the parameter* c_M *is defined as in Definition 4.1.*

Proof. The SIMPLE preconditioner requires only the approximate solution to a Poisson-type problem with system matrix $\widehat{M}_{\mathrm{BDB^T}}$ for which we use an incomplete Cholesky decomposition without fill-in. Solving with the incomplete Cholesky factors costs

$$N_{\mathrm{SIMPLE}} = N_{\mathrm{solveIC}} = 2(c_M + 1)n_p = c_{\mathrm{SIMPLE}} n_p$$

operations with $c_{\mathrm{SIMPLE}} = 2(c_M + 1)$. □

We now analyze the ratio r_{apply} of additional costs introduced by the low-rank updates to the costs without any update regarding the application of the preconditioners for the LSC and the SIMPLE preconditioner. Table 4.1 shows the percentage of the additional application costs per update rank. Furthermore, we show the maximum update rank such that the application costs of the updated preconditioners are less than twice as high as those of the initial preconditioners. The values are computed with the parameters given in Table 4.2 that displays the values for the parameters c_M, c_A, c_B, and c_u for the considered test systems. More details about the test systems are given in Section 5.1. We observe that the additional application costs can be significant, especially for the SIMPLE preconditioner.

However, we will observe that the additional costs are small compared to the costs of one GMRes iteration. The costs of the GMRes iteration are dominated by the application of the system matrix and the block preconditioner. Hence, the proportionate costs of the update application are reduced as we have significantly more velocity degrees of freedom than pressure degrees of freedom (by a factor of $c_u \approx 20$, see Table 4.2). We now analyze the additional costs introduced by the update compared to the averaged costs of one GMRes

TABLE 4.1: Percentage of additional application costs introduced by increasing the update rank by one per application of the preconditioner (4.11) or (4.12) ("costs per rank") and maximum update rank such that the application costs of the updated preconditioners are less than twice as those of the initial preconditioners. See Table 5.1 for the system dimensions.

	SIMPLE		LSC	
system	costs per rank	rank for doubling	costs per rank	rank for doubling
`shell_3`	1.9 %	53	0.04 %	2276
`shell_4`	1.7 %	58	0.04 %	2527
`cube_4`	1.9 %	51	0.04 %	2242
`cube_5`	1.7 %	56	0.04 %	2505

TABLE 4.2: Values for the parameters c_A, c_B, c_M, c_u as defined in Definition 4.1 for the shell and the cube, each with two different refinement levels, see Table 5.1 for the system dimensions.

system	c_A	c_B	c_M	c_u
`shell_3`	163.96	317.81	106.68	22.58
`shell_4`	177.60	345.15	115.74	23.27
`cube_4`	167.11	306.10	102.99	21.94
`cube_5`	179.11	338.37	113.32	22.93

iteration. We start by deriving the averaged complexity of one iteration of the restarted GMRes Algorithm 2.1 in Theorem 4.4.

Theorem 4.4. *We consider the iterative solution of problem* (4.30) *with the restarted GMRes Algorithm 2.1 with restart length $k \ll n_p$. The parameters c_A, c_B, c_u are defined as in Definition 4.1. Then after ℓ GMRes iterations, the arithmetic mean of the complexity of one iteration is given by*

$$N_{\text{Iter,avg}}(k,\ell) = \tfrac{1}{\ell}\big(\ell + \lfloor \tfrac{\ell}{k} \rfloor + 1\big)\big(N_{\text{mult}\widehat{\text{S}}^{-1}} + N_{\text{mult}\widehat{\text{A}}^{-1}}\big) + \mathcal{O}(c_{\text{gmres}}(k,\ell)n_p)$$

with the constant $c_{\text{gmres}}(k,\ell) = \frac{1}{\ell}\left[c_{\text{Init}}(k,\ell) + c_{\text{Iter}}(k,\ell) + c_{\text{upd}}(k,\ell)\right]$ *where*

$$\begin{aligned}
c_{\text{Init}}(k,\ell) &= \big(\lfloor \tfrac{\ell}{k} \rfloor + 1\big)(2c_u c_A + 4c_B + 3c_u + 3),\\
c_{\text{Iter}}(k,\ell) &= \ell(2c_u c_A + 8c_B - c_u - 1) + 2\left[\lfloor \tfrac{\ell}{k} \rfloor k(k+1) + c_\ell(c_\ell + 1) + \lfloor \tfrac{\ell}{k} \rfloor + 1\right](c_u + 1),\\
c_{\text{upd}}(k,\ell) &= 4\big(\lfloor \tfrac{\ell}{k} \rfloor + 1\big)c_B + 2\big(\lfloor \tfrac{\ell}{k} \rfloor k + c_\ell\big)(c_u + 1).
\end{aligned}$$

The parameter c_ℓ is given by $c_\ell = k$, if $\ell \bmod k = 0$ and $\ell > 0$, and by $c_\ell = \ell \bmod k$, else. The parameters $N_{\text{mult}\widehat{\text{S}}^{-1}}$ and $N_{\text{mult}\widehat{\text{A}}^{-1}}$ denote the application costs of the preconditioners $\widehat{S}^{-1}$ and $\widehat{A}^{-1}$, respectively.

Proof. We first determine the arithmetic mean of floating point operations needed for one iteration of the restarted GMRes method given in Algorithm 2.1. The initialization of the restarted GMRes method requires the computation of the initial residual per restart. We denote the related costs of all initializations with $N_{\text{Init}}(k,\ell)$. It follows the actual iteration which requires the application of the preconditioner and the system matrix followed by a Gram-Schmidt process. We call the corresponding cumulative costs $N_{\text{Iter}}(k,\ell)$. Before each restart and after the stopping criterion is satisfied, we solve a (small) minimization problem with $N_{\text{min}}(k,\ell)$ operations and update the approximate solution vector x_k with in total $N_{\text{upd}}(k,\ell)$ operations. It follows that the arithmetic mean of the costs per GMRes iteration is given by

$$N_{\text{Iter,avg}}(k,\ell) = \tfrac{1}{\ell}\left[N_{\text{Init}}(k,\ell) + N_{\text{Iter}}(k,\ell) + N_{\text{min}}(k,\ell) + N_{\text{upd}}(k,\ell)\right]$$

where ℓ is the number of GMRes iterations and k is the restart length.

We now analyze the averaged costs of a GMRes iteration $N_{\text{Iter,avg}}$ step by step. To quantify the costs, we relate all parts to the number of pressure degrees of freedom n_p.

We start with the initial computations that are required before starting the actual iteration process. In line 1 of Algorithm 2.1, we evaluate the initial residual which requires a matrix-vector product with the system matrix $\mathcal{A}$ and subtraction of a vector of size n_u+n_p. We additionally normalize the initial residual vector which requires the computation of the 2-norm of an $(n_u + n_p)$-dimensional vector and a scaling of this vector. We evaluate the initialization at the beginning of each restart. In the ℓth iteration with a restart length of k, we evaluated line 1 hence $\lfloor \frac{\ell}{k} \rfloor + 1$ times where $\lfloor \cdot \rfloor$ denotes the floor function. The initialization costs total to

$$\begin{aligned}
N_{\text{Init}}(k,\ell) &= \big(\lfloor \tfrac{\ell}{k} \rfloor + 1\big)\left[N_{\text{mult}\mathcal{A}} + N_{\text{update}} + N_{\text{norm}} + N_{\text{scale}}\right]\\
&= \big(\lfloor \tfrac{\ell}{k} \rfloor + 1\big)\left[N_{\text{mult}\mathcal{A}} + (n_u + n_p) + 2(n_u + n_p) - 1 + (n_u + n_p)\right]\\
&= \big(\lfloor \tfrac{\ell}{k} \rfloor + 1\big)\left[N_{\text{mult}\mathcal{A}} + 4(c_u + 1)n_p - 1\right]. \qquad (4.38)
\end{aligned}$$

The application of the system matrix $\mathcal{A} = \left(\begin{smallmatrix} A & B^{\mathrm{T}} \\ B & 0 \end{smallmatrix}\right)$ requires matrix-vector products with the matrix blocks A, B, B^{T} and the addition of a vector of size $n_u = c_u n_p$. We obtain

$$\begin{aligned} N_{\mathrm{mult}\mathcal{A}} &= N_{\mathrm{multA}} + N_{\mathrm{multB}} + N_{\mathrm{mult}}B^{\mathrm{T}} + N_{\mathrm{add}} \\ &= c_u n_p(2c_A - 1) + n_p(2c_B - 1) + c_u n_p(2c_{B^{\mathrm{T}}} - 1) + c_u n_p \\ &= (2c_u c_A + 4c_B - c_u - 1)n_p \end{aligned} \tag{4.39}$$

where we used that $c_{B^{\mathrm{T}}} = \frac{c_B}{c_u}$ as shown in Equation (4.31).

We insert Equation (4.39) into Equation (4.38) and obtain

$$\begin{aligned} N_{\mathrm{Init}}(k,\ell) &= \left(\lfloor \tfrac{\ell}{k} \rfloor + 1\right)\left[N_{\mathrm{mult}\mathcal{A}} + 4(c_u+1)n_p - 1\right] \\ &= \left(\lfloor \tfrac{\ell}{k} \rfloor + 1\right)\left[(2c_u c_A + 4c_B - c_u - 1)n_p + 4(c_u+1)n_p - 1\right] \\ &= \mathcal{O}(c_{\mathrm{Init}}(k,\ell)n_p) \end{aligned} \tag{4.40}$$

with $c_{\mathrm{Init}}(k,\ell) = \left(\lfloor \tfrac{\ell}{k} \rfloor + 1\right)(2c_u c_A + 4c_B + 3c_u + 3)n_p$.

We now discuss the costs of the actual iteration process given in the lines 2 to 10 in Algorithm 2.1. Per iteration, we apply once the block triangular preconditioner P_{U} given in Equation (3.35) and the system matrix $\mathcal{A}$. Furthermore, we perform Gram-Schmidt orthogonalizations of vectors of size $n_u + n_p = (c_u+1)n_p$ with related costs $N_{\mathrm{GS}}(k,\ell)$. The ℓ outer iterations require

$$N_{\mathrm{Iter}}(k,\ell) = \ell N_{\mathrm{mult}P_{\mathrm{U}}} + \ell N_{\mathrm{mult}\mathcal{A}} + N_{\mathrm{GS}}(k,\ell)$$

operations.

We now discuss the complexity of the application of the preconditioner P_{U}. Each application of the block triangular preconditioner $P_{\mathrm{U}} = \left(\begin{smallmatrix} \widehat{A}^{-1} & \widehat{A}^{-1}B^{\mathrm{T}}\widehat{S}^{-1} \\ 0 & -\widehat{S}^{-1} \end{smallmatrix}\right)$ requires the application of the preconditioners $\widehat{A}^{-1}$ and $\widehat{S}^{-1}$, a matrix-vector product with the matrix block B^{T} and the addition of a vector of the size $n_u = c_u n_p$ since

$$\begin{pmatrix} \widehat{A}^{-1} & \widehat{A}^{-1}B^{\mathrm{T}}\widehat{S}^{-1} \\ 0 & -\widehat{S}^{-1} \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \widehat{A}^{-1}(x + B^{\mathrm{T}}v) \\ -v \end{pmatrix}$$

with $v \coloneqq \widehat{S}^{-1}y$. The application of the block triangular preconditioner hence requires

$$\begin{aligned} N_{P_{\mathrm{U}}} &= N_{\mathrm{mult}\widehat{\mathrm{A}}^{-1}} + N_{\mathrm{mult}\widehat{\mathrm{S}}^{-1}} + c_u n_p(2c_{B^{\mathrm{T}}} - 1) + c_u n_p \\ &= N_{\mathrm{mult}\widehat{\mathrm{A}}^{-1}} + N_{\mathrm{mult}\widehat{\mathrm{S}}^{-1}} + 4c_B n_p \end{aligned} \tag{4.41}$$

operations.

Now, we analyze the complexity of the Gram-Schmidt orthogonalization. Per inner iteration, the Gram-Schmidt process requires j dot products, j vector additions and scaling of j vectors where j refers to the inner iteration of Algorithm 2.1 using the same notation as in Algorithm 2.1. We furthermore require one norm computation and scaling of a vector. Each evaluation is done for vectors of size $n_u + n_p = (c_u+1)n_p$. The cost at the jth inner iteration amount to

$$\begin{aligned} N_{\mathrm{GS,Iter}}(j) &= j(N_{\mathrm{w^T v}} + N_{\mathrm{scale}} + N_{\mathrm{updw}}) \\ &= j(2(n_u + n_p) - 1 + 2(n_p + n_u)) \\ &= j(4(c_u+1)n_p - 1). \end{aligned}$$

Since the size of the Arnoldi basis varies for different GMRes iteration steps we need to sum up over all iterations to obtain the total costs related to the Gram-Schmidt process. We obtain for k iterations (i.e. per restart)

$$\begin{aligned} N_{\mathrm{GS,restart}}(k) &= \sum_{j=1}^{k} N_{\mathrm{GS,Iter}}(j) = \sum_{j=1}^{k} j(4(c_u+1)n_p - 1) \\ &= \tfrac{1}{2}k(k+1)(4(c_u+1)n_p - 1). \end{aligned}$$

In the restarted GMRes iteration, we have $\lfloor \frac{\ell}{k} \rfloor$ restarts after each k iterations additionally to the last c_ℓ iterations. The parameter c_ℓ corresponds to the size of the Arnoldi basis after the last restart and is given by $c_\ell = k$, if $\ell \bmod k = 0$ and $\ell > 0$, and by $c_\ell = \ell \bmod k$, else. Per iteration, we furthermore require one norm computation and scaling of a vector of size $n_u + n_p$. The total costs related to the Gram-Schmidt orthogonalization are hence

$$\begin{aligned} N_{\mathrm{GS}}(k,\ell) &= \lfloor \tfrac{\ell}{k} \rfloor N_{\mathrm{GS,restart}}(k) + N_{\mathrm{GS,restart}}(c_\ell) + \big(\lfloor \tfrac{\ell}{k} \rfloor + 1\big)(N_{\mathrm{norm}} + N_{\mathrm{scale}}) \\ &= \mathcal{O}(c_{\mathrm{GS}}(k,\ell)n_p) \end{aligned} \tag{4.42}$$

with $c_{\mathrm{GS}}(k,\ell) = 2\left[\lfloor \frac{\ell}{k} \rfloor k(k+1) + c_\ell(c_\ell+1) + \lfloor \frac{\ell}{k} \rfloor + 1\right](c_u+1)$. Combing Equations (4.39), (4.41), and (4.42) yields the costs related to the actual iteration

$$\begin{aligned} N_{\mathrm{Iter}}(k,\ell) &= \ell N_{\mathrm{mult}\mathrm{P_U}} + \ell N_{\mathrm{mult}\mathcal{A}} + N_{\mathrm{GS}} \\ &= \ell\big(N_{\mathrm{mult}\widehat{\mathrm{A}}^{-1}} + N_{\mathrm{mult}\widehat{\mathrm{S}}^{-1}} + 4c_B n_p\big) + \ell(2c_u c_A + 4c_B - c_u - 1)n_p \\ &\qquad + \mathcal{O}(c_{\mathrm{GS}}(k,\ell)n_p) \\ &= \ell N_{\mathrm{mult}\widehat{\mathrm{A}}^{-1}} + \ell N_{\mathrm{mult}\widehat{\mathrm{S}}^{-1}} + \mathcal{O}(c_{\mathrm{Iter}}(k,\ell)n_p) \end{aligned} \tag{4.43}$$

with $c_{\mathrm{Iter}}(k,\ell) = \ell(2c_u c_A + 8c_B - c_u - 1) + c_{\mathrm{GS}}(k,\ell)$. Before each restart and additionally, if the convergence criterion is met, we solve a minimization problem at most of size $k \times k$, respectively, and update the solution vector x_k. To solve the minimization problems, we apply at most $k-1$ Givens rotations and then solve the obtained triangular system. Solving one of the minimization problems hence costs

$$\begin{aligned} N_{\mathrm{min,Iter}}(k) &= N_{\mathrm{Givens}}(k) + N_{\mathrm{solve}}(k) \\ &\leq 6(k-1) + \tfrac{1}{2}k(k+1) \end{aligned}$$

which does not depend on n_p and can hence be omitted in the derivation of the leading term in the complexity analysis for $k \ll n_p$.

Before each restart and after the last GMRes iteration, the solution vector x_k is updated. Updating the solution vector requires the application of the block triangular preconditioner, a matrix-vector product with the $(n_u + n_p) \times k$ or $(n_u + n_p) \times c_\ell$ matrix V and the update of an $(n_u + n_p) = (c_u + 1)n_p$ vector which costs

$$\begin{aligned} N_{\mathrm{upd}}(k,\ell) &= \big(\lfloor \tfrac{\ell}{k} \rfloor + 1\big)N_{\mathrm{apply}P_U} + N_{\mathrm{mult}\mathrm{V}}(k,\ell) + \big(\lfloor \tfrac{\ell}{k} \rfloor + 1\big)N_{\mathrm{add}} \\ &= \big(\lfloor \tfrac{\ell}{k} \rfloor + 1\big)\big(N_{\mathrm{mult}\widehat{\mathrm{A}}^{-1}} + N_{\mathrm{mult}\widehat{\mathrm{S}}^{-1}} + 4c_B n_p\big) \\ &\quad + \big(\lfloor \tfrac{\ell}{k} \rfloor (2k-1) + (2c_\ell - 1)\big)(c_u+1)n_p + \big(\lfloor \tfrac{\ell}{k} \rfloor + 1\big)(c_u+1)n_p \\ &= \big(\lfloor \tfrac{\ell}{k} \rfloor + 1\big)\big(N_{\mathrm{mult}\widehat{\mathrm{A}}^{-1}} + N_{\mathrm{mult}\widehat{\mathrm{S}}^{-1}}\big) + \mathcal{O}(c_{\mathrm{upd}}(k,\ell)n_p) \end{aligned} \tag{4.44}$$

operations with $c_{\text{upd}}(k,\ell) = 4\big(\lfloor\frac{\ell}{k}\rfloor + 1\big)c_B + 2\big(\lfloor\frac{\ell}{k}\rfloor k + c_\ell\big)(c_u + 1)$.

Combining Equations (4.40), (4.43), (4.44), we obtain for the averaged costs for one GMRes iteration

$$\begin{aligned} N_{\text{Iter,avg}}(k,\ell) &= \tfrac{1}{\ell}\left[N_{\text{Init}}(k,\ell) + N_{\text{Iter}}(k,\ell) + N_{\min}(k,\ell) + N_{\text{upd}}(k,\ell)\right] \\ &= \tfrac{1}{\ell}\big(\ell + \lfloor\tfrac{\ell}{k}\rfloor + 1\big)\big(N_{\text{mult}\widehat{\text{A}}^{-1}} + N_{\text{mult}\widehat{\text{S}}^{-1}}\big) + \mathcal{O}(c_{\text{gmres}}(k,\ell)n_p) \end{aligned}$$

with $c_{\text{gmres}}(k,\ell) = \frac{1}{\ell}(c_{\text{Init}}(k,\ell) + c_{\text{Iter}}(k,\ell) + c_{\text{upd}}(k,\ell))$. □

We determine the additional costs per GMRes iteration that are introduced by the application of the low-rank updates in the following corollary.

Corollary 4.5. *We consider the same setting and parameters as in Theorem 4.4. The additional application costs introduced by the low-rank updates* (4.11) *and* (4.12) *are given by*

$$N_{\text{gmres,upd}}(k,\ell) = \mathcal{O}(c_{\text{gmres,upd}}(k,\ell)n_p) \tag{4.45}$$

with $c_{\text{gmres,upd}}(k,\ell) = \frac{1}{\ell}\big(\lfloor\frac{\ell}{k}\rfloor + \ell + 1\big)c_r$ *with* $c_r = 4r + 1$ *(see Equation* (4.36)*).*

Proof. The iterative solution with Algorithm 2.1 requires $\ell + \lfloor\frac{\ell}{k}\rfloor + 1$ applications of the Schur complement preconditioner $\widehat{S}^{-1}$, i.e. also of the low-rank update if we replace the preconditioner $\widehat{S}^{-1}$ by one of the preconditioners (4.11) or (4.12). With Equations (4.35) and (4.37), we find that the additional application costs introduced by the low-rank updates comprise $\mathcal{O}(c_r n_p)$ operations with $c_r = (4r + 1)$. The arithmetic mean of additional costs introduced by the updates per GMRes iteration is hence given by $\mathcal{O}(c_{\text{gmres,upd}}(k,\ell)n_p)$ operations with $c_{\text{gmres,upd}}(k,\ell) = \frac{1}{\ell}\big(\lfloor\frac{\ell}{k}\rfloor + \ell + 1\big)c_r$. □

The following corollary relates the additional application costs introduced by a low-rank update to the averaged costs of a GMRes iteration.

Corollary 4.6. *We consider the same setting and parameters as in Theorem 4.4. The ratio of additional application costs introduced by the low-rank updates* (4.11) *and* (4.12) *to the averaged costs of a GMRes iteration is given by*

$$r_{\text{upd}}(k,\ell) = \frac{\big(\lfloor\frac{\ell}{k}\rfloor + \ell + 1\big)c_r}{\ell c_{\text{gmres}}(k,\ell)} \tag{4.46}$$

with $c_r = 4r + 1$ *(see Equation* (4.36)*) and* $c_{\text{gmres}}(k,\ell)$ *as in Theorem 4.4.*

Proof. We obtain the result (4.46) by forming the ratio of the update complexity given in Corollary 4.5 to the complexity of one GMRes iteration given in Theorem 4.4. □

We furthermore consider the ratio

$$r_{\text{constr}} = \frac{N_{\text{setup}}}{N_{\text{Iter,avg}}(k,\ell)} \tag{4.47}$$

of the complexities for the update construction N_{setup} (see Equations (4.33), (4.34)) and the averaged GMRes iteration $N_{\text{Iter,avg}}(k,\ell)$ from Theorem 4.4.

With Equation (4.46), we can estimate how many iterations we need to save depending on the update rank to reduce solver times. Now, we furthermore estimate how many iterations we need to save to reduce total computational times. We assume that we save $\Delta\ell$ iterations by applying a low-rank update to the Schur complement preconditioner. We hence save $N_{\text{saved}}(k,\ell) = N_{\text{Iter,avg}}(k,\ell)\cdot\ell - N_{\text{Iter,avg}}(k,\ell-\Delta\ell)\cdot(\ell-\Delta\ell)$ operations. To simplify the estimate, we assume that all GMRes iterations cost the same and find the approximation $N_{\text{saved}}(k,\ell) \approx \Delta\ell N_{\text{Iter,avg}}(k,\ell)$ operations. The additional costs introduced by the low-rank update comprise the construction costs N_{setup} and the application costs $N_{\text{gmres,upd}}(k,\ell-\Delta\ell)\cdot(\ell-\Delta\ell)$. The total times are hence reduced if

$$\Delta\ell N_{\text{Iter,avg}}(k,\ell) \gtrapprox N_{\text{setup}} + N_{\text{gmres,upd}}(k,\ell-\Delta\ell)\cdot(\ell-\Delta\ell).$$

We insert Equation (4.45), use $\lfloor\frac{\ell-\Delta\ell}{k}\rfloor \geq \frac{\ell-\Delta\ell}{k} - 1$, and obtain

$$\Delta\ell N_{\text{Iter,avg}}(k,\ell) \gtrapprox N_{\text{setup}} + (\ell-\Delta\ell)\left(\tfrac{1}{k}-1\right)c_r.$$

Solving for $\Delta\ell$ yields

$$\Delta\ell \gtrapprox \frac{N_{\text{setup}} + \ell(1+\frac{1}{k})(4r+1)}{N_{\text{Iter,avg}}(k,\ell) + (1+\frac{1}{k})(4r+1)}. \tag{4.48}$$

To quantify the additional application costs per GMRes iteration as shown in Corollary 4.5 and its ratio to the averaged costs of a GMRes iteration with the initial preconditioner as shown in Corollary 4.6, we furthermore need the application costs for the initial preconditioner $\widehat{A}^{-1}$. For the initial preconditioner $\widehat{A}^{-1}$, we choose the block triangular preconditioner given in Equation (3.38) with incomplete LU factorizations without fill-in for the arising inverses which we also use for the numerical experiments in Chapter 5. In the following proposition, we derive the application costs of the block triangular preconditioner (3.38) for three-dimensional spatial domains. We assume that the number of nonzero entries is the same for all nine blocks of A. This is only approximately the case due to differing boundary conditions for the different velocity directions.

Proposition 4.7. *We consider the block triangular preconditioner* (3.38) *with* 3×3 *blocks. The inverses* A_{ii}^{-1}, $i=1,2,3$, *are approximated with an inexact LU decomposition without fill-in. The parameters* c_A, c_u *are defined as in Definition 4.1. Let the number of nonzero entries be the same for all nine blocks of* A, *then the application costs of the block triangular preconditioner are given by*

$$N_{\text{mult}\widehat{\text{A}}^{-1}} = c_{\text{mult}\widehat{\text{A}}^{-1}} n_p$$

operations with $c_{\text{mult}\widehat{\text{A}}^{-1}} = \left(\frac{4}{3}c_A + 2\right)c_u$.

Proof. For three-dimensional spatial domains, this preconditioner requires the application of three ILU preconditioners of size $\frac{n_u}{3}\times\frac{n_u}{3}$, three matrix-vector products with matrices of the same size, and the subtraction of three vectors of size $\frac{n_u}{3}$. We obtain the application complexity

$$\begin{aligned} N_{\text{mult}\widehat{\text{A}}^{-1}} &= \left(\sum_{i=1}^{3} 2(c_{A_{ii}}\tfrac{n_u}{3} + \tfrac{n_u}{3}) + \sum_{j=i+1}^{3} \tfrac{n_u}{3}(2c_{A,ij}-1)\right) + n_u \\ &= 2\left(\tfrac{c_A}{3}n_u + n_u\right) + n_u(2\tfrac{c_A}{3}-1) + n_u \\ &= c_{\text{mult}\widehat{\text{A}}^{-1}} n_p \end{aligned}$$

where $c_{\text{mult}\widehat{\text{A}}^{-1}} = \left(\frac{4}{3}c_A + 2\right)c_u$. □

With the parameter values given in Table 4.2 and Propositions 4.2, 4.3, and 4.7, we find the ratio $r_{\text{upd}}(k,\ell)$ of additional application costs and the ratio r_{constr} of additional construction costs of the updates for the SIMPLE and the LSC preconditioner per GMRes iteration. Furthermore, we calculate the estimates on the reduction of iteration counts needed to reduce total times. The results are displayed in Table 4.3. The ratios $r_{\text{upd}}(k,\ell)$ are so small that any reduction in iteration counts will lead to a reduction in solver times for reasonably small update ranks. To reduce total solver times, i.e. the sum of the time consumed by the GMRes iterations and the time required to construct the (updated) preconditioners, we need to save about half an iteration per update rank for updates constructed with Arnoldi and about 1.6 (SIMPLE) and 2.4 (LSC) iterations per update rank for updates constructed with the randomized range finder with one power iteration and without oversampling. In most of our numerical experiments, we will observe that the solver times are reduced but not the total times.

TABLE 4.3: Ratios r_{upd} (4.46) and r_{constr} (4.47) for increasing the update rank by one and necessary reduction in iteration counts per update rank $\Delta\ell$ (4.48) assuming a total number of $\ell = 60$ iterations and a restart length of $k = 40$ for updates constructed with the Arnoldi iteration (A) or with the randomized range finder (R). See Table 5.1 for the system dimensions.

	SIMPLE			LSC		
system	r_{upd}	r_{constr} (A, R)	$\Delta\ell$ (A, R)	r_{upd}	r_{constr} (A, R)	$\Delta\ell$ (A, R)
`shell_3`	0.02%	1.5%, 2.4%	0.4, 1.6	0.02%	4.4%, 3.5%	0.6, 2.4
`shell_4`	0.02%	1.7%, 2.6%	0.4, 1.6	0.01%	4.9%, 3.9%	0.6, 2.4
`cube_4`	0.02%	1.5%, 2.3%	0.4, 1.6	0.02%	4.3%, 3.5%	0.6, 2.3
`cube_5`	0.02%	1.6%, 2.6%	0.4, 1.6	0.01%	4.8%, 3.9%	0.6, 2.4

4.6.3 Repeated updates

In this subsection, we compare setup and application costs of N updates of rank r to those of one update of rank Nr. We restrict the cost analysis to right updates since the derivation for left updates is analogous.

We start with the application costs of the preconditioner $\widehat{S}_{\text{R},N}^{-1}$ (4.19b) which we will need to then derive its setup costs. We furthermore discuss how we can recycle precomputed vectors and thus reduce setup costs of the repeated update based on the randomized low-rank approximation technique from Algorithm 2.2 in both updates.

Application

The application of the repeated update scheme

$$\widehat{S}_{\text{R},N}^{-1} = \widehat{S}^{-1}\left(\prod_{i=1}^{N} \alpha_i\big(\mathrm{I}_{n_p} + X_{(\cdot),\text{R},i}\big)\right)$$

requires one application of the initial preconditioner, the application of N multiplicative updates of the form $\mathrm{I}_{n_p} + X_{(\cdot),\text{R},i}$, where a rank-$r$ factorization of $X_{(\cdot),\text{R},i}$ is given, and

scaling of N vectors with the relaxation parameters α_i, $i = 1, 2, \ldots, N$. In the previous section, we have derived that the application of one update with rank r introduces $\mathcal{O}(c_r n_p)$ with $c_r = 4r + 1$ additional costs as stated in Equations (4.35) and (4.37) if $r_i \ll n_p$. The application costs for N updates hence sum up to

$$N_{\text{apply},N} = N_{\text{mult}\widehat{\text{S}}^{-1}} + \sum_{i=1}^{N} \mathcal{O}(c_{r_i} n_p) \tag{4.49}$$

with $c_{r_i} = 4r_i + 1$ where r_i is the update rank in the ith update. Assuming that we choose the same update ranks in all updates, i.e. $r_1 = r_2 = \ldots = r_N = r$, we obtain

$$N_{\text{apply},N} = N_{\text{mult}\widehat{\text{S}}^{-1}} + \mathcal{O}(N c_r n_p). \tag{4.50}$$

The application costs are still linear in the pressure degrees of freedom n_p if we have $Nr \ll n_p$. This assumption is reasonable since we typically have $r \ll n_p$ and $N \ll n_p$.

We find with Equations (4.35) and (4.37) that the application of one update of rank Nr costs

$$N_{\text{apply}} = N_{\text{mult}\widehat{\text{S}}^{-1}} + \mathcal{O}(c_{Nr} n_p)$$

with $c_{Nr} = 4Nr + 1$. Thus, the application of N updates of rank r is slightly more expensive than the application of one update of rank Nr. However, the leading term of both application complexities is of the same order.

Construction

We continue with deriving the construction costs of the repeated update scheme (4.19b). We start with the setup costs of a repeated update that utilizes the Arnoldi iteration given in Algorithm 2.4. For repeated updates computed with the Arnoldi iteration, we obtain with Equation (4.34)

$$N_{\text{setup,Arnoldi},N} = \sum_{i=1}^{N} \left(r_i N_{\text{mult}\widetilde{\text{E}}_\alpha, i} + \mathcal{O}(c_i n_p) \right)$$

with $c_i = 4r_i(r_i + 1) + 1$. We find with Equation (4.49) that

$$N_{\text{setup,Arnoldi},N} = \sum_{i=1}^{N} \left(r_i N_{\text{mult}\widetilde{\text{E}}_\alpha} + r_i \sum_{j=1}^{i} \mathcal{O}(c_{r_j} n_p) + \mathcal{O}(c_i n_p) \right).$$

Assuming that we use the same update rank $r_i = r$, $i = 1, 2, \ldots, N$, in each update, we obtain

$$\begin{aligned} N_{\text{setup},N} &= \sum_{i=1}^{N} \left(r N_{\text{mult}\widetilde{\text{E}}_\alpha} + r \sum_{j=1}^{i} \mathcal{O}(c_r n_p) + \mathcal{O}(c_i n_p) \right) \\ &= Nr N_{\text{mult}\widetilde{\text{E}}_\alpha} + \mathcal{O}(c_{\text{setup,N}} n_p) \end{aligned}$$

with $c_{\text{setup,N}} = \frac{1}{2}N(N+1)rc_r + N(4r(r+1)+1)$. We now compare the construction costs to the setup costs of one update of rank Nr which is according to Equation (4.34) given by

$$N_{\text{setup}} = NrN_{\text{mult}\widetilde{\text{E}}_\alpha} + \mathcal{O}(c_{i,N}n_p)$$

with $c_{i,N} = 4Nr(Nr+1)+1$. Since the setup is dominated by the products with $N_{\text{mult}\widetilde{\text{E}}_\alpha}$ for $N \ll n_p$, $r \ll n_p$, the setup costs for N updates with rank r are of the same order as the setup costs for one update of rank Nr.

Randomized update We now proceed with the construction costs required for repeated updates that utilize the randomized low-rank factorization given in Algorithm 2.2. In a repeated update, each with rank r, we may reuse precomputed matrix-vector products with the preconditioned Schur complement or its transpose.

We start from the setup costs for one update as discussed in Section 4.6 to derive the setup costs required for repeated updates constructed with the randomized low-rank approximation from Algorithm 2.2. Then, we discuss how we may reuse precomputed vectors from the setup of the first update in the computation of the following updates. We analyze how the setup costs of repeated updates compare to the setup costs of one accordingly large update.

With the construction costs of one update given in Equation (4.33) and the application costs of the updated preconditioner $\widehat{S}^{-1}_{\text{R},N}$ given in Equation (4.50), we obtain the following setup costs for N updates

$$\begin{aligned} N_{\text{setup,random},N} &= \sum_{i=1}^{N} \left(2(q_i+1)\ell_i N_{\widetilde{\text{E}}_{\alpha,i}}\right) \\ &= \sum_{i=1}^{N} \left(2(q_i+1)\ell_i N_{\widetilde{\text{E}}_\alpha} + 2(q_i+1)\ell_i \sum_{j=1}^{i} \mathcal{O}(c_{r_j} n_p)\right) \end{aligned}$$

where r_i is the update rank, $\ell_i = r_i + k_i$ the number of sampling vectors, k_i the oversampling parameter, and q_i the number of power iterations used in the ith update for $i = 1, 2, \ldots, N$. The constant $c_{r_j} = 4r_j + 1$ refers to the additional application costs of the updates.

In the computation of repeated updates we may reuse some resulting vectors of multiplications with the approximate Schur complement $\widetilde{S}$ and the (possibly updated) preconditioner $\widehat{S}^{-1}$. To derive this, we consider the first two updates and assume that we use the same number of sampling vectors $\ell_1 = \ell_2 = \ell_r$ and the same random test matrix in each update. Then, we generalize the recycling of vectors for larger numbers of repeated updates.

We start with the left update. In line 2 of Algorithm 2.2 we compute

$$\widetilde{E}_{\text{L},1}G = \left(\text{I}_{n_p} - \alpha_1 \widehat{S}^{-1}\widetilde{S}\right)G = G - Y$$

for the first update and

$$\begin{aligned} \widetilde{E}_{\text{L},2}G &= \left(\text{I}_{n_p} - \alpha_1\alpha_2\big(\text{I}_{n_p} + X_{\text{random,L},\alpha_1,r}\big)\widehat{S}^{-1}\widetilde{S}\right)G \\ &= G - \alpha_2\big(\text{I}_{n_p} + X_{\text{random,L},\alpha_1,r}\big)Y \end{aligned}$$

for the second update where $Y \coloneqq \alpha_1 \widehat{S}^{-1}\widetilde{S}G \in \mathbb{R}^{n_p \times \ell_r}$. If we store the matrix Y in the setup of the first update and reuse it in the setup of the second update, we can save ℓ_r applications of the approximate Schur complement $\widetilde{S}$ and the preconditioner $\widehat{S}^{-1}$.

We now consider the right updates. In the setup of two right updates, we can reuse precomputed products if we apply the range finder in Algorithm 2.2 to the transposed error matrices $\widetilde{E}_{\mathrm{R},1}^{\mathrm{T}}$ and $\widetilde{E}_{\mathrm{R},2}^{\mathrm{T}}$. We then use the low-rank approximations

$$\begin{aligned}\widetilde{E}_{\mathrm{R},1} &= \left(\widetilde{E}_{\mathrm{R},1}^{\mathrm{T}}\right)^{\mathrm{T}} \approx \left(Q_{\mathrm{R},1}N_{\mathrm{R},1}^{\mathrm{T}}\right)^{\mathrm{T}} = N_{\mathrm{R},1}Q_{\mathrm{R},1}^{\mathrm{T}},\\ \widetilde{E}_{\mathrm{R},2} &= \left(\widetilde{E}_{\mathrm{R},2}^{\mathrm{T}}\right)^{\mathrm{T}} \approx \left(Q_{\mathrm{R},2}N_{\mathrm{R},2}^{\mathrm{T}}\right)^{\mathrm{T}} = N_{\mathrm{R},2}Q_{\mathrm{R},2}^{\mathrm{T}}\end{aligned}$$

to define the preconditioner update.

In line 2 of Algorithm 2.2 (if applied to the transposed error matrices) we calculate

$$\begin{aligned}E_{\mathrm{R},1}^{\mathrm{T}}G &= \left(\mathrm{I}_{n_p} - \alpha_1 \widehat{S}^{-\mathrm{T}}\widetilde{S}^{\mathrm{T}}\right)G = G - \alpha_1 \widehat{S}^{-\mathrm{T}}S^{\mathrm{T}}G = G - Z,\\ E_{\mathrm{R},2}^{\mathrm{T}}G &= \left(\mathrm{I}_{n_p} - \alpha_1\alpha_2\left(\mathrm{I}_{n_p} + X_{\mathrm{random,R},\alpha_1,r}\right)^{\mathrm{T}}\widehat{S}^{-\mathrm{T}}\widetilde{S}^{\mathrm{T}}\right)G\\ &= G - \alpha_2\left(\mathrm{I}_{n_p} + X_{\mathrm{random,R},\alpha_1,r}\right)^{\mathrm{T}}Z,\end{aligned}$$

with $Z \coloneqq \alpha_1 \widehat{S}^{-\mathrm{T}}\widetilde{S}^{\mathrm{T}}G \in \mathbb{R}^{n_p \times \ell_r}$. We can hence store the resulting matrix Z and reuse it in the setup of the second update to save ℓ_r applications of the approximate Schur complement $\widetilde{S}$ and the preconditioner $\widehat{S}^{-1}$.

We can store and reuse sampling vectors for the following updates as well. Assuming that we have constructed the first $i-1$, $i \geq 2$, left rank-r updates with the sampled error matrix

$$\widetilde{E}_{\mathrm{L},i-1}G = \left(\mathrm{I}_{n_p} - \alpha_{i-1}\widehat{S}_{\mathrm{L},i-2}^{-1}\widetilde{S}\right)G = G - Y_{i-1}$$

with $Y_{i-1} \coloneqq \alpha_{i-1}\widehat{S}_{\mathrm{L},i-2}^{-1}\widetilde{S}G$, we find for the setup of the ith left rank-r update

$$\widetilde{E}_{\mathrm{L},i}G = \left(\mathrm{I}_{n_p} - \alpha_i\widehat{S}_{\mathrm{L},i-1}^{-1}\widetilde{S}\right)G = G - \alpha_i\left(\mathrm{I}_{n_p} + X_{\mathrm{random,L},i-1}\right)Y_{i-1}.$$

We hence not only save ℓ_r applications of $\widetilde{S}$ and $\widehat{S}^{-1}$ but also $(i-2)\ell_r$ applications of multiplicative updates of the form $\alpha_j(\mathrm{I}_{n_p} + X_{\mathrm{random,L},j})$.

The same approach is applicable to the following right updates assuming the same rank for all updates. Here, we require the sampled error matrices

$$\begin{aligned}E_{\mathrm{R,i}-1}^{\mathrm{T}}G &= \left(\mathrm{I}_{n_p} - \alpha_{i-1}\widehat{S}_{\mathrm{R},i-2}^{\mathrm{T}}\widetilde{S}^{\mathrm{T}}\right)G\\ &= G - \alpha_{i-1}\widehat{S}_{\mathrm{R},i-2}^{-\mathrm{T}}\widetilde{S}^{\mathrm{T}}G = G - Z_{i-1},\end{aligned}$$

with $Z_{i-1} \coloneqq \alpha_{i-1}\widehat{S}_{\mathrm{R},i-2}^{-\mathrm{T}}\widetilde{S}^{\mathrm{T}}G$ to set up the $(i-1)$th update and

$$\begin{aligned}E_{\mathrm{R,i}}^{\mathrm{T}}G &= \left(\mathrm{I}_{n_p} - \alpha_i\widehat{S}_{\mathrm{R},i-1}^{-\mathrm{T}}\widetilde{S}^{\mathrm{T}}\right)G\\ &= G - \alpha_i\left(\mathrm{I}_{n_p} + X_{\mathrm{random,R},i-1}\right)^{\mathrm{T}}Z_{i-1}\end{aligned}$$

to construct the ith update. Additionally to ℓ_r (transposed) applications of $\widetilde{S}$ and $\widehat{S}^{-1}$, we also save $(i-2)\ell_r$ applications of multiplicative updates of the form $\alpha_j(\mathrm{I}_{n_p}+X_{\mathrm{random,R},j})^{\mathrm{T}}$.

For the following derivation, we assume that we use the same update ranks $r_i = r$ and the same approximation parameters, i.e. the number of power iterations $q_i = q$ and the number of oversampling vectors $k_i = k_r$, in each update, i.e. for $i = 1, 2, \ldots, N$. The parameter $\ell_r = r + k_r$ is the number of sampling vectors. The setup costs for N updates thus reduce to

$$\begin{aligned} N_{\mathrm{setup,random},N} = \sum_{i=1}^{N} \left(2(q+1)\ell_r N_{\widetilde{\mathrm{E}}_\alpha} + 2(q+1)\ell_r \sum_{j=1}^{i} \mathcal{O}(c_r n_p) \right) \\ - \sum_{i=2}^{N} \ell_r \big(N_{\mathrm{mult}\widetilde{\mathrm{S}}^{-1}} + N_{\mathrm{apply},i-2} \big) \\ = 2N(q+1)\ell_r N_{\widetilde{\mathrm{E}}_\alpha} \\ + \left[(q+1)\ell_r N(N+1) - \tfrac{1}{2}\ell_r(N-2)(N-1) \right] \mathcal{O}(c_r n_p) \\ - (N-1)\ell_r \big(N_{\mathrm{mult}\widetilde{\mathrm{S}}^{-1}} + N_{\mathrm{mult}\widehat{\mathrm{S}}^{-1}} \big). \end{aligned}$$

The variable $N_{\mathrm{apply},0} = N_{\mathrm{mult}\widehat{\mathrm{S}}^{-1}}$ denotes the application costs of the initial preconditioner. We insert the application costs of $\widetilde{E}_\alpha$ (4.32) and obtain

$$\begin{aligned} N_{\mathrm{setup,random},N} &= 2N(q+1)\ell_r \big(N_{\mathrm{mult}\widetilde{\mathrm{S}}} + N_{\mathrm{mult}\widehat{\mathrm{S}}^{-1}} + 2n_p \big) \\ &\quad + \left[(q+1)\ell_r N(N+1) - \tfrac{1}{2}\ell_r(N-2)(N-1) \right] \mathcal{O}(c_r n_p) \\ &\quad - (N-1)\ell_r \big(N_{\mathrm{mult}\widetilde{\mathrm{S}}^{-1}} + N_{\mathrm{mult}\widehat{\mathrm{S}}^{-1}} \big) \\ &= (2Nq + N + 1)\ell_r \big(N_{\mathrm{mult}\widetilde{\mathrm{S}}} + N_{\mathrm{mult}\widehat{\mathrm{S}}^{-1}} \big) + \mathcal{O}(c_{\mathrm{setup,random,N}} n_p) \end{aligned}$$

with $c_{\mathrm{setup,random},N} = 4N(q+1)\ell_r + \left[(q+1)\ell_r N(N+1) - \tfrac{1}{2}\ell_r(N-2)(N-1) \right](4r+1)$.

Now, we analyze how the setup costs of N updates of rank r compare to the setup costs of one update of rank Nr. We start from Equation (4.33) and obtain that the setup of one update of rank Nr costs

$$N_{\mathrm{setup,random,Nr}} = 2(q+1)\ell_{Nr} \big(N_{\mathrm{mult}\widetilde{\mathrm{S}}} + N_{\mathrm{mult}\widehat{\mathrm{S}}^{-1}} \big) + \mathcal{O}(c_{\mathrm{setup,random,Nr}} n_p)$$

operations where $c_{\mathrm{setup,random,Nr}} = 4(q+1)\ell_{Nr} + \ell_{Nr} + 8(q+1)\ell_{Nr}(\ell_{Nr}+1) + 2r^2$ if we use $\ell_{Nr} = Nr + k_r$ sampling vectors and the same number of power iterations q as for the repeated update. Since the multiplications with the approximate Schur complement $\widetilde{S}$ and the preconditioner $\widehat{S}^{-1}$ dominate the construction costs for $N \ll n_p$, $r \ll n_p$, we only consider the number of required applications of $\widetilde{S}$ and $\widehat{S}^{-1}$ in the setup cost comparison. For N updates of rank r we require $(2Nq + N + 1)(r + k_r)$ and for one update of rank Nr we need $2(q+1)(Nr + k_r)$ applications of $\widetilde{S}$ and $\widehat{S}^{-1}$. The setup of N updates with rank r is thus expected to be cheaper if

$$(2Nq + N + 1)(r + k_r) < 2(q+1)(Nr + k_r).$$

We rearrange and obtain

$$(2Nq + N + 1 - 2N(q+1))r < (2(q+1) - (2Nq + N + 1))k_r.$$

Solving for r yields for $N > 1$

$$r > \left(2(q+1) - \frac{1}{N-1}\right) k_r \geq (2q+1)k_r. \tag{4.51}$$

It thus depends on the number of updates N, the update rank r, and the approximation parameters q, k_r whether the repeated construction of smaller updates is cheaper than the construction of one accordingly larger update. If we do not use oversampling, i.e. $k_r = 0$, the inequality (4.51) is satisfied for all ranks $r > 0$. Then, it is cheaper to compute $N > 1$ updates of rank r than to compute one update of rank Nr.

Table 4.4 compares the ratio c_{constr} (4.47) for different numbers of updates with a total update rank of $Nr = 20$. The results are exemplarily obtained for the test system `shell_3` and are very similar for the other considered test systems. We observe that increasing the number N of updates while accordingly decreasing the single update ranks r reduces the setup costs and thus the ratio c_{constr} of construction complexity to the averaged costs of an initial GMRes iteration.

TABLE 4.4: Ratio r_{constr} (4.47) for Schur complement preconditioners with repeated outer updates assuming a total number of $\ell = 60$ iterations and a restart length of $k = 40$ for updates constructed with the randomized range finder. The results are obtained for the test system `shell_3`.

N	r	SIMPLE	LSC
1	20	30.2%	71.3%
2	10	26.4%	62.4%
5	4	24.1%	57.0%
10	2	23.3%	55.2%
20	1	22.9%	54.3%

4.6.4 Complexity of the inner updates

We now discuss the computational complexity of the construction and application of the updates discussed in Section 4.5. We restrict ourselves to the derivation for the right updates since the complexities for the left updates are the same.

The main difference regarding the construction complexity of the inner updates from Section 4.5 compared to the outer Schur complement preconditioner updates derived in Section 4.3 lies in the application costs of the error matrix. The error matrix $\widetilde{E}_{\mathrm{R},\alpha} = \mathrm{I}_{n_p} - \alpha\widetilde{S}\widehat{S}^{-1}$ (4.9) is now replaced by the error matrix $E_{M,\mathrm{R}} = \mathrm{I}_{n_p} - M\widehat{M}^{-1}$ (see Equation (4.20b) or (4.25b)) which is significantly cheaper to apply. Instead of applying the approximate Schur complement $\widetilde{S} = B\widehat{A}^{-1}B^{\mathrm{T}}$ (4.8), we now require only the application of the $n_p \times n_p$ matrix $M = BD^{-1}B^{\mathrm{T}}$. If we do not precompute M, the difference in application costs is based on the difference of the application costs of the preconditioner $\widehat{A}^{-1}$ for the $n_u \times n_u$ matrix block A which is typically more expensive than the application of the inverse diagonal $n_u \times n_u$ matrix D^{-1}. Furthermore, the application of $\widehat{S}^{-1}$ is now replaced by the application of $\widehat{M}^{-1}$. For the SIMPLE preconditioner, both preconditioners, $\widehat{S}^{-1}$ and $\widehat{M}^{-1}$ are the same. However, for the LSC method, the application of $\widehat{M}^{-1}$ is significantly less

expensive than the application of $\widehat{S}^{-1}$ since the latter requires two applications of $\widehat{M}^{-1}$ additionally to three matrix-vector products and two applications of an inverse diagonal matrix per application to a vector.

For the following complexity analysis, we use the parameters c_M, c_B, and c_{B^T} defined in Definition 4.1. We start with the construction costs of the low-rank approximations with Algorithm 2.2 or 2.4. For both algorithms, we require applications of the error matrix $E_{M,\mathrm{R}} = \mathrm{I}_{n_p} - M\widehat{M}^{-1}$ to a vector of size n_p. This costs

$$N_{\mathrm{multE}} = N_{\mathrm{multM}} + N_{\mathrm{mult}\widehat{\mathrm{M}}^{-1}} + n_p.$$

We now discuss the application costs N_{multM} of the matrix $M = BD^{-1}B^\mathrm{T}$ to a vector. We discuss two options: (i) the matrix M is precomputed and thus available in explicit form and (ii) the matrix M is not precomputed. Option (i) is required if we use for example an inexact factorization or an algebraic multigrid method to approximate the solutions to the inner Poisson-type problems. The application of the precomputed matrix $M \in \mathbb{R}^{n_p \times n_p}$ requires

$$N_{\mathrm{multM(i)}} = n_p(2c_M - 1)$$

floating point operations where c_M is the sparsity factor of M that relates the number of nonzero entries M to the number of rows $\mathrm{nnz}(M) = c_M n_p$. Note that the sparsity factor c_M is equal to the sparsity factors for the `IC(0)` factorizations utilized in both Schur complement preconditioners, the LSC and the SIMPLE preconditioner. If we do not precompute the matrix $M = BD^{-1}B^\mathrm{T}$ (option (ii)), its application to a vector of size n_p requires the application of the matrices B, D^{-1} and B^T. The costs add up to

$$\begin{aligned} N_{\mathrm{multM(ii)}} &= N_{\mathrm{multB}} + N_{\mathrm{multD}^{-1}} + N_{\mathrm{multB}^\mathrm{T}} \\ &= n_p(2c_B - 1) + n_u + n_u(2c_{B^\mathrm{T}} - 1) \\ &= 4c_B n_p - n_p = \mathcal{O}(c_{\mathrm{multM(ii)}} n_p) \end{aligned}$$

where we applied Equation (4.31) and defined $c_{\mathrm{multM(ii)}} \coloneqq 4c_B - 1$. We can hence estimate the complexity of a matrix-vector multiplication with M as

$$N_{\mathrm{multM}} \leq \max\left\{N_{\mathrm{multM(i)}}, N_{\mathrm{multM(ii)}}\right\} = \mathcal{O}(c_{\mathrm{multM}} n_p)$$

with $c_{\mathrm{multM}} = \max\{2c_M, 4c_B\} - 1$. For the randomized Algorithm 2.2, the matrix $E_{M,\mathrm{R}}$ or its transpose is applied $2\ell_r(q+1)$ times where ℓ_r is the number of sampling vectors and q is the number of power iterations.

As discussed in Section 4.6, we furthermore require orthogonalization steps and possibly re-orthogonalization steps after each application of the error matrix with the computational complexity $N_{\mathrm{orth}} = \mathcal{O}(4n_p\ell_r(\ell_r+1))$ if we require re-orthogonalization.

The construction costs hence add up to

$$\begin{aligned} N_{\mathrm{setup,random}} &= 2\ell_r(q+1)N_{\mathrm{multE}} + 2(q+1)N_{\mathrm{orth}} \\ &= 2\ell_r(q+1)N_{\mathrm{mult}\widehat{\mathrm{M}}^{-1}} + \mathcal{O}(c_{\mathrm{setup,random}} n_p) \end{aligned}$$

where $c_{\mathrm{setup,random}} = 8(q+1)\ell_r(\ell_r+1) + 2\ell_r(q+1)\max\{2c_M, 4c_B\}$.

When we construct the update with the Arnoldi iteration, we require r applications of the error matrix $E_{M,\mathrm{R}}$ to a vector. We furthermore require orthogonalization and possibly

re-orthogonalization steps which cost $N_{\text{orth}} = \mathcal{O}(4n_p r(r+1))$ for r update vectors of size n_p. The construction costs add up to

$$\begin{aligned} N_{\text{setup,Arnoldi}} &= rN_{\text{multE}} + N_{\text{orth}} \\ &= rN_{\text{mult}\widehat{\text{M}}^{-1}} + \mathcal{O}(c_{\text{setup,Arnoldi}} n_p) \end{aligned}$$

with the constant $c_{\text{setup,Arnoldi}} = 4r(r+1) + r\max\{2c_M, 4c_B\}$.

The application costs are similar to those of the outer low-rank corrections. For the SIMPLE preconditioner, the application of the low-rank update based on $E_{M,\text{R}}$ is as expensive as the application of the low-rank update based on $\widetilde{E}_{\text{R},\alpha}$. For the LSC preconditioner, the application of the updated preconditioners based on $E_{M,\text{R}}$ are slightly more expensive since we now require two update applications instead of one. The application costs hence add up to

$$N_{\text{apply,SIMPLE}} = \mathcal{O}(c_r n_p) + N_{\text{mult}\widehat{\text{S}}^{-1}}$$

for the updated SIMPLE preconditioner and

$$N_{\text{apply,LSC}} = \mathcal{O}(2c_r n_p) + N_{\text{mult}\widehat{\text{S}}^{-1}}$$

for the updated LSC preconditioner with $c_r = 4r - 1$ where we used Equations (4.35) and (4.37).

We now estimate the ratio of additional application costs per GMRes iteration as well as the required reduction in iteration counts to reduce total computational times. Since the application of the inner updates is as expensive as for the updates applied to $\widehat{S}^{-1}$ for the SIMPLE preconditioner, the percentage of additional costs per GMRes iteration is given by Equation (4.46)

$$r_{\text{upd}} = \frac{\left(\lfloor \frac{\ell}{k} \rfloor + \ell + 1\right) c_r}{\ell c_{\text{gmres}}(\ell)}.$$

For the LSC preconditioner, the ratio

$$r_{\text{upd,LSC}} = 2r_{\text{upd}} \tag{4.52}$$

is twice as high for the inner updates as for the outer updates since we require two applications of low-rank corrections instead of one per application of the preconditioner. As derived in Section 4.6.2, we find the estimate for the necessary reduction in iteration counts with inequality (4.48)

$$\Delta\ell \gtrapprox \frac{N_{\text{setup}} + \ell(1 + \frac{1}{k})(4r+1)}{N_{\text{Iter,avg}} + (1 + \frac{1}{k})(4r+1)}.$$

For the LSC preconditioner, this estimate reads

$$\Delta\ell_{\text{LSC}} \gtrapprox \frac{N_{\text{setup}} + 2\ell(1 + \frac{1}{k})(4r+1)}{N_{\text{Iter,avg}} + 2(1 + \frac{1}{k})(4r+1)} \tag{4.53}$$

due to the higher application costs of the inner low-rank corrections.

We now quantify the parameters r_{upd} and $\Delta\ell$ for the inner updates for the SIMPLE and the LSC preconditioner. With the sparsity factors given in Table 4.2, we find the

ratio r_{upd} of additional application costs and the ratio r_{constr} of additional construction costs of the inner updates for the LSC and SIMPLE preconditioner per GMRes iteration and estimates on the required reduction of iteration counts $\Delta\ell$. The results are displayed in Table 4.5. The values of r_{upd} for the SIMPLE preconditioner are equal to the values obtained for the updates applied to $\widehat{S}^{-1}$, the values of r_{upd} for the LSC preconditioner with inner updates are twice as high but still very small compared to those obtained with updates applied to $\widehat{S}^{-1}$. The ratio r_{constr} and the necessary reduction of iteration counts required to reduce total times is significantly smaller than for the updates for relaxed Schur complement preconditioners. We hence expect that it is significantly more likely to not only reduce solver times but also total times with the inner updates than for the outer updates.

TABLE 4.5: Ratios r_{upd} (4.46), (4.52) and r_{constr} (4.47) for increasing the update rank by one and necessary reduction in iteration counts per update rank $\Delta\ell$ (4.48) and $\Delta\ell_{\text{LSC}}$ (4.53) for Schur complement preconditioners with inner updates assuming a total number of $\ell = 60$ iterations and a restart length of $k = 40$ for updates constructed with the Arnoldi iteration (A) or with the randomized range finder (R).

	SIMPLE			LSC		
system	r_{upd}	r_{constr} (A, R)	$\Delta\ell$ (A, R)	r_{upd}	r_{constr} (A, R)	$\Delta\ell_{\text{LSC}}$ (A, R)
`shell_3`	0.02%	0.09%, 0.35%	0.04, 0.12	0.03%	0.09%, 0.35%	0.04, 0.09
`shell_4`	0.02%	0.09%, 0.38%	0.04, 0.12	0.03%	0.09%, 0.38%	0.04, 0.09
`cube_4`	0.02%	0.08%, 0.33%	0.04, 0.12	0.03%	0.08%, 0.33%	0.04, 0.09
`cube_5`	0.02%	0.09%, 0.37%	0.04, 0.11	0.03%	0.09%, 0.37%	0.04, 0.09

4.7 Error analysis

In this section, we analyze how the updates derived in Section 4.3 change the error between the identity matrix and the preconditioned Schur complement. The same reasoning can be applied to general error-based low-rank updates as derived in Section 4.1. We conduct the analysis for the left and right updates side by side.

For the error analysis, we recall the preconditioner updates given in Equations (4.11)

$$\widehat{S}^{-1}_{\text{random,L},\alpha,r} = \alpha\Big(\mathrm{I}_{n_p} + Q_{\text{L},\alpha,r}\big(\mathrm{I}_r - N^{\mathrm{T}}_{\text{L},\alpha,r}Q_{\text{L},\alpha,r}\big)^{-1}N^{\mathrm{T}}_{\text{L},\alpha,r}\Big)\widehat{S}^{-1},$$
$$\widehat{S}^{-1}_{\text{random,R},\alpha,r} = \alpha\widehat{S}^{-1}\Big(\mathrm{I}_{n_p} + Q_{\text{R},\alpha,r}\big(\mathrm{I}_r - N^{\mathrm{T}}_{\text{R},\alpha,r}Q_{\text{R},\alpha,r}\big)^{-1}N^{\mathrm{T}}_{\text{R},\alpha,r}\Big)$$

and (4.12)

$$\widehat{S}^{-1}_{\text{Arnoldi,L},\alpha,r} = \alpha\Big(\mathrm{I}_{n_p} + V_{\text{L},\alpha,r}\Big((\mathrm{I}_r - H_{\text{L},\alpha,r})^{-1} - \mathrm{I}_r\Big)V^{\mathrm{T}}_{\text{L},\alpha,r}\Big)\widehat{S}^{-1},$$
$$\widehat{S}^{-1}_{\text{Arnoldi,R},\alpha,r} = \alpha\widehat{S}^{-1}\Big(\mathrm{I}_{n_p} + V_{\text{R},\alpha,r}\Big((\mathrm{I}_r - H_{\text{R},\alpha,r})^{-1} - \mathrm{I}_r\Big)V^{\mathrm{T}}_{\text{R},\alpha,r}\Big).$$

We omit the updated preconditioners $\widehat{S}^{-1}_{\text{ArnoldiP,L},\alpha,r}$, $\widehat{S}^{-1}_{\text{ArnoldiP,R},\alpha,r}$ (4.13) in the discussion since their formulation has the same structure as the preconditioners $\widehat{S}^{-1}_{\text{random,L},\alpha,r}$,

$\widehat{S}^{-1}_{\text{random,R},\alpha,r}$ (4.11) and hence the same reasoning holds. To obtain a more readable analysis, we simplify the notation for the updated preconditioners. The left preconditioners $\widehat{S}^{-1}_{\text{random,L},\alpha,r}$ and $\widehat{S}^{-1}_{\text{Arnoldi,L},\alpha,r}$ have the form

$$\widehat{S}^{-1}_{\text{L}} = \alpha(\text{I}_{n_p} + X_{\text{L}})\widehat{S}^{-1} \tag{4.54}$$

and the right preconditioners $\widehat{S}^{-1}_{\text{random,R},\alpha,r}$ and $\widehat{S}^{-1}_{\text{Arnoldi,R},\alpha,r}$ can be written as

$$\widehat{S}^{-1}_{\text{R}} = \alpha\widehat{S}^{-1}(\text{I}_{n_p} + X_{\text{R}}). \tag{4.55}$$

The left and right multiplicative updates have the form

$$\left(\text{I}_{n_p} - \widehat{E}_{(\cdot)}\right)^{-1} = \text{I}_{n_p} + X_{(\cdot)} \tag{4.56}$$

where $\widehat{E}_{(\cdot)} \in \mathbb{R}^{n_p \times n_p}$ is a rank-r approximation to the error between the identity matrix and the scaled preconditioned Schur complement and $X_{(\cdot)} \in \mathbb{R}^{n_p \times n_p}$ is a rank-r matrix obtained with the Sherman-Morrison-Woodbury formula. The indices of $\widehat{E}$ and X denote the type of the update scheme.

The next theorem relates the error between the identity matrix and the preconditioned Schur complement after applying a low-rank update to the approximation error of the low-rank approximation that was used for the update construction and to the multiplicative low-rank update. Table 4.6 summarizes some definitions of matrices used in the error analysis.

Theorem 4.8. *Let the updated preconditioners $\widehat{S}^{-1}_{\text{L}}$, $\widehat{S}^{-1}_{\text{R}}$ be defined as in* (4.54), (4.55) *with X_{L}, X_{R} defined as in* (4.56). *Then for any sub-multiplicative matrix norm $\|\cdot\|$, the norm of the error matrices $E_{\text{L,upd}} = \text{I}_{n_p} - \widehat{S}^{-1}_{\text{L}} S$, $E_{\text{R,upd}} = \text{I}_{n_p} - S\widehat{S}^{-1}_{\text{R}}$ can be determined and bounded by*

$$\begin{aligned}\|E_{\text{L,upd}}\| &= \|R_{\text{L},\alpha} + X_{\text{L}} R_{\text{L},\alpha}\| \leq \|\text{I}_{n_p} + X_{\text{L}}\|\|R_{\text{L},\alpha}\|,\\ \|E_{\text{R,upd}}\| &= \|R_{\text{R},\alpha} + R_{\text{R},\alpha} X_{\text{R}}\| \leq \|R_{\text{R},\alpha}\|\|\text{I}_{n_p} + X_{\text{R}}\|.\end{aligned}$$

where

$$R_{\text{L},\alpha} \coloneqq E_{\text{L},\alpha} - \widehat{E}_{\text{L}}, \qquad R_{\text{R},\alpha} \coloneqq E_{\text{R},\alpha} - \widehat{E}_{\text{R}}$$

Table 4.6: Definition of the matrices used for the error analysis.

matrix	description	matrix	description
S	$= BA^{-1}B^{\text{T}}$	$\widetilde{S}$	$= B\widehat{A}^{-1}B^{\text{T}}$
$\widehat{A}^{-1}$	initial preconditioner for A	$\widehat{S}^{-1}$	initial preconditioner for S
$\widehat{S}^{-1}_{\text{L}}$	$= \alpha(\text{I}_{n_p} + X_{\text{L}})\widehat{S}^{-1}$	$\widehat{S}^{-1}_{\text{R}}$	$= \alpha\widehat{S}^{-1}(\text{I}_{n_p} + X_{\text{R}})$
$E_{\text{L},\alpha}$	$= \text{I}_{n_p} - \alpha\widehat{S}^{-1}S$	$E_{\text{R},\alpha}$	$= \text{I}_{n_p} - \alpha S\widehat{S}^{-1}$
$\widehat{E}_{\text{L}}$	low-rank approximation of $E_{\text{L},\alpha}$	$\widehat{E}_{\text{R}}$	low-rank approximation of $E_{\text{R},\alpha}$
$\widetilde{E}_{\text{L},\alpha}$	$= \text{I}_{n_p} - \alpha\widehat{S}^{-1}\widetilde{S}$	$\widetilde{E}_{\text{R},\alpha}$	$= \text{I}_{n_p} - \alpha\widetilde{S}\widehat{S}^{-1}$
$E_{\text{L,upd}}$	$= \text{I}_{n_p} - \alpha\widehat{S}^{-1}_{\text{L}}S$	$E_{\text{R,upd}}$	$= \text{I}_{n_p} - \alpha S\widehat{S}^{-1}_{\text{R}}$

denote the approximation error of the low-rank approximations $\widehat{E}_{\mathrm{L}} \approx E_{\mathrm{L},\alpha}$, $\widehat{E}_{\mathrm{R}} \approx E_{\mathrm{R},\alpha}$ *that was used to construct the low-rank updates with* $E_{\mathrm{L},\alpha}$ *and* $E_{\mathrm{R},\alpha}$ *defined as in Equation* (4.5).

Proof. For the proof, we require the following formula

$$\begin{aligned} \mathrm{I}_{n_p} &= \mathrm{I}_{n_p} - \widehat{E}_{(\cdot)} + (\mathrm{I}_{n_p} - \widehat{E}_{(\cdot)})X_{(\cdot)} \\ \Leftrightarrow \widehat{E}_{(\cdot)} &= (\mathrm{I}_{n_p} - \widehat{E}_{(\cdot)})X_{(\cdot)} \end{aligned} \tag{4.57}$$

which we obtain by rearranging Equation (4.56).

Now, we analyze how the update changes the error between the identity matrix and the preconditioned Schur complement. We start with the new error matrices

$$\begin{aligned} E_{\mathrm{L,upd}} &= \mathrm{I}_{n_p} - \widehat{S}_{\mathrm{L}}^{-1} S = \mathrm{I}_{n_p} - \alpha(\mathrm{I}_{n_p} + X_{\mathrm{L}})\widehat{S}^{-1} S, \\ E_{\mathrm{R,upd}} &= \mathrm{I}_{n_p} - S\widehat{S}_{\mathrm{R}}^{-1} = \mathrm{I}_{n_p} - \alpha S\widehat{S}^{-1}(\mathrm{I}_{n_p} + X_{\mathrm{R}}) \end{aligned}$$

and obtain by inserting Equation (4.5)

$$\begin{aligned} E_{\mathrm{L,upd}} &= E_{\mathrm{L},\alpha} - X_{\mathrm{L}}(\mathrm{I}_{n_p} - E_{\mathrm{L},\alpha}) = E_{\mathrm{L},\alpha} - X_{\mathrm{L}}(\mathrm{I}_{n_p} - \widehat{E}_{\mathrm{L}} + \widehat{E}_{\mathrm{L}} - E_{\mathrm{L},\alpha}), \\ E_{\mathrm{R,upd}} &= E_{\mathrm{R},\alpha} - (\mathrm{I}_{n_p} - E_{R,\alpha})X_{\mathrm{R}} = E_{\mathrm{R},\alpha} - (\mathrm{I}_{n_p} - \widehat{E}_{\mathrm{R}} + \widehat{E}_{\mathrm{R}} - E_{R,\alpha})X_{\mathrm{R}}. \end{aligned}$$

Now, we apply Equation (4.57) and find

$$\begin{aligned} E_{\mathrm{L,upd}} &= E_{\mathrm{L},\alpha} - \widehat{E}_{\mathrm{L}} + X_{\mathrm{L}}(E_{\mathrm{L},\alpha} - \widehat{E}_{\mathrm{L}}) = R_{\mathrm{L},\alpha} + X_{\mathrm{L}} R_{\mathrm{L},\alpha}, \\ E_{\mathrm{R,upd}} &= E_{\mathrm{R},\alpha} - \widehat{E}_{\mathrm{R}} + (E_{\mathrm{R},\alpha} - \widehat{E}_{\mathrm{R}})X_{\mathrm{R}} = R_{\mathrm{R},\alpha} + R_{\mathrm{R},\alpha} X_{\mathrm{R}}. \end{aligned}$$

By taking the norm of both sides and using the sub-multiplicativity of the norm, we find the error estimates

$$\begin{aligned} \|E_{\mathrm{L,upd}}\| &= \|R_{\mathrm{L},\alpha} + X_{\mathrm{L}} R_{\mathrm{L},\alpha}\| \leq \|\mathrm{I}_{n_p} + X_{\mathrm{L}}\| \|R_{\mathrm{L},\alpha}\|, \\ \|E_{\mathrm{R,upd}}\| &= \|R_{\mathrm{R},\alpha} + R_{\mathrm{R},\alpha} X_{\mathrm{R}}\| \leq \|R_{\mathrm{R},\alpha}\| \|\mathrm{I}_{n_p} + X_{\mathrm{R}}\|. \end{aligned}$$

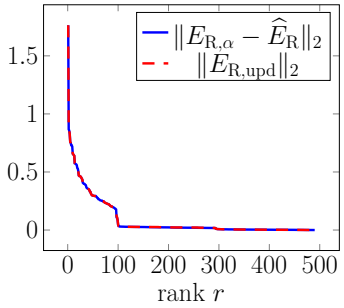
□

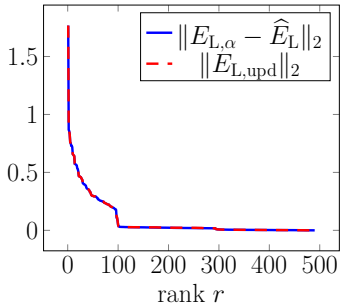
Theorem 4.8 shows that the error after applying the update does not only depend on the low-rank approximation error $R_{\mathrm{L},\alpha}$ and $R_{\mathrm{R},\alpha}$ but also the product $X_{\mathrm{L}} R_{\mathrm{L},\alpha}$ and $R_{\mathrm{R},\alpha} X_{\mathrm{R}}$, respectively. The low-rank approximation $\widehat{E}_{\mathrm{L}}$ ($\widehat{E}_{\mathrm{R}}$) is found by first approximating the error matrix $E_{\mathrm{L},\alpha}$ ($E_{\mathrm{R},\alpha}$) by $\widetilde{E}_{\mathrm{L},\alpha}$ ($\widetilde{E}_{\mathrm{R},\alpha}$), see Equation (4.9), and then applying a low-rank approximation scheme to $\widetilde{E}_{\mathrm{L},\alpha}$ ($\widetilde{E}_{\mathrm{R},\alpha}$). The approximation quality of the low-rank approximations hence depends on the quality of the Schur complement approximation $\widetilde{S} = B\widehat{A}^{-1}B^{\mathrm{T}}$ (4.8) and on the quality of the approximate low-rank factorization that we use to construct the update. We can estimate the approximation errors as

$$\begin{aligned} \|R_{\mathrm{L},\alpha}\| &= \|E_{\mathrm{L},\alpha} - \widetilde{E}_{\mathrm{L},\alpha} + \widetilde{E}_{\mathrm{L},\alpha} - \widehat{E}_{\mathrm{L}}\| \leq \|E_{\mathrm{L},\alpha} - \widetilde{E}_{\mathrm{L},\alpha}\| + \|\widetilde{E}_{\mathrm{L},\alpha} - \widehat{E}_{\mathrm{L}}\|, \\ \|R_{\mathrm{R},\alpha}\| &= \|E_{\mathrm{R},\alpha} - \widetilde{E}_{\mathrm{R},\alpha} + \widetilde{E}_{\mathrm{R},\alpha} - \widehat{E}_{\mathrm{R}}\| \leq \|E_{\mathrm{R},\alpha} - \widetilde{E}_{\mathrm{R},\alpha}\| + \|\widetilde{E}_{\mathrm{R},\alpha} - \widehat{E}_{\mathrm{R}}\| \end{aligned}$$

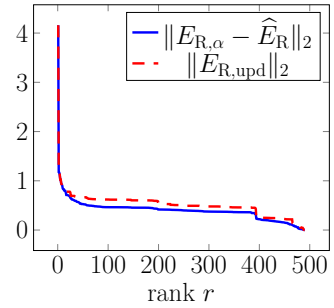
where we used the approximate error matrices $\widetilde{E}_{\mathrm{L},\alpha}$, $\widetilde{E}_{\mathrm{R},\alpha}$ from Equation (4.9).

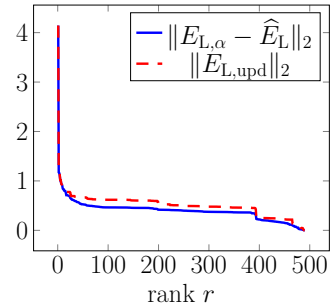
Figure 4.3 and Figure 4.4 display the error $\|E_{(\cdot),\mathrm{upd}}\|$ and the approximation error $\|E_{(\cdot),\alpha} - \widehat{E}_{(\cdot)}\|$ in spectral norm with $\alpha = 1$ for a left and a right update obtained with a



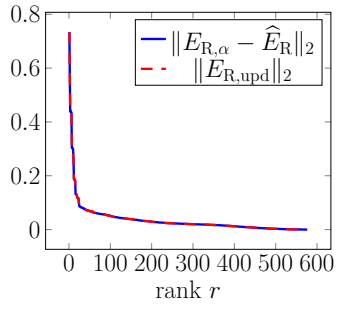


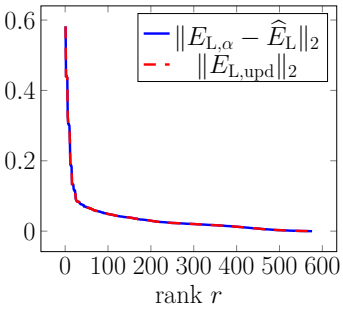
(A) Shell, LSC.



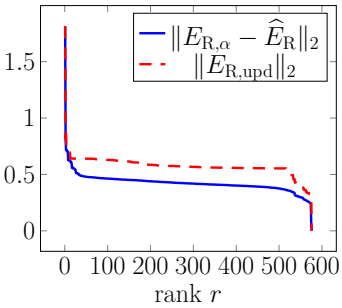


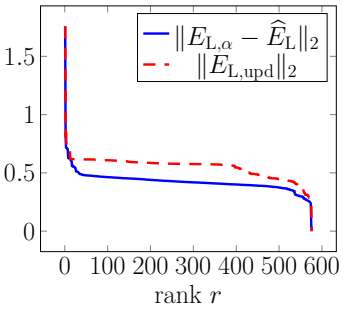
(B) Shell, SIMPLE.





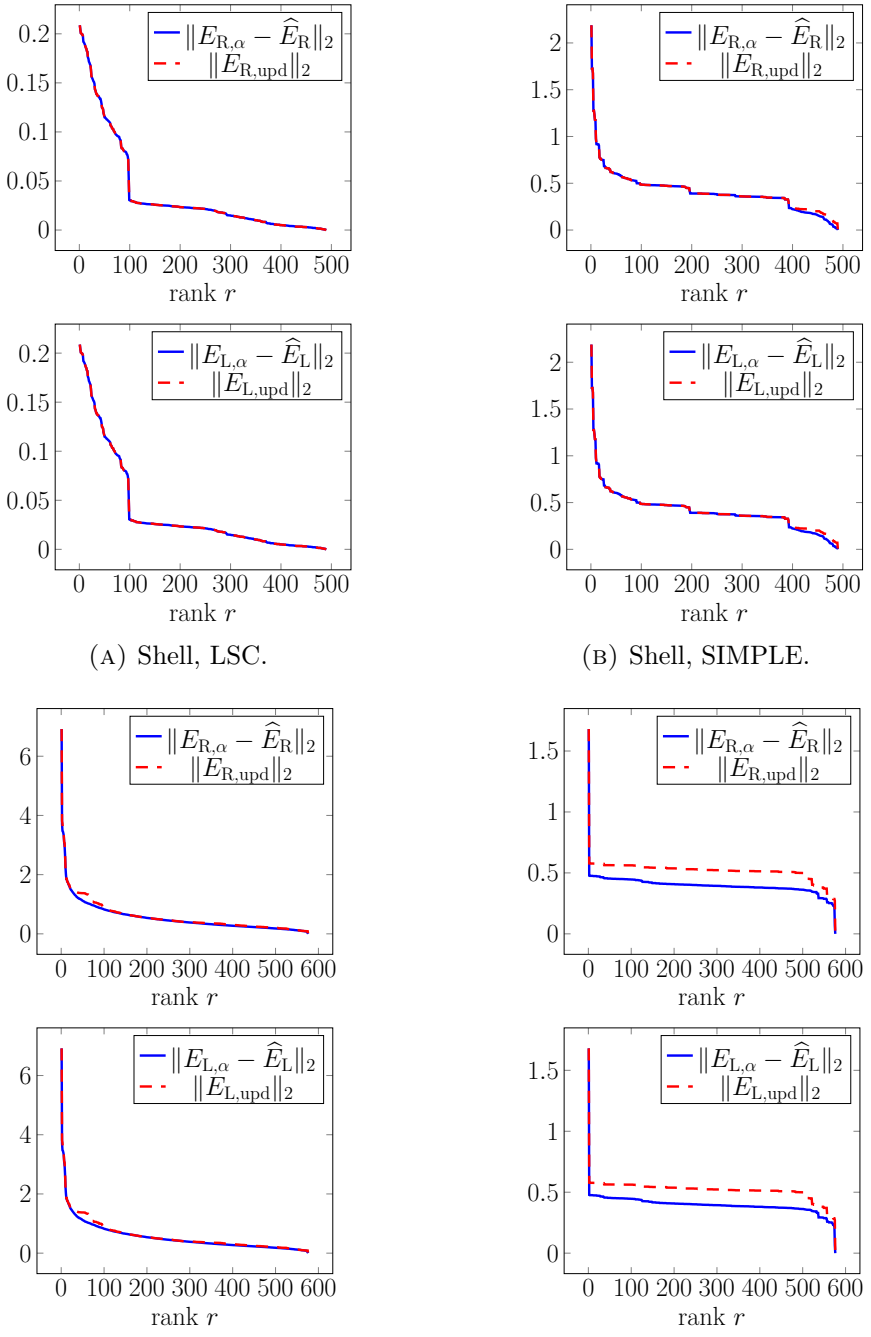
(C) Cube, LSC.





(D) Cube, SIMPLE.

FIGURE 4.3: Low-rank approximation error $\|E_{(\cdot),\alpha} - \widehat{E}_{(\cdot)}\|$ with $\alpha = 1$ and preconditioner approximation error $\|E_{(\cdot),\mathrm{upd}}\|$ after applying a right or left update constructed with a rank-r-best approximation via SVD for *Oseen* systems for a shell ($n_u = 10422$, $n_p = 490$, $k_n = 5\mathrm{e}{-4}$) and a cube ($n_u = 14739$, $n_p = 729$, $k_n = 5\mathrm{e}{-3}$).



(A) Shell, LSC. (B) Shell, SIMPLE.

(C) Cube, LSC. (D) Cube, SIMPLE.

FIGURE 4.4: Low-rank approximation error $\|E_{(\cdot),\alpha} - \widehat{E}_{(\cdot)}\|$ with $\alpha = 1$ and preconditioner approximation error $\|E_{(\cdot),\mathrm{upd}}\|$ after applying a right or left update constructed with a rank-r-best approximation via SVD for *Stokes* systems for a shell ($n_u = 10422$, $n_p = 490$, $k_n = 5\mathrm{e}{-4}$) and a cube ($n_u = 14739$, $n_p = 729$, $k_n = 5\mathrm{e}{-3}$).

rank-r best approximation using the truncated singular value decompositionof $E_{(\cdot),\alpha}$. Note that we apply the low-rank approximation scheme to the exact error matrix $E_{(\cdot),\alpha}$ and not to the approximation $\widetilde{E}_{(\cdot),\alpha}$. The results are computed in MATLAB with direct solvers for all arising inverses. The Oseen test systems are obtained on a spherical shell with time step size $k_n = 5\text{e}{-4}$ and a cube with time step size $k_n = 5\text{e}{-3}$ after five Picard correction iterations in the first time step. The Stokes test systems are obtained on the same meshes but with a quasi-Stokes discretization. Further details of the considered test systems are given in Section 5.1. The results look very similar for the left and right updates. For both update sides, we observe a significant difference between the error $\|E_{(\cdot),\text{upd}}\|$ and the approximation error $\|E_{(\cdot),\alpha} - \widehat{E}_{(\cdot)}\|$ for the SIMPLE preconditioner on the cube for the Oseen system and for the Stokes system. For the other test systems, the error $\|E_{(\cdot),\text{upd}}\|$ after applying the update is only slightly larger than the low-rank approximation error $\|E_{(\cdot),\alpha} - \widehat{E}_{(\cdot)}\|$.

We now analyze the influence of the update on the rank of the error matrices and the low-rank approximation error. Assuming that $\widehat{E}_{\text{L}}$ and $\widehat{E}_{\text{R}}$ are rank-r best approximations of $E_{\text{L},\alpha}$ and $E_{\text{R},\alpha}$, respectively, we find

$$\begin{aligned}\operatorname{rank}(R_{\text{L},\alpha}) &= \operatorname{rank}(E_{\text{L},\alpha} - \widehat{E}_{\text{L}}) = \operatorname{rank}(E_{\text{L},\alpha}) - r,\\ \operatorname{rank}(R_{\text{R},\alpha}) &= \operatorname{rank}(E_{\text{R},\alpha} - \widehat{E}_{\text{R}}) = \operatorname{rank}(E_{\text{R},\alpha}) - r.\end{aligned}$$

The low-rank update reduces the rank of the error between the identity matrix and the preconditioned Schur complement as well since

$$\begin{aligned}\operatorname{rank}\left(E_{\text{L,upd}}\right) &= \operatorname{rank}\left(R_{\text{L},\alpha} + X_{\text{L}} R_{\text{L},\alpha}\right) = \operatorname{rank}\left(\left(\mathrm{I}_{n_p} + X_{\text{L}}\right) R_{\text{L},\alpha}\right) \overset{(*)}{=} \operatorname{rank}\left(E_{\text{L},\alpha}\right) - r,\\ \operatorname{rank}\left(E_{\text{R,upd}}\right) &= \operatorname{rank}\left(R_{\text{R},\alpha} + R_{\text{R},\alpha} X_{\text{R}}\right) = \operatorname{rank}\left(R_{\text{R},\alpha}\left(\mathrm{I}_{n_p} + X_{\text{R}}\right)\right) \overset{(*)}{=} \operatorname{rank}(E_{\text{R},\alpha}) - r.\end{aligned}$$

Both last equalities, marked with $(*)$, hold since the matrices $\mathrm{I}_{n_p} + X_{\text{L}}$ and $\mathrm{I}_{n_p} + X_{\text{R}}$ are invertible as can be seen in Equation (4.56).

4.8 Spectral analysis

In this section, we analyze how the updates of Section 4.3 act on the spectrum of the preconditioned Schur complement. The analysis for the left and right updates is provided side by side following the lines in [74]. For better readability, we use the same simplified notation as in the preceding Section 4.7.

The eigenvalues of the preconditioned Schur complement are given by

$$\lambda\Big(\widehat{S}_{\text{L}}^{-1} S\Big) = \lambda\Big(\alpha\big(\mathrm{I}_{n_p} + X_{\text{L}}\big)\widehat{S}^{-1} S\Big), \tag{4.58a}$$

$$\lambda\Big(S\widehat{S}_{\text{R}}^{-1}\Big) = \lambda\Big(\alpha S\widehat{S}^{-1}\big(\mathrm{I}_{n_p} + X_{\text{R}}\big)\Big) \tag{4.58b}$$

where $\lambda(\cdot)$ denote the eigenvalues of the argument. We substitute the error matrices $E_{\text{L},\alpha}$, $E_{\text{R},\alpha}$ from Equation (4.5) into Equation (4.58), insert an additional zero, and find

$$\begin{aligned}\lambda\Big(\widehat{S}_{\text{L}}^{-1} S\Big) &= \lambda\big(\big(\mathrm{I}_{n_p} + X_{\text{L}}\big)\big(\mathrm{I}_{n_p} - E_{\text{L},\alpha}\big)\big) = \lambda\Big(\big(\mathrm{I}_{n_p} + X_{\text{L}}\big)\Big(\mathrm{I}_{n_p} - E_{\text{L},\alpha} + \widehat{E}_{\text{L},\alpha} - \widehat{E}_{\text{L},\alpha}\Big)\Big),\\ \lambda\Big(S\widehat{S}_{\text{R}}^{-1}\Big) &= \lambda\big(\big(\mathrm{I}_{n_p} - E_{\text{R},\alpha}\big)\big(\mathrm{I}_{n_p} + X_{\text{R}}\big)\big) = \lambda\Big(\Big(\mathrm{I}_{n_p} - E_{\text{R},\alpha} + \widehat{E}_{\text{L},\alpha} - \widehat{E}_{\text{L},\alpha}\Big)\big(\mathrm{I}_{n_p} + X_{\text{R}}\big)\Big).\end{aligned}$$

Then, we substitute $R_{\mathrm{L},\alpha}$ or $R_{\mathrm{R},\alpha}$ as defined in Theorem 4.8 and obtain

$$\lambda\Big(\widehat{S}_{\mathrm{L}}^{-1}S\Big) = \lambda\Big(\big(\mathrm{I}_{n_p} + X_{\mathrm{L}}\big)\Big(\mathrm{I}_{n_p} - R_{\mathrm{L}} - \widehat{E}_{\mathrm{L},\alpha}\Big)\Big),$$
$$\lambda\Big(S\widehat{S}_{\mathrm{R}}^{-1}\Big) = \lambda\Big(\Big(\mathrm{I}_{n_p} - R_{\mathrm{L}} - \widehat{E}_{\mathrm{L},\alpha}\Big)\big(\mathrm{I}_{n_p} + X_{\mathrm{R}}\big)\Big).$$

We now apply Equation (4.56) and obtain

$$\lambda\Big(\widehat{S}_{\mathrm{L}}^{-1}S\Big) = 1 - \lambda\big((\mathrm{I}_{n_p} + X_{\mathrm{L}})R_{\mathrm{L},\alpha}\big),$$
$$\lambda\Big(S\widehat{S}_{\mathrm{R}}^{-1}\Big) = 1 - \lambda\big(R_{\mathrm{R},\alpha}\big(\mathrm{I}_{n_p} + X_{\mathrm{R}}\big)\big).$$

If the eigenvalues of $(\mathrm{I}_{n_p} + X_{\mathrm{L}})R_{\mathrm{L},\alpha}$ (or $R_{\mathrm{L},\alpha}$) and $R_{\mathrm{R},\alpha}(\mathrm{I}_{n_p} + X_{\mathrm{R}})$ (or $R_{\mathrm{R},\alpha}$) are close to zero, the eigenvalues of $\widehat{S}_{\mathrm{L}}^{-1}S$ and $S\widehat{S}_{\mathrm{R}}^{-1}$, respectively, are close to one.

Now, we numerically compare spectra for the "exact" preconditioned Schur complement

$$S\widehat{S}_{(\cdot),\mathrm{R},\alpha,r}^{-1} = BA^{-1}B^{\mathrm{T}}\widehat{S}_{(\cdot),\mathrm{R},\alpha,r}^{-1} \tag{4.59}$$

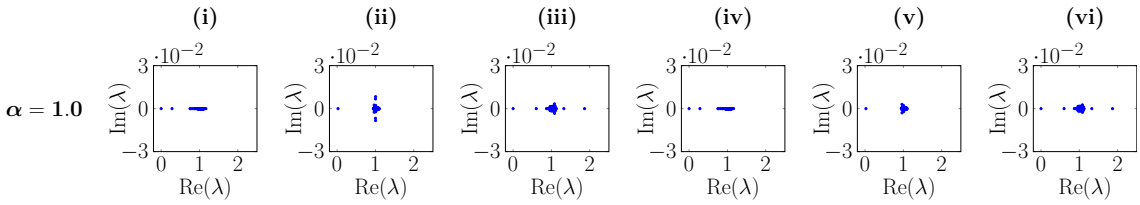
computed with a direct solver for the inverse A^{-1} to spectra for the approximate preconditioned Schur complement

$$\widetilde{S}\widehat{S}_{(\cdot),\mathrm{R},\alpha,r}^{-1} = B\widehat{A}^{-1}B^{\mathrm{T}}\widehat{S}_{(\cdot),\mathrm{R},\alpha,r}^{-1} \tag{4.60}$$

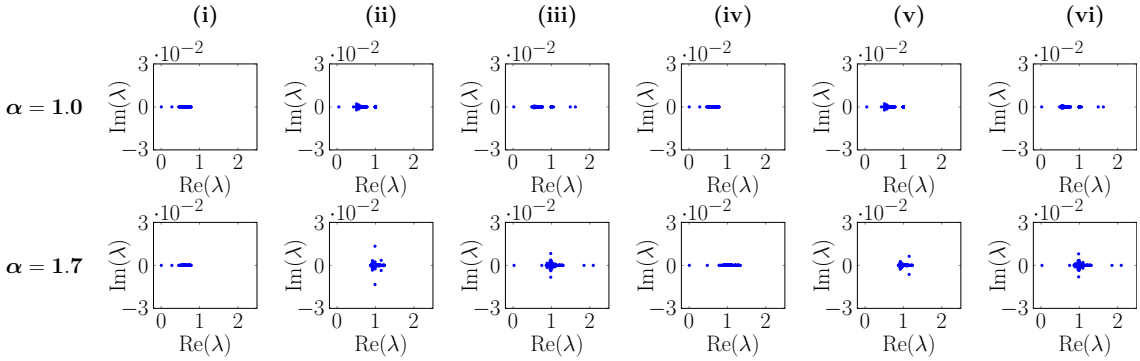
from the `C++` implementation of the methods. The updated preconditioner $\widehat{S}_{(\cdot),\mathrm{R},\alpha,r}^{-1}$ is constructed based on a low-rank approximation of the approximate error matrix $\widetilde{E}_{\mathrm{R},\alpha}$ (4.9). The eigenvalues are computed in MATLAB. Based on the results in Section 5.2 of the next chapter, we choose the following initial preconditioners: for the preconditioner $\widehat{A}^{-1}$, we choose the block triangular preconditioner (3.38) with ILU without fill-in for all arising inverses; as initial Schur complement preconditioner $\widehat{S}^{-1}$, we utilize the LSC (3.43) and the SIMPLE (3.44) preconditioner where we approximate the inner Poisson-type problems with incomplete Cholesky decompositions without fill-in. We set the relaxation parameter for the Schur complement preconditioners to $\alpha = 1$. For the SIMPLE preconditioner, we furthermore consider the parameter $\alpha = 1.7$ since we will observe in Chapter 5 that this choice leads to a higher reduction of GMRes iterations than $\alpha = 1$. The test matrices are obtained on a cube with time step size $k_n = 5\mathrm{e}{-3}$. We consider Oseen systems obtained after five Picard correction iterations in the first time step and quasi-Stokes systems. Further details of the considered test systems are given in Section 5.1.

Figure 4.5 shows spectra of the (approximate) preconditioned Schur complement before and after applying an update of rank $r = 30$ obtained from a randomized low-rank approximation with Algorithm 2.2 (applied to the transposed error matrix) and from the Arnoldi Algorithm 2.4. The spectra for the Oseen systems for reusing the initial `IC(0)` factorization of the Schur complement preconditioners and for recomputing it for each saddle-point system had no visible difference. Therefore, we only show the spectra for recomputing the `IC(0)` factorizations. For both initial Schur complement preconditioners, we observe that the eigenvalues of the approximate preconditioned Schur complement (4.60) appear to be very similar to those of the exact preconditioned Schur complement (4.59) for the quasi-Stokes as well as for the Oseen-type systems. The relaxation parameter α shifts the

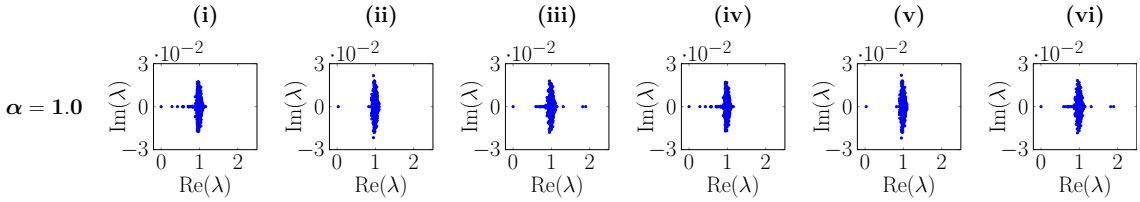
center of the spectrum and the spectral radius. A center close to one is expected to be beneficial regarding convergence behavior obtained with the updated preconditioner. For the SIMPLE preconditioner, we observe that the center of the spectrum is shifted towards one with the relaxation parameter $\alpha = 1.7$. This is advantageous for the considered update techniques since the updates aim to shift eigenvalues (or singular values) to one. The randomized updates, built by approximating the range of the transposed error matrix, improve the clustering of eigenvalues more than the updates based on the Arnoldi iteration that have only a slight visible influence on the spectrum. For the Stokes systems, we furthermore observe that the updates introduce small imaginary parts for some eigenvalues since the update scheme is not symmetric.



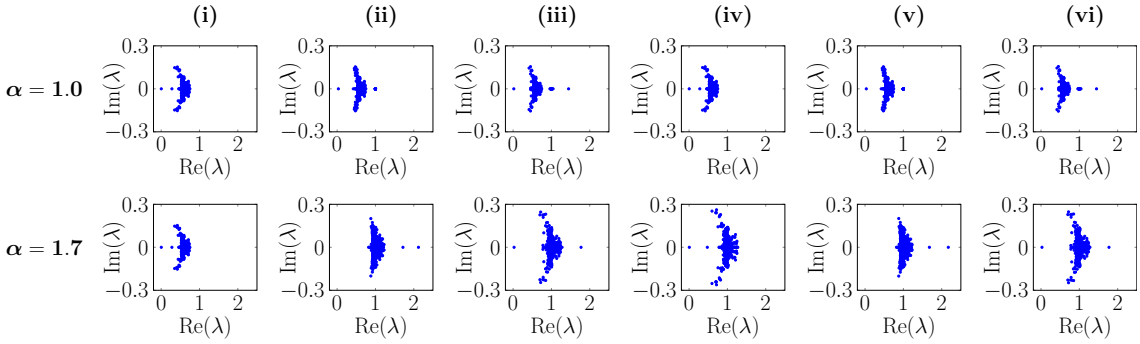
(A) Stokes, LSC.



(B) Stokes, SIMPLE.



(C) Oseen, LSC.



(D) Oseen, SIMPLE.

FIGURE 4.5: Spectra of the preconditioned (approximate) Schur complement for the cube ($n_u = 14739$, $n_p = 729$, $k_n = 5\text{e}{-3}$). The images from left to right show: spectra of the Schur complement preconditioned with (i) $\alpha\widehat{S}^{-1}$ without any update, (ii) the preconditioner $\widehat{S}^{-1}_{\text{R,random},\alpha,r}$ (4.11b) with update rank $r = 30$, (iii) the preconditioner $\widehat{S}^{-1}_{\text{R,Arnoldi},\alpha,r}$ (4.12b) with update rank $r = 30$, and spectra of the approximate Schur complement from Equation (4.8) preconditioned with (iv) $\alpha\widehat{S}^{-1}$ without any update, (v) the preconditioner $\widehat{S}^{-1}_{\text{R,random},\alpha,r}$ (4.11b) with update rank $r = 30$, and (vi) the preconditioner $\widehat{S}^{-1}_{\text{R,Arnoldi},\alpha,r}$ (4.12b) with update rank $r = 30$.

Chapter 5

Numerical results for preconditioners with low-rank updates

In this chapter, we illustrate and analyze the effectiveness of the updated preconditioners derived in Chapter 4 with various numerical experiments. We test the update techniques for Oseen and Stokes systems originating from the Boussinesq problem described in Chapter 3.

Preliminarily, we define the test setting in Section 5.1. We start the experiments with a numerical analysis of the initial preconditioners in Section 5.2. In Section 5.3, we illustrate the influence of the relaxation parameter on the convergence behavior of the iterative solver. Then, we analyze the effectiveness of the derived low-rank corrections for different preconditioner components. Section 5.4 is concerned with (outer) low-rank updates for relaxed Schur complement preconditioners. In Section 5.5, we numerically analyze Schur complement preconditioners with inner updates. For completeness, we briefly discuss low-rank updates for the preconditioner $\widehat{A}^{-1}$ in Section 5.6.

The presentation of the results in this chapter is based on our article [9]. In this thesis, we extend the numerical analysis to a wider range of test systems, parameters, and update schemes. Furthermore, we apply the low-rank updates to different initial preconditioners.

5.1 Test setting

In this section, we define the test setting that we use for the numerical experiments. We consider two domains, a unit cube and a spherical shell with inner radius R_0 and a thickness of R_1. For both domains, we use a globally refined mesh with hexahedral cells motivated by the implementation of finite elements in the software package DEAL.II [5, 6]. Figure 5.1 shows two example grids, one for the cube and one for the spherical shell. The test systems are obtained with three different refinement levels for the spherical shell and the cube. Table 5.1 shows the sizes of the considered test systems. The number in the name of the test systems denotes the number of global mesh refinement steps. The coarsest systems are only used for the numerical illustration of the error and spectral analysis in the previous chapter except for the coarsest cube systems furthermore used for increasing the update rank to a maximum. Most experiments are performed for the larger two system sizes. For the numerical experiments, we consider systems obtained with two linearization methods for each of the meshes: a quasi-Stokes system (3.25) (in the following: Stokes

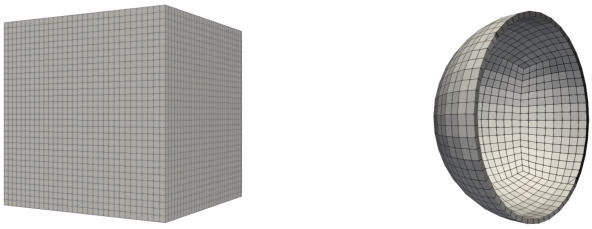


FIGURE 5.1: Example grids with uniformly refined hexahedral cells for the cube and the spherical shell.

TABLE 5.1: Sizes of the test systems.

identifier	geometry	n_u	n_p	outer cell diameter
`shell_2`	spherical shell	10422	490	1
`shell_3`	spherical shell	78438	3474	0.518
`shell_4`	spherical shell	608454	26146	0.255
`cube_3`	cube	14739	729	0.217
`cube_4`	cube	107811	4913	0.108
`cube_5`	cube	823875	35937	0.054

systems) and a Picard-linearized system (3.27) (in the following: Oseen systems). For the Oseen systems, we use the systems obtained after five Picard correction iterations as test systems. The following experiments are performed with test systems obtained for the first time step.

The physical parameters included in our model are given in Table 5.2. The definition and values of the nondimensional parameters of our model are shown in Table 5.3. We chose the velocity and length scale such that the Reynolds number, the Rossby number, and the Péclet number are moderate.

Software and Hardware We implemented the methods in `C++` using the finite element library DEAL.II in version 9.4.0 [5, 6] with the Trilinos collection in version 13.4.1 [38]. For

TABLE 5.2: Parameters for the atmospheric model.

description	symbol	value	unit
inner radius	R_0	2	m
atmospheric height	R_1	0.1	m
reference atmosphere temperature	T_{ref}	2	K
atmosphere temperature change	ΔT	0.5	K
reference air density	ρ_{ref}	1.29	$\text{kg}\,\text{m}^{-3}$
dynamic viscosity	ν	1.82e-5	$\text{kg}\,\text{m}^{-1}\,\text{s}^{-1}$
Earth angular velocity	ω	7.272205e-5	s^{-1}
thermal expansion coefficient	β	3.661e-3	K^{-1}
thermal conductivity	κ	2.62e-2	$\text{W}\,\text{m}^{-1}\,\text{K}^{-1}$
gravity constant	g	9.81	$\text{m}\,\text{s}^{-2}$

TABLE 5.3: Dimensionless numbers obtained with the velocity scale $U = 0.01ms^{-1}$ and the length scale $L = 1.0m$ and the physical parameters given in Table 5.2.

description	symbol	definition	value
Reynolds number	Re	$UL\rho_{\text{ref}}/\nu$	708.791
Péclet number	Pe	UL/κ	494.828
Rossby number	Ro	$U/(L\omega)$	137.51

creating the mesh, the software package `p4est` [19] is exploited which could be used for distributing the mesh on different processes. An implementation of the Boussinesq model provided by Konrad Simon served as a basis for our implementation of the numerical methods. The implementation used to produce the results in this thesis is based on code provided in a GitHub repository[1] that was published along with our article [10]. We performed the numerical experiments on a desktop computer with 32 GB RAM and an Intel Xeon Gold 6240 processor with 2.60 GHz and 18 cores. All numerical experiments are conducted in serial, i.e. by using one thread on one process.

For the numerical tests, we use a right preconditioned GMRes solver with a restart length of 40. The chosen restart length balances solver costs influenced by the size of the Arnoldi basis in the solver and iteration counts. We choose right-oriented preconditioning since then the residuals equate to the residuals of the original equation. For the preconditioners that involve an inner iterative solver, we use a flexible GMRes solver as the outer solver. The (flexible) GMRes iteration is stopped when the relative residual drops below 10^{-8}. Most of the shown computational times are obtained from one test run since the results in Sections 5.4.2 and 5.5.2 show that the standard deviation of run times is only about 1-3% from the measured time.

Setting for the low-rank approximations Constructing the derived low-rank updates requires low-rank approximation methods where we use the following setting. For the low-rank approximation via the Arnoldi Algorithm 2.4 (with and without projection), we set the tolerance for re-orthogonalization to $\xi = 0.5$ since smaller values for ξ led to a loss of orthogonality in numerical experiments. The randomized low-rank updates can be computed based on an error matrix or its transpose. For right updates, we compute the randomized low-rank approximation by applying the randomized power range finder in Algorithm 2.2 to the transposed error matrix since we derived in Section 4.6.3 that sample vectors can then be reused for following updates. This means that we call the Algorithm 4.1 with the flag `approx_transposed` set to `true`. For the left updates, sampling vectors can be reused when we apply Algorithm 2.2 to the error matrix and not to its transpose. Hence, we construct and define the left updates as described in Algorithms 4.1 with the flag `approx_transposed` set to `false`.

Reusing factorizations For both Schur complement preconditioners, the SIMPLE and the LSC preconditioner given in (3.44) and (3.43), we require approximations to inner Poisson-type problems. For most numerical experiments, we will utilize approximations by

[1] https://github.com/rbeddig/LowRankUpdates/tree/YRMCSE22-proceedings, accessed on Nov. 05, 2023.

incomplete Cholesky factorizations. The preconditioner construction for both approaches comprises precomputing a matrix of the type $M = BD^{-1}B^{\mathrm{T}}$ with a diagonal matrix D and precomputing the algebraic multigrid, involving the construction of a multilevel hierarchy and precomputing the LU factorization on the coarse level, or the factorizations. For the LSC preconditioner, the matrix M does not change with Picard-type iterations or time steps for the considered setting which does not apply to the SIMPLE preconditioner. However, to save construction costs, we will perform the precomputations only for the first saddle-point system, i.e. a Stokes system in the first time step, and then reuse the approximations to the Poisson-type problems in the following Picard-type iterations. This approach is used in all experiments in this chapter if not stated otherwise. Due to the small changes between the different saddle-point systems, this approach results in very similar iteration counts for the initial preconditioner as well as the updated preconditioners with smaller cumulative construction times. A brief numerical comparison in the next Section 5.2 for reusing and recomputing the factorizations justifies our approach.

5.2 Initial preconditioners

We begin by comparing different initial preconditioners $\widehat{A}^{-1}$ and $\widehat{S}^{-1}$ for the block triangular preconditioner

$$P_{\mathrm{U}} = \begin{pmatrix} \widehat{A}^{-1} & \widehat{A}^{-1}B^{\mathrm{T}}\widehat{S}^{-1} \\ 0 & -\widehat{S}^{-1} \end{pmatrix}.$$

First, we study the preconditioners for the upper left matrix block that are discussed in Section 3.5.1. Then, we compare the SIMPLE (3.44) and the LSC (3.43) preconditioner for the Schur complement with various approximations of the inner Poisson-type problems.

Preconditioners $\widehat{A}^{-1}$ We compare preconditioners that are based on (a) an $n_u \times n_u$ matrix and (b) on a block-structured matrix. For approach (a), we consider preconditioners built from the whole matrix A, the almost block diagonal replacement matrix $\widetilde{A}$ defined in (3.36), and the velocity mass matrix M_u. For approach (b), we choose a block diagonal (referred to as 'b-diag') and an upper block triangular preconditioner (referred to as 'b-tri') based on the subblocks of A as defined in (3.37) and (3.38). We compare an algebraic multigrid method with an inexact LU factorization to approximate the arising inverses for all approaches for $\widehat{A}^{-1}$.

For the multigrid approach, we use an algebraic multigrid preconditioner (AMG) with smoothed aggregation from the `ML` package [69, 32], included in Trilinos. We use two smoothing steps with a Chebyshev smoother for the symmetric matrices and a Gauss-Seidel smoother for the asymmetric matrices. We solve the systems on the coarsest levels with a LU factorization and use one V-cycle of the algebraic multigrid to approximate the arising inverses in the preconditioners. The levels of the multigrid method are obtained using smoothed aggregation with an aggregation threshold of 0.024. This value seemed to be favorable in numerical experiments. Furthermore, we consider preconditioners $\widehat{A}$ that employ an inexact LU factorization (ILU) with zero additional fill of the `Ifpack` package [63], included in Trilinos.

For the preconditioners built from A, $\widetilde{A}$, and M_u, the setup involves the construction of either an ILU factorization or an AMG of size $n_u \times n_u$. For the block diagonal and block

triangular preconditioner, the setup requires the construction of either three incomplete factorizations or three AMGs, each of size $n_u/3 \times n_u/3$ for the considered three-dimensional domains.

For $\widehat{S}^{-1}$, we use the SIMPLE preconditioner defined in (3.44) where we approximate the Poisson-type problem with an incomplete Cholesky factorization (IC) with zero additional fill from the `Ifpack` package [63].

We test the preconditioners for the Stokes and the Oseen systems, defined in (3.25) and (3.27), on a spherical shell and a cube with two different refinement levels for both domains. The results obtained with the preconditioners that lead to the smallest total times are printed in bold.

Table 5.4 shows iteration counts, setup and solver times obtained for the Oseen systems. We observe that the preconditioners utilizing ILU lead to lower setup and solver CPU times than the preconditioners utilizing AMG for all systems. Furthermore, the GMRes solver fails to converge within 1000 iterations (marked with "nc") for the preconditioners with AMG for the cube systems.

The ILU preconditioner based on A leads to the lowest iteration counts for all test problems. However, the setup and application costs are significantly higher than for the other considered ILU preconditioners due to the coupling of velocity components. The ILU preconditioners based on M_u, $\widetilde{A}$ as well as the block ILU preconditioners require similar construction times. The block triangular ILU preconditioners lead to the lowest total times for the test systems except for the coarser cube (`cube_4`) for which the block diagonal preconditioner leads to the lowest total times.

Table 5.5 shows iteration counts, setup and solver times obtained for the Stokes systems. As well as observed for the Oseen systems, we see that the preconditioners with ILU lead to smaller setup times and faster convergence than the preconditioners with AMG for all considered systems. The ILU preconditioners based on A lead as well to the smallest iteration counts but come along with the highest setup costs. The preconditioners based on the velocity mass matrix M_u perform better for the Stokes systems than for the Oseen systems but are only one of the most effective preconditioners for the coarser shell system (`shell_3`). For most of the Stokes systems (except for `cube_5`), the ILU preconditioner based on the replacement matrix $\widetilde{A}^{-1}$ is one of the preconditioners that yield the smallest setup and solver times. The block triangular ILU preconditioner leads to similar computational times. For the coarse cube (`cube_4`), the block diagonal preconditioner results in smallest computational times.

In the following, we use the block triangular ILU preconditioner for $\widehat{A}^{-1}$ since it leads to the smallest computational times for the majority of the test systems and second smallest times (and iteration counts) for the other systems. To obtain comparable results, we use the same initial preconditioner for all remaining numerical experiments.

Schur complement preconditioners $\widehat{S}^{-1}$ We now compare various initial Schur complement preconditioners $\widehat{S}^{-1}$ to be used in the upper block triangular preconditioner P_U (3.35).

We compare the LSC preconditioner given in (3.43) and the SIMPLE preconditioner given in (3.44). Both preconditioners require the (approximate) solution to inner Poisson-type problem(s) with system matrices of the type $BD^{-1}B^{\mathrm{T}}$ with a diagonal matrix D. We compare three approaches: (a) an inner iterative solver, (b) an algebraic multigrid

TABLE 5.4: Setup and solver times in seconds and iteration counts obtained with GMRes preconditioned with the upper block triangular preconditioner (3.35) using various preconditioners $\widehat{A}^{-1}$. The Schur complement is preconditioned with the SIMPLE preconditioner with `IC(0)`. The results are obtained for the *Oseen* systems, see Table 5.1 for the system dimensions.

		ILU					AMG				
		b-diag	b-tri	A	$\widetilde{A}$	M_u	b-diag	b-tri	A	$\widetilde{A}$	M_u
`shell_3`,	setup $\widehat{A}^{-1}$	0.42	**0.39**	1.50	0.40	0.39	1.14	1.12	9.91	2.17	0.62
$k_n = 5e{-}4$	solve	1.00	**0.94**	1.19	1.86	5.14	9.17	9.15	16.14	7.54	9.86
	iterations	38	**31**	30	32	190	224	201	146	141	190
`shell_4`,	setup $\widehat{A}^{-1}$	3.14	**3.16**	12.41	3.09		7.56	7.61	111.38	28.87	
$k_n = 2.5e{-}4$	solve	24.01	**17.36**	22.57	19.35		34.31	31.16	73.82	44.42	
	iterations	108	**67**	67	79	nc	101	82	74	85	nc
`cube_4`,	setup $\widehat{A}^{-1}$	**0.55**	0.55	2.14	0.50						
$k_n = 5e{-}3$	solve	**2.77**	3.25	4.22	2.90						
	iterations	**75**	75	75	76	nc	nc	nc	nc	nc	nc
`cube_5`,	setup $\widehat{A}^{-1}$	4.27	**4.35**	17.33	4.10						
$k_n = 2.5e{-}3$	solve	33.10	**31.75**	39.80	35.01						
	iterations	106	**83**	84	109	nc	nc	nc	nc	nc	nc

TABLE 5.5: Setup and solver times in seconds and iteration counts obtained with GMRes preconditioned with the upper block triangular preconditioner (3.35) using various preconditioners $\widehat{A}^{-1}$. The Schur complement is preconditioned with the SIMPLE preconditioner with `IC(0)`. The results are obtained for the *Stokes* systems, see Table 5.1 for the system dimensions.

		ILU					AMG				
		b-diag	b-tri	A	$\widetilde{A}$	M_u	b-diag	b-tri	A	$\widetilde{A}$	M_u
`shell_3`,	setup $\widehat{A}^{-1}$	0.41	**0.42**	1.54	**0.48**	**0.42**	0.84	0.86	1.89	0.72	0.63
$k_n = 5e{-}4$	solve	0.85	**0.74**	0.90	**0.69**	**0.76**	1.59	1.48	3.50	1.69	1.68
	iterations	32	**24**	22	**25**	**28**	35	30	29	30	32
`shell_4`,	setup $\widehat{A}^{-1}$	3.27	3.30	12.53	**3.23**	3.09	3.13	3.05	17.08	8.47	5.89
$k_n = 2.5e{-}4$	solve	33.41	27.16	25.15	**26.30**	35.48	67.82	51.12	123.68	80.42	111.83
	iterations	145	99	73	**109**	149	181	120	118	141	239
`cube_4`,	setup $\widehat{A}^{-1}$	**0.55**	0.55	2.19	**0.50**	0.50	2.12	2.09	3.37	0.81	0.81
$k_n = 5e{-}3$	solve	**0.98**	1.14	1.53	**1.04**	1.17	2.36	2.60	6.49	2.29	2.39
	iterations	**27**	27	27	**28**	31	36	36	32	32	33
`cube_5`,	setup $\widehat{A}^{-1}$	4.34	**4.38**	17.55	3.99	3.99	4.43	4.38	32.90	11.96	7.17
$k_n = 2.5e{-}3$	solve	19.17	**17.34**	22.85	20.08	24.81	38.45	41.90	133.03	60.05	48.52
	iterations	62	**47**	46	64	78	72	71	68	71	79

method, and (c) an incomplete Cholesky factorization (IC). The approach has negligible construction costs while leading to higher application costs for $\widehat{S}^{-1}$ while approaches (b) and (c) have comparably higher construction costs while leading to a cheaper application of $\widehat{S}^{-1}$. We do not use preconditioned iterations to avoid the additional construction costs required to set up a preconditioner while having also higher application costs for $\widehat{S}^{-1}$ due to the inner iterations. For the inner iterative solver, we use the conjugate gradient method (CG) with a relative (stopping) tolerance of 0.1 for the relative residual. For the approximation with a multigrid method, we use one V-cycle of an algebraic multigrid from the `ML` package in TRILINOS with two pre- and post-smoothing steps with a Chebyshev smoother. We obtain the multigrid levels with smoothed aggregation and an aggregation threshold of 0.02 which seemed to be favorable in numerical experiments. Furthermore, we consider an inexact Cholesky factorization with zero additional fill from the `Ifpack` package. We also compared inexact Cholesky factorizations with more fill-in but this led to significantly higher setup times and did hardly or even not improve the convergence of the outer solver. The setup costs for preconditioners with IC or AMG include the calculation of a matrix-matrix product to precompute the matrix $BD^{-1}B^{\mathrm{T}}$ with $D = \mathrm{diag}(A)$ for the SIMPLE preconditioner and $D = \mathrm{diag}(M_u)$ for the LSC preconditioner. The setup costs furthermore include the setup of the (inner) IC or AMG preconditioner. The setup costs for the preconditioners with inner CG solver include the precomputation of the inverse diagonal matrix required for the Poisson-type problems. Based on the results so far, we choose the block triangular preconditioner (3.38) with ILU without any fill-in to approximate all arising inverses for the preconditioner $\widehat{A}^{-1}$.

We compare the Schur complement preconditioners for the same Oseen and Stokes systems as above. Results for the preconditioners with smallest total times are printed in bold. We furthermore consider the costs per iteration where the smallest values are printed in blue.

Table 5.6 shows iteration counts, setup, solver times and time per iteration obtained with the Schur complement preconditioners for the Oseen systems. The preconditioners with inner inexact CG solves, led to the lowest iteration counts but high solver times. Furthermore, the number of outer iterations is similar for both mesh sizes. A boundary correction as discussed in Section 3.5.2.1 thus does not seem to be necessary. The application costs for the preconditioners with inner iterative solver vary significantly depending on the number of required inner iterations. For the cube systems, the number of inner iterations per outer iteration varies between three to 60. For the shell systems, the number of required inner iterations for the CG solves per outer iteration varies from five iterations to nearly 1500 iterations. Due to the high number of required inner iterations, the solver times blow up for the shell systems.

For most of the shell systems, the preconditioners that lead to the lowest total times are the ones that utilize an incomplete Cholesky decomposition. However, the LSC preconditioner does not lead to convergence on the finer meshes when the inner Poisson-type problems are approximated by IC factorizations. For the finer meshes (`shell_4`), the LSC preconditioner only leads to convergence within less than 1000 iterations when the inner Poisson-type problems are approximately solved with the inner iterative solver which requires too high solver times.

For the cube systems on coarser grids (`cube_4`), the preconditioners with `IC(0)` appear to be favorable concerning computational times whereas the AMG option seems to be

TABLE 5.6: Setup, solver times and cost per iteration in seconds and iteration counts (inner CG iterations) obtained with (F)GMRes preconditioned with the upper block triangular preconditioner (3.35) using various preconditioners for $\widehat{S}^{-1}$ and the fixed preconditioner (3.38) for $\widehat{A}^{-1}$. The results are obtained for the *Oseen* systems, see Table 5.1 for the system dimensions.

		SIMPLE			LSC		
		CG	AMG	IC	CG	AMG	IC
`shell_3`,	setup $\widehat{S}^{-1}$	0.01	0.15	**0.17**	0.01		**0.17**
$k_n = 5e{-}4$	solve	11.42	7.14	**0.95**	27.26		**1.51**
	iterations	26 (5098)	216	**31**	24 (13570)	nc	**32**
	time/iteration	0.439	0.033	**0.031**	1.136		**0.047**
`shell_4`,	setup $\widehat{S}^{-1}$	0.09	1.18	**1.29**	**0.10**		
$k_n = 2.5e{-}4$	solve	189.36	67.61	**17.40**	**502.02**		
	iterations	23 (9709)	250	**67**	**20** (26087)	nc	nc
	time/iteration	8.233	0.270	**0.260**	**25.101**		
`cube_4`,	setup $\widehat{S}^{-1}$		0.19	**0.21**		**0.21**	0.22
$k_n = 5e{-}3$	solve		4.03	**3.36**		**5.16**	5.37
	iterations	nc	86	**75**	nc	**74**	76
	time/iteration		0.047	**0.045**		**0.070**	0.071
`cube_5`,	setup $\widehat{S}^{-1}$	0.12	**1.83**	1.84	0.13	**1.82**	
$k_n = 2.5e{-}3$	solve	35.96	**23.25**	30.97	61.07	**33.24**	
	iterations	37 (855)	**60**	83	26 (1767)	**56**	nc
	time/iteration	0.972	0.388	**0.373**	2.349	**0.594**	

favorable for the finer discretized cubes (`cube_5`).

Table 5.7 shows iteration counts, setup, solver times and costs per iteration for the Stokes systems. For the shell systems, the preconditioners with IC lead to the smallest total times except for LSC on the finer mesh. Here again, only the LSC preconditioner with the inner CG solver leads to convergence within 1000 GMRes iterations but requires long iteration times. For the cube systems, the preconditioners with AMG lead to the fastest convergence. The range of required inner CG iterations is very similar to that observed for the Oseen systems.

We observe that the quality of the approximations to the inner Poisson-type problems has a significant impact on the required number of iterations. The preconditioners with inner iterative solver lead to convergence within small numbers of iterations (in most cases up to 20-30 iterations). However, the inner iterations can lead to high solver times. The application costs for the Schur complement preconditioners with IC are just the lowest of the considered options. In Section 5.4, we investigate whether an (outer) low-rank update applied to a Schur complement preconditioner that is cheap to apply while not too expensive to construct can reduce iteration counts and speed up convergence. In the Sections 5.4 and 5.5, we analyze the effectiveness of outer and inner low-rank updates for the Schur complement preconditioners with `IC(0)`. AMG leads to similar costs per

TABLE 5.7: Setup, solver times and cost per iteration in seconds and iteration counts (inner CG iterations) obtained with (F)GMRes preconditioned with the upper block triangular preconditioner (3.35) using various preconditioners for $\widehat{S}^{-1}$ and the fixed preconditioner (3.38) for $\widehat{A}^{-1}$. The results are obtained for the *Stokes* systems, see Table 5.1 for the system dimensions.

		SIMPLE			LSC		
		CG	AMG	IC	CG	AMG	IC
`shell_3`,	setup $\widehat{S}^{-1}$	0.01	0.15	**0.16**	0.01		**0.16**
$k_n = 5e{-}4$	solve	10.91	5.67	**0.73**	23.76		**1.43**
	iterations	17 (4977)	178	**24**	18 (11003)	nc	**30**
	time/iteration	0.642	0.032	**0.030**	1.320		**0.048**
`shell_4`,	setup $\widehat{S}^{-1}$	0.10	1.24	**1.36**	**0.10**		
$k_n = 2.5e{-}4$	solve	207.00	59.49	**25.87**	**512.41**		
	iterations	17 (10522)	218	**99**	**17** (26313)	nc	nc
17	time/iteration	12.176	0.273	**0.261**	30.14		
`cube_4`,	setup $\widehat{S}^{-1}$	0.03	**0.32**	0.34	0.03	**0.33**	0.37
$k_n = 5e{-}3$	solve	2.96	**1.69**	1.95	4.16	**3.92**	3.94
	iterations	15 (347)	**23**	27	12 (527)	**33**	35
	time/iteration	0.197	0.073	**0.072**	0.347	0.119	**0.113**
`cube_5`,	setup $\widehat{S}^{-1}$	0.12	**1.81**	1.85	0.13	**1.81**	
$k_n = 2.5e{-}3$	solve	22.06	**8.72**	17.45	33.21	**19.99**	
	iterations	15 (634)	**23**	47	14 (959)	**33**	nc
	time/iteration	1.471	0.379	**0.371**	2.372	**0.606**	

application as IC but its transposed application is typically not implemented which is required in the methods that rely on a randomized low-rank approximation.

TABLE 5.8: Solver times and iteration counts obtained with GMRes preconditioned with the upper block triangular preconditioner (3.35) using the preconditioner (3.38) for $\widehat{A}^{-1}$ and two versions of the SIMPLE preconditioner (3.44) with `IC(0)` for $\widehat{S}^{-1}$: reusing the initial `IC(0)` factorization and recomputing a new one. The results are obtained for the *Oseen* systems, see Table 5.1 for the system dimensions.

		recompute IC	reuse IC
`shell_3`,	solve	0.95	0.95
$k_n = 5\text{e}{-4}$	iterations	31	31
`shell_4`,	solve	17.66	17.68
$k_n = 2.5\text{e}{-4}$	iterations	66	67
`cube_4`,	solve	3.36	3.28
$k_n = 5\text{e}{-3}$	iterations	75	75
`cube_5`,	solve	30.97	31.34
$k_n = 2.5\text{e}{-3}$	iterations	83	83

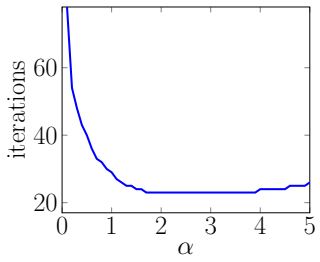
Since we observed that the Schur complement preconditioners with IC tend to be the cheapest of the considered options per application, we now discuss an option for the SIMPLE preconditioner (3.44) that reduces the required construction times. Motivated by the assumption that the change between saddle-point systems is small, we now compare results obtained by reusing the SIMPLE preconditioner instead of recomputing it. Table 5.8 compares iteration counts and solver times for reusing the IC factorization in the SIMPLE preconditioner to those obtained by recomputing the factorizations from scratch for the Oseen systems. The iteration counts for both approaches are very similar and the deviations in solver times are in the range of inaccuracy of time measurement. Regarding total computational times, it is therefore beneficial to reuse the precomputed factorizations. In the following, we will hence always reuse the `IC(0)` factorization computed for the initial Stokes systems for solving the following Oseen systems.

Table (5.9) summarizes the iterative solver and initial preconditioners used in the remaining numerical experiments.

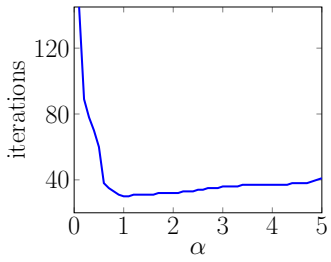
5.3 Relaxation parameter

In this section, we analyze whether relaxing the Schur complement preconditioner in the block triangular preconditioner (3.35) can accelerate the convergence of the outer iterative solver. As stated in [26], such scaling can accelerate convergence for the SIMPLE preconditioner. We are interested to see whether this can be observed for other preconditioner choices as well. For the numerical experiments in this section, we use the block preconditioner

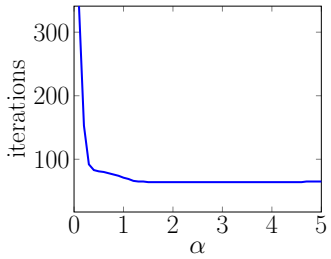
$$P_{\mathrm{U},\alpha} = \begin{pmatrix} \widehat{A}^{-1} & \alpha \widehat{A}^{-1} B^{\mathrm{T}} \widehat{S}^{-1} \\ 0 & -\alpha \widehat{S}^{-1} \end{pmatrix}. \tag{5.1}$$



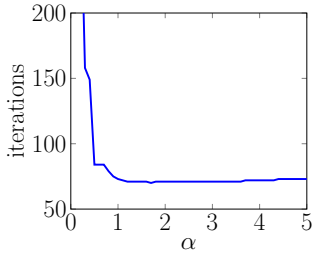
(A) Shell, SIMPLE.



(B) Shell, LSC.



(C) Cube, SIMPLE.



(D) Cube, LSC.

FIGURE 5.2: Influence of the relaxation parameter α on GMRes iteration counts for the *Oseen* systems on the shell ($n_u = 78438$, $n_p = 3474$, $k_n = 5\text{e}{-4}$) and the cube ($n_u = 107811$, $n_p = 4913$, $k_n = 5\text{e}{-3}$) with the block preconditioner $P_{\text{U},\alpha}$ (5.1).

We vary the relaxation parameter α for the Schur complement preconditioner in the block triangular preconditioner from 0.1 to 5 with step sizes 0.1. The chosen initial preconditioners $\widehat{A}^{-1}$ and $\widehat{S}^{-1}$ are given in Table 5.9. We test the influence of the relaxation parameter α for the Oseen and Stokes systems on a shell ($n_u = 78438$, $n_p = 3474$) with time step size $k_n = 5\text{e}{-4}$ and a cube ($n_u = 107811$, $n_p = 4913$) with time step size $k_n = 5\text{e}{-3}$.

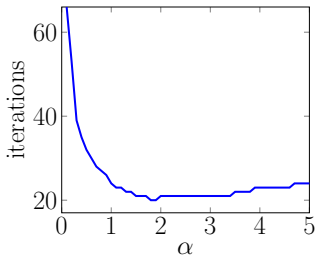
We observe in Figure 5.2 for the Oseen systems and in Figure 5.3 for the Stokes systems that the scaling has a significant influence on the required number of GMRes iterations. For the considered test systems, the optimal values for α regarding the number of GMRes iterations are given in Table 5.10. For all test systems, the optimal values for α are larger

TABLE 5.9: Solver and initial preconditioners for the numerical experiments if not stated otherwise.

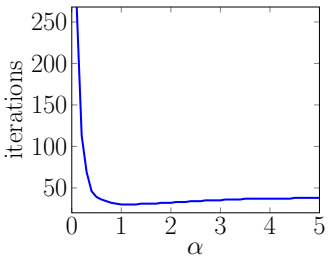
	description
solver	restarted GMRes with restart length $k = 40$
$P_{\text{U},\alpha}$	block triangular preconditioner (5.1) with relaxed Schur complement preconditioner
$\widehat{A}^{-1}$	block triangular preconditioner (3.38) with `ILU(0)` for all arising inverses
$\widehat{S}^{-1}$	LSC (3.43) or SIMPLE (3.44) with `IC(0)` to approximate the inner Poisson-type problems, where the `IC(0)` factorization is based on the first saddle-point systems, i.e. the Stokes systems

or equal to one. For values smaller than one, the iteration counts increase significantly. For larger scaling parameters ($\alpha \in [1, 5]$) the variation in iteration counts is small compared to the variation for values in the interval (0,1].

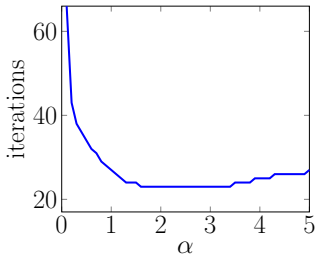
For the SIMPLE preconditioner, the observed optimal values for α differ from observations in the literature. In [57], the SIMPLE iteration is used as a smoother in a multigrid method. There, the scaling is restricted to damping, i.e. $\alpha \in (0, 1]$. Optimal damping was observed for values significantly smaller than one for the incompressible Navier-Stokes equations, tested for the lid-driven cavity benchmark on a unit square. As far as we know, there are no reference results for relaxing the LSC preconditioner.



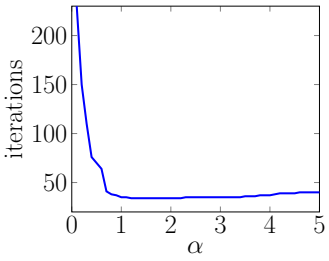
(A) Shell, SIMPLE.



(B) Shell, LSC.



(C) Cube, SIMPLE.



(D) Cube, LSC.

FIGURE 5.3: Influence of the relaxation parameter α on GMRes iteration counts for the *Stokes* systems on the shell ($n_u = 78438$, $n_p = 3474$, $k_n = 5\text{e}{-4}$) and the cube ($n_u = 107811$, $n_p = 4913$, $k_n = 5\text{e}{-3}$) with the block preconditioner $P_{\text{U},\alpha}$ (5.1).

TABLE 5.10: Optimal values for $\alpha \in \{0.1, 0.2, \ldots, 5.0\}$ regarding GMRes iteration counts obtained with the block preconditioner $P_{\mathrm{U},\alpha}$ (5.1) using a block triangular ILU preconditioner for $\widehat{A}^{-1}$ and the LSC and SIMPLE preconditioner for $\widehat{S}^{-1}$ with `IC(0)` for the inner Poisson-type problems.

	Oseen		Stokes	
system	SIMPLE	LSC	SIMPLE	LSC
`shell_3`	1.7–3.9	1.0–1.1	1.8–1.9	1.0–1.3
`cube_4`	1.5–4.6	1.7	1.6–3.4	1.2–2.2

5.4 Updates for relaxed Schur complement preconditioners

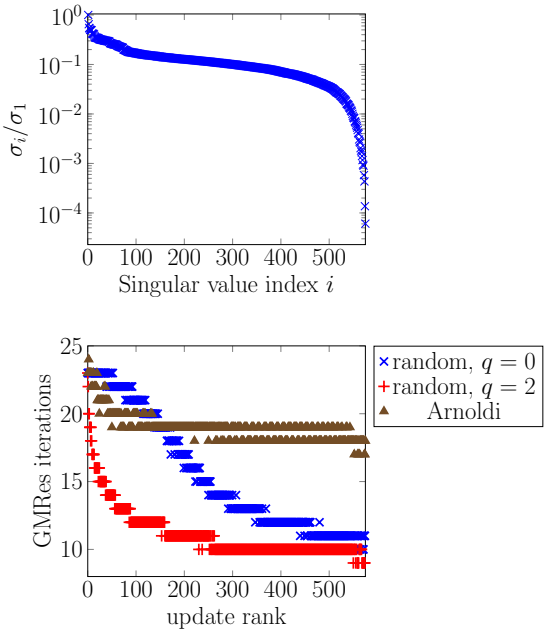
In this section, we investigate whether a low-rank update can further improve the relaxed Schur complement preconditioners. We focus on the low-rank updates that are based on a low-rank approximation of the difference between the identity matrix and the scaled right preconditioned approximate Schur complement (4.9) since the results for right and left updates are very similar. In Section 5.4.2, we nevertheless compare both update sides.

5.4.1 Update rank and low-rank property

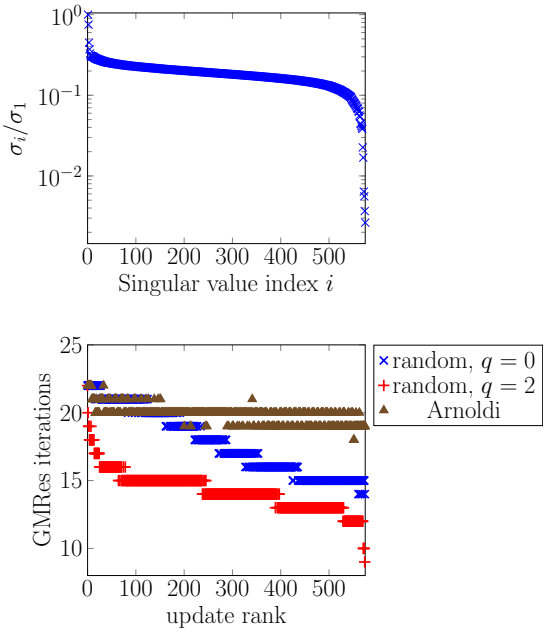
We now analyze the potential of the derived low-rank updates (4.11b) and (4.12b) by increasing the update rank from zero to the number of unconstrained pressure degrees of freedom which is the maximum reasonable update rank. Both low-rank updates are based on a low-rank approximation of the matrix $E_{\mathrm{R},\alpha}$ given in (4.9). With a best approximation of the matrix $E_{\mathrm{R},\alpha}$, the updated Schur complement preconditioners are ideal if the update rank equates the number of unconstrained pressure degrees of freedom. We are interested whether the considered low-rank approximations are suitable to approach the ideal Schur complement preconditioner.

We compare the low-rank updates, (4.11b) and (4.12b), which are constructed with three different low-rank approximations: the randomized Algorithm 2.2 with (a) $q = 0$ and (b) $q = 2$ power iterations, and (c) the Arnoldi iteration given in Algorithm 2.4. We use a block triangular ILU preconditioner for $\widehat{A}^{-1}$ and compare the update quality for the SIMPLE and the LSC preconditioner. For the Schur complement preconditioners, we approximate the inner Poisson-type problems with an inexact Cholesky factorization without fill-in. We set the relaxation parameter to $\alpha = 1.8$ for the SIMPLE preconditioner and to $\alpha = 1.2$ for the LSC preconditioner which turned out to be in the optimal range of relaxation parameters in numerical tests for the considered test problems.

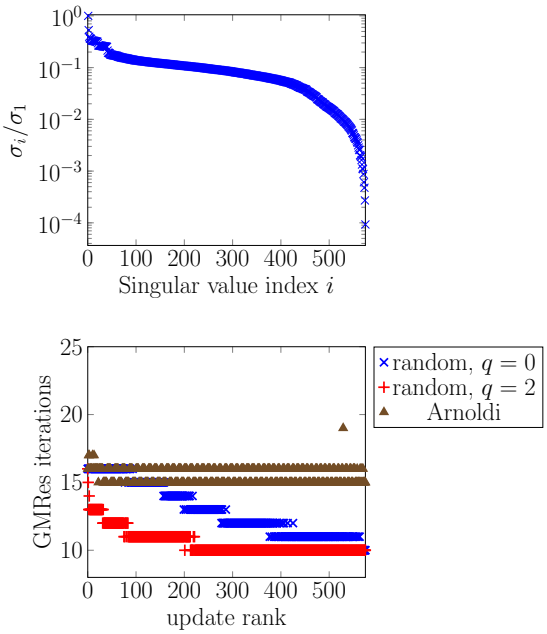
Figure 5.4 shows GMRes iteration counts and the decay of singular values of $E_{\mathrm{R},\alpha}$ in dependence on the update rank r for the Stokes and Oseen systems on a coarse cube ($n_u = 14739$, $n_p = 729$, $k_n = 5\mathrm{e}{-3}$). We observe that the quality of the low-rank update significantly depends on the underlying low-rank approximation. For the randomized approach with $q = 2$ power iterations, the smallest number of GMRes iterations that we achieve is ten. With using $q = 2$ power iterations, the trend in the reduction of GMRes iterations follows the decay of singular values. However, without any power iterations



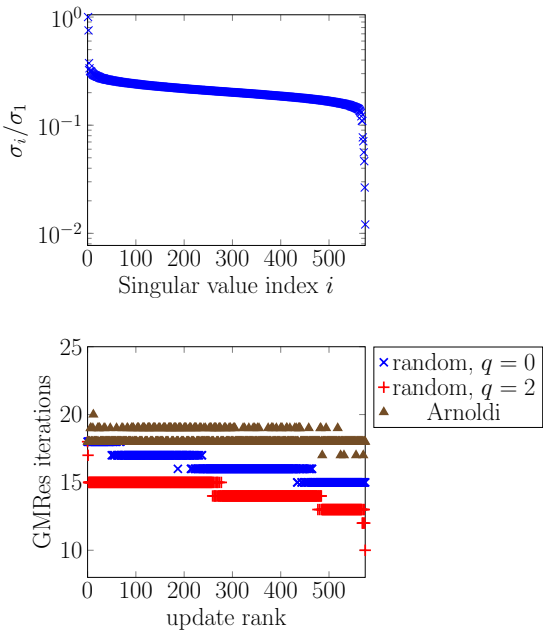
(A) Oseen, SIMPLE, $\alpha = 1.8$.



(B) Oseen, LSC, $\alpha = 1.2$.



(C) Stokes, SIMPLE, $\alpha = 1.8$.



(D) Stokes, LSC, $\alpha = 1.2$.

FIGURE 5.4: Decay of the singular values of $\widetilde{E}_{\mathrm{R},\alpha}$ from Equation (4.9) (upper images) and GMRes iteration counts (lower images) for increasing the update rank from $r = 0$ to $r = 575$ (the number of unconstrained pressure degrees of freedom) and for small *Oseen* and *Stokes* systems on the cube ($n_u = 14739$, $n_p = 729$, $k_n = 5\mathrm{e}{-3}$). The shown results are obtained with updates constructed with Algorithm 2.2 ("random") using $q = 0$ and $q = 2$ power iterations and Algorithm 2.4 ("Arnoldi").

or with using the Arnoldi iterations for the update construction, the iteration counts are clearly less reduced. For most cases, the Arnoldi iteration leads to the highest iteration counts. Only for the SIMPLE preconditioner for the Oseen systems with update ranks up to 145, the randomized approach without any power iterations leads to higher iteration counts. For the Arnoldi iteration, we observe that we cannot reduce the GMRes iterations as much as with the randomized low-rank approximations. Furthermore, we observed that reorthogonalization of the Arnoldi vectors is required especially for higher update ranks to avoid a loss of robustness.

5.4.2 Influence of the low-rank approximation

The preconditioners with low-rank updates from Section 4.3 are based on a low-rank approximation of the matrix

$$E_{\mathrm{R},\alpha} = \mathrm{I}_{n_p} - \alpha S \widehat{S}^{-1}$$

or

$$E_{\mathrm{L},\alpha} = \mathrm{I}_{n_p} - \alpha \widehat{S}^{-1} S.$$

In this section, we discuss the influence of the underlying low-rank approximation of both error matrices on the quality of the low-rank update. We compare the updated preconditioner (4.11) constructed with the randomized low-rank approximation from Algorithm 2.2 to the updated preconditioner (4.12) obtained with the Arnoldi iteration given in Algorithm 2.4. We furthermore consider low-rank updates (4.13) that are constructed with the Arnoldi iteration followed by projection (abbreviated as "Arnoldi+P" in the Tables 5.11 and 5.12).

We additionally discuss the influence of the parameters in the randomized low-rank approximation, the number of power iterations and oversampling vectors, on the quality of the update. For the randomized low-rank approximation, we expect a higher accuracy with increasing the number of power iterations, see Corollary 2.3, and thus hope that the number of GMRes iterations can be further reduced with the updated preconditioner. However, applying more power iterations leads to higher setup costs. The number of power iterations is therefore a trade-off of increasing setup costs and decreasing solver costs. We compare results obtained with $q = 0$ to $q = 3$ power iterations. Using a few oversampling vectors typically also improves the accuracy of the randomized low-rank approximation. However, oversampling requires (expensive) additional multiplications with the error matrix or its transpose. We have observed that the additional construction costs are not profitable since oversampling has only a negligible impact on convergence obtained with the updated preconditioners. Therefore, we set the oversampling parameter to $k_r = 0$ in the following experiments.

We compare the updated preconditioners for an update rank of $r = 20$. The chosen values for the relaxation parameter α are in the range of optimal values in terms of iteration counts given in Table 5.10. For the updates that are computed with a randomized low-rank approximation, we show the median and standard deviation obtained from 200 test runs. We use the same test systems and initial preconditioners $\widehat{A}^{-1}$ as in the previous section. For the initial Schur complement preconditioner $\widehat{S}^{-1}$, we use the LSC and SIMPLE preconditioner where we approximate the inner Poisson-type problems with an incomplete Cholesky factorization without fill-in. For comparison, we also show results

obtained without any update using the same relaxation parameter as for the updated preconditioners.

Table 5.11 shows the results obtained for the Oseen systems on the shell ($n_u = 78438$, $n_p = 3474$, $k_n = 5\text{e}{-4}$) and the cube ($n_u = 107811$, $n_p = 4913$, $k_n = 5\text{e}{-3}$). We observe that the setup costs increase significantly with the number of power iterations. The setup costs for the updates computed with the Arnoldi iteration are slightly lower than those for the update constructed with the randomized method without any power iteration. For the Arnoldi iteration, we only require about half of the applications of the error matrix compared to the randomized method without any power iterations but the error matrix is applied to single vectors and not to block vectors as done in the randomized method. The setup costs for the Arnoldi method followed by projection lie between the costs with the randomized method with $q = 0$ and $q = 1$ power iterations. We observe that both update

TABLE 5.11: Results for varying the number of power iterations q with rank $r = 20$, oversampling parameter $k_r = 0$, for the *Oseen* systems on the shell ($n_u = 78438$, $n_p = 3474$, $k_n = 5\text{e}{-4}$) and the cube ($n_u = 107811$, $n_p = 4913$, $k_n = 5\text{e}{-3}$). The needed GMRes iterations and computational times in seconds are given as the median (standard deviation) obtained from 200 test runs.

(A) Shell, LSC, $\alpha = 1.1$.

q	iters.	setup (update)	solve
Right update			
0	33 (0.39)	0.60 (1.8e−3)	1.58 (2.0e−2)
1	26 (0.48)	1.17 (3.8e−3)	1.24 (2.4e−2)
2	25 (0.31)	1.75 (6.0e−3)	1.19 (1.5e−2)
3	25 (0.49)	2.32 (1.8e−2)	1.18 (2.4e−2)
Arnoldi	30	0.59	1.42
Arnoldi+P	24	0.89	1.11
Left update			
0	31 (0.51)	0.60 (2.1e−3)	1.49 (2.5e−2)
1	24 (0.48)	1.18 (2.0e−3)	1.15 (2.3e−2)
2	24 (0.48)	1.77 (4.3e−3)	1.15 (2.3e−2)
3	24 (0.49)	2.32 (1.3e−2)	1.13 (2.3e−2)
Arnoldi	27	0.57	1.25
Arnoldi+P	24	0.92	1.14
no update	30	0	1.41

(B) Cube, LSC, $\alpha = 1.7$.

q	iters.	setup (update)	solve
Right update			
0	70 (0.38)	0.92 (1.2e−2)	4.65 (5.7e−2)
1	70 (2.53)	1.84 (2.3e−2)	4.66 (0.17)
2	68 (4.25)	2.79 (7.1e−2)	4.56 (0.30)
3	65 (5.66)	3.67 (6.9e−2)	4.30 (0.37)
Arnoldi	65	0.82	4.34
Arnoldi+P	42	1.32	2.87
Left update			
0	71 (0.48)	0.92 (1.7e−3)	4.64 (3.2e−2)
1	70 (0.41)	1.82 (4.4e−3)	4.58 (2.7e−2)
2	69 (7.77)	2.73 (5.5e−3)	4.51 (0.50)
3	66 (4.16)	3.61 (1.3e−2)	4.28 (0.26)
Arnoldi	66	0.79	4.30
Arnoldi+P	62	1.40	4.45
no update	70	0	4.66

(C) Shell, SIMPLE, $\alpha = 1.9$.

q	iters.	setup (update)	solve
Right update			
0	24 (0.31)	0.26 (1.2e−3)	0.73 (9.5e−3)
1	22 (0.31)	0.51 (1.8e−3)	0.67 (9.6e−3)
2	21 (0.50)	0.76 (3.0e−3)	0.65 (1.5e−2)
3	21 (0.39)	1.01 (3.6e−3)	0.64 (1.2e−2)
Arnoldi	24	0.26	0.73
Arnoldi+P	20	0.40	0.60
Left update			
0	24 (0.45)	0.26 (1.2e−3)	0.73 (1.4e−2)
1	21 (0.50)	0.52 (1.6e−3)	0.64 (1.5e−2)
2	20 (0.44)	0.77 (1.9e−3)	0.61 (1.3e−2)
3	20 (0.43)	1.03 (2.4e−3)	0.61 (1.3e−2)
Arnoldi	22	0.26	0.66
Arnoldi+P	20	0.41	0.61
no update	23	0	0.70

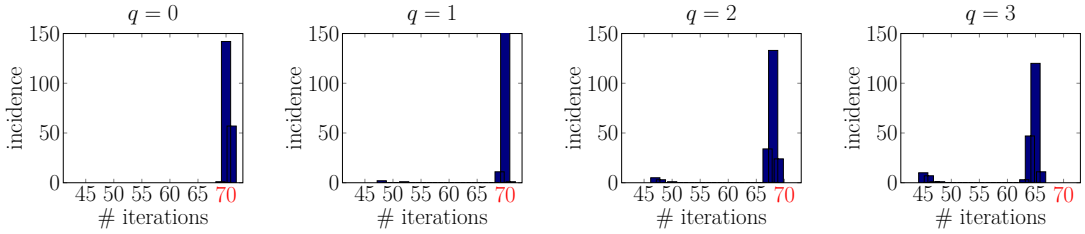
(D) Cube, SIMPLE, $\alpha = 1.9$.

q	iters.	setup (update)	solve
Right update			
0	71 (0.75)	0.40 (1.1e−2)	3.12 (6.3e−2)
1	77 (2.23)	0.78 (1.9e−2)	3.37 (0.13)
2	71 (2.26)	1.19 (3.1e−2)	3.10 (0.12)
3	64 (2.41)	1.59 (4.7e−2)	2.82 (0.12)
Arnoldi	60	0.36	2.60
Arnoldi+P	61	0.54	2.66
Left update			
0	71 (0.78)	0.40 (1.8e−3)	3.04 (3.6e−2)
1	79 (1.67)	0.79 (3.2e−3)	3.42 (7.2e−2)
2	72 (1.63)	1.18 (3.8e−3)	3.09 (7.4e−2)
3	66 (1.73)	1.57 (4.7e−3)	2.82 (7.6e−2)
Arnoldi	61	0.36	2.58
Arnoldi+P	65	0.62	2.95
no update	64	0	2.74

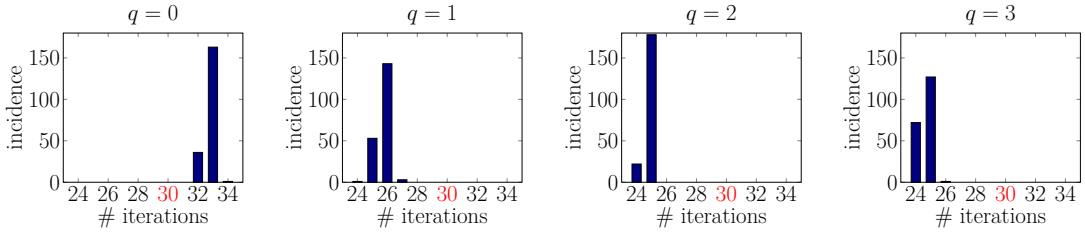
sides lead to very similar iteration counts. The left updates tend to result in a slightly higher reduction of iterations for the shell systems and a slightly lower reduction of iterations for the cube systems than the right updates. The highest reduction in iteration counts is obtained by the Arnoldi iteration followed by projection, except for the updated SIMPLE preconditioner on the cube systems where the Arnoldi iteration without projection leads to (slightly) better results. For the shell systems, the update computed with the Arnoldi iteration does not reduce iteration counts for the LSC preconditioner and increases the required number of iterations by one for the SIMPLE preconditioner. For the randomized approach used for the shell systems, we observe that the number of GMRes iterations increases with an update with $q = 0$ compared to no update. Increasing the number of power iterations to $q = 1$ leads to a reduction in GMRes iterations but increasing the power iterations by more than $q = 1$ does not seem to be beneficial. The GMRes iteration counts are only reduced by one for $q = 2$ and $q = 3$ compared to $q = 1$. For the randomized approach on the cube systems, we observe that increasing the number of power iterations decreases the number of required GMRes iterations for the LSC preconditioner. However, for the SIMPLE preconditioner on the cube, the randomized approach with $q = 0$ to $q = 2$ power iterations leads to an increase in iterations. For $q = 3$ power iterations, the number of GMRes iterations is the same as obtained without any update. However, for the cube systems, the update based on the Arnoldi iteration reduces iteration counts with significantly lower setup costs compared to the update based on the randomized approach. We furthermore observe that the standard deviations of iteration counts are significantly higher for the cube than for the shell.

To gain an insight into the reasons for the large standard deviations for the Oseen systems on the cube, we furthermore illustrate the distribution of iteration counts within the 200 test runs for the right updates in Figure 5.5. We observe that with increasing the number of power iterations, the distribution of GMRes iteration counts is shifted towards smaller iteration counts. Only for the SIMPLE preconditioner on the cube systems the number of iterations increases from $q = 0$ to $q = 1$ power iterations. For the LSC preconditioner on the cube, we observe that with increasing the number of power iterations, a second cluster of iteration counts builds up with only around 45 iterations instead of around 60-61 iterations. This explains the large standard deviation in iteration counts. Applying more power iterations thus might reduce iteration counts further. Figure 5.6 shows the distribution of iteration counts obtained with 4 and 5 power iterations. We observe that iteration counts of 44–45 get significantly more likely. For the SIMPLE preconditioner, the number of GMRes iterations is now reduced by the update but with the cost of high construction times.

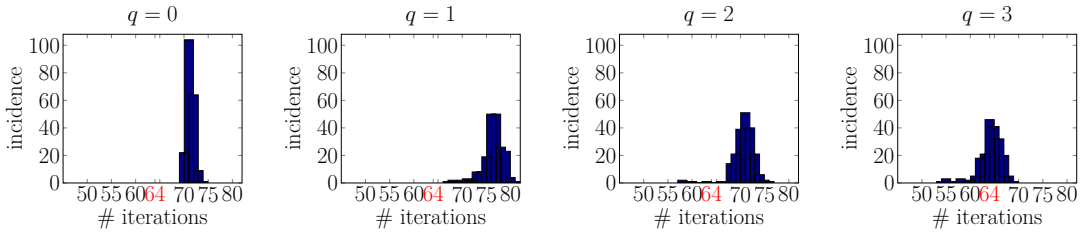
Table 5.12 shows results obtained for the Stokes systems on a cube and a spherical shell. As for the Oseen systems, the setup costs for the updates constructed with the randomized method are higher than those for the update computed with the Arnoldi iteration and increase with the number of power iterations. The setup costs for updates constructed with the Arnoldi method followed by projection are between the costs with $q = 0$ and $q = 1$ power iterations. Both update sides lead again to very similar results. Left updates tend to result in a slightly higher reduction of iterations for the shell systems than the right updates. For the cube systems, the left updates tend to lead to a slightly smaller reduction in iteration counts than the right updates. The update constructed with the Arnoldi iteration decreases iteration counts for the cube but increases iteration counts



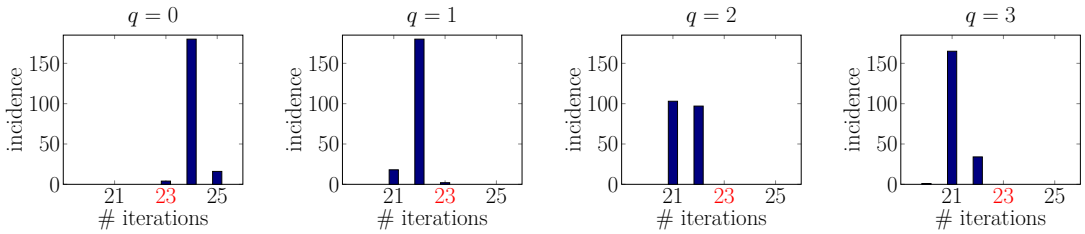
(A) Cube, LSC, $\alpha = 1.6$.



(B) Shell, LSC, $\alpha = 1.1$.

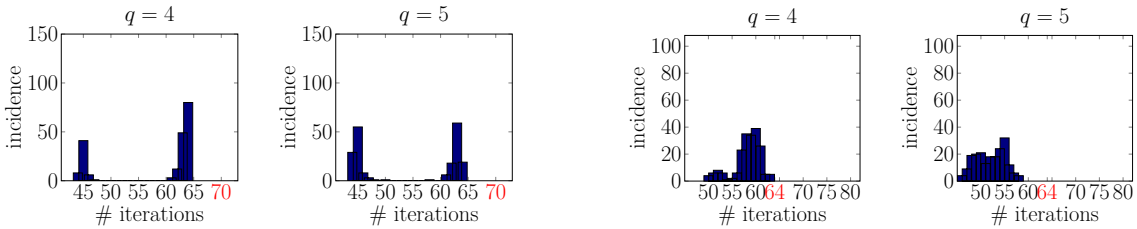


(C) Cube, SIMPLE, $\alpha = 1.9$.



(D) Shell, SIMPLE, $\alpha = 1.9$.

FIGURE 5.5: Distribution of GMRes iterations obtained for 200 test runs for the *Oseen* systems (shell with $n_u = 78438$, $n_p = 3474$, $k_n = 5\text{e}{-4}$, cube with $n_u = 107811$, $n_p = 4913$, $k_n = 5\text{e}{-3}$) with the updated preconditioner (4.11b) with update rank $r = 20$ constructed with a randomized low-rank approximation with $q = 0$ to $q = 3$ power iterations.



(A) Cube, LSC, $\alpha = 1.6$. (B) Cube, SIMPLE, $\alpha = 1.9$.

FIGURE 5.6: Distribution of GMRes iterations obtained for 200 test runs for the *Oseen* systems (cube with $n_u = 107811$, $n_p = 4913$, $k_n = 5\text{e}{-3}$) with the updated preconditioner (4.11b) with update rank $r = 20$ constructed with a randomized low-rank approximation with $q = 4$ to $q = 5$ power iterations.

TABLE 5.12: Results for varying the number of power iterations q with rank $r = 20$, oversampling parameter $k_r = 0$, for the *Stokes* systems on the shell ($n_u = 78438$, $n_p = 3474$, $k_n = 5\text{e}{-4}$) and the cube ($n_u = 107811$, $n_p = 4913$, $k_n = 5\text{e}{-3}$). The needed GMRes iterations and computational times in seconds are given as the median (standard deviation) obtained from 200 test runs.

(A) Shell, LSC, $\alpha = 1.2$.

q	iters.	setup (update)	solve
Right update			
0	31 (0.17)	0.59 (5.7e−3)	1.48 (1.0e−2)
1	26 (0.40)	1.16 (8.5e−3)	1.23 (1.9e−2)
2	25 (0.12)	1.73 (1.2e−2)	1.19 (7.7e−3)
3	25 (0.26)	2.31 (1.4e−2)	1.19 (1.3e−2)
Arnoldi	26	0.58	1.23
Arnoldi+P	22	0.91	1.03
Left update			
0	31 (0.48)	0.59 (1.6e−3)	1.48 (2.3e−2)
1	25 (0.45)	1.18 (2.6e−3)	1.19 (2.2e−2)
2	24 (0.50)	1.76 (3.6e−3)	1.15 (2.4e−2)
3	24 (0.44)	2.35 (6.1e−3)	1.15 (2.1e−2)
Arnoldi	24	0.57	1.11
Arnoldi+P	21	0.92	0.99
no update	30	0	1.45

(B) Cube, LSC, $\alpha = 1.2$.

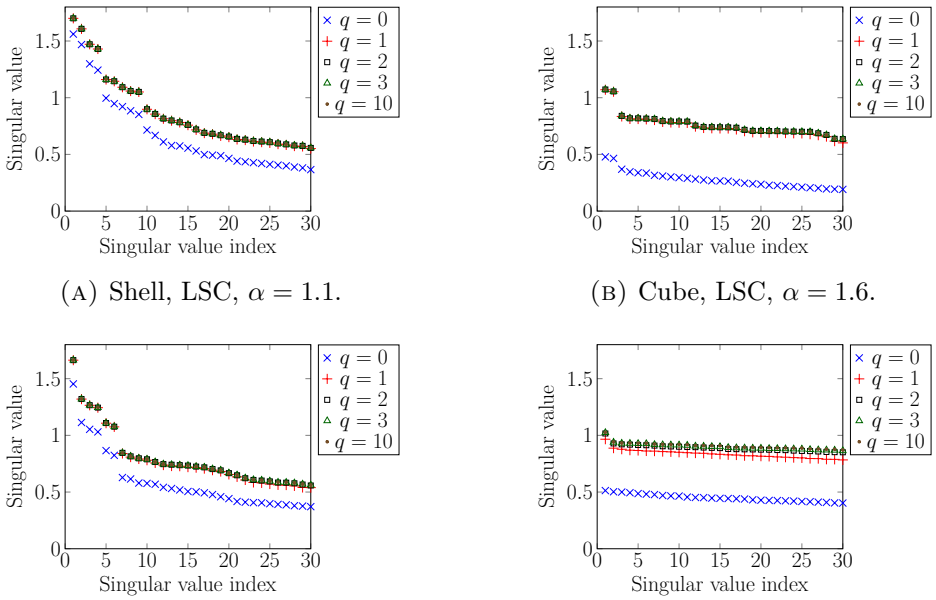
q	iters.	setup (update)	solve
Right update			
0	34 (0.47)	0.94 (2.4e−2)	2.30 (5.5e−2)
1	25 (0.44)	1.84 (3.9e−2)	1.65 (4.2e−2)
2	21 (0.48)	2.80 (5.9e−2)	1.39 (3.9e−2)
3	20 (0.12)	3.78 (8.5e−2)	1.36 (3.6e−2)
Arnoldi	29	0.80	1.92
Arnoldi+P	26	1.31	1.72
Left update			
0	34 (0.42)	0.92 (6.2e−3)	2.34 (2.9e−2)
1	26 (0.46)	1.86 (8.6e−3)	1.75 (3.2e−2)
2	22 (0.47)	2.77 (1.4e−2)	1.48 (3.2e−2)
3	21 (0.12)	3.70 (1.5e−2)	1.41 (9.7e−3)
Arnoldi	30	0.79	1.95
Arnoldi+P	26	1.34	1.78
no update	34	0	2.25

(C) Shell, SIMPLE, $\alpha = 1.9$.

q	iters.	setup (update)	solve
Right update			
0	21 (0.51)	0.26 (5.4e−3)	0.64 (1.6e−2)
1	19 (7.1e−2)	0.50 (8.5e−3)	0.58 (3.9e−3)
2	19 (0.12)	0.75 (1.1e−2)	0.58 (5.4e−3)
3	19 (0.10)	1.00 (1.1e−2)	0.58 (4.9e−3)
Arnoldi	21	0.27	0.64
Arnoldi+P	18	0.41	0.55
Left update			
0	22 (0.47)	0.26 (1.7e−3)	0.67 (1.5e−2)
1	19 (0.24)	0.51 (2.1e−3)	0.58 (7.7e−3)
2	19 (0.47)	0.76 (1.9e−3)	0.58 (1.5e−2)
3	19 (0.34)	1.01 (2.6e−3)	0.58 (1.0e−2)
Arnoldi	18	0.26	0.53
Arnoldi+P	17	0.41	0.52
no update	20	0	0.63

(D) Cube, SIMPLE, $\alpha = 1.9$.

q	iters.	setup (update)	solve
Right update			
0	23 (0)	0.40 (1.4e−2)	0.99 (1.9e−2)
1	20 (0.42)	0.79 (2.6e−2)	0.85 (2.9e−2)
2	18 (0.21)	1.16 (3.5e−2)	0.77 (2.2e−2)
3	18 (0.45)	1.57 (3.8e−2)	0.76 (2.7e−2)
Arnoldi	21	0.36	0.90
Arnoldi+P	19	0.58	0.81
Left update			
0	23 (0)	0.40 (5.0e−3)	1.00 (4.7e−3)
1	21 (0.19)	0.79 (7.7e−3)	0.91 (9.5e−3)
2	20 (0.25)	1.16 (9.5e−3)	0.87 (1.2e−2)
3	20 (0.48)	1.54 (1.3e−2)	0.86 (2.1e−2)
Arnoldi	22	0.36	0.92
Arnoldi+P	20	0.61	0.88
no update	23	0	0.97



(A) Shell, LSC, $\alpha = 1.1$.

(B) Cube, LSC, $\alpha = 1.6$.

(C) Shell, SIMPLE, $\alpha = 1.9$.

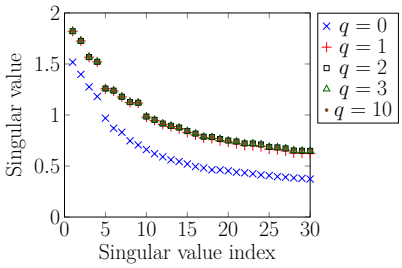
(D) Cube, SIMPLE, $\alpha = 1.9$.

FIGURE 5.7: Approximate 30 largest singular values of $\widetilde{E}_{\mathrm{R},\alpha}$ (4.9) computed with the randomized singular value decomposition with $q = 0$ to $q = 3$ and $q = 10$ power iterations obtained for the *Oseen* systems on the cube ($n_u = 107811$, $n_p = 4913$, $k_n = 5\mathrm{e}{-3}$) and the shell ($n_u = 78438$, $n_p = 3474$, $k_n = 5\mathrm{e}{-4}$).

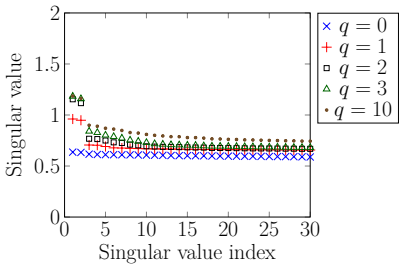
for the shell. Updates based on Arnoldi vectors with additional projection using the error matrix result in lower iteration counts than updates without projection. The updates based on Arnoldi vectors with projection lead to the highest reduction in iterations for the shell systems. For the cube systems, the obtained iteration counts are similar to those obtained with the randomized method with $q = 1$ power iterations. For the randomized approach, we observe that increasing the number of power iterations from $q = 0$ to $q = 3$ reduces the number of GMRes iterations. For the shell systems, the randomized approach increases iteration counts for the LSC preconditioner. For the SIMPLE preconditioner, the iterations increase slightly for $q = 0$ and decrease slightly for $q = 1$ to $q = 3$. For the cube systems, the iteration counts decrease for $q = 1$ to $q = 2$ power iterations. The results for $q = 2$ and $q = 3$ power iterations only differ slightly. We furthermore observe that the standard deviations of the iteration counts are lower than those obtained for the Oseen systems.

To explain the not entirely satisfying results, we additionally show the 30 largest approximate singular values of the error matrix $\widetilde{E}_{\mathrm{R},\alpha}$ computed with the randomized singular value decomposition [34, 51] for varying the number of power iterations. We refer to Section 4.2 for a description of the randomized singular value decomposition.

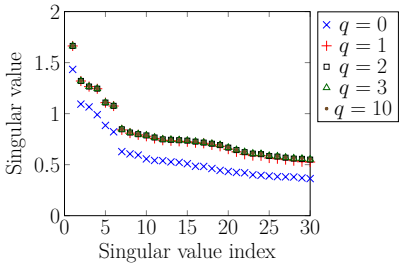
Figure 5.7 shows the largest 30 approximate singular values of the error matrix $\widetilde{E}_{\mathrm{R},\alpha}$ computed with the randomized singular value decomposition for varying the number of power iterations from $q = 0$ to $q = 3$ and for $q = 10$. For the shell systems, we observe that the singular values obtained with $q = 0$ power iterations are significantly smaller than the singular values obtained with $q = 1$ to $q = 3$ and $q = 10$ power iterations. Increasing the



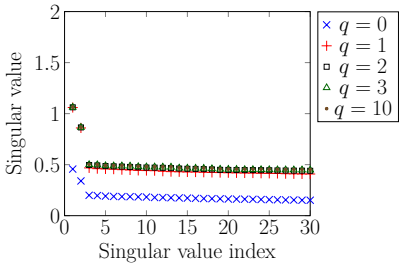
(A) Shell, LSC, $\alpha = 1.1$.



(B) Cube, LSC, $\alpha = 1.6$.



(C) Shell, SIMPLE, $\alpha = 1.9$.



(D) Cube, SIMPLE, $\alpha = 1.9$.

FIGURE 5.8: Approximate 30 largest singular values of $\widetilde{E}_{\mathrm{R},\alpha}$ (4.9) computed with the randomized singular value decomposition with $q = 0$ to $q = 3$ and $q = 10$ power iterations obtained for the *Stokes* systems on the cube ($n_u = 107811$, $n_p = 4913$, $k_n = 5\mathrm{e}{-3}$) and the shell ($n_u = 78438$, $n_p = 3474$, $k_n = 5\mathrm{e}{-4}$).

number of power iterations from $q = 1$ to $q = 3$ and $q = 10$, the approximate singular values are close to each other. For the cube system with the SIMPLE preconditioner, the results look similar. However, for the cube system with the LSC preconditioner, the approximate singular values obtained with $q = 0$ to $q = 3$ and $q = 10$ power iterations differ. Only the two largest singular values are close for $q = 2$ to $q = 3$ and $q = 10$ power iterations. We furthermore observe that the singular values decay faster for the shell systems than for the cube systems.

Figure 5.8 illustrates the largest 30 approximate singular values obtained with $q = 0$ to $q = 3$ and $q = 10$ power iterations for the Stokes systems. For all systems, the results for $q = 0$ power iterations differ significantly from those obtained with more power iterations. Increasing the number of power iterations from one to ten only shows slight differences in the singular values. We furthermore observe that the singular values decay faster for the Stokes systems than for the Oseen systems.

5.4.3 Rank and relaxation

Now we investigate the quality of the update scheme regarding iteration counts and computational times for varying the update rank r and the relaxation parameter α. We vary the update rank r from 0 to 120 with step size 5 and the relaxation parameter α from 0.1 to 2.9 with step size 0.1. We present the results for the iteration counts as matrix plots where white areas refer to the results obtained without any update and without relaxing the Schur complement preconditioner. Red areas denote an increase in iterations and green

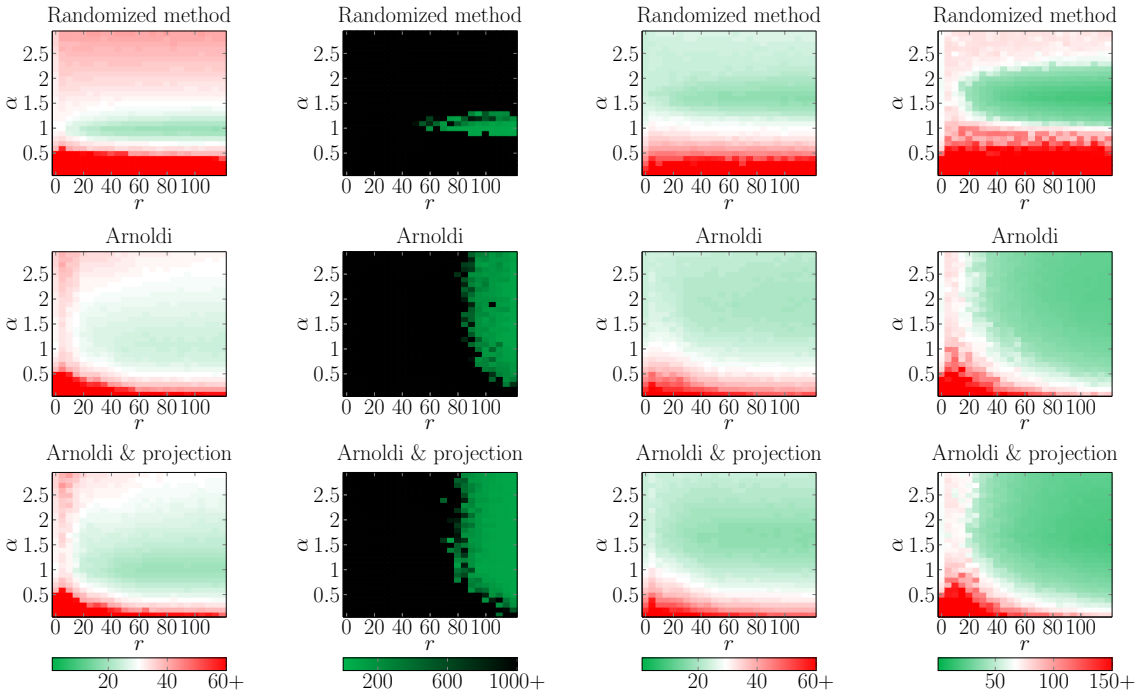
areas denote a reduction in iterations. Black areas correspond to parameter sets that did not lead to convergence within 1000 iterations.

Figure 5.9 shows iteration counts for the updated SIMPLE and LSC preconditioner, both tested for the Oseen systems on the shell and cube with two different refinement levels. The system dimensions and time step sizes are given in the caption. We observe a significant dependence on the relaxation parameter. However, the "optimal" values for the parameter α may be different from those obtained for the preconditioners without any update. We observe that the dependence on the relaxation parameter is very similar for the different refinement levels, at least in the considered test setting. For the updated LSC preconditioner, a relaxation parameter of around $\alpha = 1.1$ seems to be beneficial for the randomized update for both test systems. For the updated SIMPLE preconditioner, a relaxation parameter of around 1.6–1.7 seems to be beneficial. Convergence depends less on the relaxation parameter for $\alpha > 1$ for the updates based on the Arnoldi iteration (with and without projection) than in the randomized updates. It is interesting to observe that the updates first increase iteration counts with smaller ranks before they lead to a reduction in GMRes iterations.

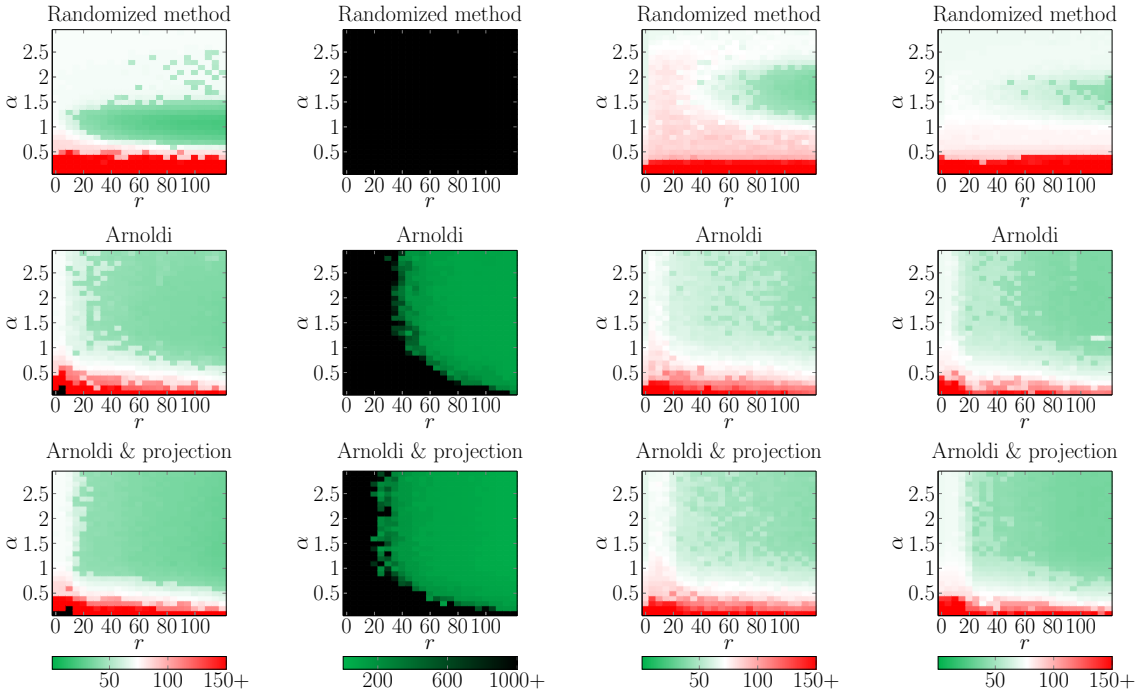
Figure 5.10 shows iteration counts for the Stokes systems. For the LSC preconditioner, the optimal value for α is around 1.0 and for the SIMPLE preconditioner, it is around 1.7. For the update versions based on the Arnoldi iteration (with and without projection), convergence depends less on the relaxation parameter for $\alpha > 1$ than for the updates constructed with the randomized approach. For the coarser systems, we observe a higher reduction of iteration counts with the randomized approach. This looks different for the finer systems with LSC preconditioner. For the cube, the randomized approach does not lead to convergence within 1000 iterations for an update rank up to 120. For the shell, convergence within 1000 iterations is only observed for a narrow range of relaxation parameters but with a smaller update rank than with the Arnoldi iteration. We observe that we require reorthogonalization steps in the update construction with the Arnoldi iteration especially for small relaxation parameters ($\alpha < 1$) and for increasing the update rank. Additional projection leads to a higher reduction in iteration counts for the updates based on the Arnoldi method.

Table 5.13 displays iteration counts and computational times obtained with relaxation parameters that are close to "optimal" for a selection of update ranks. Note that the shown results for $r = 0$ may differ due to variations in the relaxation parameter. We observe that the updates reduce iteration counts for sufficiently large update ranks. However, the construction of the update introduces large additional times. The additional application costs of the low-rank updates are small compared to the averaged costs of one iteration of the restarted GMRes method as shown in Section 4.6.2. So, reduced iteration counts typically also lead to reduced solver times. However, the low-rank updates do not reduce total computational times for most of the systems. This may be somewhat different for a parallelized implementation of the methods. Although most of the shown total times are not reduced, this may differ for sequences of saddle-point systems where we can reuse update vectors. The bottleneck concerning setup times for the update is the application of the Schur complement approximation which requires an approximation to the inverse A^{-1}.

In the next Section 5.5, we discuss inner low-rank corrections for the considered Schur complement preconditioners that avoid the application of a Schur complement approxima-

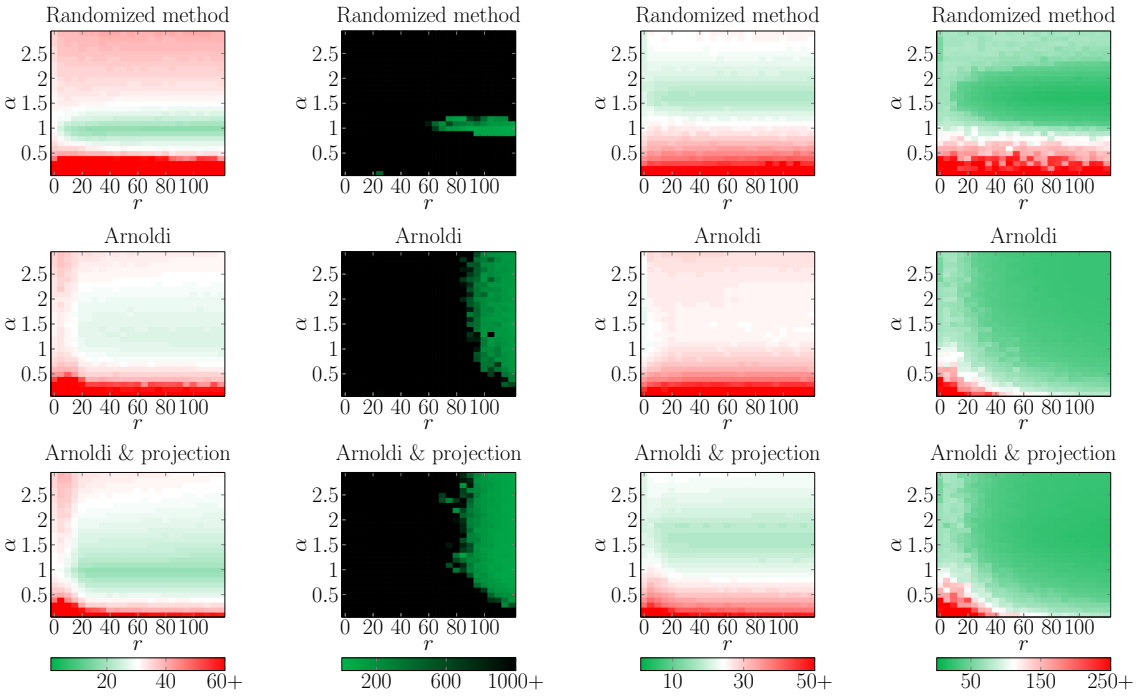


(A) Shell, LSC. (B) Shell, SIMPLE.

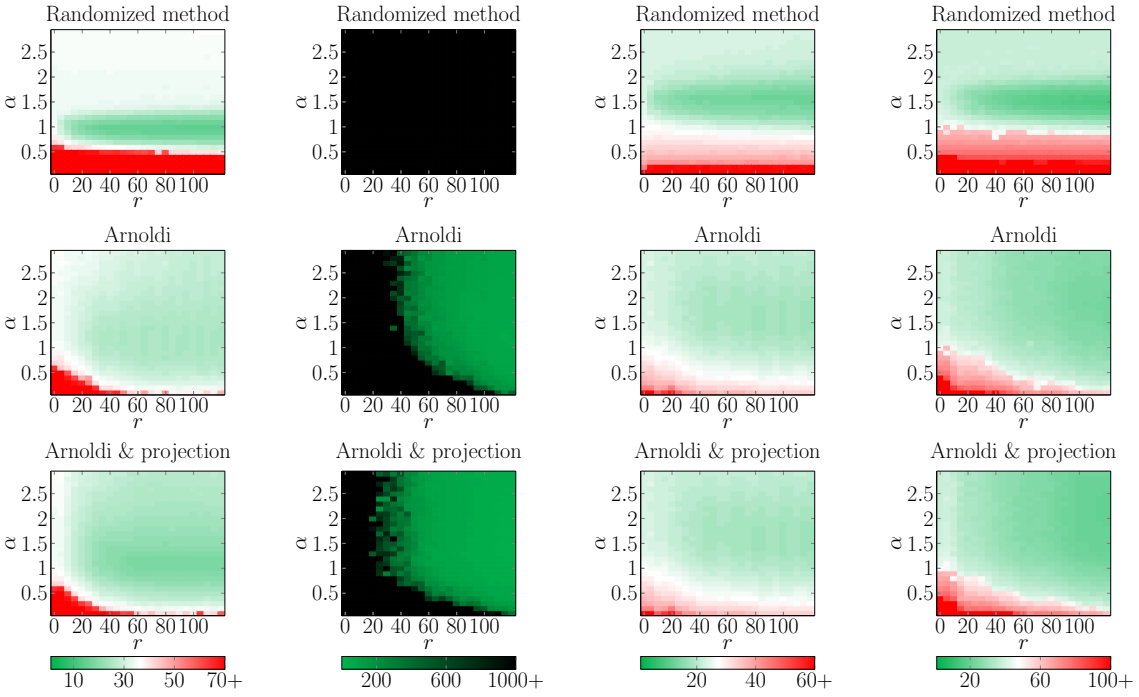


(C) Cube, LSC. (D) Cube, SIMPLE.

FIGURE 5.9: GMRes iteration counts for varying the update rank r and relaxation parameter α. The low-rank updates are constructed with the randomized Algorithm 2.2 ($q = 1$), the Arnoldi Algorithm 2.4 (with and without projection). The results are obtained for the *Oseen* systems with two refinement levels (left column of each subfigure: `shell_3` or `cube_4`, right column of each subfigure: `shell_4` or `cube_5`, see Table 5.1).



(A) Shell, LSC. (B) Shell, SIMPLE.



(C) Cube, LSC. (D) Cube, SIMPLE.

FIGURE 5.10: GMRes iteration counts for varying the update rank r and relaxation parameter α. The low-rank updates are constructed with the randomized Algorithm 2.2 ($q = 1$), the Arnoldi Algorithm 2.4 (with and without projection). The results are obtained for the *Stokes* systems with two refinement levels (left column of each subfigure: `shell_3` or `cube_4`, right column of each subfigure: `shell_4` or `cube_5`, see Table 5.1).

tion and thus lead to significantly smaller construction times.

TABLE 5.13: GMRes iteration counts as well as update construction and solver times (both in seconds) for different update ranks and low-rank approximation methods. The results are obtained for the *Oseen* systems for the shell and the cube, each with two refinement levels.

SIMPLE, random ($q = 1$)						SIMPLE, Arnoldi						LSC, random ($q = 1$)						LSC, Arnoldi					
α	r	iters.	setup	solve	sum	α	r	iters.	setup	solve	sum	α	r	iters.	setup	solve	sum	α	r	iters.	setup	solve	sum
Shell ($n_u = 78438$, $n_p = 3474$, $k_n =$ 5e−4), Oseen.																							
1.7	0	23	0	0.69	**0.69**	1.7	0	23	0	0.69	**0.69**	1.0	0	30	0	1.39	**1.39**	1.0	0	30	0	1.39	**1.39**
1.7	20	20	0.51	0.60	1.11	1.7	20	23	0.26	0.69	0.95	1.0	20	23	1.16	1.07	2.23	1.0	20	29	0.57	1.35	1.92
1.7	40	18	1.02	0.54	1.56	1.7	40	21	0.52	0.63	1.15	1.0	40	20	2.31	0.94	3.25	1.0	40	26	1.15	1.21	2.36
1.7	60	17	1.51	0.51	2.02	1.7	60	20	0.79	0.61	1.40	1.0	60	19	3.45	0.88	4.33	1.0	60	24	1.73	1.12	2.85
1.7	80	17	2.03	0.51	2.54	1.7	80	20	1.05	0.61	1.66	1.0	80	19	4.64	0.90	5.54	1.0	80	24	2.30	1.12	3.42
Shell ($n_u = 608454$, $n_p = 26146$, $k_n =$ 2.5e−4), Oseen.																							
1.6	0	64	0	16.65	16.65	2.0	0	63	0	16.08	16.08	1.0	0	nc				1.0	0	nc			
1.6	20	41	4.96	10.92	**15.88**	2.0	20	55	2.15	14.12	16.27	1.0	20	nc				1.0	20	nc			
1.6	40	29	9.97	7.54	17.51	2.0	40	38	4.31	9.81	**14.12**	1.0	40	nc				1.0	40	nc			
1.6	60	23	15.57	5.97	21.54	2.0	60	32	6.48	8.20	14.68	1.0	60	201	35.78	79.69	115.47	1.0	60	nc			
1.6	80	21	20.89	5.50	26.39	2.0	80	28	8.67	7.20	15.87	1.0	80	77	47.92	30.84	**78.76**	1.0	80	nc			
SIMPLE, random ($q = 1$)						SIMPLE, Arnoldi						LSC, random ($q = 1$)						LSC, Arnoldi					
α	r	iters.	setup	solve	sum	α	r	iters.	setup	solve	sum	α	r	iters.	setup	solve	sum	α	r	iters.	setup	solve	sum
Cube ($n_u = 107811$, $n_p = 4913$, $k_n =$ 5e-3), Oseen.																							
1.7	0	64	0	2.72	**2.72**	1.9	0	64	0	2.74	**2.74**	1.1	0	72	0	4.72	4.72	1.3	0	71	0	4.72	4.72
1.7	20	78	0.80	3.36	4.16	1.9	20	60	0.36	2.60	2.96	1.1	20	40	1.86	2.67	**4.53**	1.3	20	61	0.81	4.04	4.85
1.7	40	69	1.55	2.94	4.48	1.9	40	58	0.73	2.51	3.24	1.1	40	33	3.60	2.14	5.74	1.3	40	42	1.60	2.84	**4.44**
1.7	60	61	2.37	2.61	4.98	1.9	60	47	1.09	2.05	3.14	1.1	60	29	5.38	1.88	7.26	1.3	60	38	2.42	2.54	4.96
1.7	80	53	3.09	2.27	5.36	1.9	80	54	1.46	2.33	3.79	1.1	80	25	7.18	1.62	8.81	1.3	80	37	3.22	2.48	5.70
Cube ($n_u = 823875$, $n_p = 35937$, $k_n =$ 2.5e-3), Oseen.																							
1.7	0	71	0	26.14	**26.14**	1.7	0	71	0	26.12	26.12	1.1	0	nc				1.5	0	nc			
1.7	20	71	7.59	26.24	33.83	1.7	20	56	3.05	20.58	**23.62**	1.1	20	nc				1.5	20	nc			
1.7	40	68	16.05	25.42	41.46	1.7	40	53	6.12	19.45	25.58	1.1	40	nc				1.5	40	912	14.33	533.39	547.72
1.7	60	65	24.00	24.32	48.32	1.7	60	52	9.20	19.26	28.45	1.1	60	nc				1.5	60	113	21.49	66.33	87.82
1.7	80	61	32.19	22.86	55.05	1.7	80	40	12.29	15.23	27.52	1.1	80	nc				1.5	80	79	28.56	46.81	**75.38**

5.5 Schur complement preconditioners with inner updates

We now analyze the inner low-rank updates defined in (4.22), (4.24), (4.27), and (4.29) numerically with similar experiments as done in the preceding Section 5.4. We use the same settings, iterative solver, and initial preconditioners as described in Section 5.1 and table 5.9.

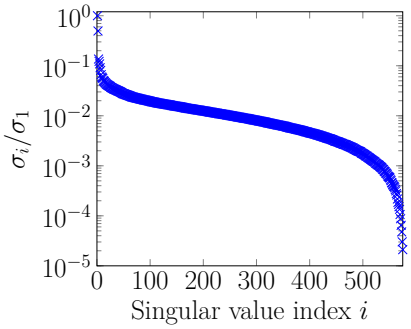
5.5.1 Update rank and low-rank property

We now analyze numerically the effectiveness of the low-rank update for the inner Poisson-type problems for increasing the update rank to a maximum. The numerical experiments are conducted with the same test systems, initial preconditioners and relaxation parameters as in the analogous tests from Section 5.4.1. We relate the results to the decay of singular values of the error matrices $E_{M,\mathrm{R}}$ (4.20) and $E_{D,\mathrm{R}}$ (4.25). The inner Poisson-type problems are approximated by incomplete Cholesky factorizations which are symmetric, so $(E_{M,\mathrm{R}})^{\mathrm{T}} = E_{M,\mathrm{L}}$ and $(E_{M,\mathrm{R}})^{\mathrm{T}} = E_{M,\mathrm{L}}$. Both matrices thus have the same singular values and we restrict ourselves to the analysis of the right updates with corresponding error matrices.

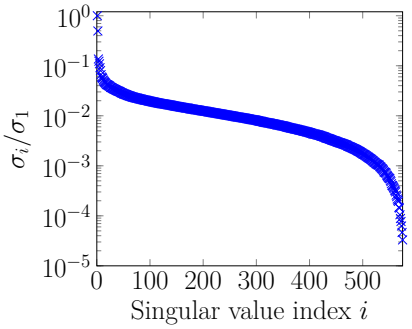
Figure 5.11 shows iteration counts obtained for updates constructed with a randomized low-rank approximation with (a) $q = 0$ power iterations, (b) $q = 2$ power iterations, and (c) with an Arnoldi iteration. We furthermore show the decay of singular values of $E_{D,\mathrm{R}}$ for the SIMPLE and of $E_{M,\mathrm{R}}$ for the LSC preconditioner for the Stokes systems. Note that the matrix $E_{M,\mathrm{R}}$ is the same for the considered Stokes and Oseen systems for the LSC preconditioner since it does not include any advective parts. For the SIMPLE preconditioner, the matrix $E_{D,\mathrm{R}}$ includes advective parts and thus changes with Picard-type iterations but only via the diagonal matrix $D = \mathrm{diag}(A)$. Furthermore, the change in the diagonal matrix D is proportional to the (small) time step size. Therefore, we reuse the approximation $\widehat{M}_{\mathrm{BDB^T}}$ in (3.44) corresponding to the first saddle-point system, i.e. the Stokes system, in the Schur complement preconditioner and in the update construction in the following Picard-type iterations. With this approach, we save construction costs while the results obtained with the inner updates tend to be the same for reusing and recomputing the approximation $\widehat{M}_{\mathrm{BDB^T}}$. We furthermore observed that the singular values of the error matrix $E_{D,\mathrm{R}}$ only differ by a relative value of up to 1e−3 between the reused and the recomputed matrix $M_{\mathrm{BDBT^T}}$ and its approximation $\widehat{M}_{\mathrm{BDB^T}}$.

We observe that the first few singular values decay very fast for both Schur complement preconditioners. We hence hope that a small update rank is sufficient to improve the initial preconditioners. This is resembled in the iteration numbers. For small update ranks, the number of iterations is reduced if we use a randomized low-rank approximation. The Arnoldi iteration only slightly reduced iteration counts. However, for increasing the update rank further to a maximum rank, the preconditioners are not further improved. Compared to computing the update for the Schur complement preconditioner $\widehat{S}^{-1}$, the minimum iteration counts are higher for most systems. This may be explained by the fact that we now only improve the approximation of the inner Poisson-type problems but do not act on the approximation error of the Schur complement preconditioning technique itself. However, as observed in Section 5.2, the approximation quality of the inner Poisson-type problems has a significant impact on the convergence of the outer iterative solver. Another advantage of the updates for the Poisson-type problems is that the underlying low-rank

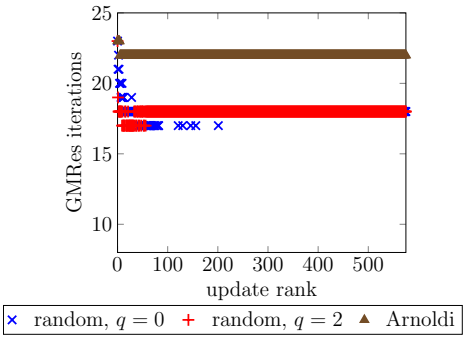
approximation techniques can be applied to the exact error matrices $E_{M,\mathrm{R}}$ (4.20) or $E_{D,\mathrm{R}}$ (4.25) or their left-oriented versions. For the Schur complement preconditioner updates, the low-rank approximation methods can only be applied to an approximation of the error matrices $E_{\mathrm{L},\alpha}$ or $E_{\mathrm{R},\alpha}$, defined in (4.5), since they depend on the exact Schur complement which is typically not available.



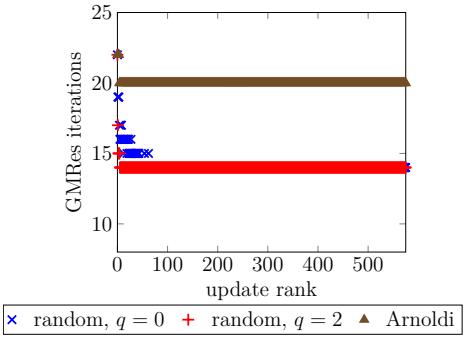
(A) SIMPLE, $\alpha = 1.8$.



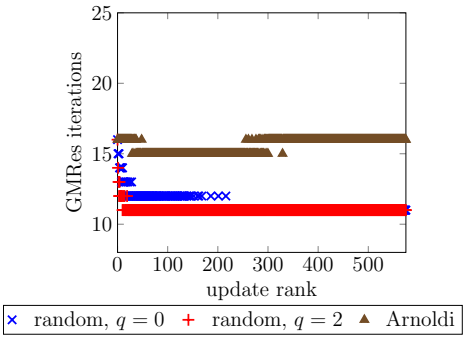
(B) LSC, $\alpha = 1.2$.



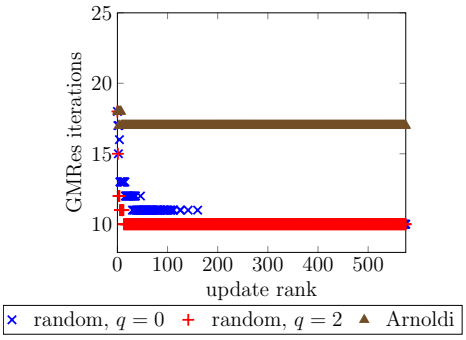
(C) Oseen, SIMPLE, $\alpha = 1.8$.



(D) Oseen, LSC, $\alpha = 1.2$.



(E) Stokes, SIMPLE, $\alpha = 1.8$.



(F) Stokes, LSC, $\alpha = 1.2$.

FIGURE 5.11: Decay of singular values of the matrix $E_{M,\mathrm{R}} = \mathrm{I}_{n_p} - M\widehat{M}^{-1}$ for the Stokes systems and GMRes iteration counts for increasing the update rank from $r = 0$ to $r = 575$ for small *Oseen* and *Stokes* systems on the cube ($n_u = 14739$, $n_p = 729$, $k_n = 5\mathrm{e}{-3}$). The shown results are obtained with updates constructed with Algorithm 2.2 ("random") using $q = 0$ and $q = 2$ power iterations and Algorithm 2.4 ("Arnoldi").

5.5.2 Influence of the low-rank approximation

We now investigate how the low-rank approximation affects the quality of the update scheme. For this purpose, we compare the results obtained with an update rank of $r = 20$ computed with a randomized low-rank approximation for increasing the number of power iterations from $q = 0$ to $q = 3$ and the Arnoldi iteration (with as well as without projection). Furthermore, we compare the results obtained with the same relaxation parameter but without any update. The numerical experiments are performed with the same systems, initial preconditioners, and relaxation parameters as in the analogous experiments from Section 5.4.2, see Table 5.9 for the initial solver. The shown results are restricted to the right updates since the results for the experiments of this section appear to be very similar for both update sides.

Table 5.14 shows the median and standard deviation of the iteration counts obtained from 200 test runs for the Oseen systems and Table 5.15 shows the results obtained for the Stokes systems with two different refinement levels for the cube and the shell systems. We discuss the observations for the Oseen and the Stokes systems side by side since the results appear to be very similar for all considered test systems. For all considered low-rank approximation techniques, we observe that the iteration counts obtained with the updated Schur complement preconditioners with inner updates are significantly smaller than those obtained with the preconditioners $\widehat{S}^{-1}_{\text{random,R},\alpha,r}$, $\widehat{S}^{-1}_{\text{Arnoldi,R},\alpha,r}$, and $\widehat{S}^{-1}_{\text{ArnoldiP,R},\alpha,r}$ with outer updates, defined in (4.11b), (4.12b), and (4.13b). The deviation in iteration counts obtained in repeated test runs is also much smaller. Not only the setup times but also the construction times are now significantly smaller (by about two orders of magnitude) than for the outer updates. Hence, a more accurate but more expensive low-rank approximation is now beneficial not only regarding the solver costs but also regarding the total costs. The smallest construction costs are obtained with the updates that are based on the Arnoldi iteration. However, the iteration counts are less reduced than with the randomized approach for most test systems. Additional projection leads to higher setup times as well as a higher reduction in iteration counts and solver times. The highest reduction of iteration counts is obtained with inner updates built with the randomized method with $q > 0$ power iterations. For the smaller systems, applying one power iteration in the randomized low-rank approximation appears to balance the accuracy and construction times of the updates. For the larger systems, the application of $q > 1$ power iterations leads to a further reduction of the median as well as the standard deviation of GMRes iteration counts. Since the construction of the inner updates is comparably cheap, the sum of construction and solver times is also reduced with $q = 2$ or even $q = 3$ power iterations. Especially for the LSC preconditioner on the larger shell systems, using more than one power iteration appears to be beneficial.

5.5.3 Rank and relaxation

In this section, we investigate numerically the influence of the update rank and the relaxation parameter for the preconditioners with updated inner Poisson-type problems. We compare GMRes iterations obtained with the update rank $r \in \{0, 5, \dots, 120\}$ and the relaxation parameter $\alpha \in \{0.1, 0.2, \dots, 2.9\}$ for the same systems and initial preconditioners as in Section 5.4.3 that are summarized in Table 5.9.

For the Schur complement preconditioners with inner low-rank updates, the relaxation

TABLE 5.14: Results for varying the number of power iterations q with rank $r = 20$ for the *Oseen* systems on the shell and the cube for two different refinement levels. The needed GMRes iterations and computational times in seconds are given as the median (standard deviation) obtained from 200 test runs.

(A) Shell, LSC, $\alpha = 1.1$.

q	iters.	setup (update)	solve
$n_u = 78438$, $n_p = 3474$, $k_n = 5\text{e}{-4}$.			
0	21 (0.69)	2.5e−2 (2.3e−3)	0.99 (3.8e−2)
1	18 (0)	4.2e−2 (1.8e−3)	0.86 (1.4e−2)
2	18 (0)	6.0e−2 (1.7e−3)	0.87 (1.9e−2)
3	18 (0)	7.7e−2 (2.0e−3)	0.88 (1.9e−2)
Arnoldi	26	2.0e−2	1.21
Arnoldi+P	19	3.0e−2	0.89
no update	30	0	1.41
$n_u = 608454$, $n_p = 26146$, $k_n = 2.5\text{e}{-4}$.			
0	nc	0.19 (3.1e−3)	nc
1	190 (132.5)	0.33 (8.2e−3)	79.4 (54.8)
2	79 (11.4)	0.47 (8.6e−3)	32.8 (4.6)
3	64 (5.1)	0.61 (4.2e−3)	26.1 (2.1)
Arnoldi	340	0.18	140.8
Arnoldi+P	259	0.26	108.6
no update	nc	0	nc

(B) Cube, LSC, $\alpha = 1.7$.

q	iters.	setup (update)	solve
$n_u = 107811$, $n_p = 4913$, $k_n = 5\text{e}{-3}$.			
0	36.5 (0.98)	3.4e−2 (2.0e−3)	2.51 (8.6e−2)
1	27 (0)	5.7e−2 (1.7e−3)	1.83 (2.1e−3)
2	27 (0)	8.0e−2 (2.3e−3)	1.83 (4.2e−2)
3	27 (0)	1.0e−1 (2.4e−3)	1.78 (3.5e−2)
Arnoldi	43	2.7e−2	2.92
Arnoldi+P	36	4.1e−2	2.42
no update	71	0	4.66
$n_u = 823875$, $n_p = 35937$, $k_n = 2.5\text{e}{-3}$.			
0	420.5 (212.2)	0.27 (2.8e−3)	243.9 (122.9)
1	68 (3.5)	0.46 (2.0e−3)	39.4 (2.1)
2	37 (3.2)	0.65 (2.7e−3)	21.5 (2.0)
3	36 (0.74)	0.85 (3.8e−3)	20.8 (0.44)
Arnoldi	227	0.24	131.8
Arnoldi+P	144	0.34	86.0
no update	nc	0	nc

(C) Shell, SIMPLE, $\alpha = 1.9$.

q	iters.	setup (update)	solve
$n_u = 78438$, $n_p = 3474$, $k_n = 5\text{e}{-4}$.			
0	20 (0.66)	2.5e−2 (2.3e−3)	0.61 (2.4e−2)
1	19 (0)	4.2e−2 (1.1e−3)	0.58 (2.4e−4)
2	19 (0)	5.9e−2 (1.7e−3)	0.58 (8.4e−3)
3	19 (0)	7.6e−2 (2.3e−3)	0.58 (1.6e−2)
Arnoldi	22	1.9e−2	0.66
Arnoldi+P	19	3.0e−2	0.58
no update	23	0	0.70
$n_u = 608454$, $n_p = 26146$, $k_n = 2.5\text{e}{-4}$.			
0	60 (5.2)	0.19 (2.2e−3)	15.7 (1.3)
1	33 (1.1)	0.33 (2.6e−3)	8.6 (0.30)
2	28 (0.96)	0.47 (3.3e−3)	7.3 (0.26)
3	26 (0.69)	0.61 (3.5e−3)	6.8 (0.18)
Arnoldi	61	0.18	16.6
Arnoldi+P	41	0.26	11.3
no update	63	0	16.1

(D) Cube, SIMPLE, $\alpha = 1.9$.

q	iters.	setup (update)	solve
$n_u = 107811$, $n_p = 4913$, $k_n = 5\text{e}{-3}$.			
0	44 (1.2)	3.3e−2 (8.9e−4)	1.96 (7.0e−2)
1	37 (0.18)	5.6e−2 (1.6e−3)	1.61 (2.8e−2)
2	37 (0)	8.0e−2 (1.9e−3)	1.65 (3.8e−2)
3	37 (0)	0.10 (1.9e−4)	1.64 (3.6e−2)
Arnoldi	46	2.7e−2	2.01
Arnoldi+P	44	4.1e−2	1.95
no update	64	0	2.74
$n_u = 823875$, $n_p = 35937$, $k_n = 2.5\text{e}{-3}$.			
0	63 (3.1)	0.27 (2.7e−3)	23.4 (1.1)
1	38 (1.0)	0.46 (3.2e−3)	14.2 (0.47)
2	33 (0.59)	0.65 (4.4e−3)	12.3 (0.23)
3	32 (0.51)	0.85 (4.1e−3)	11.9 (0.20)
Arnoldi	62	0.23	23.4
Arnoldi+P	58	0.35	21.8
no update	71	0	25.8

TABLE 5.15: Results for varying the number of power iterations q with rank $r = 20$ for the *Stokes* systems on the shell and the cube for two different refinement levels. The needed GMRes iterations and computational times in seconds are given as the median (standard deviation) obtained from 200 test runs.

(A) Shell, LSC, $\alpha = 1.2$.

q	iters.	setup (update)	solve
$n_u = 78438$, $n_p = 3474$, $k_n = 5\text{e}{-4}$.			
0	19 (0.66)	2.5e−2 (7.9e−4)	0.93 (3.7e−2)
1	16 (0.21)	4.3e−2 (1.3e−3)	0.79 (2.2e−2)
2	16 (0.31)	5.9e−2 (1.5e−3)	0.78 (2.6e−2)
3	16 (0.32)	7.5e−2 (1.8e−3)	0.76 (2.1e−2)
Arnoldi	25	1.9e−2	1.16
Arnoldi+P	17	3.0e−2	0.81
no update	30	0	1.45
$n_u = 608454$, $n_p = 26146$, $k_n = 2.5\text{e}{-4}$.			
0	nc	0.19 (3.8e-3)	nc
1	306 (166.8)	0.33 (4.4e-3)	124.2 (67.5)
2	106 (18.9)	0.47 (3.5e-3)	42.9 (7.6)
3	70 (5.5)	0.61 (3.2e-3)	28.3 (2.2)
Arnoldi	624	0.17	264.7
Arnoldi+P	225	0.25	95.7
no update	nc	0	nc

(B) Cube, LSC, $\alpha = 1.2$.

q	iters.	setup (update)	solve
$n_u = 107811$, $n_p = 4913$, $k_n = 5\text{e}{-3}$.			
0	21 (0.63)	3.4e−2 (2.1e−3)	1.41 (4.4e−2)
1	13 (0.12)	5.7e−2 (1.6e−3)	0.89 (2.3e−2)
2	13 (0.26)	7.9e−2 (2.0e−3)	0.89 (2.4e−2)
3	13 (0.38)	0.10 (2.2e−3)	0.87 (3.2e−2)
Arnoldi	27	2.7e−2	1.80
Arnoldi+P	22	4.1e−2	1.48
no update	34	0	2.25
$n_u = 823875$, $n_p = 35937$, $k_n = 2.5\text{e}{-3}$.			
0	646 (185.7)	0.27 (3.2e−3)	371.4 (106.8)
1	68 (3.6)	0.46 (3.2e−3)	39.1 (2.1)
2	37 (1.9)	0.65 (4.2e−3)	21.3 (1.3)
3	35 (0.67)	0.85 (4.6e−3)	20.1 (0.40)
Arnoldi	241	0.23	141.9
Arnoldi+P	113	0.35	65.8
no update	nc	0	nc

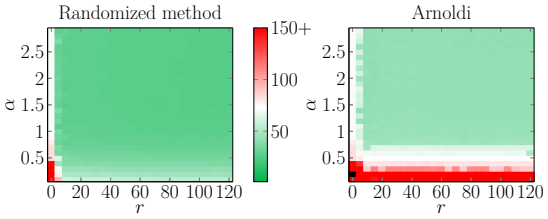
(C) Shell, SIMPLE, $\alpha = 1.9$.

q	iters.	setup (update)	solve
$n_u = 78438$, $n_p = 3474$, $k_n = 5\text{e}{-4}$.			
0	16 (0.41)	2.5e−2 (1.4e−3)	0.49 (1.5e−2)
1	14 (0.47)	4.2e−2 (9.2e−4)	0.43 (1.6e−2)
2	14 (0.17)	5.9e−2 (1.1e−3)	0.43 (9.5e−3)
3	14 (0)	7.8e−2 (1.7e−3)	0.45 (8.6e−3)
Arnoldi	21	2.0e−2	0.63
Arnoldi+P	15	3.0e−2	0.45
no update	20	0	0.63
$n_u = 608454$, $n_p = 26146$, $k_n = 2.5\text{e}{-4}$.			
0	62 (18.3)	0.20 (3.8e−3)	16.4 (1.1)
1	33 (1.2)	0.34 (4.4e−3)	8.7 (0.33)
2	28 (0.89)	0.48 (4.2e−3)	7.4 (0.24)
3	26 (0.73)	0.62 (4.4e−3)	6.8 (0.20)
Arnoldi	57	0.16	15.1
Arnoldi+P	37	0.24	9.9
no update	67	0	17.3

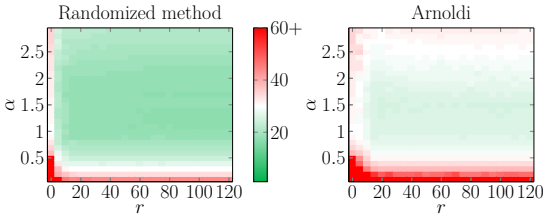
(D) Cube, SIMPLE, $\alpha = 1.9$.

q	iters.	setup (update)	solve
$n_u = 107811$, $n_p = 4913$, $k_n = 5\text{e}{-3}$.			
0	18 (0.50)	3.3e−2 (7.9e−4)	0.76 (2.4e−2)
1	13 (0)	5.6e−2 (1.0e−3)	0.56 (7.0e−3)
2	13 (0)	8.0e−2 (2.3e−3)	0.57 (1.3e−2)
3	13 (7.1e−2)	0.10 (1.7e−3)	0.56 (8.0e−3)
Arnoldi	20	2.7e-2	0.84
Arnoldi+P	18	4.0e-2	0.77
no update	23	0	0.97
$n_u = 823875$, $n_p = 35937$, $k_n = 2.5\text{e}{-3}$.			
0	35 (0.54)	0.27 (2.7e−3)	13.0 (0.21)
1	27 (0.70)	0.46 (3.6e−3)	9.9 (0.27)
2	23 (0.52)	0.65 (4.2e−3)	8.4 (0.19)
3	22 (0.33)	0.85 (4.1e−3)	8.1 (0.12)
Arnoldi	34	0.24	12.8
Arnoldi+P	32	0.36	12.4
no update	38	13.9	13.9

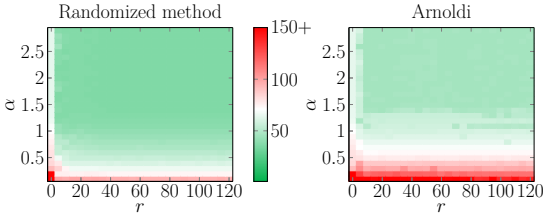
parameter has only a slight influence on convergence behavior for parameter choices $1.0 < \alpha < 2.9$. We thus show exemplary results for the Oseen systems for one refinement level (`cube_4` from Table 5.1) in Figure 5.12 and omit the results for the other systems. Instead, we show iteration counts and computational times in seconds for the update construction and the sum with solver times in Figure 5.13 for the Oseen systems and in Figure 5.14 for the Stokes systems, each for two refinement levels. In both figures, we set $\alpha = 1.7$ for the SIMPLE preconditioner and $\alpha = 1.1$ for the LSC preconditioner. These parameter choices are within the range of "optimal" values for α regarding GMRes iteration counts. We observe that iteration counts and computational times decrease significantly for small update ranks, especially for the randomized update. The ratio of the construction time increases with the update rank. The low-rank updates constructed with the randomized low-rank approximation with $q = 1$ lead to a higher reduction in iterations than the updates constructed with the Arnoldi iteration without projection. The iteration counts and computational times obtained with inner updates based on the Arnoldi iteration in combination with projection lie between the results obtained with both other low-rank approximations for most of the test systems. Only for the LSC preconditioner on the Stokes system on the shell with the coarser mesh, the highest reduction in iteration counts and total times are obtained with updates based on the Arnoldi iteration followed by projection. Compared to the cheapest considered initial Schur complement preconditioners from Section 5.2, the preconditioners with inner updates based on a randomized low-rank approximation lead to smaller total times with suitable update ranks and relaxation parameters.



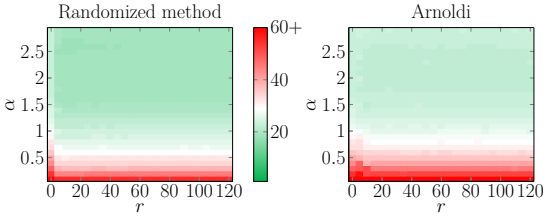
(A) Cube, LSC.



(B) Shell, LSC.

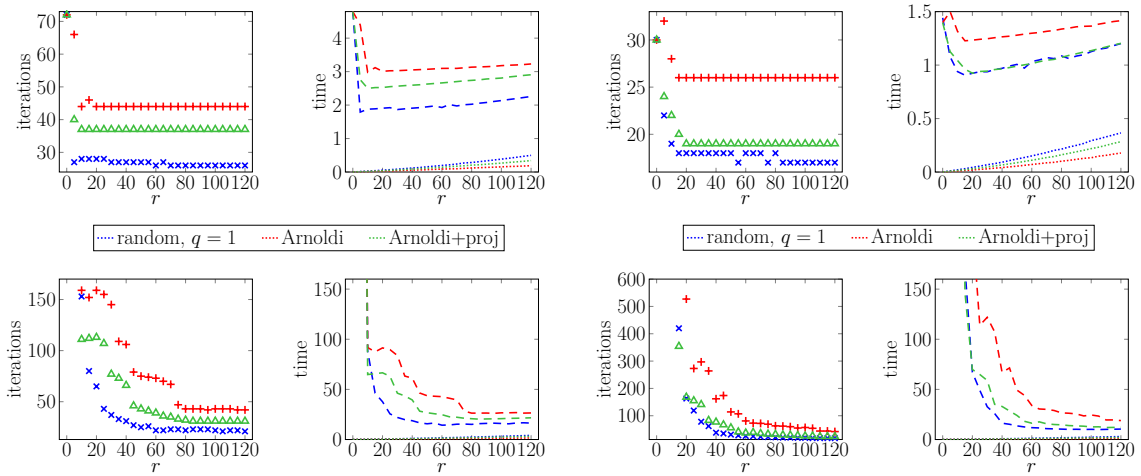


(C) Cube, SIMPLE.



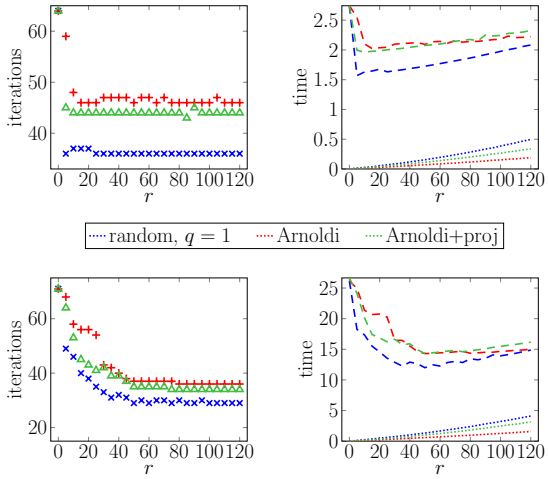
(D) Shell, SIMPLE.

FIGURE 5.12: GMRes iteration counts for varying the update rank r and relaxation parameter α. The results are obtained for the *Oseen* systems with the randomized Algorithm 2.2 ($q = 1$) and the Arnoldi Algorithm 2.4.

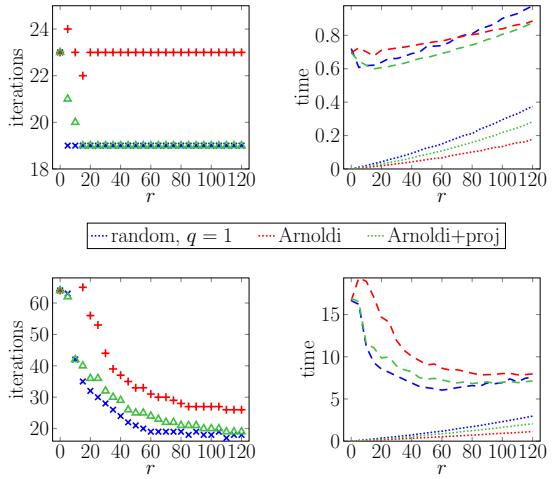


(A) Cube, LSC, $\alpha = 1.1$.

(B) Shell, LSC, $\alpha = 1.1$.

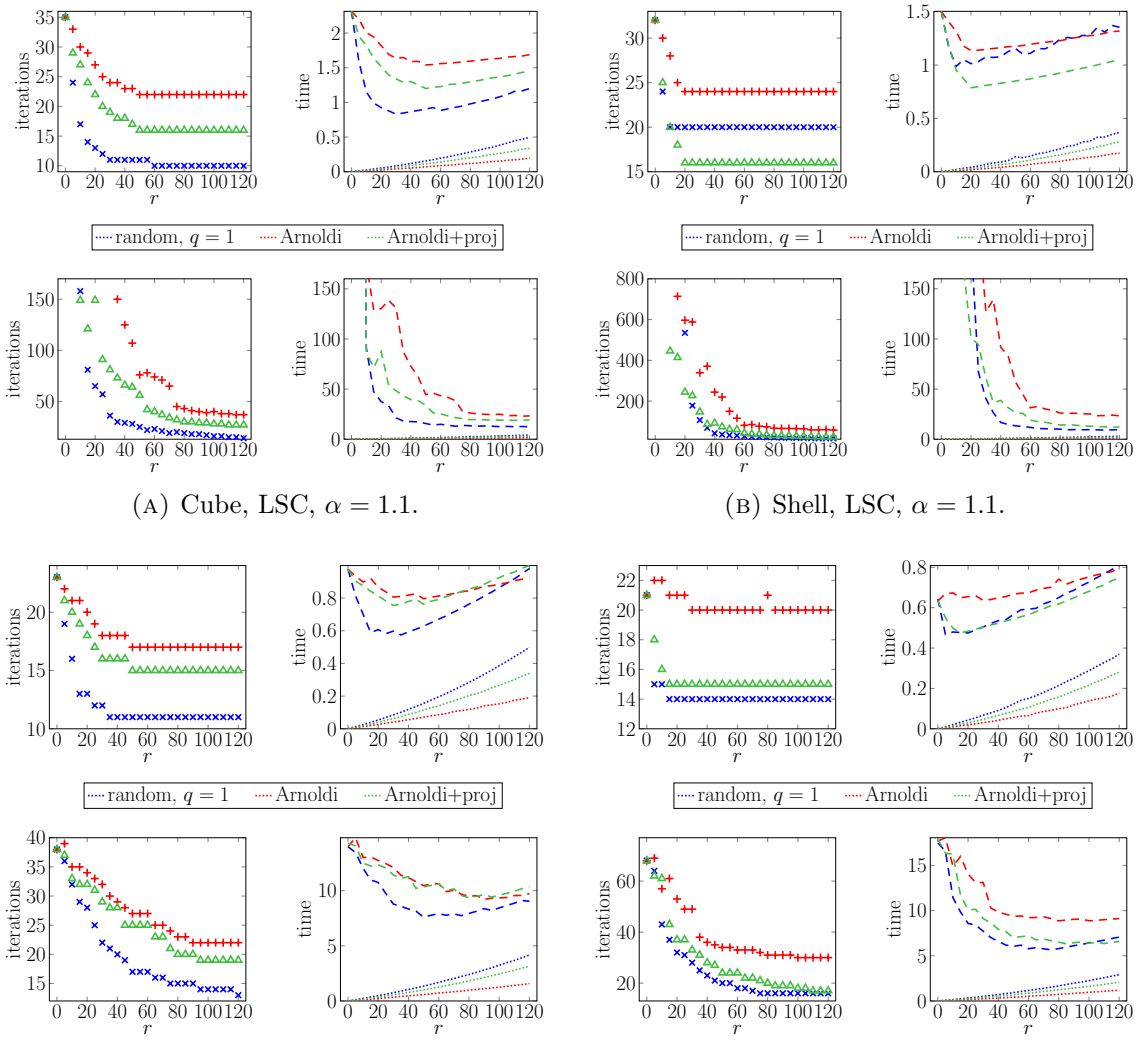


(C) Cube, SIMPLE, $\alpha = 1.7$.



(D) Shell, SIMPLE, $\alpha = 1.7$.

FIGURE 5.13: GMRes iteration counts and computational times in seconds (update construction: dotted, total of solve and update construction: dashed) for varying the update rank r. The results are obtained for the *Oseen* systems with two refinement levels (upper row of each subfigure: `shell_3` or `cube_4`, lower row of each subfigure: `shell_4` or `cube_5`, see Table 5.1).



(A) Cube, LSC, $\alpha = 1.1$.

(B) Shell, LSC, $\alpha = 1.1$.

(C) Cube, SIMPLE, $\alpha = 1.7$.

(D) Shell, SIMPLE, $\alpha = 1.7$.

FIGURE 5.14: GMRes iteration counts and computational times in seconds (update construction: dotted, total of solve and update construction: dashed) for varying the update rank r. The results are obtained for the *Stokes* systems with two refinement levels (upper row of each subfigure: `shell_3` or `cube_4`, lower row of each subfigure: `shell_4` or `cube_5`, see Table 5.1).

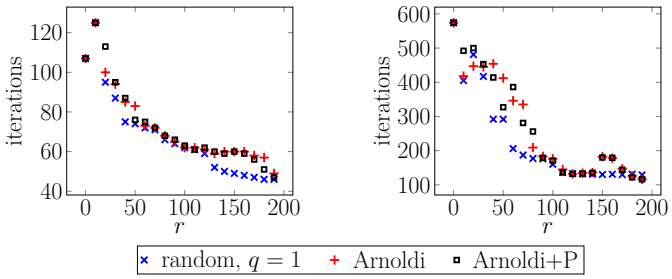
5.6 Updates for preconditioning the upper left matrix block

For the sake of completeness, we now investigate the effectiveness of the error-based low-rank corrections from Section 4.1 for the preconditioner $\widehat{A}^{-1}$. We consider two initial preconditioners for $\widehat{A}^{-1}$: the block-triangular preconditioner (3.38) with ILU without any fill-in for all arising inverses which turned out to be the most effective preconditioner for the majority of the test systems (see Section 5.2); and the ILU preconditioner based on the velocity mass matrix M_u (also without any fill-in) which is interesting since the velocity mass matrix does not change with Picard iterations and varying time step sizes

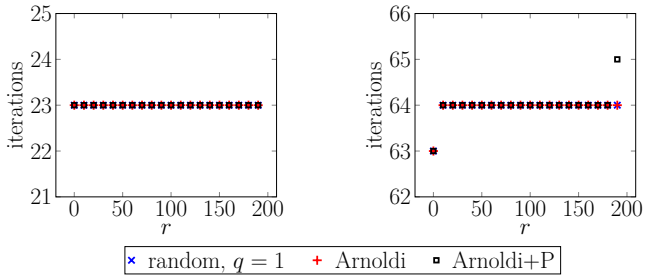
and dominates the matrix block A.

As Schur complement preconditioner, we use the relaxed SIMPLE preconditioner with an incomplete Cholesky decomposition without fill-in and set the relaxation parameter to $\alpha = 1.8$ based on the results achieved so far. We test the low-rank corrections with the Oseen and Stokes systems on a cube and a spherical shell, each with two refinement levels. The low-rank updates are built with three different low-rank approximation techniques: the randomized range finder with $q = 1$ power iterations and without oversampling, the Arnoldi iteration, and the Arnoldi iteration followed by projection.

Figure 5.15 shows exemplarily GMRes iteration counts obtained for the preconditioners $\widehat{A}^{-1}$ with low-rank updates for ranks up to 190 for the Oseen systems on the shell with two refinement levels. The remaining results are omitted since we found that an update with a rank of up to 190 does not improve the initial preconditioners $\widehat{A}^{-1}$ for most of the considered test systems. We hence only show selected numerical results for the Oseen systems on the spherical shell domain that show the highest improvement with low-rank updates. These observations are consistent with the slow decay of singular values of the error matrix as illustrated in Section 4.2. For the block triangular preconditioner, the low-rank updates increase iteration counts by 1-2 iterations for an update rank of up to $r = 190$ whereas for the ILU preconditioner based on the velocity mass matrix, we observe a significant decrease in iteration counts. However, we still require about twice as many iterations with rank $r = 190$ as with the block-triangular ILU preconditioner without any update. The iteration counts are similar for the different low-rank approximation methods. A low-rank correction for $\widehat{A}^{-1}$ is hence not beneficial regarding iteration counts and solver times for the considered test systems.



(A) Updates for ILU(M_u).



(B) Updates for the block-triangular ILU preconditioner (3.38).

FIGURE 5.15: GMRes iteration counts for the preconditioner $\widehat{A}^{-1}$ with low-rank updates for *Oseen* systems on the spherical shell domain with two refinement levels (left images of each subfigure: `shell_3` with k_n = 5e−4, right images of each subfigure: `shell_4` with k_n = 2.5e−4, see Table 5.1).

Chapter 6

Conclusion and outlook

We have developed and analyzed low-rank updates for preconditioners for saddle-point systems coming from fluid dynamic applications and applied the low-rank corrections in a numerical solver for the Boussinesq equations. In the following, we summarize the main results of this thesis.

We started in Chapter 3 with our model of interest, a model for buoyancy-driven atmospheric dynamics described by the rotating Boussinesq equations. A spatial discretization with the finite-element method and a semi-implicit temporal discretization led to a sequence of saddle-point systems for the fluid velocity and pressure as well as positive definite linear systems for the temperature field. As the simulation time is dominated by the numerical solution of the saddle-point systems (Table 3.1), we focused on preconditioners for saddle-point systems. We initially reviewed common preconditioning strategies for saddle-point systems which we used as a basis for the derived low-rank corrections. Block preconditioners for saddle-point systems comprise typically preconditioners $\widehat{A}^{-1}$ and $\widehat{S}^{-1}$ for the upper left matrix block and the pressure Schur complement, respectively. Based on the numerical comparison of several initial preconditioners in Section 5.2 (Tables 5.4–5.7), we chose a block triangular preconditioner with `ILU(0)` for $\widehat{A}^{-1}$ and the LSC and SIMPLE preconditioner with `IC(0)` for the inner Poisson-type problems for $\widehat{S}^{-1}$.

We derived error-based low-rank corrections which are constructed from a (randomized) low-rank approximation of the preconditioning error, i.e. the difference between the identity matrix and the left or right preconditioned matrix. To obtain an effective low-rank correction, we require that the preconditioning error has a small numerical rank, i.e. sufficiently fast decaying singular values. However, we observed that the preconditioning error has not always (numerical) low-rank character (Section 4.2, Figures 4.1–4.2). The developed low-rank updates can be applied to any given preconditioner. We considered different components for the low-rank updates: the preconditioner $\widehat{A}^{-1}$, the Schur complement preconditioner $\widehat{S}^{-1}$, and preconditioners for inner Poisson-type problems required for the Schur complement preconditioners. We focused on low-rank corrections for the latter two components since preconditioning the Schur complement is typically a bottleneck for numerical solvers of saddle-point systems. Furthermore, we observed in Section 4.2 that the preconditioning error for preconditioners $\widehat{A}^{-1}$ has slowly decaying singular values (Figures 4.1–4.2) for the considered test systems which also resembled in numerical experiments as the developed low-rank updates for the considered preconditioners $\widehat{A}^{-1}$ for the upper left matrix block are not effective for the considered test systems (Section 5.6, Figure 5.15). We

furthermore considered inner updates applied to the Poisson-type problems in the Schur complement preconditioning techniques SIMPLE and LSC with significantly smaller setup times than those required for the outer updates (Sections 4.5 and 5.5, Figures 5.13–5.14). These updates turned out to be more effective than the updates for the Schur complement preconditioners since the underlying error matrix has fast decaying singular values (Figure 5.11) and can be well approximated by a low-rank factorization.

To construct the updates, we require a low-rank approximation of a matrix that is defined via its action on a vector. We compared low-rank factorizations based on the Arnoldi iteration and a randomized power range finder. We compared the different low-rank approximation strategies regarding the computational complexity of the construction and application of the updates (Section 4.6) and the effectiveness concerning the reduction of GMRes iteration counts (Sections 5.4.2 and 5.5.2). The construction complexity depends on the low-rank approximation technique as well as parameters for the approximation methods such as the number of power iterations or oversampling vectors. We have observed that the decision on a suitable low-rank approximation technique requires balancing setup costs and accuracy. Low-rank updates based on too inaccurate approximations may deteriorate the initial preconditioner. It may hence be advantageous to apply a more expensive but more accurate low-rank approximation technique. We found that the randomized range finder with one power iteration and without oversampling balanced setup costs and accuracy for the considered test systems (Section 5.4.2, Tables 5.11–5.12, and Section 5.5.2, Tables 5.14–5.15).

Additionally to the low-rank updates, we introduced a relaxation parameter (Section 4.3) that scales the Schur complement preconditioner in the block preconditioner. We observed that this parameter has a significant influence on the convergence behavior (Figures 5.2–5.3) as well as the effectiveness of the low-rank corrections while introducing only minor additional costs. An optimal choice of this parameter seems to depend mainly on the initial preconditioner and the range of (nearly) optimal parameters is relatively large at least for the considered test systems (Sections 5.3, 5.4.3, and 5.5.3).

We analyzed the computational complexity of the construction and application of the considered update schemes for Schur complement preconditioners for a low-rank approximation with the Arnoldi iteration and a randomized range finder (Section 4.6). We related the additional application costs introduced by the updates to the application costs of the initial Schur complement preconditioners and the computational complexity of one iteration of the restarted GMRES solver. We observed that the additional application costs are linear in the number of pressure degrees of freedom but may be significant for cheap initial preconditioners. However, the additional costs are negligible compared to the computational complexity of one iteration of the restarted GMRes solver (Tables 4.3 and 4.5). A reduction of total costs requires the reduction of about half an iteration for updates based on the Arnoldi iteration and about 1.5-2.5 iterations per update rank for updates based on the randomized range finder with one power iteration and without oversampling (Table 4.3). We furthermore discussed the recycling of sample vectors in the update construction with the randomized range finder for a repeated application of low-rank updates with the same update rank. Reusing sample vectors may reduce setup times depending on the update rank, the number of power iterations, and the number of oversampling vectors. If we do not use oversampling, the setup times are reduced for every update rank (Table 4.4).

We furthermore provided an error analysis that links the low-rank approximation error

to the preconditioning error after applying a low-rank correction (Section 4.7). We found that the spectral norm of the difference between the identity matrix and the preconditioned matrix or Schur complement is not guaranteed to be reduced by a low-rank correction but may even increase (Figures 4.3–4.4). A numerical spectral analysis showed that the considered class of updates may enhance the clustering of eigenvalues but does not guarantee improvement of spectral properties of the preconditioned matrix (Section 4.8, Figure 4.5).

We conclude with an outlook on possible future work based on this thesis.

High-performance computing The application of the developed methods in the simulation of large-scale models demands a high-performance implementation. A numerical analysis regarding scalability, speedup, and efficiency may be interesting to learn about required algorithmic adaptations for example to reduce communication between the processes. The current implementation is already based on matrix and vector types of the `Epetra` package from TRILINOS [38] that are implemented in parallel to be used with distributed memory and hence can be used as a basis for a parallel implementation of the developed algorithms. A high-performance implementation may somewhat change the drawn conclusions regarding computational costs such as the ratio of update construction costs to saved solver times. Furthermore, the decision on suitable initial preconditioners might be different with distributed memory than with a serial implementation.

Recycling of update vectors For the unsteady Navier-Stokes equations, the matrix block A changes with each Picard iteration and for varying time step sizes. Computing new preconditioners $\widehat{A}^{-1}$ in every Picard iteration or in every time step is computationally expensive. With low-rank updates, we might avoid computing new preconditioners from scratch. However, the setup of new updates also requires a significant amount of time. To save construction costs, an option is the recycling of update vectors. We could reuse the vectors that approximate the range of an error matrix E_{old}, stored as the columns of the matrix Q, and project with the new error matrix E_{new} to obtain new update vectors. Namely, starting with updates based on the low-rank approximation

$$E_{\text{old}} = \mathrm{I}_{n_u} - A_{\text{old}}\widehat{A}^{-1} \approx QN_{\text{old}}^{\mathrm{T}}$$

with $N_{\text{old}}^{\mathrm{T}} = Q^{\mathrm{T}}E_{\text{old}}$, we construct the new updates via the approximation

$$E_{\text{new}} = \mathrm{I}_{n_u} - A_{\text{new}}\widehat{A}^{-1} \approx QN_{\text{new}}^{\mathrm{T}}$$

with $N_{\text{new}}^{\mathrm{T}} = Q^{\mathrm{T}}E_{\text{new}}$. However, we have observed that we require sufficiently accurate approximations of the range to improve the initial preconditioners. This approach hence involves an investigation of how much the system matrix A can change before we have to compute a new preconditioner or low-rank update from scratch.

Adaptive updates The effectiveness of the presented low-rank strategies depends on the update rank. To determine a suitable rank, we could use adaptive low-rank approximation strategies such as the (randomized) incremental range finder. Another approach would be to apply repeatedly further new updates of small ranks as discussed in Section 4.4. The first approach requires a suitable tolerance for the low-rank approximation as well as cheap (randomized) error estimators. First numerical experiments showed that the effectiveness

of the updates depends significantly on the approximation tolerance. Suitable values for the tolerance can differ for different update types and initial preconditioners. Balancing computational costs and reliability of randomized error estimators requires a reasonable choice of parameters such as the number of sampling vectors. Further investigation is required to find sets of suitable parameters that can be used in a black box manner. For the second approach we also need to decide when to compute a further update. One option could be to compute an additional update if the solver does not converge in a predefined number of iterations. The update rank could be determined with the incremental range finder or could be set to a small (constant) number. As derived in Section 4.6.3, depending on the update parameters, the construction of repeated updates may be cheaper than computing a larger new update. However, we observed that new updates of small ranks may deteriorate a given preconditioner. This approach hence requires an extensive numerical analysis.

Further numerical analysis The numerical analysis of the developed methods covers saddle-point systems coming from time-dependent problems with small time step sizes. Another class of saddle-point systems in fluid flow problems is for example given by the discretized steady-state Navier-Stokes equations. For unsteady problems, the matrix block A is dominated by the (positive definite) velocity mass matrix. For the steady-state systems, we distinguish two regimes for the matrix block A: advection-dominated systems for large Reynolds numbers and diffusion-dominated systems for small Reynolds numbers. Especially advection-dominated systems challenge typically iterative solvers. The presented low-rank updates can be applied to any preconditioner for any linear system. However, numerical experiments are required to investigate the effectiveness of low-rank updates for steady-state problems. For the steady-state systems, low-rank updates for preconditioners $\widehat{A}^{-1}$ might be more effective than updates for Schur complement preconditioners $\widehat{S}^{-1}$. Furthermore, different initial preconditioners might be more efficient such as the augmented Lagrangian preconditioner. Especially for advection-dominated problems, the LSC preconditioner might be more effective than the SIMPLE preconditioner. Furthermore, numerical studies for different discretizations might help to assess black box parameters such that the low-rank update strategies can be directly applied in simulations. Further experiments could be done with other iterative methods such as the biconjugate gradient stabilized method.

Bibliography

[1] N. Ahmed, C. Bartsch, V. John, and U. Wilbrandt. An assessment of some solvers for saddle point problems emerging from the incompressible Navier-Stokes equations. *Comput. Methods Appl. Math.*, 331:492–513, 2018. doi:10.1016/j.cma.2017.12.004.

[2] J. Ahrens, B. Geveci, and C. Law. 36 - ParaView: An end-user tool for large-data visualization. In C. D. Hansen and C. R. Johnson, editors, *Visualization Handbook*, pages 717–731. Butterworth-Heinemann, Burlington, 2005. doi:10.1016/B978-012387582-2%2F50038-1.

[3] H. Al Daas and L. Grigori. A class of efficient locally constructed preconditioners based on coarse spaces. *SIAM J. Matrix Anal. Appl.*, 40(1):66–91, 2019. doi:10.1137/18M1194365.

[4] H. Al Daas, T. Rees, and J. Scott. Two-level Nyström–Schur preconditioner for sparse symmetric positive definite matrices. *SIAM J. Sci. Comput.*, 43(6):A3837–A3861, 2021. doi:10.1137/21M139548X.

[5] D. Arndt, W. Bangerth, D. Davydov, T. Heister, L. Heltai, M. Kronbichler, M. Maier, J.-P. Pelteret, B. Turcksin, and D. Wells. The deal.II finite element library: Design, features, and insights. *Comput. Math. Appl.*, 81:407–422, 2021. doi:10.1016/j.camwa.2020.02.022.

[6] D. Arndt, W. Bangerth, M. Feder, M. Fehling, R. Gassmöller, T. Heister, L. Heltai, M. Kronbichler, M. Maier, P. Munch, J.-P. Pelteret, S. Sticko, B. Turcksin, and D. Wells. The `deal.II` library, version 9.4. *J. Numer. Math.*, 30(3):231–246, 2022. doi:10.1515/jnma-2022-0054.

[7] W. Bangerth, C. Burstedde, T. Heister, and M. Kronbichler. Algorithms and data structures for massively parallel generic adaptive finite element codes. *ACM Trans. Math. Software*, 38(2):14/1–28, 2012. doi:10.1145/2049673.2049678.

[8] W. Bangerth and O. Kayser-Herold. Data structures and requirements for hp finite element software. *ACM Trans. Math. Softw.*, 36(1):4/1–31, 2009. doi:10.1145/1486525.1486529.

[9] R. S. Beddig, J. Behrens, and S. Le Borne. A low-rank update for relaxed Schur complement preconditioners in fluid flow problems. *Numer. Algorithms*, 2023. doi:10.1007/s11075-023-01548-3.

[10] R. S. Beddig, J. Behrens, S. Le Borne, and K. Simon. An error-based low-rank correction for pressure Schur complement preconditioners. In A. Iske and T. Rung, editors, *Modeling, Simulation and Optimization of Fluid Dynamic Applications*, volume 148 of *Lect. Notes Comput. Sci. Eng.* Springer, Cham, 2023. doi:10.1007/978-3-031-45158-4_5.

[11] M. Benzi, G. H. Golub, and J. Liesen. Numerical solution of saddle point problems. *Acta Numer.*, 14:1–137, 2005. doi:10.1017/S0962492904000212.

[12] M. Benzi and M. A. Olshanskii. An augmented Lagrangian–based approach to the Oseen problem. *SIAM J. Sci. Comput.*, 28(6):2095–2113, 2006. doi:10.1137/050646421.

[13] M. Benzi and M. A. Olshanskii. Field-of-values convergence analysis of augmented Lagrangian preconditioners for the linearized Navier–Stokes problem. *SIAM J. Numer. Anal.*, 49(2):770–788, 2011. doi:10.1137/100806485.

[14] M. Benzi, M. A. Olshanskii, and Z. Wang. Modified augmented Lagrangian preconditioners for the incompressible Navier–Stokes equations. *Internat. J. Numer. Methods Fluids*, 66(4):486–508, 2011. doi:10.1002/fld.2267.

[15] M. Benzi and M. Tůma. A comparative study of sparse approximate inverse preconditioners. *Appl. Numer. Math.*, 30(2-3):305–340, 1999. doi:10.1016/S0168-9274(98)00118-4.

[16] L. Bergamaschi. A survey of low-rank updates of preconditioners for sequences of symmetric linear systems. *Algorithms (Basel)*, 13(4):100, 2020. doi:10.3390/a13040100.

[17] F. Brezzi and R. S. Falk. Stability of higher-order hood–taylor methods. *SIAM J. on Numer. Anal.*, 28(3):581–590, 1991. doi:10.1137/0728032.

[18] W. L. Briggs, V. E. Henson, and S. F. McCormick. *A multigrid tutorial*. SIAM, Philadelphia, PA, 2nd edition, 2000. doi:10.1137/1.9780898719505.

[19] C. Burstedde, L. C. Wilcox, and O. Ghattas. `p4est`: Scalable algorithms for parallel adaptive mesh refinement on forests of octrees. *SIAM J. Sci. Comput.*, 33(3):1103–1133, 2011. doi:10.1137/100791634.

[20] S. Börm and S. Le Borne. $\mathcal{H}$-LU factorization in preconditioners for augmented Lagrangian and grad-div stabilized saddle point systems. *Internat. J. Numer. Methods Fluids*, 68(1):83–98, 2010. doi:10.1002/fld.2495.

[21] E. Carson and N. Khan. Mixed precision iterative refinement with sparse approximate inverse preconditioning. *SIAM J. Sci. Comput.*, 45(3):C131–C153, 2023. doi:10.1137/22M1487709.

[22] T. F. Chan and T. P. Mathew. Domain decomposition algorithms. *Acta Numer.*, 3(7-8):61–143, 1994. doi:10.1017/S0962492900002427.

[23] B. Cushman-Roisin and J.-M. Beckers. *Introduction to geophysical fluid dynamics: physical and numerical aspects*, volume 101 of *Int. Geophys. Ser.* Elsevier/Academic Press, 2nd edition, 2011.

[24] D. R. Durran. *Numerical methods for fluid dynamics with applications in geophysics*, volume 32 of *Texts Appl. Math.* Springer, 2nd edition, 2010. doi:10.1007/978-1-4419-6412-0.

[25] H. Elman, V. E. Howle, J. Shadid, R. Shuttleworth, and R. Tuminaro. Block preconditioners based on approximate commutators. *SIAM J. Sci. Comput.*, 27(5):1651–1668, 2006. doi:10.1137/040608817.

[26] H. Elman, D. Silvester, and A. Wathen. *Finite Elements and Fast Iterative Solvers: with Applications in Incompressible Fluid Dynamics.* Oxford Univ. Press, 2014. doi:10.1093/acprof:oso/9780199678792.001.0001.

[27] H. Elman and R. Tuminaro. Boundary conditions in approximate commutator preconditioners for the Navier–Stokes equations. *Elec. Trans. Numer. Math.*, 35:257–280, 2009.

[28] H. C. Elman. Preconditioners for saddle point problems arising in computational fluid dynamics. *Appl. Numer. Math.*, 43(1):75–89, 2002. doi:10.1016/S0168-9274(02)00118-6.

[29] P. E. Farrell, L. Mitchell, and F. Wechsung. An augmented Lagrangian preconditioner for the 3D stationary incompressible Navier–Stokes equations at high Reynolds number. *SIAM J. Sci. Comput.*, 41(5):A3073–A3096, 2019. doi:10.1137/18M1219370.

[30] J. Fiordilino, W. Layton, and Y. Rong. An efficient and modular grad–div stabilization. *Comput. Methods Appl. Math.*, 335:327–346, 2018. doi:10.1016/j.cma.2018.02.023.

[31] J. Frank and C. Vuik. On the construction of deflation-based preconditioners. *SIAM J. Sci. Comput.*, 23(2):442–462, 2001. doi:10.1137/S1064827500373231.

[32] M. Gee, C. Siefert, J. Hu, R. Tuminaro, and M. Sala. ML 5.0 smoothed aggregation user's guide. Technical Report SAND2006-2649, Sandia National Laboratories, 2006.

[33] L. Grigori, F. Nataf, and S. Yousef. Robust algebraic Schur complement preconditioners based on low rank corrections. Research Report RR-8557, INRIA, 2014.

[34] N. Halko, P. G. Martinsson, and J. A. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Rev.*, 53(2):217–288, 2011. doi:10.1137/090771806.

[35] X. He and C. Vuik. Comparison of some preconditioners for the incompressible Navier–Stokes equations. *Numer. Math. Theory Methods Appl.*, 9(2):239–261, 2016. doi:10.4208/nmtma.2016.m1422.

[36] X. He and C. Vuik. Efficient and robust Schur complement approximations in the augmented Lagrangian preconditioner for the incompressible laminar flows. *J. Comput. Phys.*, 408:109286, 2020. doi:10.1016/j.jcp.2020.109286.

[37] T. Heister, J. Dannberg, R. Gassmöller, and W. Bangerth. High accuracy mantle convection simulation through modern numerical methods – II: realistic models and problems. *Geophys. J. Int.*, 210(2):833–851, 2017. doi:10.1093/gji/ggx195.

[38] M. A. Heroux, R. A. Bartlett, V. E. Howle, R. J. Hoekstra, J. J. Hu, T. G. Kolda, R. B. Lehoucq, K. R. Long, R. P. Pawlowski, E. T. Phipps, A. G. Salinger, H. K. Thornquist, R. S. Tuminaro, J. M. Willenbring, A. Williams, and K. S. Stanley. An overview of the Trilinos project. *ACM Trans. Math. Software*, 31(3):397–423, 2005. doi:10.1145/1089014.1089021.

[39] N. J. Higham and T. Mary. A new preconditioner that exploits low-rank approximations to factorization error. *SIAM J. Sci. Comput.*, 41(1):A59–A82, 2019. doi:10.1137/18M1182802.

[40] W. Hoffmann. Iterative algorithms for Gram-Schmidt orthogonalization. *Computing*, 41(4):335–348, 1989. doi:10.1007/BF02241222.

[41] V. John. On the efficiency of linearization schemes and coupled multigrid methods in the simulation of a 3D flow around a cylinder. *Internat. J. Numer. Methods Fluids*, 50(7):845–862, 2006. doi:10.1002/fld.1080.

[42] V. John. *Finite element methods for incompressible flow problems*, volume 51 of *Springer Series in Computational Mathematics*. Springer, Cham, 2016. doi:10.1007/978-3-319-45750-5.

[43] D. Kay, D. Loghin, and A. Wathen. A preconditioner for the steady-state Navier–Stokes equations. *SIAM J. Sci. Comput.*, 24(1):237–256, 2002. doi:10.1137/S106482759935808X.

[44] M. Kronbichler, T. Heister, and W. Bangerth. High accuracy mantle convection simulation through modern numerical methods. *Geophys. J. Int.*, 191(1):12–29, 2012. doi:10.1111/j.1365-246X.2012.05609.x.

[45] S. Le Borne. Hierarchical matrix preconditioners for the Oseen equations. *Comput. Vis. Sci.*, 11(3):147–157, 2008. doi:10.1007/s00791-007-0065-x.

[46] S. Le Borne. Preconditioned nullspace method for the two-dimensional Oseen problem. *SIAM J. Sci. Comput.*, 31(4):2494—2509, 2009. doi:10.1137/070691577.

[47] S. Le Borne and L. Rebholz. Preconditioning sparse grad-div/augmented Lagrangian stabilized saddle point systems. *Comput. Vis. Sci.*, 16(6):259–269, 2015. doi:10.1007/s00791-015-0236-0.

[48] R. Li and Y. Saad. Low-rank correction methods for algebraic domain decomposition preconditioners. *SIAM J. Matrix Anal. Appl.*, 38(3):807–828, 2017. doi:10.1137/16M110486X.

[49] R. Li, Y. Xi, and Y. Saad. Schur complement-based domain decomposition preconditioners with low-rank corrections. *Numer. Linear Algebra Appl.*, 23(4):706–729, 2016. doi:10.1002/nla.2051.

[50] P.-G. Martinsson, V. Rokhlin, and M. Tygert. A randomized algorithm for the decomposition of matrices. *Appl. Comput. Harmon. Anal.*, 30(1):47–68, 2011. doi:10.1016/j.acha.2010.02.003.

[51] P.-G. Martinsson and J. Tropp. Randomized numerical linear algebra: Foundations and algorithms. *Acta Numer.*, 29:403–572, 2020. doi:10.1017/S0962492920000021.

[52] M. F. Murphy, G. H. Golub, and A. J. Wathen. A note on preconditioning for indefinite linear systems. *SIAM J. on Sci. Comput.*, 21(6):1969–1972, 2000. doi:10.1137/S1064827599355153.

[53] M. A. Olshanskii and Y. V. Vassilevski. Pressure Schur complement preconditioners for the discrete Oseen problem. *SIAM J. Sci. Comput.*, 29(6):2686–2704, 2007. doi:10.1137/070679776.

[54] M. A. Olshanskii and A. Zhiliakov. Recycling augmented Lagrangian preconditioner in an incompressible fluid solver. *Numer. Linear Algebra Appl.*, 29(2):e2415, 2022. doi:10.1002/nla.2415.

[55] S. V. Patankar and D. B. Spalding. A calculation procedure for heat, mass and momentum transfer in three-dimensional parabolic flows. *Int. J. Heat Mass Transf.*, 15:1787–1806, 1972. doi:10.1016/0017-9310(72)90054-3.

[56] J. Pedlosky. *Geophysical fluid dynamics*. Springer study edition. Springer New York, NY, 2nd edition, 1987. doi:10.1007/978-1-4612-4650-3.

[57] M. Pernice and M. D. Tocci. A multigrid-preconditioned Newton–Krylov method for the incompressible Navier–Stokes equations. *SIAM J. Sci. Comput.*, 23(2):398–418, 2001. doi:10.1137/S1064827500372250.

[58] M. Rehman, C. Vuik, and G. Segal. SIMPLE-type preconditioners for the Oseen problem. *Internat. J. Numer. Methods Fluids*, 61(4):432–452, 2009. doi:10.1002/fld.1957.

[59] J. Rudi, G. Stadler, and O. Ghattas. Weighted BFBT preconditioner for Stokes flow problems with highly heterogeneous viscosity. *SIAM J. Sci. Comput.*, 39(5):S272–S297, 2017. doi:10.1137/16M108450X.

[60] Y. Saad. A flexible inner-outer preconditioned GMRES algorithm. *SIAM J. Sci. Comput.*, 14(2):461–469, 1993. doi:10.1137/0914028.

[61] Y. Saad. *Iterative methods for sparse linear systems*. SIAM, Philadelphia, PA, 2nd edition, 2003. doi:10.1137/1.9780898718003.

[62] Y. Saad and M. H. Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput.*, 7(3):856–869, 1986. doi:10.1137/0907058.

[63] M. Sala and M. Heroux. Robust algebraic preconditioners with IFPACK 3.0. Technical Report SAND-0662, Sandia National Laboratories, 2005. doi:https://doi.org/10.2172/1127118.

[64] Y.-H. Shih, G. Stadler, and F. Wechsung. Robust multigrid techniques for augmented Lagrangian preconditioning of incompressible Stokes equations with extreme viscosity variations. *SIAM J. Sci. Comput.*, 45(3):S27–S53, 2022. doi:10.1137/21M1430698.

[65] D. Silvester, H. Elman, D. Kay, and A. Wathen. Efficient preconditioning of the linearized Navier–Stokes equations for incompressible flow. *J. Comput. Appl. Math.*, 128(1):261–279, 2001. doi:10.1016/S0377-0427(00)00515-X.

[66] D. Silvester and A. Wathen. Fast iterative solution of stabilised Stokes systems part ii: Using general block preconditioners. *SIAM J. on Numer. Anal.*, 31(5):1352, 1994. doi:10.1137/0731070.

[67] H. S. Tang, R. D. Haynes, and G. Houzeaux. A review of domain decomposition methods for simulation of fluid flows: Concepts, algorithms, and applications. *Arch. Comput. Methods Eng.*, 28(3):841–873, 2021. doi:10.1007/s11831-019-09394-0.

[68] U. Trottenberg, C. Oosterlee, and A. Schüller. *Multigrid*. Elsevier Academic Press, London, 2001.

[69] R. S. Tuminaro and C. Tong. Parallel smoothed aggregation multigrid: Aggregation strategies on massively parallel machines. In *Proceedings of the 2000 ACM/IEEE Conference on Supercomputing*, SC '00, pages 5–es, USA, 2000. IEEE Computer Society. doi:10.1109/SC.2000.10008.

[70] C. Vuik and A. Saghir. The Krylov accelerated SIMPLE(R) method for incompressible flow. Technical Report 02-01, Delft University of Technology, 2002.

[71] C. Vuik, A. Saghir, and G. P. Boerstoel. The Krylov accelerated SIMPLE(R) method for flow problems in industrial furnaces. *Internat. J. Numer. Methods Fluids*, 33(7):1027–1040, 2000. doi:10.1002/1097-0363(20000815)33:7%3C1027::AID-FLD41%3E3.0.CO;2-S.

[72] F. Zanetti and L. Bergamaschi. Scalable block preconditioners for linearized Navier-Stokes equations at high Reynolds number. *Algorithms (Basel)*, 13(8):199, 2020. doi:10.3390/a13080199.

[73] Q. Zheng. Domain decomposition based preconditioner combined local low-rank approximation with global corrections. *Comput. Math. Appl.*, 114:41–46, 2022. doi:10.1016/j.camwa.2022.03.006.

[74] Q. Zheng, Y. Xi, and Y. Saad. A power Schur complement low-rank correction preconditioner for general sparse linear systems. *SIAM J. Matrix Anal. Appl.*, 42(2):659–682, 2021. doi:10.1137/20M1316445.

Symbols and acronyms

Acronyms

abbreviation	description
AMG	algebraic multigrid
CFL	Courant-Friedrichs-Lewy
CG	conjugate gradient method
DEAL.II	`C++` finite element library (Differential Eqn.s Analysis Library)
dof	degree of freedom
FGMRes	flexible generalized minimum residual (method)
GMRes	generalized minimum residual (method)
IC	incomplete Cholesky (factorization)
`IC(0)`	incomplete Cholesky factorization without fill-in
ILU	incomplete LU (factorization)
`ILU(0)`	incomplete LU factorization without fill-in
LSC	least-squares commutator
Pe	Péclet number
Re	Reynolds number
Ro	Rossby number
SIMPLE	semi-implicit method of pressure-linked equations
SVD	singular value decomposition

Mathematical symbols

symbol	description
$\mathbb{N}$	field of natural numbers including zero
$\mathbb{R}$	field of real numbers
X^{T}	transpose of the matrix X
X^{-1}	inverse of the matrix X
I_n	$n \times n$ identity matrix
$\mathrm{diag}(X)$	diagonal matrix that holds the diagonal of the matrix X
$\mathrm{rank}(X)$	rank of the matrix X
$\ker(X)$	kernel of the matrix X
$[X]_{ij}$	(i,j)th element of the matrix X
$[X]_j$	jth column of the matrix X
$[\mathbf{v}]_i$	ith element of the vector $\mathbf{v}$
$\|\cdot\|_F$	Frobenius norm
$\|\cdot\|_2$	Euclidean (vector) norm or spectral (matrix) norm
$\partial_t f$	partial time derivative of the function f
$\partial_{x_i} f$	partial derivative of the function f with respect to the variable x_i
∇x	gradient of the variable x
$\nabla \cdot x$	divergence of the variable x
$\boldsymbol{\varepsilon}(\mathbf{v})$	$= \frac{1}{2}(\nabla\mathbf{v} + (\nabla\mathbf{v})^{\mathrm{T}})$ strain-rate tensor

symbol	description
$\mathbf{v} \times \mathbf{w}$	cross products of the vectors $\mathbf{v}$, $\mathbf{w}$
$\mathbf{v} \otimes \mathbf{w}$	outer product of the vectors $\mathbf{v}$, $\mathbf{w}$
$\sum$	summation symbol
Ω	domain
Γ	boundary of the domain Ω
k_n	nth time step size
$\mathbf{u}$	velocity
p	pressure
T	temperature
n_u	number of velocity degrees of freedom
n_p	number of pressure degrees of freedom
n_T	number of temperature degrees of freedom
$\mathbf{n}$	outwards showing unit normal vector
$\subseteq$	subset of
$\cup$	union of sets
$\max\{\cdot,\cdot\}$	maximum of the arguments
$\mathcal{O}(\cdot)$	big O Landau symbol
$j \bmod k$	$= m$ for $j = mk + b$, $0 \leq b < k$, $k > 0$, $m, k, b \in \mathbb{N}$
$\lfloor j \rfloor$	floor function, greatest integer less than or equal to j
$\mathrm{nnz}(X)$	number of nonzero entries of the sparse matrix X
$\lambda(\cdot)$	eigenvalue of the argument

Further symbols

symbol	description
S	$= BA^{-1}B^{\mathrm{T}}$, Schur complement
$\widetilde{S}$	$= B\widehat{A}^{-1}B^{\mathrm{T}}$, approximate Schur complement (4.8)
P_{U}	block triangular preconditioner, see (3.35)
$P_{\mathrm{U},\alpha}$	block triangular preconditioner with relaxation, see (5.1)
$\widehat{A}^{-1}$	initial preconditioner for the upper left matrix block A
$\widehat{S}^{-1}$	initial preconditioner for the Schur complement S
$\widehat{S}^{-1}_{\mathrm{LSC}}$	LSC preconditioner, see (3.43)
$\widehat{S}^{-1}_{\mathrm{SIMPLE}}$	SIMPLE preconditioner, see (3.44)
$\widehat{A}^{-1}_{\mathrm{btri}}$	block triangular preconditioner, see (3.38)
$\widehat{S}^{-1}_{\mathrm{L}}$	preconditioner with left update, see (4.54)
$\widehat{S}^{-1}_{\mathrm{R}}$	preconditioner with right update, see (4.55)
$\widehat{S}^{-1}_{\mathrm{L},N}$, $\widehat{S}^{-1}_{\mathrm{R},N}$	left/right repeated update scheme, see (4.19a)

symbol	description
$\widehat{S}^{-1}_{\text{random,L},\alpha,r}$, $\widehat{S}^{-1}_{\text{random,R},\alpha,r}$	left/right update constructed with the randomized Algorithm 2.2, see (4.11)
$\widehat{S}^{-1}_{\text{Arnoldi,L},\alpha,r}$, $\widehat{S}^{-1}_{\text{Arnoldi,R},\alpha,r}$	left/right update constructed with the Arnoldi Algorithm 2.4 and projection, see (4.13)
$\widehat{S}^{-1}_{\text{ArnoldiP,L},\alpha,r}$, $\widehat{S}^{-1}_{\text{ArnoldiP,R},\alpha,r}$	left/right update constructed with the Arnoldi Algorithm 2.4, see (4.12)
$E_{\text{L},\alpha}$	$= \mathrm{I}_{n_p} - \alpha\widehat{S}^{-1}S$, see (4.5)
$E_{\text{R},\alpha}$	$= \mathrm{I}_{n_p} - \alpha S\widehat{S}^{-1}$, see (4.5)
$\widehat{E}_{\text{L}}$	low-rank approximation of $E_{\text{L},\alpha}$
$\widehat{E}_{\text{R}}$	low-rank approximation of $E_{\text{R},\alpha}$
$\widetilde{E}_{\text{L},\alpha}$	$= \mathrm{I}_{n_p} - \alpha\widehat{S}^{-1}\widetilde{S}$, see (4.9)
$\widetilde{E}_{\text{R},\alpha}$	$= \mathrm{I}_{n_p} - \alpha\widetilde{S}\widehat{S}^{-1}$, see (4.9)
$E_{\text{L,upd}}$	$= \mathrm{I}_{n_p} - \alpha\widehat{S}^{-1}_{\text{L}}S$
$E_{\text{R,upd}}$	$= \mathrm{I}_{n_p} - \alpha S\widehat{S}^{-1}_{\text{R}}$