Construction of hierarchical matrices for the preconditioning of the three-dimensional Navier-Stokes equations

Vom Promotionsausschuss der
Technischen Universität Hamburg

zur Erlangung des akademischen Grades

Doktor der Naturwissenschaften (Dr. rer. nat.)

genehmigte Dissertation (Monografie)

von
Jonas David Grams

aus
Dortmund

2025

1. Gutachterin: Prof. Dr. Sabine Le Borne
2. Gutachter: Prof. Dr. Steffen Börm

Tag der mündlichen Prüfung: 20. Dezember 2024

doi:https://doi.org/10.15480/882.14508
ORCiD: https://orcid.org/0009-0009-3521-9030

Dedicated to Henny and Friedrich.

Summary

In this thesis, we consider linear systems from the discretization of the Navier-Stokes equations. These systems of (non-linear) partial differential equations model the velocity and the pressure of a fluid or gas. In a first step, the Navier-Stokes equations have to be linearized to solve the system of differential equations. This linearization results in a sequence of linear partial differential equations, so-called Oseen equations.

The discretization of the Oseen equations with the finite element method yields a typically unsymmetric and indefinite (block) system matrix. The unfavorable (spectral) properties of the system matrix often result in a slow convergence of most iterative solvers. The performance of these solvers can be improved by preconditioning of the linear (block) system. A popular approach is the use of block preconditioners derived from a block LDU factorization of the system matrix. These preconditioners require easily invertible approximations to two matrices. One is the upper left block of the 2×2 block system matrix, the other one is the so-called Schur complement.

We consider the application of hierarchical matrices ($\mathcal{H}$-matrices) for (block) preconditioners of saddle point problems. This requires the representation of two sparse matrices as hierarchical matrices as well as the computation of a third matrix, the (approximate) Schur complement, with $\mathcal{H}$-matrix arithmetic. In this thesis, we pursue two different strategies to improve the time required for the preconditioner set-up.

The first is the construction of the hierarchical block structures of the sparse matrices. We present a new partitioning of the index sets allowing for a sparser block structure of the matrices required to compute the (approximate) Schur complement. This partitioning is based on the connection between the pressure and velocity field described by the underlying problem, the Oseen equations. Numerical experiments presented in this thesis illustrate the effectiveness of this block structure for the time-consuming task of computing an approximate Schur complement. In our tests, we observed a speed-up of about 40%–50% with the new partitioning. However, this approach also has a disadvantage. Although the numerical experiments nonetheless indicate the effectiveness of the partitioning, we introduce an adapted variant aiming to tackle the disadvantage while preserving the effectiveness. However, we present results that suggest that, although the adapted variant indeed tackles the disadvantages, is not as effective as the coupled clustering.

The second option we pursue is the utilization of different approaches to the hierarchical matrix ($\mathcal{H}$-matrix) arithmetic. Recently, two new variants of the (truncated) $\mathcal{H}$-matrix multiplication were introduced, both aiming to reduce the time required to perform the multiplication through a reordering of operations. Since the numerical experiments presented in the corresponding publication only considered $\mathcal{H}$-matrix approximation of (dense) matrices stemming from a discretization of integral operators, we evaluate the effectiveness of both variants for our model problem. For the first variant, the $\mathcal{H}$-matrix arithmetic with accumulated updates, we observed (if at all) only a negligible benefit for the total time compared to the standard arithmetic. However, a mix of the standard $\mathcal{H}$-matrix arithmetic and the arithmetic with accumulated updates yields a speed-up of about 10%–20% in our tests. The second variant, the $\mathcal{H}$-matrix arithmetic with sum-expressions, effectively reduces the time required to compute an approximate Schur complement through the use of less accurate but fast low-rank truncation methods.

Acknowledgements

In my almost five years at the institute of Mathematics of the Hamburg University of Technology I was accompanied by friends and colleagues without whom writing this thesis most certainly would have been more frustrating, if even possible. Therefore, I want to thank them here.

First, I am deeply thankful to my supervisor, Sabine Le Borne, for her invaluable advice, feedback and guidance. This has allowed me to learn a great deal from her and has been incredibly helpful in constantly improving my work.

I would also like to thank Rebekka Beddig, with whom I had the pleasure of sharing an office for almost three years and who has become a valued friend. For many fruitful and often encouraging conversations as well as for proofreading large parts of this thesis and her helpful comments I am deeply grateful to her.

Furthermore, I wish to thank my colleagues and friends Lina Fesefeldt and Thorben Abel from the Chair of Numerical Mathematics who also proofread parts of this thesis. After two years of working from home, they let me look forward to travel to work every day with the welcoming nature and were able to create an enjoyable working environment. The latter also extends to all other colleagues from the institute of Mathematics.

Finally, I thank my friends and family who accompanied and supported me on my journey, or even started it by convincing me to study mathematics.

Contents

Abbreviations and Acronyms **ix**

List of Symbols **x**

Notational conventions **xii**

1 Introduction **1**

2 Linear algebra basics **5**
2.1 Low-rank approximation techniques 5
2.1.1 Singular value decomposition 6
2.1.2 Lanczos bidiagonalization 7
2.1.3 Randomized low-rank approximation 10
2.2 Krylov subspace methods 13
2.2.1 GMRes method 13
2.2.2 BiCGStab method 15
2.2.3 Preconditioning 16
2.3 Saddle point problems 17
2.3.1 Properties of saddle point problems 17
2.3.2 Preconditioning of saddle point problems 18

3 Navier-Stokes equations **20**
3.1 Variational formulation for Oseen problems 21
3.2 Galerkin discretization 24
3.3 Finite element discretization 26
3.4 Upwinding 31

4 Hierarchical matrices **34**
4.1 Definitions 34
4.2 Clustering 40
4.2.1 Geometric bisection clustering 40
4.2.2 Domain decomposition clustering 44
4.3 Admissibility conditions 47
4.4 Arithmetic 52
4.4.1 $\mathcal{H}$-matrix-vector multiplication 52
4.4.2 (Standard) $\mathcal{H}$-matrix multiplication 53
4.4.3 $\mathcal{H}$-matrix inversion 62

4.4.4 $\mathcal{H}$-matrix factorizations 68

5 $\mathcal{H}$-LU preconditioning of saddle point problems 71
5.1 Uncoupled clustering 74
5.2 Coupled clustering 76
5.3 Coupled clustering with interface decomposition 84
5.4 Numerical results 89
5.4.1 Model problem 89
5.4.2 Results 91

6 Variants of the $\mathcal{H}$-matrix multiplication 103
6.1 $\mathcal{H}$-matrix multiplication with accumulated updates 104
6.1.1 The algorithm 104
6.1.2 Numerical experiments 107
6.2 $\mathcal{H}$-matrix multiplication with sum-expressions 112
6.2.1 The algorithm 112
6.2.2 Numerical experiments 116

7 Conclusion and outlook 121

Bibliography 124

Abbreviations and Acronyms

Abbreviation	**Description**	**First page**
BEM	Boundary element method	2
BiCG method	Biconjugate gradient method	15
BiCGStab method	Biconjugate gradient stabilized method	3
CG method	Conjugate gradient method	13
FEM	Finite element method	1
GMRes method	Generalized minimal residual method	3
$\mathcal{H}$-matrix	Hierarchical matrix	v
$\mathcal{H}$-LU factorization	$\mathcal{H}$-matrix LU factorization (cf. Section 4.4.4)	2
HODLR	Hierarchically off-diagonal low-rank	51
i.b.p.	Integration by parts	22
ILU	Incomplete LU	2
LSC	Least squares commutator	19
PBiCGStab method	Preconditioned BiCGStab method	90
PDE	Partial differential equation	1
SVD	Singular value decomposition	6

List of Symbols

Notation	Description
$\|\cdot\|_p$	p-norm of a vector for $1 \leq p$.
$\|\cdot\|_\infty$	Maximum norm of a vector.
$\text{dist}_2(\cdot,\cdot)$	Distance $\text{dist}_2(X,Y) := \inf_{x\in X, y\in Y} \|x-y\|_2$ with respect to the Euclidean norm $\|\cdot\|_2$.
$\text{dist}_\infty(\cdot,\cdot)$	Distance $\text{dist}_\infty(X,Y) := \inf_{x\in X, y\in Y} \|x-y\|_\infty$ with respect to the maximum norm $\|\cdot\|_\infty$.
$\text{diam}_2(\cdot)$	Diameter $\text{diam}_2(X) := \sup_{x,y\in X} \|x-y\|_2$ with respect to the Euclidean norm $\|\cdot\|_2$.
$\text{diam}_\infty(\cdot)$	Diameter $\text{diam}_\infty(X) := \sup_{x,y\in X} \|x-y\|_\infty$ with respect to the maximum norm $\|\cdot\|_\infty$.
supp(f)	Support $\text{supp}(f) = \overline{\{\mathbf{x}\in\Omega : f(\mathbf{x}) \neq 0\}}$ of a function $f : \Omega \to \mathbb{R}$ ($\Omega \subseteq \mathbb{R}^n$, $n \in \mathbb{N}$)
$\nabla \cdot \mathbf{u}$	Divergence of a function $\mathbf{u} : \mathbb{R}^d \to \mathbb{R}^d$.
∇f	The gradient of a function $f : \mathbb{R}^d \to \mathbb{R}$.
Δu or $\Delta \mathbf{u}$	Laplace operator applied to a scalar function $u : \mathbb{R}^d \to \mathbb{R}$ or componentwise to a vector-valued function $\mathbf{u} : \mathbb{R}^d \to \mathbb{R}^d$.
$\mathsf{L}^2(\Omega;\mathbb{R}^m)$, $\mathsf{L}^2(\Omega)$	Lebesgue space of square-integrable functions $u : \Omega \to \mathbb{R}^m$. If $m = 1$, $\mathbb{R}^m$ is omitted.
$\|f\|_{\mathsf{L}^2(\Omega)}$	Norm of the function space $\mathsf{L}^2(\Omega;\mathbb{R}^m)$ defined by $\|f\|^2_{\mathsf{L}^2(\Omega)} := \int_\Omega \|f(x)\|_2^2 \, dx$.
$\mathsf{H}^k(\Omega;\mathbb{R}^m)$, $\mathsf{H}^k(\Omega)$	k-th Sobolev space (space of k-times (weak) differentiable functions).
$\mathsf{H}^1_0(\Omega;\mathbb{R}^m;\Gamma)$, $\mathsf{H}^1_0(\Omega;\Gamma)$	Functions $\mathbf{u} \in \mathsf{H}^1(\Omega;\mathbb{R}^m)$ with $\mathbf{u}\|_\Gamma = 0$ for $\Gamma \subseteq \partial\Omega$. Note that $\mathbf{u}\|_{\partial\Omega}$ is typically referred to as trace operator mapping from $\mathsf{H}^k(\Omega;\mathbb{R}^m)$ to $\mathsf{L}^2(\partial\Omega;\mathbb{R}^m)$ (see, e.g., [7, p.1.6])
$\text{span}(v_1,\ldots,v_n)$	Linear span of vectors $v_1,\ldots,v_n \in V$ in a vector space V.

Notation	**Description**
$\mathbf{A}\vert_{t\times s}$	Restriction of a matrix $\mathbf{A} \in \mathbb{R}^{\mathcal{I}\times\mathcal{J}}$ to $\mathbb{R}^{t\times s}$ for a subset $t \times s \subseteq \mathcal{I} \times \mathcal{J}$.
$\mathbf{A}\vert^{\mathcal{I}\times\mathcal{J}}$	Natural embedding of a matrix $\mathbf{A} \in \mathbb{R}^{t\times s}$ into $\mathbb{R}^{\mathcal{I}\times\mathcal{J}}$ for a superset $\mathcal{I} \times \mathcal{J} \supseteq t \times s$.
$\rho(\mathbf{A})$	Spectral radius $\rho(\mathbf{A}) := \{\vert\lambda\vert : \lambda \in \sigma(\mathbf{A})\}$ of a matrix $\mathbf{A}$.
$\sigma(\mathbf{A})$	Spectrum $\sigma(\mathbf{A}) := \{\lambda \in \mathbb{C} : \lambda$ is an eigenvalue of $\mathbf{A}\}$ of a matrix $\mathbf{A}$.
$\mathcal{H}(\mathcal{T}_{\mathcal{I}\times\mathcal{J}}, k)$	Set of hierarchical matrices with local rank k with respect to a block cluster tree $\mathcal{T}_{\mathcal{I}\times\mathcal{J}}$ (cf. Definition 4.11).
$\mathcal{R}(t, s, k)$	Set of $t \times s$ rank-k factors (cf. Definition 2.1).
$\mathbb{R}^{t\times s}$	Set of $t \times s$ matrices for finite index sets t and s.

Notational conventions

Throughout this thesis, we will use the following naming and notational conventions.

- Matrices will be named with bold and uppercase letters, e.g., $\mathbf{M} \in \mathbb{R}^{t \times s}$ in most situations. We will differ from this convention only in a few cases when we have to name certain block matrices (e.g., in Section 2.3).
- We will use finite index sets to denote the rows and columns of matrices, e.g., in

  $$\mathbf{M} \in \mathbb{R}^{t \times s}.$$

  Additionally, we may replace one or both index sets with a positive integer if the corresponding set is of the form $\{1, \ldots, m\}$ for $m \in \mathbb{N}$. For example, we will write $\mathbf{M} \in \mathbb{R}^{m \times n}$ instead of $\mathbf{M} \in \mathbb{R}^{t \times s}$ with $t = \{1, \ldots, m\}$ and $s = \{1, \ldots, n\}$.
- Vectors from the vector space $\mathbb{R}^n$ ($n \in \mathbb{N}$) will be named with bold and lowercase letters, e.g., $\mathbf{x} \in \mathbb{R}^n$.
- Functions will be named with lowercase Latin or Greek letters. Multidimensional functions, i.e., functions mapping to $\mathbb{R}^n$ for some $n \in \mathbb{N}$, will be named with bold letters.

Chapter 1

Introduction

Many applications, as, e.g., climate simulations or aerodynamic simulations, require modeling (incompressible) fluid or gas flows. A popular tool for describing these flows are the Navier-Stokes equations. This system of non-linear partial differential equations (PDEs) models the velocity field and the pressure of a fluid or gas by describing two conservation laws: the conservation of momentum and the conservation of mass.

Although there are some examples for which a solution to the Navier-Stokes equations is known, one has, in general, to rely on numerical methods to compute approximate solutions.

First, the problem has to be linearized, e.g., with the so-called Picard iteration. Next, we have to discretize the problem, i.e., formulate a finite dimensional problem yielding an approximate solution. Well-known examples for discretization techniques are, e.g., the finite difference method, the finite volume method, or the finite element method (FEM).

In combination with the linearization, the discretization techniques mentioned above yield a sequence of linear systems one has to solve. Since the Navier-Stokes equations model both the velocity field and the pressure of a fluid in a bounded domain, these linear systems have a natural 2×2 block structure, i.e., are of the form

$$\mathcal{M} \begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix} := \begin{pmatrix} \mathbf{F} & \mathbf{B}^{\mathsf{T}} \\ \mathbf{B} & \mathbf{0} \end{pmatrix} \begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix} = \begin{pmatrix} \mathbf{r} \\ \mathbf{s} \end{pmatrix}. \tag{1.1}$$

This kind of linear 2×2 block system is called saddle point problem.

Achieving a sufficient accuracy of the discretization leads to linear systems with a large number of degrees of freedom, especially for three-dimensional problems. Although the system matrix $\mathcal{M}$ is sparse, solving the system (1.1) with direct methods, as, e.g., Gauß elimination, is not feasible in a proper amount of time for a large number of degrees of freedom since they typically require $\mathcal{O}(n^3)$ floating point operations where $n \in \mathbb{N}$ is the number of degrees of freedom (see, e.g., [18, §3.2]). Instead, one has to rely on (iterative) approximate methods. A popular choice for these are the so-called Krylov subspace methods. These iterative methods compute the iterates by projecting the error onto certain subspaces, the so-called Krylov subspaces. The iterates are then updated by the scaled projection of the error. An advantage of these methods is that they only use vector and matrix-vector arithmetic and therefore require $\mathcal{O}(n^2)$, or in case of sparse matrices $\mathcal{O}(n)$ operations, per iteration step. Hence, Krylov subspace methods are efficient if they converge within few iteration steps.

However, saddle point matrices, as the system matrix in (1.1), are typically unsymmetric and indefinite, especially the ones obtained from the discretization of the Navier-Stokes equations, and the lack of symmetry and the indefiniteness of the problem on the other hand restrict the choice of methods. Typically, one has to decide between a minimization of the error and a short-term recursion for the update of the approximate solution.

Methods that minimize the error in the Krylov space have to store information from the previous update steps. Especially for large systems, this can be problematic since the number of iteration steps to reach a certain accuracy often grows with the size of the system, and thus the storage requirement may exceed the available memory.

Methods with short-term recursions on the other hand do not have to store information from all previous iteration steps. However, since they do not minimize the error they typically converge slower.

Although spectral properties of the system matrix may only describe the worst-case behavior for both approaches (see, e.g., [9, 35]), saddle point problems as in (1.1) tend to require a larger number of iteration steps to achieve an acceptable residual error. A popular approach to reduce the required number of iteration steps is preconditioning, i.e., solving

$$\mathcal{P}^{-1}\mathcal{M}\begin{pmatrix}\mathbf{x}\\ \mathbf{y}\end{pmatrix} = \mathcal{P}^{-1}\begin{pmatrix}\mathbf{r}\\ \mathbf{s}\end{pmatrix}$$

instead of (1.1) for a suitable matrix $\mathcal{P}$. For fluid flow problems block preconditioners obtained from a block factorization of the system matrix are a popular and often effective choice (cf. [4, 30]). These require easily invertible approximations to the matrix block $\mathbf{F}$ and the so-called Schur complement $\mathbf{S} = \mathbf{B}\mathbf{F}^{-1}\mathbf{B}^{\mathsf{T}}$.

In this thesis, we will consider approximate triangular factorizations of $\mathbf{F}$ and $\mathbf{S}$ as such approximations. Classical examples for such approximate triangular factorizations are the (sparse) incomplete LU (ILU) factorizations which exist in various variants such as ILU(p) for $p \in \mathbb{N}_0$ or ILUT (see, e.g., [35, §10.3]). However, while $\mathbf{F}$ is sparse and these factorizations would be applicable, this is not the case for the Schur complement since it is typically dense due to the presence of the (dense) inverse of $\mathbf{F}$ in its definition.

A data-sparse alternative to sparse factorizations are hierarchical matrix ($\mathcal{H}$-matrix) factorizations. $\mathcal{H}$-matrices were originally introduced in [25] as a data-sparse matrix format for discretizations of integral operators. In comparison to dense matrices, they offer almost linear memory and computational complexity. Since they were introduced, they were also considered for other applications, as, e.g., the data-sparse representation of matrices from the discretizations with the boundary element method (BEM) (cf. [1]) or approximate inverses or factorizations of FEM matrices (cf. [2, 3, 16]). They were even already considered for the preconditioning of Oseen equations (cf. [31]). The approximation to $\mathbf{F}$ in this case is an (approximate) $\mathcal{H}$-matrix LU factorization ($\mathcal{H}$-LU factorization) $\mathbf{F} \approx \mathbf{L}_\mathbf{F}\mathbf{U}_\mathbf{F}$. For the Schur complement, we first need a suitable representation. We compute an approximate representation with $\mathcal{H}$-matrix arithmetic, i.e.,

$$\mathbf{S} \approx \mathbf{S}_\mathcal{H} = \left(\mathbf{B}\mathbf{U}_\mathbf{F}^{-1}\right)\left(\mathbf{L}_\mathbf{F}^{-1}\mathbf{B}^{\mathsf{T}}\right)$$

where all solves with triangular matrices and the multiplication are done with the (approximate) $\mathcal{H}$-matrix arithmetic. From this approximate representation $\mathbf{S}_\mathcal{H}$, we can then compute an $\mathcal{H}$-LU factorization $\mathbf{S} \approx \mathbf{S}_\mathcal{H} \approx \mathbf{L}_\mathbf{S}\mathbf{U}_\mathbf{S}$.

However, these approximations to $\mathbf{F}$ and $\mathbf{S}$ have a major disadvantage. Although all arithmetic operations of $\mathcal{H}$-matrices can be realized with a nearly linear computational complexity, the set-up is a time-consuming task in view of rather large constants in the complexity estimates. In [31], we can observe that the set-up of the preconditioner is dominated by the explicit computation of the (approximate) Schur complement $\mathbf{S}_{\mathcal{H}}$, which requires two solves with triangular $\mathcal{H}$-matrices and a multiplication with $\mathcal{H}$-matrices. As we will see later, the estimates of the computational complexity of these operations depend on the block structure of the involved matrices. Another thing to note here is that $\mathcal{H}$-matrix arithmetics heavily rely on rank truncations in order to control the block-wise ranks of the results. In this thesis, we pursue two options for an improvement of the set-up time of the $\mathcal{H}$-LU factorizations:

1. In [31], the index sets were divided to achieve a suitable representation of an (approximate) Schur complement $\mathbf{S}_{\mathcal{H}}$ and to achieve a block structure of $\mathbf{F}$ optimized for the computation of an $\mathcal{H}$-LU factorization. We will introduce a new approach for the division of the index sets, optimized for the time-consuming $\mathcal{H}$-matrix update in the computation of an (approximate) Schur complement $\mathbf{S}_{\mathcal{H}}$.

2. Recently, [5] and [11] proposed different concepts for the $\mathcal{H}$-matrix multiplication improving the standard algorithm originally introduced in [25] by changing the truncation operations. While [5] reduces the number of necessary truncations, [11] reformulates the multiplication algorithm to allow for an easy application of matrix-vector operations based truncation methods. We will examine the applicability of both variants to our problem, the set-up of (hierarchical) block preconditioners for the saddle point problem (1.1), by analyzing numerical experiments.

The remainder of this thesis is structured as follows. First, we will discuss some basics from numerical linear algebra in Chapter 2. First, in Section 2.1 we will explore some low-rank approximation techniques. This task is an important part of many $\mathcal{H}$-matrix operations, especially the multiplication and the factorization, in order to control the rank of the resulting blocks. Additionally, we will briefly describe two Krylov subspace methods for unsymmetric systems, namely the generalized minimal residual method (GMRes method) and the biconjugate gradient stabilized method (BiCGStab method) in Section 2.2, as well as basic results regarding saddle point systems and the preconditioning of such in Section 2.3.

Next, in Chapter 3, we will briefly discuss the finite element discretization of Oseen problems, which are a result of the linearization of the Navier-Stokes equations.

Then, in Chapter 4, we will give a detailed description of hierarchical matrices. This includes methods for the generation of the block structures for the application we explore in this thesis, namely cluster strategies in Section 4.2 and admissibility conditions in Section 4.3. We also present standard (approximate) arithmetic operations for $\mathcal{H}$-matrices such as the multiplication and the LU factorization of $\mathcal{H}$-matrices in Section 4.4.

Following the description of basics in Chapters 2 to 4, we will introduce our new approach to the construction of the block structures for the matrices necessary for the preconditioner set-up. We start with a brief description of the approach proposed in [31] in Section 5.1 before introducing and analyzing our new approach in Section 5.2. We already presented this approach in [19] but expand these results here by a further analysis.

Additionally, we will introduce an adaption of our block structure in Section 5.3 which aims to tackle some disadvantages of the approach presented in prior section Section 5.2. Finally, we will compare all approaches with numerical experiments which we will present in Section 5.4.

While Chapter 5 is about the first option for improvements of the set-up time of the $\mathcal{H}$-LU factorizations, an optimization of the block structures of the involved $\mathcal{H}$-matrices, we will examine the variants of the $\mathcal{H}$-matrix multiplication in Chapter 6. We note that the numerical tests for both variants in [5] and [11], respectively, were done with (typically dense) matrices from a discretization via the boundary element method. Hence, we review their applicability to our problem with numerical experiments, since the blocks of the saddle point problem (1.1) are typically sparse. First, we briefly describe the $\mathcal{H}$-matrix multiplication with accumulated updates from [5] followed by a presentation of the results of the numerical experiments in Section 6.1. Then, in Section 6.2, we will briefly describe the $\mathcal{H}$-matrix multiplication with sum-expressions from [11] also followed by the presentation of the results from the numerical experiments.

Finally, we conclude this thesis with a brief summary of the main discoveries from Chapters 5 and 6 and open tasks for future work in Chapter 7.

Chapter 2

Linear algebra basics

The main part of this thesis addresses the preconditioning of saddle point problems utilizing hierarchical matrices. This chapter deals with the necessary linear algebra background.

Arithmetic operations with hierarchical matrices require the approximation of matrices by low-rank matrices (cf. Section 4.4). Consequently, Section 2.1 introduces some notation and discusses a selection of low-rank approximation techniques used in this thesis.

Furthermore, we will consider unsymmetric saddle point problems which require efficient solvers, e.g., the Krylov subspace methods examined in Section 2.2. These methods tend to be less efficient for large-scale indefinite systems and can be enhanced by using a preconditioner.

Last, we will examine linear saddle point problems in Section 2.3. Since these systems are typically indefinite, we will also consider block preconditioning techniques for saddle point problems.

2.1 Low-rank approximation techniques

An important concept in this thesis is the approximation of matrices by matrices of a lower rank, i.e., for a matrix $\mathbf{M} \in \mathbb{R}^{m\times n}$ and a rank $k < \min(m,n)$, find $\mathbf{A}_\mathbf{M} \in \mathbb{R}^{m\times k}$ and $\mathbf{B}_\mathbf{M} \in \mathbb{R}^{n\times k}$ with $\mathbf{M} \approx \mathbf{A}_\mathbf{M}\mathbf{B}_\mathbf{M}^\mathsf{T}$. We define the following set to simplify the notation.

Definition 2.1 (Rank-k factors)**.** *Let t and s be finite index sets. Then we define the set of $t \times s$* rank-k factors *by*

$$\mathcal{R}(t,s,k) := \{(\mathbf{A},\mathbf{B}) : \mathbf{A} \in \mathbb{R}^{t\times k}, \mathbf{B} \in \mathbb{R}^{s\times k}\}.$$

Note that a matrix $\mathbf{M} \in \mathbb{R}^{t\times s}$ has at most rank k if and only if there are rank-k factors $(\mathbf{A}_\mathbf{M}, \mathbf{B}_\mathbf{M}) \in \mathcal{R}(t,s,k)$ with

$$\mathbf{M} = \mathbf{A}_\mathbf{M}\mathbf{B}_\mathbf{M}^\mathsf{T}. \tag{2.1}$$

While the singular value decomposition of a matrix, presented in Section 2.1.1, can be used to construct a best approximation of any given rank, this is a computationally expensive task. Computing the singular value decomposition of an $n \times n$ matrix ($n \in \mathbb{N}$) requires $\mathcal{O}(n^3)$ operations (see Remark 2.2). Additionally, the matrix is required explicitly which is either not always possible or may further increase the memory requirements and the computational complexity.

In addition to the approximation utilizing the SVD, we will briefly present two methods, both relying upon matrix-vector operations:

1. low-rank approximation utilizing the Lanczos bidiagonalization (Section 2.1.2),
2. randomized low-rank approximation (Section 2.1.3).

2.1.1 Singular value decomposition

Let $\mathbf{M} \in \mathbb{R}^{m\times n}$. Without loss of generality we assume $m \geq n$. Then the singular value decomposition (SVD) of $\mathbf{M}$ is given by

$$\mathbf{M} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\mathsf{T}}$$

with $\mathbf{U} \in \mathbb{R}^{m\times m}, \mathbf{V} \in \mathbb{R}^{n\times n}$ orthogonal, $\mathbf{\Sigma} \in \mathbb{R}^{m\times n}$ diagonal such that the diagonal entries σ_i of $\mathbf{\Sigma}$ are non-negative and in descending order $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0$. Then

$$\widetilde{\mathbf{M}} := \mathbf{U}|_{m\times k}\mathbf{\Sigma}|_{k\times k}(\mathbf{V}|_{n\times k})^{\mathsf{T}}$$

is a rank-k best approximation of $\mathbf{M}$ in the spectral norm and the Frobenius norm (cf. [18, Thm. 2.4.8]). The approximation error with respect to the spectral and Frobenius norm, respectively, can be determined with the remaining singular values:

$$||\mathbf{M} - \widetilde{\mathbf{M}}||_2 = \sigma_{k+1} \quad \text{and} \quad \|\mathbf{M} - \widetilde{\mathbf{M}}\|_{\mathrm{F}} = \sqrt{\sum_{i=k+1}^{\min(n,m)} \sigma_i^2}.$$

Rank-k factors of $\widetilde{\mathbf{M}}$ are given, e.g., via

$$\mathbf{A}_k := \mathbf{U}|_{m\times k}, \quad \mathbf{B}_k := \mathbf{V}|_{n\times k}\mathbf{\Sigma}|_{k\times k}. \tag{2.2}$$

Remark 2.2 (Computational complexity)**.** *The computational complexity for the SVD can be estimated by*

$$W_{\mathrm{SVD}}(n,m) \leq 4m^2n + 8mn^2 + 9n^3 \leq 21n^3, \tag{2.3}$$

as discussed in [18, §8.6.3]. We define the mapping

$$\mathfrak{T}_{\mathrm{SVD},k} : \mathbb{R}^{m\times n} \to \mathcal{R}(m,n,k)$$

with

$$\mathfrak{T}_{\mathrm{SVD},k}(\mathbf{M}) = (\mathbf{A}_k, \mathbf{B}_k)$$

and $\mathbf{A}_k, \mathbf{B}_k$ from (2.2). By combining (2.3) with the cost for the multiplication with the diagonal matrix $\mathbf{\Sigma}|_{k\times k}$ in (2.2), we can estimate the computational complexity of the truncation $\mathfrak{T}_{\mathrm{SVD},k}(\mathbf{M})$ by

$$W_{\mathfrak{T},\mathrm{SVD}}(m,n,k) \leq 4m^2n + 8mn^2 + 9n^3 + kn \leq 21n^3 + kn. \tag{2.4}$$

Remark 2.3 (Thin SVD)**.** *If* $\mathbf{M} = (\mathbf{A}^\ell, \mathbf{B}^\ell)$ *for* $(\mathbf{A}^\ell, \mathbf{B}^\ell) \in \mathcal{R}(m, n, \ell)$*, i.e.,* $\mathbf{M}$ *already has a low rank* $\ell < m, n$*, the size of the SVD matrices can be reduced by using QR factorizations to compute a thin SVD*

$$\mathbf{M} = \widetilde{\mathbf{U}}\widetilde{\mathbf{\Sigma}}\widetilde{\mathbf{V}}^\mathsf{T}$$

with $\widetilde{\mathbf{U}} \in \mathbb{R}^{m\times\ell}, \widetilde{\mathbf{V}} \in \mathbb{R}^{n\times\ell}, \widetilde{\mathbf{\Sigma}} \in \mathbb{R}^{\ell\times\ell}$ *(cf. [28, §I.2.5]). As shown in [28, §I.2.5], we can estimate the computational complexity of finding the thin SVD by*

$$W_{\text{thin}}(m, n, \ell) \leq 6\ell^2(m + n) + \frac{65}{3}\ell^3. \tag{2.5}$$

A low-rank representation of the rank-k best approximation can be determined even more efficiently, e.g., by using

$$\mathbf{A}_k^\ell := \widetilde{\mathbf{U}}|_{m\times k}, \quad \mathbf{B}_k^\ell := \widetilde{\mathbf{V}}|_{n\times k}\widetilde{\mathbf{\Sigma}}|_{k\times k}. \tag{2.6}$$

As in Remark 2.2, we define the truncation mapping

$$\mathfrak{T}^\ell_{\text{thin},k} : \mathcal{R}(m, n, \ell) \to \mathcal{R}(m, n, k) \tag{2.7}$$

with

$$\mathfrak{T}^\ell_{\text{thin},k}(\mathbf{M}) = (\mathbf{A}_k^\ell, \mathbf{B}_k^\ell)$$

with $\mathbf{A}_k, \mathbf{B}_k$ *from (2.6). By combining* (2.5) *with the cost for the multiplication with the diagonal matrix* $\mathbf{\Sigma}|_{k\times k}$ *in* (2.6)*, we can estimate the computational complexity of the truncation* $\mathfrak{T}^\ell_{\text{thin},k}(\mathbf{M})$ *by (cf. [28, Remark 2.18])*

$$W_{\mathfrak{T},\text{thin}}(m, n, \ell, k) \leq 6\ell^2(n + m) + 22\ell^3.$$

2.1.2 Lanczos bidiagonalization

A popular approach to compute the SVD of a matrix $\mathbf{M} \in \mathbb{R}^{m\times n}$ is the sparsification of the matrix before computing the singular values. The idea of this approach is to compute an orthogonally equivalent bidiagonal matrix, i.e., (assuming without loss of generality $m \geq n$) constructing

$$\mathbf{T} = \begin{pmatrix} \alpha_1 & \beta_1 & & & \\ 0 & \alpha_2 & \beta_2 & & \\ \vdots & \ddots & \ddots & \ddots & \\ 0 & & 0 & \alpha_{n-1} & \beta_{n-1} \\ 0 & & & 0 & \alpha_n \end{pmatrix} \in \mathbb{R}^{n\times n}$$

with

$$\mathbf{QTW}^\mathsf{T} = \mathbf{M}$$

for suitable matrices $\mathbf{Q} \in \mathbb{R}^{m\times n}$ and $\mathbf{W} \in \mathbb{R}^{n\times n}$ with orthonormal columns, as first described by Golub and Kahan [17]. Then the matrix $\mathbf{T}^\mathsf{T}\mathbf{T}$ is a symmetric tridiagonal

matrix. This kind of matrices allows for an efficient computation of the eigenvalues and eigenvectors utilizing Givens rotations. Hence, the singular values and singular vectors of $\mathbf{T}$ (and $\mathbf{M}$) can be computed efficiently. The orthogonal matrices $\mathbf{Q}$ and $\mathbf{W}$ can be computed column-wise, as it is done by the Golub-Kahan-Lanczos bidiagonalization algorithm (see, e.g., [18, Algorithm 10.4.1]). By stopping at step $k \leq n$, we obtain $\mathbf{Q}_k \in \mathbb{R}^{m\times k}$ and $\mathbf{W}_k \in \mathbb{R}^{n\times k}$ with orthonormal columns as well as

$$\mathbf{T}_k = \begin{pmatrix} \alpha_1 & \beta_1 & & & \\ & \alpha_2 & \beta_2 & & \\ & & \ddots & \ddots & \\ & & & \alpha_{k-1} & \beta_{k-1} \\ & & & & \alpha_k \end{pmatrix} \in \mathbb{R}^{k\times k}.$$

Let $\mathbf{A}_k := \mathbf{Q}_k \in \mathbb{R}^{m\times k}$ and $\mathbf{B}_k := \mathbf{T}_k\mathbf{W}_k \in \mathbb{R}^{n\times k}$. Then, we have $(\mathbf{A}_k, \mathbf{B}_k) \in \mathcal{R}(m, n, k)$. The singular values of $\mathbf{T}_k$ yield a good approximation to the largest singular values of $\mathbf{M}$ (cf. [18, §10.4.1]). Hence, [36] proposed to use $\mathbf{A}_k\mathbf{B}_k^\mathsf{T}$ as a rank-k approximation for $\mathbf{M}$. It is shown that the error

$$\|\mathbf{M} - \mathbf{A}_k\mathbf{B}_k^\mathsf{T}\|_\mathrm{F}$$

approaches the error of a rank-j best approximation for $j < k$ with increasing k. Numerical tests in [36] indicate that just slightly larger k suffice in most cases.

The computation of $(\mathbf{A}_k, \mathbf{B}_k)$ follows [11] and is summarized in Algorithm 2.1. The matrices $\mathbf{Q}_k, \mathbf{W}_k$ and $\mathbf{T}_k$ are computed by the Golub-Kahan-Lanczos algorithm stopped after k steps in lines 2–12.

Remark 2.4 (Computational complexity)**.** *As before (see Remark 2.2), we define the truncation mapping*

$$\mathfrak{T}_{\mathrm{Lanczos},k} : \mathbb{R}^{m\times n} \to \mathcal{R}(m, n, k).$$

Each iteration of the loop (lines 5–12) in Algorithm 2.1 requires

- *two matrix-vector multiplications with* $\mathbf{M}$ *and* $\mathbf{M}^\mathsf{T}$*, respectively,*
- *two scaled vector additions requiring* $2m$ *operations each,*
- *two scalings of vectors requiring* m *operations each,*
- *and two norm computations requiring* $2m$ *operations each.*

Additionally, the algorithm requires the multiplication of an $m \times k$ *matrix by a* $k \times k$ *bidiagonal matrix requiring* $3m(k-1)+m$ *operations. Therefore, the computational complexity of the low-rank approximation using Lanczos bidiagonalization is given by*

$$\begin{aligned} W_\mathrm{Lanczos}(\mathbf{M}, k) &= 2kW_\mathrm{MV}(\mathbf{M}) + 4km + 2km + 4km + 3km - 2m \\ &= 2kW_\mathrm{MV}(\mathbf{M}) + 13km - 2m \end{aligned}$$

where $W_\mathrm{MV}(\mathbf{M})$ *is the computational complexity required for multiplying* $\mathbf{M}$ *or its transpose with a vector.*

Algorithm 2.1 Low-rank approximation via Lanczos bidiagonalization

Input: Matrix $\mathbf{M} \in \mathbb{R}^{m \times n}$, rank $k \in \mathbb{N}$
Output: Rank-k factors $(\mathbf{A}_k, \mathbf{B}_k) \in \mathcal{R}(m, n, k)$ such that $\mathbf{A}_k \mathbf{B}_k^{\mathsf{T}}$ approximates $\mathbf{M}$

function `BiLanczosLowRankApproximation`($\mathbf{M}$, k)
  Generate a random start vector $\mathbf{w}_1 \in \mathbb{R}^n$
  $\mathbf{q}_0 \leftarrow \mathbf{0}$
  $\beta_0 \leftarrow 0$
  for $j = 1, \ldots, k$ **do**
    $\mathbf{q}_j \leftarrow \mathbf{M}\mathbf{w}_j - \beta_{j-1}\mathbf{q}_{j-1}$
    $\alpha_j \leftarrow \|\mathbf{q}_j\|_2$
    $\mathbf{q}_j \leftarrow \mathbf{q}_j / \alpha_j$
    $\mathbf{w}_{j+1} \leftarrow \mathbf{M}^{\mathsf{T}} - \alpha_j \mathbf{w}_j$
    $\beta_j \leftarrow \|\mathbf{w}_{j+1}\|_2$
    $\mathbf{w}_{j+1} \leftarrow \mathbf{w}_{j+1} / \beta_j$
  end for
  $\boldsymbol{\alpha} \leftarrow (\alpha_1, \ldots, \alpha_k)$
  $\boldsymbol{\beta} \leftarrow (\beta_1, \ldots, \beta_{k-1})$
  $\mathbf{T}_k \leftarrow \mathrm{bidiag}(\boldsymbol{\alpha}, \boldsymbol{\beta})$
  $\mathbf{Q}_k \leftarrow (\mathbf{q}_1 \cdots \mathbf{q}_k) \in \mathbb{R}^{m \times k}$
  $\mathbf{W}_k \leftarrow (\mathbf{w}_1 \cdots \mathbf{w}_k) \in \mathbb{R}^{n \times k}$
  $\mathbf{A}_k \leftarrow \mathbf{Q}_k \mathbf{T}_k, \quad \mathbf{B}_k \leftarrow \mathbf{W}_k$
  return $(\mathbf{A}_k, \mathbf{B}_k)$
end function

An advantage of this approach over the computation of a full SVD is that we do not need the matrix $\mathbf{M}$ explicitly but only its action on vectors. Additionally, we do not have to solve an eigenvalue problem which reduces the computational costs.

If we require an approximation with a fixed error bound instead of a fixed rank, Algorithm 2.1 can be adapted using a stopping criterion as, e.g., developed in [36].

2.1.3 Randomized low-rank approximation

Computing a rank-k approximation $\widetilde{\mathbf{M}}$ of an arbitrary matrix $\mathbf{M} \in \mathbb{R}^{m\times n}$ via its SVD can be computationally expensive (see Remark 2.2) since one has to compute the full SVD first. While there exists a wide range of deterministic algorithms avoiding solving an eigenvalue problem as, e.g., the method presented in Section 2.1.2, randomized approaches have become popular in the last few years. The key idea of the randomized low-rank approximation is to reduce the problem dimension by sampling the range of the matrix with a few random vectors, i.e., generating random vectors $\boldsymbol{\omega}_1, \dots, \boldsymbol{\omega}_k \in \mathbb{R}^n$ and considering

$$\mathbf{Y} := \mathbf{M}\boldsymbol{\Omega} \in \mathbb{R}^{m\times k}$$

with $\boldsymbol{\Omega} = (\boldsymbol{\omega}_1 \cdots \boldsymbol{\omega}_k) \in \mathbb{R}^{n\times k}$ instead of $\mathbf{M}$. Then, we can obtain a rank-k approximation of $\mathbf{M}$ by constructing a matrix $\mathbf{Q} \in \mathbb{R}^{m\times k}$ whose columns form an orthonormal basis of the range of $\mathbf{Y}$. If $\mathbf{M}$ has rank k, this basis is a basis of the range of $\mathbf{M}$, and we obtain a rank-k representation of $\mathbf{M}$ with

$$\widetilde{\mathbf{M}} := \mathbf{Q}\mathbf{Q}^\mathsf{T}\mathbf{M} = \mathbf{A}_k\mathbf{B}_k$$

where $\mathbf{A}_k = \mathbf{Q}$ and $\mathbf{B}_k = \mathbf{Q}^\mathsf{T}\mathbf{M}$. If, on the other hand, $k < \operatorname{rank}(\mathbf{M})$, it is very likely that $\widetilde{\mathbf{M}}$ is a good rank-k approximation of $\mathbf{M}$ (depending on the singular values of $\mathbf{M}$). We refer to [29, 32] for a detailed error analysis and discussion on the probabilities.

The matrix $\mathbf{Q}$ can be computed incrementally with the Gram-Schmidt orthogonalization method. However, the standard Gram-Schmidt method may lead to a severe loss of orthogonality among the computed columns of $\mathbf{Q}$ (see, e.g., [18, §5.2.7]). Two popular improvements are

1. The so-called modified Gram-Schmidt method obtained by a rearrangement of the computations,
2. Additional re-orthogonalizing steps for the computed columns.

Algorithm 2.2 summarizes the randomized low-rank approximation with the Gram-Schmidt method using additional orthogonalization steps.

Remark 2.5 (Improvements)**.** *The results of the method above can be improved in two ways:*

1. *by a slight oversampling, i.e., using $k+p$ instead of k random vectors for some $p \in \mathbb{N}$, to increase the probability of a small error,*
2. *by combining the standard approximation method described above with a power iteration, i.e., using*

   $$\mathbf{Y} = (\mathbf{M}\mathbf{M}^\mathsf{T})^q\mathbf{M}\boldsymbol{\Omega}$$

   for some $q \in \mathbb{N}$,

Algorithm 2.2 Adaptive randomized low-rank approximation

Input: Matrix $\mathbf{M} \in \mathbb{R}^{m \times n}$, rank k, number of orthogonalization steps γ.
Output: Rank-k factors $(\mathbf{A}_k, \mathbf{B}_k) \in \mathcal{R}(m, n, k)$

function `RandomLowRank`($\mathbf{M}$, k, γ)
  Generate a random vector $\boldsymbol{\omega}_1 \in \mathbb{R}^n$
  $\mathbf{y}_1 \leftarrow \mathbf{M}\boldsymbol{\omega}_1$
  $\mathbf{q}_1 \leftarrow \mathbf{y}_1 / \|\mathbf{y}_1\|_2$
  $\mathbf{Q}_1 \leftarrow \begin{pmatrix} \mathbf{q}_1 \end{pmatrix} \in \mathbb{R}^{m \times 1}$
  for $j = 1, \ldots, k-1$ **do**
    Generate a random vector $\boldsymbol{\omega}_{j+1} \in \mathbb{R}^n$
    $\mathbf{y}_{j+1} \leftarrow \mathbf{M}\boldsymbol{\omega}_{j+1}$
    for $\ell = 1, \ldots, \gamma$ **do**
      $\mathbf{y}_{j+1} \leftarrow \mathbf{y}_{j+1} - \mathbf{Q}_j \mathbf{Q}_j^\mathsf{T} \mathbf{y}_{j+1}$
    end for
    $\mathbf{q}_{j+1} \leftarrow \mathbf{y}_{j+1} / \|\mathbf{y}_{j+1}\|$
    $\mathbf{Q}_{j+1} \leftarrow \begin{pmatrix} \mathbf{Q}_j & q_{j+1} \end{pmatrix} \in \mathbb{R}^{m \times (j+1)}$
  end for
  $\mathbf{A}_k \leftarrow \mathbf{Q}_k$
  $\mathbf{B}_k \leftarrow \mathbf{M}^\mathsf{T} \mathbf{Q}_k$
  return $(\mathbf{A}_k, \mathbf{B}_k)$
end function

3. *the outer loop (lines 6–14) of Algorithm 2.2 computes a QR decomposition of*

$$\mathbf{Y} = \mathbf{M}\begin{pmatrix}\boldsymbol{\omega}_1 & \dots & \boldsymbol{\omega}_k\end{pmatrix}$$

using the Gram-Schmidt method with γ orthogonalization steps. This can be replaced by any other method for the construction of a QR factorization.

The second approach is especially useful for matrices with slowly decaying singular values (cf. [29, §4.5]).

Remark 2.6 (Computational complexity)**.** *As before, we define the truncation mapping*

$$\mathfrak{T}_{\mathrm{rand},k} : \mathbb{R}^{m\times n} \to \mathcal{R}(m,n,k)$$

with

$$\mathfrak{T}_{\mathrm{rand},k}(\mathbf{M}) = (\mathbf{Q}, \mathbf{Q}^{\mathsf{T}}\mathbf{M}).$$

The computational complexity of Algorithm 2.2 can be estimated as follows. Step j of the outer for-loop (lines 6–14) of Algorithm 2.2 requires

- *one matrix-vector multiplication with $\mathbf{M}$,*
- *γ matrix-vector multiplications with $\mathbf{Q}_j^{\mathsf{T}}$ (lines 9–11) requiring $j(2m-1)$ operations,*
- *γ matrix-vector multiplications with $\mathbf{Q}_j$ (lines 9–11) requiring $m(2j-1)$ operations,*
- *γ vector subtractions requiring m operations,*
- *one norm computation requiring $2m$ operations,*
- *one vector scaling requiring m operations.*

Additionally, the algorithm requires one extra matrix-vector multiplication with $\mathbf{M}$, one extra norm computation, and one extra vector scaling in the beginning as well as one matrix multiplication of $\mathbf{M}^{\mathsf{T}}$ with $\mathbf{Q}_k$. The matrix multiplication can be performed with k matrix-vector multiplications with $\mathbf{M}^{\mathsf{T}}$. Therefore, the computational complexity of Algorithm 2.2 is

$$\begin{aligned}
W_{\mathrm{rand}}(\mathbf{M},k,\gamma) &= \sum_{j=1}^{k-1}(W_{\mathrm{MV}}(\mathbf{M}) + \gamma(2jm - j + 2jm - m + m + 2m + m)) \\
&\quad + W_{\mathrm{MV}}(\mathbf{M}) + 2m + m + kW_{\mathrm{MV}}(\mathbf{M}) \\
&= 2kW_{\mathrm{MV}}(\mathbf{M}) + 3\gamma km + 4\gamma m\sum_{j=1}^{k-1} j - \gamma\sum_{j=1}^{k-1} j \\
&= 2kW_{\mathrm{MV}}(\mathbf{M}) + 3\gamma km + 4\gamma m\frac{k(k-1)}{2} - \gamma\frac{k(k-1)}{2} \\
&= 2kW_{\mathrm{MV}}(\mathbf{M}) + 3\gamma km + \gamma k(4m-1)\frac{k-1}{2}
\end{aligned}$$

where $W_{\mathrm{MV}}(\mathbf{M})$ again is the computational complexity for multiplying $\mathbf{M}$ or $\mathbf{M}^{\mathsf{T}}$ with a vector (cf. Remark 2.4).

If we do not require an approximation with a fixed rank but rather an approximation with a fixed error bound, the method can be adapted to yield such an approximation with a high probability (cf. [29, §4.3] or [32, §12]).

2.2 Krylov subspace methods

Solving large-scale linear systems

$$\mathbf{M}\mathbf{x} = \mathbf{b}$$

for $\mathbf{M} \in \mathbb{R}^{n\times n}, \mathbf{b} \in \mathbb{R}^n$ $(n \in \mathbb{N})$ exactly typically requires $\mathcal{O}\left(n^3\right)$ operations. Due to the cubic complexity, they can often not be solved directly in a reasonable amount of time. Therefore, they are usually solved approximately. Among the most popular methods for solving such large-scale linear systems are Krylov subspace methods some of which are outlined in this section.

The basic idea of Krylov subspace methods is to find an approximation such that the residual is contained in a so-called Krylov subspace.

Definition 2.7 (Krylov subspace)**.** *Let $\mathbf{M} \in \mathbb{R}^{n\times n}$, $\mathbf{r} \in \mathbb{R}^n$, and $k \leq n$. Then we call*

$$\mathcal{K}_k(\mathbf{M}, \mathbf{r}) := (\mathbf{r}, \mathbf{M}\mathbf{r}, \ldots, \mathbf{M}^{k-1}\mathbf{r})$$

the k-th Krylov subspace of $\mathbf{M}$ and $\mathbf{r}$.

For symmetric positive definite problems, the conjugate gradient method (CG method) is a well-known method yielding

1. minimization of the error in the energy norm,
2. a short-term recursion for computing the next iterate

in each step. For asymmetric and indefinite systems, on the other hand, one typically has to compromise on one of these properties. The GMRes method gives up on the short-term recursion in favor of the minimization of the residual norm while the BiCGStab method gives up on the error minimization property in favor of a short-term recursion.

2.2.1 GMRes method

The GMRes method minimizes the residuals in each step. It is based on the construction of an orthonormal basis $\mathbf{v}_1, \ldots, \mathbf{v}_m$ of the m-th Krylov subspace. Then there is an unreduced upper Hessenberg matrix $\underline{\mathbf{H}}_m \in \mathbb{R}^{(m+1)\times m}$ with

$$\mathbf{M}\mathbf{V}_m = \mathbf{V}_{m+1}\underline{\mathbf{H}}_m, \tag{2.8}$$

where $\mathbf{V}_m$ contains the basis vectors $\mathbf{v}_1, \ldots, \mathbf{v}_m$ column-wise. The m-th iterate can be constructed by solving a least squares problem. Details about the GMRes method can be found in [35, Section 6.5], while the GMRes method is summarized in Algorithm 2.3 following the description there.

As we can observe in Algorithm 2.3, we have to store the whole basis $\mathbf{v}_1, \ldots, \mathbf{v}_m$ for the GMRes method. Therefore, the storage requirements can become too large for a large number of iterations. There are some variations of the GMRes method, one of them the restarted GMRes method. The idea of this method is to restart the GMRes method with the current iterate as a start vector after k steps for some $k \in \mathbb{N}$. Note that there is no guaranteed convergence of the restarted GMRes method. In some cases, it may converge for smaller restart cycles while failing to converge for larger restart cycles (see, e.g., [14]).

Algorithm 2.3 Preconditioned GMRes method method

Input: System matrix $\mathbf{M} \in \mathbb{R}^{n\times n}$, right hand side $\mathbf{b} \in \mathbb{R}^n$, start vector $\mathbf{x}_0 \in \mathbb{R}^n$, maximal relative residual norm ϵ, maximal number of iteration steps m_{iter}

Output: Approximate solution $\mathbf{x}$ with $\|\mathbf{b} - \mathbf{M}\mathbf{x}\| \leq \epsilon\|\mathbf{b} - \mathbf{M}\mathbf{x}_0\|$

function `GMRes`($\mathbf{M}$, $\mathbf{b}$, $\mathbf{x}_0$, ϵ, m_{iter})
  $\mathbf{r}_0 \leftarrow \mathbf{b} - \mathbf{M}\mathbf{x}_0$s
  $\rho \leftarrow \|\mathbf{r}_0\|_2$
  $\mathbf{v}_0 \leftarrow \rho^{-1}\mathbf{r}_0$
  $k \leftarrow 0$
  while $\|\mathbf{r}_k\| > \epsilon\|\mathbf{r}_0\|$ and $k < m_{\text{iter}}$ **do**
    $k \leftarrow k + 1$
    $\mathbf{w}_k \leftarrow \mathbf{M}\mathbf{v}_k$
    for $i = 1, \ldots, k$ **do** ▷ Orthogonalize $\mathbf{w}_j$ w.r.t. the first k residuals.
      $h_{i,k} \leftarrow \langle \mathbf{w}_k, \mathbf{v}_i \rangle_2$
      $\mathbf{w}_k \leftarrow \mathbf{w}_k - h_{i,j}\mathbf{v}_i$
    end for
    $h_{k+1,k} \leftarrow \|\mathbf{w}_k\|_2$
    if $h_{k+1,k} = 0$ **then**
      break
    end if
    $\mathbf{v}_{k+1} \leftarrow (h_{k+1,k})^{-1}\mathbf{w}_k$
    $\underline{\mathbf{H}}_k \leftarrow (h_{i,j}) \in \mathbb{R}^{(k+1)\times k}$
    $\mathbf{V}_k \leftarrow (\mathbf{v}_1 \cdots \mathbf{v}_k) \in \mathbb{R}^{n\times m}$
    $\mathbf{y}_k \leftarrow \arg\min_{\mathbf{y}\in\mathbb{R}^m} \|\rho\mathbf{e}_1 - \underline{\mathbf{H}}_k\mathbf{y}\|$
    $\mathbf{x}_k \leftarrow x_0 + \mathbf{V}_k\mathbf{y}_k$
    $\mathbf{r}_k \leftarrow \mathbf{b} - \mathbf{M}\mathbf{x}_k$
  end while
  return $\mathbf{x}_k$
end function

Remark 2.8 (Breakdown)**.** *The GMRes method as depicted in Algorithm 2.3 breaks down in line 14 if in step k the vector* $\mathbf{w}_k$ *is the zero vector after the orthogonalization. This is only the case if the approximate solution* $\mathbf{x}_k$ *is exact (see [35, Proposition 6.10]).*

2.2.2 BiCGStab method

The BiCGStab method was derived from the biconjugate gradient method (BiCG method) (cf. [35, Section 7.3]) and was originally described in [38]. The BiCG method computes biorthogonal bases $\mathbf{v}_1, \dots, \mathbf{v}_m$ and $\mathbf{w}_1, \dots, \mathbf{w}_m$ of the two Krylov subspaces

$$\mathcal{K}_m(\mathbf{M}, \mathbf{r}_0) = (\mathbf{v}_1, \dots, \mathbf{v}_m) \text{ and } \mathcal{K}_m(\mathbf{M}^\mathsf{T}, \widehat{\mathbf{r}}_0) = (\mathbf{w}_1, \dots, \mathbf{w}_m)$$

for two initial residuals $\mathbf{r}_0, \widehat{\mathbf{r}}_0 \in \mathbb{R}^n$. This process is also called Lanczos biorthogonalization. Then

$$\mathbf{W}_m^\mathsf{T} \mathbf{M} \mathbf{V}_m = \mathbf{T}_m,$$

where $\mathbf{V}_m$ and $\mathbf{W}_m$ contain columnwise basis vectors $\mathbf{v}_i$ and $\mathbf{w}_i$, respectively, and the matrix $\mathbf{T}_m$ is tridiagonal (cf. [35, §7.1]). The m-th iterate is then constructed by implicitly solving a system with the matrix $\mathbf{T}_m$. A detailed description can be found, e.g., in [35, §7.3.1].

The BiCGStab method is based on the observation that all residuals of the BiCG method are of the form

$$\mathbf{r}_j = \phi_j(\mathbf{M}) \mathbf{r}_0$$

for polynomials ϕ_j. The BiCGStab method on the other hand uses residuals represented by two polynomials, i.e.

$$\mathbf{r}_j = \psi_j(\mathbf{M}) \phi_j(\mathbf{M}) \mathbf{r}_0,$$

where ϕ_j is the same polynomial as for the BiCG method and ψ_j is a recursively defined polynomial which aims to stabilize the convergence. In the algorithm, shown in Algorithm 2.4, these polynomials are not evaluated explicitly, but used to derive a recursive formula for the iterates and the residuals. A detailed description of the BiCGStab method can be found in [35].

Remark 2.9 (Breakdown)**.** *At several points of Algorithm 2.4 a breakdown can occur (see, e.g., [24]) due to coefficients becoming zero and are closely related to breakdowns of the Lanczos biorthogonalization as they are discussed in [35, §7.1.2].*

If $\langle \mathbf{M}\mathbf{p}_j, \widehat{\mathbf{r}}_0 \rangle = 0$ *or* $\langle \mathbf{M}\mathbf{p}_j, \widehat{\mathbf{r}}_0 \rangle \approx 0$ *in line 7 of Algorithm 2.4, then we would divide by 0 or by small values in the computation of* α_j*. Although this produces a (near-)breakdown of the BiCGStab method, the vectors of the next step can still be defined anyway. This leads to so-called look-ahead variants of the BiCGStab method avoiding this kind of breakdown in most cases (see, e.g., [8]).*

If ω_j *in line 9 of Algorithm 2.4 is zero, this would result in a division by 0 in line 12. In this case, the degree of the (implicitly) constructed polynomials does not increase anymore.*

If $\langle \mathbf{r}_j, \widehat{\mathbf{r}}_0 \rangle = 0$ *or* $\langle \mathbf{r}_j, \widehat{\mathbf{r}}_0 \rangle \approx 0$ *in line 7 of Algorithm 2.4 also results in a division by 0 or by small values in the computation of* β_j*. But in this case, we also have* $\beta_{j-1} = 0$ *(or* ≈ 0*) and* $\alpha_j = 0$ *(or* ≈ 0*) which causes the method to be no longer valid (see, e.g., [24]).*

Algorithm 2.4 BiCGStab method

Input: System matrix $\mathbf{M} \in \mathbb{R}^{n\times n}$, right hand side $\mathbf{b} \in \mathbb{R}^n$, start vector $\mathbf{x}_0 \in \mathbb{R}^n$, maximal relative residual norm ϵ, maximal number if iteration steps m_{iter}

Output: Approximate solution $\mathbf{x}$ with $\|\mathbf{b} - \mathbf{M}\mathbf{x}\| \leq \epsilon\|\mathbf{b} - \mathbf{M}\mathbf{x}_0\|$

function `BiCGStab`($\mathbf{A}$, $\mathbf{b}$, $\mathbf{x}_0$, ϵ, m_{iter})
  $\mathbf{r}_0 \leftarrow \mathbf{b} - \mathbf{M}\mathbf{x}_0$
  Choose an arbitrary vector $\widehat{\mathbf{r}}_0 \in \mathbb{R}^n$ with $\langle \mathbf{r}_0, \widehat{\mathbf{r}}_0\rangle \neq 0$
  $\mathbf{p}_0 \leftarrow \mathbf{r}_0$
  $j \leftarrow 0$
  while $\|\mathbf{r}_j\| > \epsilon\|\mathbf{r}_0\|$ and $j < m_{\text{iter}}$ **do**
    $\alpha_j \leftarrow \langle \mathbf{r}_j, \widehat{\mathbf{r}}_0\rangle_2 / \langle \mathbf{M}\mathbf{p}_j, \widehat{\mathbf{r}}_0\rangle_2$
    $\mathbf{s}_j \leftarrow \mathbf{r}_j - \alpha_j \mathbf{M}\mathbf{p}_j$
    $\omega_j \leftarrow \langle \mathbf{M}\mathbf{s}_j, \mathbf{s}_j\rangle / \langle \mathbf{M}\mathbf{s}_j, \mathbf{M}\mathbf{s}_j\rangle_2$
    $\mathbf{x}_{j+1} \leftarrow \mathbf{x}_j + \alpha_j \mathbf{p}_j + \omega_j \mathbf{s}_j$
    $\mathbf{r}_{j+1} \leftarrow \mathbf{s}_j - \omega_j \mathbf{M}\mathbf{s}_j$
    $\beta_j \leftarrow \frac{\alpha_j}{\omega_j} \frac{\langle \mathbf{r}_{j+1}, \widehat{\mathbf{r}}_0\rangle_2}{\langle \mathbf{r}_j, \widehat{\mathbf{r}}_0\rangle_2}$
    $\mathbf{p}_{j+1} \leftarrow \mathbf{r}_{j+1} + \beta_j(\mathbf{p}_j - \omega_j \mathbf{M}\mathbf{p}_j)$
    $j \leftarrow j + 1$
  end while
  return $\mathbf{x}_j$
end function

2.2.3 Preconditioning

Many iterative methods for solving sparse linear systems depend on the spectral properties of the system matrix. These tend to get worse with an increasing size of the system in many practical applications. One approach to deal with these worse spectral properties would be to choose schemes, often tailored to the problem, not depending on these spectral properties. Another popular approach, which allows using the same iterative method for different problems, is to precondition the system, i.e. altering the system to obtain favorable spectral properties without changing the solution. For a regular $P \in \mathbb{R}^{n\times n}$ we can

1. solve the system $\mathbf{P}^{-1}\mathbf{M}\mathbf{x} = \mathbf{P}^{-1}\mathbf{b}$,
2. solve the system $\mathbf{M}\mathbf{P}^{-1}\mathbf{y} = \mathbf{b}$ and obtain $\mathbf{x}$ via $\mathbf{x} = \mathbf{P}^{-1}\mathbf{y}$
3. using a decomposition $\mathbf{P} = \mathbf{P}_L\mathbf{P}_R$ with regular $\mathbf{P}_L, \mathbf{P}_R$, solve

$$\mathbf{P}_L^{-1}\mathbf{M}\mathbf{P}_R^{-1}\mathbf{y} = \mathbf{P}_L^{-1}\mathbf{b}$$

   and then obtain $\mathbf{x} = \mathbf{P}_R^{-1}\mathbf{y}$.

The first approach is widely known as left preconditioning and can be applied directly to every iterative method, including the GMRes method and the BiCGStab method discussed in the previous two sections, while the second approach, the so-called right preconditioning, involves an additional change of variables. The third approach is typically used (with $\mathbf{P}_L = \mathbf{P}_R^{\mathsf{T}}$) if symmetry has to be preserved, i.e., for the CG method. But often these

methods can be reformulated. Then the factorization $\mathbf{P} = \mathbf{P}_L\mathbf{P}_R$ only has to exist in theory and the preconditioner is applied by solving with $\mathbf{P}$ instead (see, e.g., [35, §9.2]).

Remark 2.10. *Note that the spectra of the preconditioned matrices are the same in all three cases, i.e.,*

$$\sigma(\mathbf{P}^{-1}\mathbf{M}) = \sigma(\mathbf{M}\mathbf{P}^{-1}) = \sigma(\mathbf{P}_L^{-1}\mathbf{M}\mathbf{P}_R^{-1}).$$

This holds, since each of the three matrices $\mathbf{P}^{-1}\mathbf{M}$, $\mathbf{M}\mathbf{P}^{-1}$, *and* $\mathbf{P}_L^{-1}\mathbf{M}\mathbf{P}_R^{-1}$ *can be obtained via an equivalence transformation from the others.*

2.3 Saddle point problems

In this section, we will discuss saddle point problems, the class of linear systems discussed in this thesis. We consider saddle point problems, i.e., linear 2×2 block systems, of the form

$$\mathcal{M}\begin{pmatrix}\mathbf{x}\\ \mathbf{y}\end{pmatrix} = \begin{pmatrix}\mathbf{F} & \mathbf{B}^\mathsf{T}\\ \mathbf{B} & -\mathbf{C}\end{pmatrix}\begin{pmatrix}\mathbf{x}\\ \mathbf{y}\end{pmatrix} = \begin{pmatrix}\mathbf{r}\\ \mathbf{s}\end{pmatrix} \tag{2.9}$$

with matrix blocks $\mathbf{F} \in \mathbb{R}^{n\times n}$, $\mathbf{B} \in \mathbb{R}^{m\times n}$, and $\mathbf{C} \in \mathbb{R}^{m\times m}$ as well as right hand side vectors $\mathbf{r} \in \mathbb{R}^n$, and $\mathbf{s} \in \mathbb{R}^m$ for $n, m \in \mathbb{N}$. Saddle point systems can arise from a wide variety of problems, including the discretization of the Navier-Stokes equations discussed in Chapter 3. This section discusses some properties of saddle point problems as well as preconditioning techniques utilizing the block structure following the description in [4]. A larger list of problems yielding saddle point systems can also be found in [4].

2.3.1 Properties of saddle point problems

We first note that, if $\mathbf{F}$ is regular, the block structure in (2.9) imposes the natural block-LDU factorization

$$\begin{pmatrix}\mathbf{F} & \mathbf{B}^\mathsf{T}\\ \mathbf{B} & -\mathbf{C}\end{pmatrix} = \begin{pmatrix}\mathbf{I} & \\ \mathbf{B}\mathbf{F}^{-1} & \mathbf{I}\end{pmatrix}\begin{pmatrix}\mathbf{F} & \\ & \mathbf{S}\end{pmatrix}\begin{pmatrix}\mathbf{I} & \mathbf{F}^{-1}\mathbf{B}^\mathsf{T}\\ & \mathbf{I}\end{pmatrix} \tag{2.10}$$

where $\mathbf{S} := -\mathbf{C} - \mathbf{B}\mathbf{F}^{-1}\mathbf{B}^\mathsf{T}$ is the so-called Schur complement. From this block factorization, we can obtain a first condition for the solvability of the block system (2.9):

- $\mathbf{F}$ is regular,
- the Schur complement $\mathbf{S}$ is regular.

It turns out that these conditions can be relaxed. Even if $\mathbf{F}$ is singular and the Schur complement therefore does not exist, the saddle point matrix may be regular nonetheless. This leads to the following theorem from [4].

Theorem 2.11. *If the symmetric part* $\mathbf{H} = \frac{1}{2}(\mathbf{F} + \mathbf{F}^\mathsf{T})$ *of* $\mathbf{F}$ *is positive semidefinite,* $\mathbf{B}$ *has full rank, and* $\mathbf{C}$ *is symmetric positive semidefinite, then the following holds:*

1. *if* $\ker(\mathbf{H}) \cap \ker(\mathbf{B}) = \{0\}$ *holds, then the saddle point matrix* $\mathcal{M}$ *from* (2.9) *is regular,*

2. if the saddle point matrix $\mathcal{M}$ is regular, then $\ker(\mathbf{F}) \cap \ker(\mathbf{B}) = \{0\}$ *holds.*

Proof. See [4, Theorem 3.4]. □

Remark 2.12 (Spectral properties)**.** *If the symmetric part* $\mathbf{H} = \frac{1}{2}(\mathbf{F} + \mathbf{F}^\mathsf{T})$ *is positive definite and* $\mathbf{C}$ *is symmetric positive semidefinite, then*

$$\mathcal{H} := \frac{1}{2}(\mathcal{M} + \mathcal{M}^\mathsf{T}) = \begin{pmatrix} \mathbf{H} & \mathbf{B}^\mathsf{T} \\ \mathbf{B} & -\mathbf{C} \end{pmatrix}$$

is indefinite (cf. [4, §3.4]), i.e., has both positive and negative eigenvalues. Therefore, the eigenvalues of $\mathcal{M}$ *are on both sides of the imaginary axis.*

2.3.2 Preconditioning of saddle point problems

The convergence analysis of Krylov subspace methods, especially for non-normal systems, is complicated and the eigenvalues of the system matrix may not describe the convergence behavior. Nonetheless, it can be beneficial to deal with a system with eigenvalues clustered away from the origin. Although this may not be the case for arbitrary saddle point problems as in (2.9), it may be achieved by preconditioning. A popular strategy for preconditioning saddle point problems as in (2.9) is to utilize the natural block structure of these problems. This yields so-called block preconditioners. To derive block preconditioners for saddle point problems (2.9), we consider the following block LDU decomposition of the system matrix as in (2.10). This decomposition can be used to construct several block preconditioners. Apart from the obvious choice

$$\mathcal{P}_{\mathrm{LDU}} := \begin{pmatrix} \mathbf{I} & \\ \mathbf{B}\mathbf{F}^{-1} & \mathbf{I} \end{pmatrix} \begin{pmatrix} \mathbf{F} & \\ & \mathbf{S} \end{pmatrix} \begin{pmatrix} \mathbf{I} & \mathbf{F}^{-1}\mathbf{B}^\mathsf{T} \\ & \mathbf{I} \end{pmatrix}, \tag{2.11}$$

which is just as impractical, we can use approximations by omitting either the upper or lower triangular factor of (2.10), respectively. This leads to the block triangular preconditioner

$$\mathcal{P}_{\mathrm{ut}} = \begin{pmatrix} \mathbf{F} & \mathbf{B}^\mathsf{T} \\ \mathbf{0} & \mathbf{S} \end{pmatrix} \text{ and } \mathcal{P}_{\mathrm{lt}} = \begin{pmatrix} \mathbf{F} & \mathbf{0} \\ \mathbf{B} & \mathbf{S} \end{pmatrix}. \tag{2.12}$$

Using the upper triangular block preconditioner $\mathcal{P}_{\mathrm{ut}}$, the (right) preconditioned system matrix is given by

$$\mathcal{M}\mathcal{P}_{\mathrm{ut}}^{-1} = \begin{pmatrix} \mathbf{I} & \\ \mathbf{B}\mathbf{F}^{-1} & \mathbf{I} \end{pmatrix} \begin{pmatrix} \mathbf{F} & \\ & \mathbf{S} \end{pmatrix} \begin{pmatrix} \mathbf{I} & \mathbf{F}^{-1}\mathbf{B}^\mathsf{T} \\ & \mathbf{I} \end{pmatrix} \begin{pmatrix} \mathbf{I} & -\mathbf{F}^{-1}\mathbf{B}^\mathsf{T} \\ & \mathbf{I} \end{pmatrix} \begin{pmatrix} \mathbf{F}^{-1} & \\ & \mathbf{S}^{-1} \end{pmatrix} = \begin{pmatrix} \mathbf{I} & \\ \mathbf{B}\mathbf{F}^{-1} & \mathbf{I} \end{pmatrix}.$$

Therefore,

$$\sigma(\mathcal{P}_{\mathrm{ut}}^{-1}\mathcal{M}) = \sigma(\mathcal{M}\mathcal{P}_{\mathrm{ut}}^{-1}) = \sigma\left(\begin{pmatrix} \mathbf{I} & \\ \mathbf{B}\mathbf{F}^{-1} & \mathbf{I} \end{pmatrix}\right) = \{1\}$$

holds and the minimal polynomial is given by

$$\mu(t) = (t-1)^2.$$

This results in a convergence of the GMRes method after at most two steps (see, e.g., [35, §6.2]).

Instead of omitting only one factor, we can also only use the block-diagonal factor as preconditioner. For this preconditioner,

$$\mathcal{P}_{\mathrm{d}} = \begin{pmatrix} \mathbf{F} & \\ & -\mathbf{S} \end{pmatrix}$$

[33] shows (in the case of $\mathbf{C} = 0$) that the preconditioned system matrix $\mathcal{P}_{\mathrm{d}}^{-1}\mathcal{M}$ has the three eigenvalues 1, $\frac{1}{2}(1+\sqrt{5})$, and $\frac{1}{2}(1-\sqrt{5})$ and that the minimal polynomial of $\mathcal{P}_{\mathrm{d}}^{-1}\mathcal{M}$ is of degree 3. Hence, the GMRes method applied to the preconditioned system would terminate after at most 3 steps.

Although the preconditioners described above yield a fast convergence of the GMRes method, they are only of theoretical interest. The application requires the exact inverses of $\mathbf{F}$ and the Schur complement $\mathbf{S}$. Irrespective of possible rounding errors preventing the computation of the exact inverses, they are typically dense matrices and therefore require too much memory for large-scale problems. However, we can replace $\mathbf{F}^{-1}$ and $\mathbf{S}^{-1}$ by suitable approximations. This may still be enough to decrease the number of iterations to a small number.

In most cases, suitable approximations are problem-dependent constructions as, e.g., the least squares commutator (LSC) for $\mathbf{S}^{-1}$ (see, e.g., [12]). In this thesis, we will examine easily invertible approximations to $\mathbf{F}$ and $\mathbf{S}$ utilizing hierarchical matrix factorizations (see Chapter 5).

Chapter 3

Navier-Stokes equations

One tool for modeling incompressible fluid flows are the Navier-Stokes equations, a system of (non-linear) PDEs, we will describe in this chapter. The Navier-Stokes equations aim to model the velocity field $\mathbf{u} : \mathbb{R}^d \to \mathbb{R}^d$ and the pressure $p : \mathbb{R}^d \to \mathbb{R}$ of a fluid in a domain $\Omega \subseteq \mathbb{R}^d$ $(d = 2, 3)$. Some properties of the fluid are modeled via the kinematic viscosity $\nu > 0$. The equations consist of an equation describing the conservation of momentum of the fluid as well as an equation describing the conservation of mass.

Now, let $\Omega \subseteq \mathbb{R}^d$ be a polygonal domain with a Lipschitz continuous boundary $\Gamma := \partial\Omega$ and normal vector $\mathbf{n}_\Gamma : \Gamma \to \mathbb{R}^d$. We consider boundary value problems with mixed Neumann and Dirichlet boundary conditions. Hence, let $\Gamma_N, \Gamma_D \subseteq \Gamma$ with $\Gamma_N \cup \Gamma_D = \Gamma$. Then a stationary Navier-Stokes boundary value problem is given by:

Find $\mathbf{u} : \mathbb{R}^d \to \mathbb{R}^d$ and $p : \mathbb{R}^d \to \mathbb{R}$ with

$$-\nu\Delta\mathbf{u} + (\mathbf{u} \cdot \nabla)\mathbf{u} + \nabla p = \mathbf{r} \quad \text{on } \Omega \tag{3.1a}$$

$$\nabla \cdot \mathbf{u} = 0 \quad \text{on } \Omega \tag{3.1b}$$

$$\mathbf{u} = \mathbf{g}_D \quad \text{on } \Gamma_D \tag{3.1c}$$

$$\nu \frac{\partial}{\partial \mathbf{n}_\Gamma}\mathbf{u} - \mathbf{n}_\Gamma p = 0 \quad \text{on } \Gamma_N \tag{3.1d}$$

for a suitable right hand side $\mathbf{r} : \mathbb{R}^d \to \mathbb{R}^d$ and Dirichlet boundary value $g_D : \Gamma_D \to \mathbb{R}^d$. Since the solution $\mathbf{u}$ also describes the advection term, $(\mathbf{u} \cdot \nabla)\mathbf{u}$, the problem is non-linear and thus has to be linearized, e.g., with Picard iterations or Newton iterations. The linearization with Picard iterations results in a sequence of boundary value problems:

Find $\mathbf{u}_k : \Omega \to \mathbb{R}^d$ and $p_k : \Omega \to \mathbb{R}$ with

$$-\nu\Delta\mathbf{u}_k + (\mathbf{u}_{k-1} \cdot \nabla)\mathbf{u}_k + \nabla p_k = \mathbf{r} \quad \text{on } \Omega \tag{3.2a}$$

$$\nabla \cdot \mathbf{u}_k = 0 \quad \text{on } \Omega \tag{3.2b}$$

$$\mathbf{u}_k = \mathbf{g}_D \quad \text{on } \Gamma_D \tag{3.2c}$$

$$\nu\frac{\partial}{\partial \mathbf{n}_\Gamma}\mathbf{u}_k - \mathbf{n}_\Gamma p_k = 0 \quad \text{on } \Gamma_N \tag{3.2d}$$

with a divergence-free initial guess $\mathbf{u}_0 : \mathbb{R}^d \to \mathbb{R}^d$, i.e., $\nabla \cdot \mathbf{u}_0 = 0$. For example, we can choose $\mathbf{u}_0 = 0$. These problems are called Oseen problems. Therefore, we examine Oseen problems with a divergence-free convection $\mathbf{w} \in \mathsf{L}^\infty(\Omega; \mathbb{R}^d)$:

Find $\mathbf{u} : \Omega \to \mathbb{R}^d$ and $p : \Omega \to \mathbb{R}$ with

$$-\nu\Delta\mathbf{u} + (\mathbf{w} \cdot \nabla)\mathbf{u} + \nabla p = \mathbf{r} \quad \text{on } \Omega \tag{3.3a}$$

$$\nabla \cdot \mathbf{u} = 0 \quad \text{on } \Omega \tag{3.3b}$$

$$\mathbf{u} = \mathbf{g}_D \quad \text{on } \Gamma_D \tag{3.3c}$$

$$\nu\frac{\partial}{\partial \mathbf{n}_\Gamma}\mathbf{u} - \mathbf{n}_\Gamma p = 0 \quad \text{on } \Gamma_N \tag{3.3d}$$

The remainder of this chapter is concerned with the discretization of Oseen problems as in (3.3) with the FEM. Hence, we will consider the variational formulation of these boundary value problems in Section 3.1. While Section 3.2 deals with the Galerkin discretization of the variational problem, Section 3.3 introduces the corresponding finite element spaces following the presentation in [13]. This discussion is followed by a brief description of a stabilization technique, the so-called upwinding, in Section 3.4. The linear system obtained from the FEM discretization is a saddle point problem as examined in Section 2.3. Last, we will evaluate numerical tests using an Oseen problem or the Navier-Stokes equations as a model problem. Therefore, we will introduce all used model problems in **??**.

3.1 Variational formulation for Oseen problems

The starting point of the FEM is the variational formulation of the Oseen problem (3.3). It is derived from (3.3) by multiplying all equations with suitable test functions combined with integration of the results. This yields

$$-\nu \int_\Omega \Delta\mathbf{u} \cdot \mathbf{v}\, d\mathbf{x} + \int_\Omega (\mathbf{w} \cdot \nabla)\mathbf{u} \cdot \mathbf{v}\, d\mathbf{x} + \int_\Omega \nabla p \cdot \mathbf{v}\, d\mathbf{x} = \int_\Omega \mathbf{r} \cdot \mathbf{v}\, d\mathbf{x}, \tag{3.4a}$$

$$\int_\Omega (\nabla \cdot \mathbf{u}) q\, d\mathbf{x} = 0, \tag{3.4b}$$

where $\mathbf{v} : \mathbb{R}^d \to \mathbb{R}^d$ and $q : \mathbb{R}^d \to \mathbb{R}$ are the test functions. For the first summand, we use integration by parts (i.b.p.) to obtain

$$\begin{aligned}\int_\Omega \Delta\mathbf{u} \cdot \mathbf{v}\, d\mathbf{x} &= \sum_{i=1}^d \int_\Omega v_i \Delta u_i \, d\mathbf{x} \\ &\overset{\text{i.b.p.}}{=} \sum_{i=1}^d \left(\int_\Gamma v_i (\mathbf{n}_\Gamma \cdot \nabla u_i)\, ds - \int_\Omega \nabla v_i \cdot \nabla u_i \, d\mathbf{x} \right) \\ &= \sum_{i=1}^d \int_{\Gamma_D} v_i \frac{\partial u_i}{\partial \mathbf{n}_\Gamma}\, ds + \sum_{i=1}^d \int_{\Gamma_N} v_i \frac{\partial u_i}{\partial \mathbf{n}_\Gamma}\, ds - \sum_{i=1}^d \int_\Omega \nabla v_i \cdot \nabla u_i \, d\mathbf{x} \\ &= \int_{\Gamma_D} \mathbf{v} \cdot \frac{\partial \mathbf{u}}{\partial \mathbf{n}_\Gamma}\, ds + \int_{\Gamma_N} \mathbf{v} \cdot \frac{\partial \mathbf{u}}{\partial \mathbf{n}_\Gamma}\, ds - \sum_{i=1}^d \int_\Omega \nabla v_i \cdot \nabla u_i \, d\mathbf{x}.\end{aligned} \tag{3.5}$$

By choosing test functions $\mathbf{v} \in \mathbf{V} := \mathrm{H}_0^1(\Omega; \mathbb{R}^d; \Gamma_D)$ vanishing on the Dirichlet boundary Γ_D, the reformulation from (3.5) simplifies to

$$\begin{aligned}\int_\Omega \Delta\mathbf{u} \cdot \mathbf{v}\, d\mathbf{x} &\overset{\mathbf{v}\in\mathbf{V}}{\underset{(3.5)}{=}} \int_{\Gamma_N} \mathbf{v} \cdot \frac{\partial \mathbf{u}}{\partial \mathbf{n}_\Gamma}\, d\mathbf{x} - \sum_{i=1}^d \int_\Omega \nabla v_i \cdot \nabla w_i \, d\mathbf{x} \\ &\overset{(3.3d)}{=} \frac{1}{\nu} \int_{\Gamma_N} \mathbf{v} \cdot \mathbf{n}_\Gamma p \, ds - \sum_{i=1}^d \int_\Omega \nabla v_i \cdot \nabla w_i \, d\mathbf{x}.\end{aligned} \tag{3.6}$$

Additionally, integration by parts yields for $p \in Q := \mathsf{L}^2(\Omega)$ and $\mathbf{v} \in \mathbf{V}$

$$\begin{aligned}\int_\Omega \nabla p \cdot v\, d\mathbf{x} &= \int_\Gamma \mathbf{n}_\Gamma p \cdot \mathbf{v}\, ds - \int_\Omega p(\nabla \cdot \mathbf{v})\, d\mathbf{x} \\ &\overset{\text{i.b.p.}}{=} \int_{\Gamma_D} \mathbf{n}_\Gamma p \cdot \mathbf{v}\, ds + \int_{\Gamma_N} \mathbf{n} p \cdot \mathbf{v}\, ds - \int_\Omega p(\nabla \cdot \mathbf{v})\, d\mathbf{x} \\ &\overset{\mathbf{v}\in\mathbf{V}}{=} \int_{\Gamma_N} \mathbf{n}_\Gamma p \cdot \mathbf{v}\, ds - \int_\Omega p(\nabla \cdot \mathbf{v})\, d\mathbf{x}\end{aligned} \tag{3.7}$$

By using (3.6) and (3.7) in (3.4a), we obtain

$$\begin{aligned}\int_\Omega \mathbf{r} \cdot \mathbf{v}\, d\mathbf{x} &\overset{(3.4a)}{=} -\nu \int_\Omega \Delta\mathbf{u} \cdot \mathbf{v}\, d\mathbf{x} + \int_\Omega (\mathbf{w} \cdot \nabla)\mathbf{u} \cdot \mathbf{v}\, d\mathbf{x} + \int_\Omega \nabla p \cdot \mathbf{v}\, d\mathbf{x} \\ &\overset{(3.6)}{=} -\int_{\Gamma_N} \mathbf{v} \cdot \mathbf{n}_\Gamma p \, ds + \nu \sum_{i=1}^d \int_\Omega \nabla v_i \cdot \nabla u_i \, d\mathbf{x} \\ &\quad + \int_\Omega (\mathbf{w} \cdot \nabla)\mathbf{u} \cdot \mathbf{v}\, d\mathbf{x} + \int_\Omega \nabla p \cdot \mathbf{v}\, d\mathbf{x} \\ &\overset{(3.7)}{=} -\int_{\Gamma_N} \mathbf{v} \cdot \mathbf{n}_\Gamma p \, ds + \nu \sum_{i=1}^d \int_\Omega \nabla v_i \cdot \nabla u_i \, d\mathbf{x} \\ &\quad + \int_\Omega (\mathbf{w} \cdot \nabla)\mathbf{u} \cdot \mathbf{v}\, d\mathbf{x} + \int_{\Gamma_N} \mathbf{v} \cdot \mathbf{n}_\Gamma p \, ds - \int_\Omega p(\nabla \cdot \mathbf{v})\, d\mathbf{x} \\ &= \nu \sum_{i=1}^d \int_\Omega \nabla v_i \cdot \nabla u_i \, d\mathbf{x} + \int_\Omega (\mathbf{w} \cdot \nabla)\mathbf{u} \cdot \mathbf{v}\, d\mathbf{x} - \int_\Omega p(\nabla \cdot \mathbf{v})\, d\mathbf{x}.\end{aligned} \tag{3.8}$$

To summarize the aforementioned, we multiply the equations in (3.3) with functions from the so-called test spaces

$$\mathbf{V} = \mathsf{H}_0^1(\Omega;\mathbb{R}^d;\Gamma_D) \quad \text{and} \quad Q = \mathsf{L}^2(\Omega) \tag{3.9}$$

and use integration by parts to obtain the variational (or weak) formulation of the Oseen problem:

Find $\mathbf{u} \in \mathsf{H}^1(\Omega;\mathbb{R}^d)$ and $p \in \mathsf{L}^2(\Omega)$ with

$$\nu a(\mathbf{u},\mathbf{v}) + w(\mathbf{u},\mathbf{v}) + b(\mathbf{v},p) = r(\mathbf{v}) \qquad \text{for all } \mathbf{v} \in \mathbf{V} = \mathsf{H}_0^1(\Omega;\mathbb{R}^d;\Gamma_D) \tag{3.10a}$$

$$b(\mathbf{u},q) = 0 \qquad \text{for all } q \in Q = \mathsf{L}^2(\Omega), \tag{3.10b}$$

$$\mathbf{u} = \mathbf{g}_D \qquad \text{on } \Gamma_D \tag{3.10c}$$

where a, b and w are continuous bilinear forms and $r \in \mathbf{V}'$ is a functional defined by

$$a(\mathbf{u},\mathbf{v}) := \sum_{i=1}^{3} \int_\Omega \nabla u_i \cdot \nabla v_i \, d\mathbf{x}, \tag{3.11a}$$

$$w(\mathbf{u},\mathbf{v}) := \sum_{i=1}^{3} \int_\Omega v_i(\mathbf{w} \cdot \nabla u_i) \, d\mathbf{x}, \tag{3.11b}$$

$$b(\mathbf{u},p) := -\int_\Omega p(\nabla \cdot \mathbf{u}) \, d\mathbf{x}, , \tag{3.11c}$$

$$r(\mathbf{v}) := \int_\Omega \mathbf{r} \cdot \mathbf{v} \, d\mathbf{x}. \tag{3.11d}$$

Remark 3.1. *The bilinear forms defined above in* (3.11a) – (3.11c) *can also be seen as repeated application of scalar variants for each spatial direction:*

$$a(\mathbf{u},\mathbf{v}) = \sum_{i=1}^{d} \check{a}(u_i,v_i) \quad \textit{with} \quad \check{a}(u,v) := \int_\Omega \nabla u \cdot \nabla v \, d\mathbf{x},$$

$$w(\mathbf{u},\mathbf{v}) = \sum_{i=1}^{d} \check{w}(u_i,v_i) \quad \textit{with} \quad \check{w}(u,v) := \int_\omega v(\mathbf{w} \cdot \nabla u) \, d\mathbf{x},$$

and

$$b(\mathbf{u},p) = \sum_{i=1}^{d} \check{b}^k(u_k,p) \quad \textit{with} \quad \check{b}^k(u,p) := -\int_\Omega p \frac{\partial u}{\partial x_k} \, d\mathbf{x} \quad \textit{for } k = 1,\ldots,d.$$

The existence of a solution is determined by two properties. These are obtained from a reformulation of the problem to a single equation in a reduced space (cf. [30, Section 3.1]). The equivalence of this reduced problem and the weak Oseen problem (3.10) is guaranteed by the so-called inf-sup stability, i.e. the existence of a constant $\beta > 0$ with

$$\inf_{q \in Q, q \neq \text{constant}} \sup_{\mathbf{v} \in \mathbf{V}, \mathbf{v} \neq 0} \frac{b(\mathbf{v},q)}{\|\mathbf{v}\|_{\mathsf{H}^1(\Omega)} \|q\|_{\mathsf{L}^2(\Omega)}} \geq \beta. \tag{3.12}$$

The reduced problem then can be handled by the Lax-Milgram theorem (see, e.g., [30, Theorem B.4]) which depends on the coercivity of the bilinear form

$$f : \mathbf{V} \times \mathbf{V} \to \mathbb{R};\ f(\mathbf{u}, \mathbf{v}) = \nu a(\mathbf{u}, \mathbf{v}) + w(\mathbf{u}, \mathbf{v}), \tag{3.13}$$

i.e. the existence of a constant $\alpha > 0$ with

$$f(\mathbf{v}, \mathbf{v}) \geq \alpha \|\mathbf{v}\|^2_{\mathsf{H}^1(\Omega)}. \tag{3.14}$$

Remark 3.2. *While the inf-sup stability* (3.12) *and the coercivity of f suffice to guarantee the existence of a solution, this is not the case for the uniqueness of a solution. The uniqueness depends on the choice of the search space for the pressure solution as well as the boundary conditions. For pure Dirichlet boundary conditions a pressure solution $p \in \mathsf{L}^2(\Omega)$ is only unique up to a constant. The solution would be unique if we would search for a solution*

$$p \in \mathsf{L}^2_0(\Omega) := \{f \in \mathsf{L}^2(\Omega) : \int_\Omega f \, d\mathbf{x} = 0\}$$

instead. For mixed boundary conditions, the existence of Neumann boundary conditions imposes the uniqueness of the pressure solution. A detailed discussion can be found, e.g., in [30, §3.2].

3.2 Galerkin discretization

The Galerkin discretization of the weak Oseen problem (3.10) is based on the choice of finite-dimensional subspaces $\mathbf{V}^h \subseteq \mathbf{V} = \mathsf{H}^1(\Omega; \mathbb{R}^d)$ and $Q^h \subseteq Q = \mathsf{L}^2(\Omega)$. Additionally, we assume that we have bases

$$(\mathbf{\Phi}_1, \ldots, \mathbf{\Phi}_N, \mathbf{\Phi}_{N+1}, \ldots \mathbf{\Phi}_{N+N_D})$$

of $\mathbf{V}^h$ and

$$(\Psi_1, \ldots, \Psi_M)$$

of Q^h such that

$$(\mathbf{\Phi}_1, \ldots, \mathbf{\Phi}_N) = \mathbf{V}^h_0 := \mathbf{V}^h \cap \mathsf{H}^1_0(\Omega; \mathbb{R}^d; \Gamma_D).$$

In order to apply the Dirichlet boundary conditions, we assume that $\mathbf{g}_D \in \mathbf{V}^h$, i.e.,

$$\mathbf{g}_D = \sum_{i=N+1}^{N+N_D} u^h_j \mathbf{\Phi}_j$$

for $u^h_{N+1}, \ldots, u^h_{N+N_D} \in \mathbb{R}$. The discrete problem then is:

Find $\mathbf{x} \in \mathbb{R}^N$ and $\mathbf{y} \in \mathbb{R}^M$ such that $\mathbf{u}^h := \sum_{j=1}^N x_j \mathbf{\Phi}_j + \sum_{j=N+1}^{N+N_D} u^h_j \mathbf{\Phi}_j$ and $p^h := \sum_{k=1}^M y_k \Psi_k$ satisfy

$$a(\mathbf{u}^h, \mathbf{\Phi}_i) + w(\mathbf{u}^h, \mathbf{\Phi}_i) + b(\mathbf{\Phi}_i, p^h) = r(\mathbf{\Phi}_i) \text{ for } i = 1, \ldots, N, \tag{3.15a}$$

$$b(\mathbf{u}^h, \Psi_k) = 0 \text{ for } k = 1, \ldots, M. \tag{3.15b}$$

Finding solutions $\mathbf{x} \in \mathbb{R}^N$ and $\mathbf{y} \in \mathbb{R}^M$ corresponds to solving the linear saddle point problem (cf. [13, §8.3])

$$\mathcal{M} = \begin{pmatrix} \mathbf{F} & \mathbf{B}^\mathsf{T} \\ \mathbf{B} & \mathbf{0} \end{pmatrix} \begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix} = \begin{pmatrix} \mathbf{r}^h \\ \mathbf{s}^h \end{pmatrix} \tag{3.16}$$

with

$$\begin{aligned} \mathbf{F} &= (f_{ij}) \in \mathbb{R}^{N\times N}, & f_{ij} &= f(\mathbf{\Phi}_j, \mathbf{\Phi}_i) = \nu a(\mathbf{\Phi}_j, \mathbf{\Phi}_i) + w(\mathbf{\Phi}_j, \mathbf{\Phi}_i), \\ \mathbf{B} &= (b_{ki}) \in \mathbb{R}^{M\times N}, & b_{ki} &= b(\mathbf{\Phi}_i, \Psi_k), \\ \mathbf{r}^h &= (r_i) \in \mathbb{R}^N, & r_i &= r(\mathbf{\Phi}_i) - \sum_{j=N+1}^{N+N_D} u_j^h a(\mathbf{\Phi}_j, \mathbf{\Phi}_i), \\ \mathbf{s}^h &= (s_k) \in \mathbb{R}^M, & s_k &= -\sum_{j=N+1}^{N+N_D} u_j^h b(\mathbf{\Phi}_j, \Psi_k). \end{aligned} \tag{3.17}$$

The space $\mathbf{V}^h$ is typically defined as a Cartesian product

$$\mathbf{V}^h = \prod_{i=1}^{d} V^h \tag{3.18}$$

for a finite-dimensional space $V^h \subseteq \mathsf{H}^1(\Omega)$ with a basis

$$(\varphi_1, \varphi_2, \ldots, \varphi_n, \varphi_{n+1}, \ldots \varphi_{n+n_D}),$$

where

$$\operatorname{span}(\varphi_1, \ldots, \varphi_n) = V_0^h := V^h \cap \mathsf{H}_0^1(\Omega; \Gamma_D).$$

Then, we obtain a basis $(\mathbf{\Phi}_1, \ldots, \mathbf{\Phi}_{N+N_D})$ with $N = n \cdot d$ and $N_D = n_D \cdot d$

$$\mathbf{\Phi}_{i+(k-1)n} = \mathbf{e}_k \varphi_i$$

for $i \in \{1, \ldots n\}$, $k \in \{1, \ldots, d\}$, where $\mathbf{e}_k \in \mathbb{R}^d$ is the k-th unit vector. The boundary basis functions can be defined as

$$\mathbf{\Phi}_{N+i+(k-1)n_D} = \mathbf{e}_k \varphi_{n+i}$$

for $i \in \{1, \ldots, n_D\}$, $k \in \{1, \ldots, d\}$. In this case, Remark 3.1 implies the following block structure of the system matrix:

$$\begin{pmatrix} \mathbf{F} & \mathbf{B}^\mathsf{T} \\ \mathbf{B} & \mathbf{0} \end{pmatrix} = \begin{pmatrix} \check{\mathbf{F}} & & & (\mathbf{B}^1)^\mathsf{T} \\ & \ddots & & \vdots \\ & & \check{\mathbf{F}} & (\mathbf{B}^d)^\mathsf{T} \\ \mathbf{B}^1 & \ldots & \mathbf{B}^d & \mathbf{0} \end{pmatrix} = \begin{pmatrix} \nu\mathbf{A} + \mathbf{W} & & & (\mathbf{B}^1)^\mathsf{T} \\ & \ddots & & \vdots \\ & & \nu\mathbf{A} + \mathbf{W} & (\mathbf{B}^d)^\mathsf{T} \\ \mathbf{B}^1 & \ldots & \mathbf{B}^d & \mathbf{0} \end{pmatrix}, \tag{3.19}$$

where

$$\mathbf{A} = (a_{ij}), \quad a_{ij} = \check{a}(\varphi_j, \varphi_i),$$

$$\mathbf{W} = (w_{ij}), \quad w_{ij} = \check{w}(\varphi_j, \varphi_i),$$

and

$$\mathbf{B}^k = (b^k_{ij}), \quad b^k_{ij} = \check{b}^k(\varphi_j, \Psi_i)$$

for $k = 1, \ldots, d$.

Remark 3.3. *The existence of a solution to the discrete problem depends again on*

1. *the inf-sup stability* (3.12) *of the (discrete) saddle point problem,*
2. *the coercivity* (3.14) *of the bilinearform f from* (3.13)*.*

While the second is directly inherited from the continuous case, this does not apply to the inf-sup stability. The discrete inf-sup stability depends on the actual choice of the finite-dimensional spaces $\mathbf{V}^h$ and Q^h. We will see an example of a stable pair of spaces in the next section and refer to [30, §3.5] for abstract criteria and more examples of inf-sup stable choices.

3.3 Finite element discretization

The goal of the finite element discretization is the construction of spaces $\mathbf{V}^h$ and Q^h for the Galerkin discretization of the variational formulation from (3.10). These finite-dimensional spaces are constructed as continuous piecewise polynomial spaces.

For this, we first decompose the domain Ω into simpler domains, i.e. triangles for $d = 2$ or tetrahedra for $d = 3$.

Definition 3.4 (Simplex)**.** *Let $d \in \{2, 3\}$, $k \leq d + 1$ and $\mathbf{p}_1, \ldots, \mathbf{p}_{k+1} \in \mathbb{R}^d$ such that $\{\mathbf{p}_i - \mathbf{p}_j : j \neq i\}$ is linear independent for some $i \in \{1, \ldots k+1\}$. Then $K := \{\mathbf{p}_1, \ldots \mathbf{p}_{k+1}\}$ is called a (d-dimensional) k-*simplex*.*

1. *$\mathbf{p}_1, \ldots \mathbf{p}_{d+1}$ are called* vertices *of K.*
2. *If $d \geq 2$, then a 2-simplex $E \subseteq K$ is called* edge *of K.*
3. *If $d \geq 3$, then a 3-simplex $F \subseteq K$ is called* face *of K.*

We call 3-simplices triangles *and 4-simplices* tetrahedra*.*

Definition 3.5 (Simplex domain)**.** *Let $K = \{\mathbf{p}_1, \ldots, \mathbf{p}_{k+1}\} \subseteq \mathbb{R}^d$ be a d-dimensional k-simplex. Then, we denote the (open) convex hull of a simplex K by*

$$\mathring{K} := \{\mathbf{x} \in \mathbb{R}^d : \sum_{i=1}^{k+1} \alpha_i \mathbf{p}_i \text{ with } \alpha_1, \ldots, \alpha_{d+1} > 0 \text{ and } \sum_{i=1}^{k+1} \alpha_i = 1\}$$

and its closure by

$$\overline{K} := \{\mathbf{x} \in \mathbb{R}^d : \sum_{i=1}^{k+1} \alpha_i \mathbf{p}_i \text{ with } \alpha_1, \ldots, \alpha_{k+1} \geq 0 \text{ and } \sum_{i=1}^{d+1} \alpha_i = 1\}$$

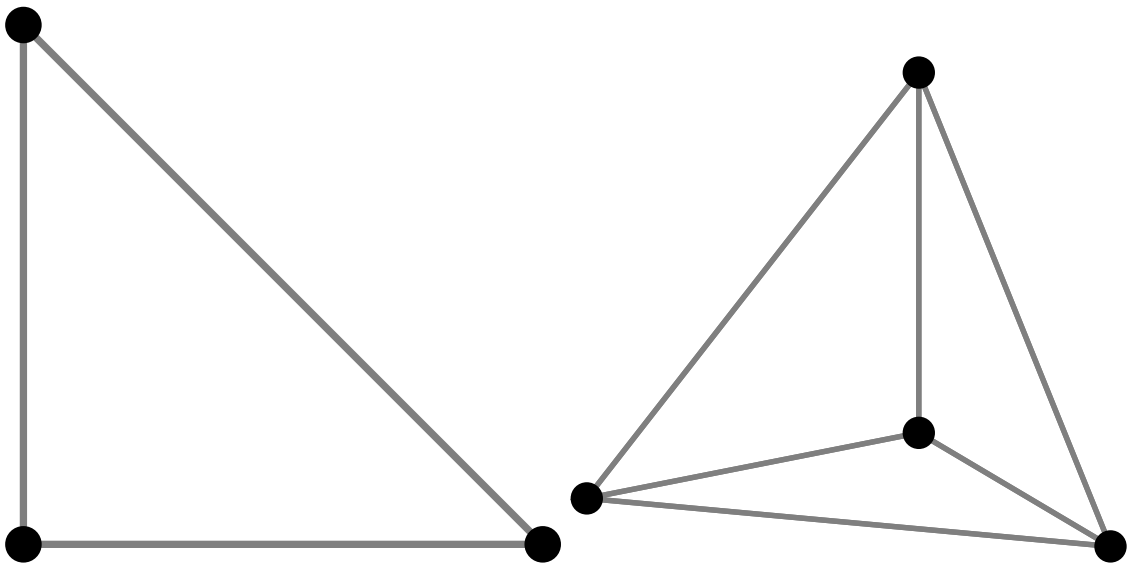

Figure 3.1: d-dimensional simplex for $d = 2$ (left) and $d = 3$ (right). The figures show the vertices(black) and edges(gray) of the simplices.

Definition 3.6 (Triangulation)**.** *Let $d \in \{2, 3\}$ and $\Omega \subseteq \mathbb{R}^d$. Let $\mathcal{T}$ be a set of d-simplices. Then we call $\mathcal{T}$ a* triangulation *of Ω, if*

1. $\bigcup_{K\in\mathcal{T}} \overline{K} = \overline{\Omega}$,
2. $\mathring{K}_1 \cap \mathring{K}_2 = \emptyset$ *for all* $K_1, K_2 \in \mathcal{T}$ *with* $K_1 \neq K_2$,
3. *for* $K_1, K_2 \in \mathcal{T}$ *with* $K_1 \neq K_2$ *and* $I := \overline{K}_1 \cap \overline{K}_2 \neq \emptyset$, *the intersection I is either a common edge, a common vertex, or (if $d = 3$) a common face of K_1 and K_2.*

For a triangulation $\mathcal{T}$, we denote the set of vertices *of $\mathcal{T}$ by*

$$\mathcal{V}(\mathcal{T}) := \bigcup_{K\in\mathcal{T}} K$$

and the set of edges *of $\mathcal{T}$ by*

$$\mathcal{E}(\mathcal{T}) := \{E : E \subseteq K \text{ is an edge of a simplex } K \in \mathcal{T}\}.$$

We call

$$h := \max_{K\in\mathcal{T}} \operatorname{diam}_2(\overline{K}) = \max_{K\in\mathcal{T}} \max_{\mathbf{p},\mathbf{q}\in K} \|\mathbf{p} - \mathbf{q}\|$$

the mesh width *of $\mathcal{T}$. In the following, we will denote a triangulation $\mathcal{T}$ with mesh width h by $\mathcal{T}^h$.*

Remark 3.7. *Typically, one is not only interested in a single triangulation but rather in a sequence of triangulations with decreasing mesh widths. This sequence can be constructed from a starting triangulation $\mathcal{T}^H$. Let $K \in \mathcal{T}^H$ and $v_1, \ldots, v_{d+1}$ the vertices of K. Let $m_1, \ldots, m_k$ be the $k = \binom{d+1}{2}$ edge midpoints. By connecting these midpoints, we obtain*

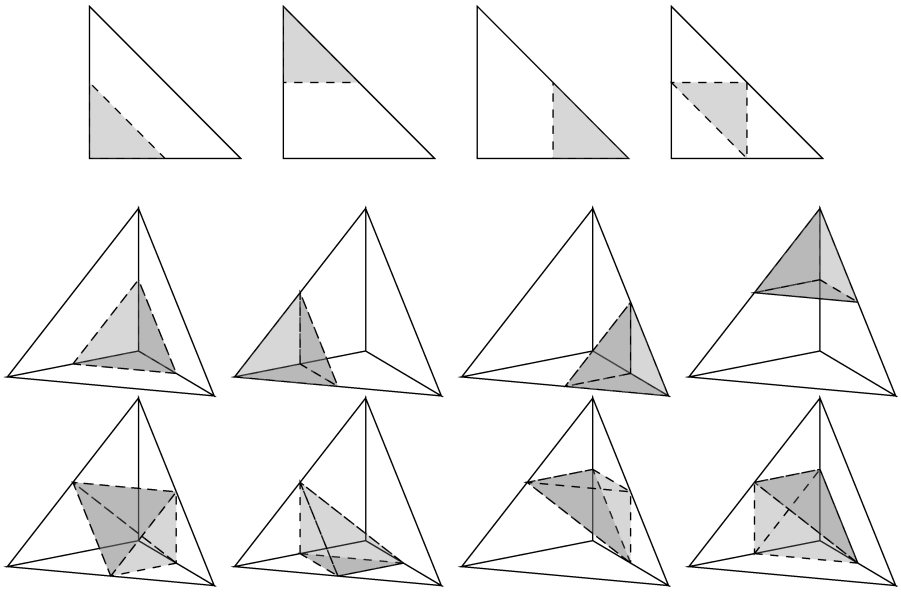

Figure 3.2: Division of a triangle into 4 smaller triangles (top) and of a tetrahedron into 8 smaller tetrahedra (bottom) by connecting the edge midpoints.

$m = 4$ (for $d = 2$) or $m = 8$ (for $d = 3$) pairwise disjoint smaller triangles or tetrahedra $K_1, \dots, K_m$ with $\cup_{i=1}^{m} \overline{K}_i = \overline{K}$, respectively (cf. Figure 3.2). Doing this for every element of the triangulation $\mathcal{T}^H$ yields the triangulation

$$\mathcal{T}^h = \bigcup_{K \in \mathcal{T}^H} \{K_1, \dots, K_m\}$$

with mesh width $h = H/2$.

The FEM uses piecewise polynomial spaces as finite-dimensional subspaces for the Galerkin discretization.

Definition 3.8. *Let $X \subseteq \mathbb{R}^d$ and $k \in \mathbb{N}$. Then we denote the* space of polynomials with at most degree k *by*

$$\Pi_k(X) := \{p : X \to \mathbb{R} : p(\mathbf{x}) = \sum_{\substack{\mathbf{i} \in \mathbb{N}_0^d \\ |\mathbf{i}| \le k}} a_{\mathbf{i}} x_1^{i_1} \cdot x_d^{i_d}, a_{\mathbf{i}} \in \mathbb{R} \text{ for all } \mathbf{i} \in \mathbb{N}_0^d\},$$

where $|\mathbf{i}| := i_1 + \cdots + i_d$ for $\mathbf{i} \in \mathbb{N}^d$.

Definition 3.9. *Let $\mathcal{T}^h$ be a triangulation of Ω. Then we define*

1. the space of piecewise polynomials *with at most degree $k \in \mathbb{N}$ on $\mathcal{T}^h$ by*

$$\mathrm{P}^k(\mathcal{T}^h) := \{u \in \mathrm{C}(\overline{\Omega}) : u|_{\overline{K}} \in \Pi_k(\overline{K}) \text{ for all } \Delta \in \mathcal{T}^h\} \tag{3.20}$$

and correspondingly the space of piecewise polynomials with at most degree k on $\mathcal{T}^h$ and with zero boundary on Γ_D by

$$\mathrm{P}_0^k(\mathcal{T}^h; \Gamma_D) := \mathrm{P}^k(\mathcal{T}^h) \cap \mathsf{H}_0^1(\Omega; \Gamma_D).$$

2. *the* space of discontinuous piecewise polynomials *with at most degree* $k \in \mathbb{N}$ *on* $\mathcal{T}^h$ *by*

$$\mathrm{P}^{k,\mathrm{disc}}(\mathcal{T}^h) := \{u \in \mathsf{L}^2(\Omega) : u|_{\mathring{K}} \in \Pi_k(\mathring{K}) \text{ for all } K \in \mathcal{T}^h\}$$

We will only consider piecewise linear polynomials. Hence the remaining task is to construct a suitable basis of the space $\mathrm{P}^1(\mathcal{T}^h)$. Since all $\mathbf{u} \in \mathrm{P}^1(\mathcal{T}^h)$ satisfy $\mathbf{u}|_{\overline{K}} \in \Pi_1(\overline{K})$, it is reasonable to choose a basis $(\varphi_1, \ldots, \varphi_n)$, for a suitable $n \in \mathbb{N}$, such that

$$\operatorname{span}(\varphi_1|_{\overline{K}}, \ldots, \varphi_n|_{\overline{K}}) = \Pi_1(\overline{K}).$$

The simplest choice are the so-called hat functions (cf. [13, §1.3]) $\varphi_{\mathbf{p}} \in \mathrm{P}^1(\mathcal{T}^h)$ with

$$\varphi_{\mathbf{p}}(\mathbf{q}) = \begin{cases} 1 & \text{if } \mathbf{p} = \mathbf{q}, \\ 0 & \text{else} \end{cases} \tag{3.21}$$

for all $\mathbf{p}, \mathbf{q} \in \mathcal{V}(\mathcal{T}^h)$. By enumerating the vertices of the triangulation

$$\{\mathbf{p}_1, \ldots, \mathbf{p}_{n+n_D}\} = \mathcal{V}(\mathcal{T}^h),$$

where $\mathbf{p}_1, \ldots, \mathbf{p}_n \in \Omega$ and $\mathbf{p}_{n+1}, \ldots, \mathbf{p}_{n+n_D} \in \Gamma$, we obtain the basis $(\varphi_1, \ldots, \varphi_{n+n_D})$ with

$$\varphi_i := \varphi_{\mathbf{p}_i} \tag{3.22}$$

for $i = 1, \ldots, n + n_D$.

Remark 3.10. *1. There exist similar constructions for higher polynomial degrees* $k \geq 2$*. For example, we can use the edge midpoints*

$$\mathcal{M}(\mathcal{T}^h) := \left\{ \frac{1}{2}(\mathbf{p} + \mathbf{q}) : \mathbf{p}, \mathbf{q} \in K, K \in \mathcal{T}^h \right\}$$

of the simplices in $\mathcal{T}^h$ *as additional auxilliary points. On the extended set of vertices*

$$\mathcal{V}^{\mathrm{ext}}(\mathcal{T}^h) := \mathcal{V}(\mathcal{T}^h) \cup \mathcal{M}(\mathcal{T}^h)$$

we can construct piecewise quadratic polynomials $\varphi_{\mathbf{p}} \in \mathrm{P}^2(\mathcal{T}^h)$ *with*

$$\varphi_{\mathbf{p}}(\mathbf{q}) = \begin{cases} 1 \text{ if } \mathbf{p} = \mathbf{q} \\ 0 \text{ else.} \end{cases}$$

Figure 3.3 illustrates such a piecewise polynomial.

2. *It is also possible to use a decomposition of* Ω *into a set* $\mathcal{Q}^h$ *of rectangles* ($d = 2$) *or cuboids* ($d = 3$) *instead of a triangulation with simplices. Then the used space of piecewise polynomials is of the form*

$$\left\{ f : f|_B(\mathbf{x}) = \sum_{\substack{\mathbf{i} \in \mathbb{N}_0^d \\ |\mathbf{i}|_\infty \leq k}} a_{B,\mathbf{i}} x_1^{i_1} \cdots x_d^{i_d}, a_{B,\mathbf{i}} \in \mathbb{R} \text{ for all } \mathbf{i} \in \mathbb{N}_0^d,\ B \in \mathcal{Q}^h \right\},$$

where $|\mathbf{i}|_\infty := \max_{j \in \{1,\ldots,d\}} i_j$*. A detailed description can be found, e.g., in [13, §1.3].*

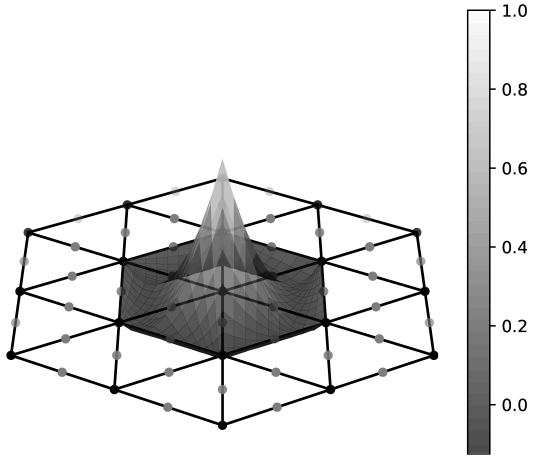

Figure 3.3: Surface plot of a basis function for polynomial degree 2 introduced in Remark 3.10.

Stable pair of finite element spaces

As stated in Remark 3.3, the discrete inf-sup stability is not inherited from the continuous, infinite-dimensional spaces $\mathbf{V}$ and Q. The validity of the discrete inf-sup stability

$$\inf_{q\in Q^h, q\neq \text{constant}} \sup_{\mathbf{v}\in\mathbf{V}^h,\mathbf{v}\neq 0} \frac{b(\mathbf{v},q)}{\|\mathbf{v}\|_{\mathsf{H}^1(\Omega)}\|q\|_{\mathsf{L}^2\Omega}} \geq \beta > 0$$

depends on the choice of the spaces $\mathbf{V}^h$ (or V^h) and Q^h. A popular choice for these spaces is the family of Taylor-Hood finite elements, where

$$V^h = \mathrm{P}^k(\mathcal{T}^h) \text{ and } Q^h = \mathrm{P}^{k-1}(\mathcal{T}^h).$$

for $k \geq 2$. Since the discrete inf-sup stability does not depend on the smoothness of the basis functions but rather on the dimension of the used spaces, another popular approach is to refine a triangulation $\mathcal{T}^h$ by connecting the midpoints of the edges (cf. Figure 3.2) to obtain a triangulation $\mathcal{T}^{h/2}$ with halved mesh width. Then $\mathrm{P}^1(\mathcal{T}^{h/2})$ has the same dimension as $\mathrm{P}^2(\mathcal{T}^h)$, and we obtain an inf-sup stable pair of spaces with

$$V^h = \mathrm{P}^1(\mathcal{T}^{h/2}) \text{ and } Q^h = \mathrm{P}^1(\mathcal{T}^h) \tag{3.23}$$

We refer to [30, Section 3.6] for a detailed discussion on the inf-sup stability of these pairs of spaces.

Unstable pair of finite element spaces

If the discrete inf-sup condition is not satisfied, the system has to be stabilized. A popular approach is to add a penalty term to the second equation. The idea presented here originates from [10]. We will follow the description in [13, Section 3.3.3].

We start with a triangulation $\mathcal{T}^h$ and set

$$Q^h = V^h = \mathrm{P}^1(\mathcal{T}^h). \tag{3.24}$$

For this choice of spaces, there is a mismatch between the range of the divergence restricted to $\mathbf{V}^h$ and the pressure space Q^h. This mismatch can be described by the L^2-orthogonal projection

$$\pi : \mathsf{L}^2(\Omega) \to \mathrm{P}^{0,\mathrm{disc}}(\mathcal{T}^h)$$

onto the space of discontinuous piecewise constant functions (cf. Definition 3.9) defined elementwise by

$$(\pi q)|_{\mathring{K}} = \frac{1}{|\mathring{K}|} \int_{\mathring{K}} q. \tag{3.25}$$

Since $\nabla \cdot \mathbf{v} \in \mathrm{P}^{0,\mathrm{disc}}(\mathcal{T}^h)$ for all $\mathbf{v} \in \mathbf{V}^h = \prod_{i=1}^d V^h$ holds, we obtain

$$b(\mathbf{v}, q - \pi q) = 0$$

for all $q \in Q^h$ and $\mathbf{v} \in \mathbf{V}^h$. The key idea of the stabilization is to penalize $q \in Q^h$ with $q - \pi q \neq 0$ by adding the bilinear form $c : Q^h \times Q^h \to \mathbb{R}$ with

$$c(p, q) = \int_\Omega (p - \pi p)(q - \pi q)\, d\mathbf{x} \tag{3.26}$$

to the second equation of (3.10). This stabilization technique was originally developed for the FEM discretization of the Stokes equations. In order to apply it to the discrete Oseen problem, we have to correct for the scaling of the bilinear form a with ν. That is done by scaling c by $\frac{1}{\nu}$ which yields the stabilized discrete Oseen problem

find $\mathbf{u} \in \mathbf{V}^h$ and $p \in Q^h$ with

$$f(\mathbf{u}, \mathbf{v}) + n(\mathbf{w}; \mathbf{u}, \mathbf{v}) + b(\mathbf{v}, p) = r(\mathbf{v}) \quad \text{for all } \mathbf{v} \in \mathbf{V}^h, \tag{3.27a}$$

$$b(\mathbf{u}, q) - \frac{1}{\nu} c(p, q) = 0 \quad \text{for all } q \in Q^h, \tag{3.27b}$$

$$\mathbf{u} = \mathbf{g}_D \quad \text{on } \Gamma_D. \tag{3.27c}$$

This results in the linear saddle point system

$$\begin{pmatrix} \mathbf{F} & \mathbf{B}^\mathsf{T} \\ \mathbf{B} & -\frac{1}{\nu}\mathbf{C} \end{pmatrix} \begin{pmatrix} \mathbf{u}^h \\ \mathbf{p}^h \end{pmatrix} = \begin{pmatrix} \mathbf{r} \\ \mathbf{s} \end{pmatrix}, \tag{3.28}$$

where $\mathbf{F}$, $\mathbf{B}$, $\mathbf{r}$, and $\mathbf{s}$ are as before in (3.16) and

$$\mathbf{C} = (c_{ij}) \in \mathbb{R}^{M \times M} \quad c_{ij} = c(\varphi_j, \varphi_i).$$

3.4 Upwinding

It can be shown that the stability of the discrete saddle point problem (3.16) depends inversely on the viscosity constant ν, (see, e.g., [30, Lemma 5.13]). Hence, the system gets unstable for $\nu \ll 1$. A popular approach to tackle these numerical instabilities are

so-called upwind techniques. In this section, we will describe the upwind triangle method first introduced in [37]. We will follow the description in [34].

First, let V^h and Q^h as in (3.23) or (3.24) and $\mathbf{V}^h = \prod_{i=1}^d V^h$. Then

$$V^h = \operatorname{span}(\varphi_1, \dots, \varphi_{n+n_D})$$

for the basis functions introduced in (3.22).

The key idea of the triangle upwind method is to replace the convection part in (3.10), i.e. the bilinear form w, by an approximation w^h. We start by considering the integral elementwise:

$$\begin{aligned} w(\mathbf{u}, \mathbf{v}) = \sum_{i=1}^d \check{w}(u_i, v_i) &= \sum_{i=1}^d \int_\Omega v_i(\mathbf{w} \cdot \nabla u_i)\, d\mathbf{x} \\ &= \sum_{i=1}^d \sum_{K \in \mathcal{T}^h} \int_K v_i(\mathbf{w} \cdot \nabla u_i)\, d\mathbf{x}. \end{aligned} \tag{3.29}$$

On an element $K \in \mathcal{T}^h$, we can approximate the integral by the quadrature rule

$$\int_K \Phi\, d\mathbf{x} \approx Q_K[\Phi] := \frac{|\overline{K}|}{d+1} \sum_{\mathbf{p} \in K} \Phi(\mathbf{p}) \tag{3.30}$$

to obtain

$$\begin{aligned} w(\mathbf{u}, \mathbf{v}) &\overset{(3.29)}{=} \sum_{i=1}^d \sum_{K \in \mathcal{T}^h} \int_K v_i(\mathbf{w} \cdot \nabla u_i)\, d\mathbf{x} \\ &\overset{(3.30)}{\approx} \sum_{i=1}^d \sum_{K \in \mathcal{T}^h} \frac{|\overline{K}|}{d+1} \sum_{\mathbf{p} \in K} v_i(\mathbf{p})(\mathbf{w} \cdot \nabla u_i)(\mathbf{p}). \end{aligned} \tag{3.31}$$

Next, we replace the evaluation of the directional derivative in the vertices with a difference scheme,using that functions $u \in V^h$ are piecewise linear polynomials and therefore have piecewise constant derivatives:

$$(\mathbf{w} \cdot \nabla u)(\mathbf{p}) \approx \begin{cases} \mathbf{w}(\mathbf{p}) \cdot \nabla u|_{K_\mathbf{p}} & \text{if } \mathbf{w}(\mathbf{p}) \neq 0, \\ 0 \quad \text{else,} \end{cases}$$

where $K_\mathbf{p} \in \mathcal{T}^h$ is a so-called upwind triangle of $\mathbf{p}$.

Definition 3.11. *Let $\mathbf{p} \in \mathcal{V}(\mathcal{T}^h)$. Let $K \in \mathcal{T}^h$ with*

1. *$\mathbf{p}$ is a vertex of K,*
2. *$-\mathbf{w}(\mathbf{p}) \in \mathbb{R}^d$ points from $\mathbf{p}$ into K.*

Then K is called upwind triangle *of $\mathbf{p}$.*

Note that there exists at least one upwind triangle for all vertices $\mathbf{p}$ with $\mathbf{w}(\mathbf{p}) \neq 0$. We then define the approximate bilinear form w^h via

$$w^h(\mathbf{u}, \mathbf{v}) := \sum_{i=1}^{d} \check{w}^h(u_i, v_i) \tag{3.32}$$

with

$$\check{w}^h(u_i, v_i) = \sum_{K \in \mathcal{T}^h} \frac{|\overline{K}|}{d+1} \sum_{\mathbf{p} \in K} (\mathbf{w}(\mathbf{p}) \cdot \nabla u_i|_{K_\mathbf{p}}) v_i(\mathbf{p}). \tag{3.33}$$

Remark 3.12. *1. For basis functions $\varphi_i, \varphi_j \in V^h$ the formula in (3.33) simplifies to*

$$\check{w}^h(\varphi_j, \varphi_i) = \sum_{K \in \mathcal{T}^h, \mathbf{p}_i \in K} \frac{|\overline{K}|}{d+1} \mathbf{w}(\mathbf{p}_i) \cdot \nabla \varphi_j|_{K_{\mathbf{p}_i}}, \tag{3.34}$$

since φ_i is a hat functions (cf. (3.21)) and therefore

$$\varphi_i(\mathbf{p}) = \begin{cases} 1 & \text{if } \mathbf{p} = \mathbf{p}_i, \\ 0 & \text{else} \end{cases}$$

holds. Since φ_j is also a hat function, (3.34) is non-zero if and only if $\mathbf{p}_j \in K_{\mathbf{p}_i}$.

2. *There are some examples (see e.g. [34, Example 3.3 in Section III.3]), for which the upwind scheme presented above coincides with simple upwind schemes for the finite difference method.*

3. *The upwind scheme presented here stabilizes the system by ensuring a discrete maximum principle for the system matrix. See [34, Section III.3.1] for more details.*

Chapter 4

Hierarchical matrices

$\mathcal{H}$-matrices were first introduced in [25] in 1999 as a data-sparse matrix format. This data sparsity is achieved by a hierarchical block structure with dense and low-rank matrix blocks. Since then, $\mathcal{H}$-matrices have been developed further. Arithmetic operations, like matrix-vector operations, $\mathcal{H}$-matrix-matrix multiplication, inversion, and matrix factorizations were developed and further analyzed, e.g., in [20, 21, 25].

For dense $n \times n$ matrices these arithmetic operations have a computational complexity of $\mathcal{O}\left(n^2\right)$ (matrix-vector arithmetic) or $\mathcal{O}\left(n^3\right)$ (for matrix-matrix arithmetic), respectively, while [21, 25] show a typical complexity of $\mathcal{O}\left(n \log^{\alpha}(n)\right)$, $\alpha = 1, 2$, for $\mathcal{H}$-matrices. The data sparsity and almost linear computational complexity encourage an application of hierarchical matrices as preconditioners, as considered, e.g., in [3, 16, 22, 31] for FEM problems, or in [1] for BEM problems.

The remainder of this chapter is structured as follows. First, all necessary definitions for the construction of $\mathcal{H}$-matrices, as well as the storage complexity, are discussed in Section 4.1. The hierarchical block structure of $\mathcal{H}$-matrices is based on a partitioning of the row and column index sets via certain tree structures which can be constructed from problem dependent information. Hence, we will discuss several strategies for the construction of these structures in Section 4.2. The differentiation between dense and low-rank blocks is made utilizing so-called admissibility conditions. Therefore, we will motivate and define some choices for these in Section 4.3. Last we will provide definitions and algorithms for arithmetic operations with $\mathcal{H}$-matrices as well as several complexity estimates in Section 4.4.

4.1 Definitions

Let $\mathcal{I}, \mathcal{J}$ be finite index sets. As already noted in the introduction of this chapter, the block structure of an $\mathcal{H}$-matrix $\mathbf{H} \in \mathbb{R}^{\mathcal{I} \times \mathcal{J}}$ is obtained from a hierarchical partitioning of the product index set $\mathcal{I} \times \mathcal{J}$. The hierarchy of the $\mathcal{H}$-matrix blocks is obtained by partitioning the index sets $\mathcal{I}$ and $\mathcal{J}$ first, and then using these to define a suitable partitioning of $\mathcal{I} \times \mathcal{J}$. All these partitionings are described through tree structures. Therefore, we first define necessary basics (Definitions 4.1 and 4.2) and (labeled) trees (Definition 4.3) following [28, Appendix A].

Definition 4.1 (Directed Graph)**.** *Let V be a non-empty, finite set, and $E \subseteq V \times V$. Then we call $G := (V, E)$ a directed graph.*

1. *The elements of V are called* vertices*, and the elements of E are called* edges*.*
2. *In the following, we will use G as an identifier for the vertex set V. Therefore, we may use the notation $v \in G$ instead of $v \in V$.*
3. *For each vertex $v \in V$ we define the* degree *of v by the number of outgoing edges*
$$\deg(v) := |\{w \in V : (v, w) \in E\}|.$$
4. *For $k \in \mathbb{N}_0$ a $(k+1)$-tuple $(v_0, \ldots, v_k) \in V^{k+1}$ is called* path *of length k from v_0 to v_k, if $(v_i, v_{i+1}) \in E$ holds for all $0 \leq i < k$.*

Definition 4.2 (Son Mapping)**.** *Let $\mathcal{T} = (V, E)$ be a directed graph (cf. Definition 4.1).*

1. *We call a mapping*
$$\text{sons} : V \to \mathcal{P}(V)$$
son mapping*, if for all $v \in V$ there holds $w \in \text{sons}(v)$ if and only if $(v, w) \in E$. The vertices w are called* sons *of v, and v is called* father *of the vertices w.*
2. *If there is a path from a vertex $v \in V$ to a vertex $w \in V$, we call v* predecessor *of w and w* successor *of v.*

Definition 4.3 ((Labeled) Tree)**.** *Let $\mathcal{T} = (V, E)$ be a directed graph (cf. Definition 4.1).*

1. *We call $\mathcal{T}$ a* tree*, if there is a son mapping (cf. Definition 4.2)*
$$\text{sons} : V \to \mathcal{P}(V)$$
such that the following holds:
   (a) *There is a unique vertex $r \in V$ that has no father, i.e.,*
$$\bigcup_{v \in V} \text{sons}(v) = V \setminus \{r\}.$$
   ,
   (b) *All vertices $v \in V$ are successors of the root r,*
   (c) *All $v \in V \setminus \{r\}$ have exactly one father.*
2. *We call the vertex r* root *of $\mathcal{T}$ and denote it by $\text{root}(\mathcal{T})$.*
3. *We call a tree $\mathcal{T}' = (V', E')$ with $V' \subseteq V$ and $E' \subseteq E$* subtree *of $\mathcal{T}$. If additionally $\text{sons}(w) = \text{sons}'(w)$ holds for all $w \in V'$, where sons' is the son mapping of $\mathcal{T}'$, then we also denote $\mathcal{T}'$ by $\mathcal{T}(v)$ with $v := \text{root}(\mathcal{T}')$.*
4. *We call $\mathcal{T}$ a* labeled tree*, if there is a set L of labels and a map*
$$\widehat{\ }: V \to L$$
which assigns a label $\widehat{v} \in L$ to all vertices $v \in \mathcal{T}$.

Remark 4.4. *Note that Definition 4.2 implies that the edges of a tree are uniquely determined by the son mapping (and vice versa). If we want to build a labeled tree, as we will do in Section 4.2, we therefore only have to define the son mapping and the labels for all vertices.*

Aside from the definitions above, we will later need the following concepts.

Definition 4.5 (Leaves, level, depth)**.** *Let $\mathcal{T} = (V, E)$ be a tree.*

1. *We call the nodes $v \in \mathcal{T}$ with* $\operatorname{sons}(v) = \emptyset$ leaves *and denote the set of leaves with*

$$\mathcal{L}(\mathcal{T}) := \{v \in \mathcal{T} : v \text{ is a leaf}\}.$$

2. *For vertices $v \in \mathcal{T}$, we define the* level *of v recursively through*

$$\operatorname{level}(\operatorname{root}(\mathcal{T})) := 0$$

and

$$\operatorname{level}(v) = \operatorname{level}(w) + 1,$$

where w is the father of v.

3. *We define the* depth *of the tree $\mathcal{T}$ as*

$$\operatorname{depth}(\mathcal{T}) := \max_{v \in \mathcal{T}} \operatorname{level}(v).$$

Now, these labeled trees can be used to describe a partitioning of the index sets $\mathcal{I}$ and $\mathcal{J}$, respectively.

Definition 4.6 (Cluster tree, [28])**.** *Let $\mathcal{I}$ be an index set and $\mathcal{T} = (V, E)$ a labeled tree (cf. Definition 4.3) with labels $L \subseteq \mathcal{P}(\mathcal{I}) \setminus \{\emptyset\}$. Then we call $\mathcal{T}$ a* cluster tree *for the index set $\mathcal{I}$, if*

1. *the root $r = \operatorname{root} \mathcal{T}$ is labeled with $\widehat{r} = \mathcal{I}$.*

2. *for each node $t \in \mathcal{T} \setminus \mathcal{L}(\mathcal{T})$ the label is the disjoint union of the sons' labels, i.e.*

$$\widehat{t} = \dot{\bigcup}_{t' \in \operatorname{sons}(t)} \widehat{t'}.$$

We denote cluster trees for an index set $\mathcal{I}$ as $\mathcal{T}_\mathcal{I}$ and call vertices $t \in \mathcal{T}_\mathcal{I}$ clusters*. Subtrees $\mathcal{T}_\mathcal{I}(t)$ for $t \in \mathcal{T}_\mathcal{I}$ are denoted by $\mathcal{T}_t$. In the following, we will also use clusters t as identifiers for their labels $\widehat{t}$, e.g.*

$$\mathbf{M}|_{t \times s} := \mathbf{M}|_{\widehat{t} \times \widehat{s}}$$

for matrices $\mathbf{M} \in \mathbb{R}^{\mathcal{I} \times \mathcal{J}}$.

If the cluster tree $\mathcal{T}_\mathcal{I}$ is clear from the context, we denote the set of leaves with $\mathcal{L}_\mathcal{I}$.

Remark 4.7. *Let $\mathcal{T}_\mathcal{I}$ be a cluster tree, and $t \in \mathcal{T}_\mathcal{I}$. Then the subtree $\mathcal{T}_t = \mathcal{T}_\mathcal{I}(t)$ is a cluster tree for the index set $\widehat{t}$. All properties are inherited directly from $\mathcal{T}_\mathcal{I}$ since the son mapping of $\mathcal{T}_t$ is just the son mapping of $\mathcal{T}_\mathcal{I}$ restricted to $\mathcal{T}_t$.*

The second property in Definition 4.6 guarantees that the set of leaves $\mathcal{L}_\mathcal{I}$ of a cluster tree $\mathcal{T}_\mathcal{I}$ defines a partition of the index set $\mathcal{I}$. Hence, a partition of $\mathcal{I} \times \mathcal{J}$ could be directly obtained from two cluster trees $\mathcal{T}_\mathcal{I}$ and $\mathcal{T}_\mathcal{J}$ as

$$\{\widehat{t} \times \widehat{s} : t \in \mathcal{L}_\mathcal{I}, s \in \mathcal{L}_\mathcal{J}\}. \tag{4.1}$$

Theoretically, we could also use an arbitrary cluster tree for $\mathcal{I} \times \mathcal{J}$, but storing this tree would cost $\mathcal{O}(|\mathcal{I}||\mathcal{J}|)$ (see [28, Section 5.5]). Therefore, we use the two cluster trees $\mathcal{T}_\mathcal{I}$ and $\mathcal{T}_\mathcal{J}$ to construct a product tree for which we only have to store the cluster trees with a cost of $\mathcal{O}(|\mathcal{I}| + |\mathcal{J}|)$.

Definition 4.8 (Block cluster tree, [28])**.** *Let $\mathcal{I}$ and $\mathcal{J}$ be index sets with cluster trees $\mathcal{T}_\mathcal{I}$ and $\mathcal{T}_\mathcal{J}$, respectively. Let $\mathcal{T} = (V, E)$ be a labeled tree with $V \subseteq \mathcal{T}_\mathcal{I} \times \mathcal{T}_\mathcal{J}$ where the label for $b \in \mathcal{T}$ is denoted with $\widehat{b}$. Then $\mathcal{T}$ is called a* block cluster tree *if the following holds:*

1. *The root is given by* $\operatorname{root}(\mathcal{T}) = (\operatorname{root}(\mathcal{T}_\mathcal{I}), \operatorname{root}(\mathcal{T}_\mathcal{J}))$.
2. *For $b = (t, s) \in \mathcal{T}$ the label is $\widehat{b} = \widehat{t} \times \widehat{s} \in \mathcal{P}(\mathcal{I} \times \mathcal{J})$.*
3. *If $b = (t, s) \in \mathcal{T}$ has sons* $\operatorname{sons}(b)$*, then there holds*

$$\operatorname{sons}(b) = \{(t', s') : t' \in \operatorname{sons}(t), s' \in \operatorname{sons}(s)\}.$$

In the following, we denote a block cluster tree for cluster trees $\mathcal{T}_\mathcal{I}$ and $\mathcal{T}_\mathcal{J}$ with $\mathcal{T}_{\mathcal{I}\times\mathcal{J}}$ and vertices $b \in \mathcal{T}_{\mathcal{I}\times\mathcal{J}}$ as blocks*. Subtrees $\mathcal{T}_{\mathcal{I}\times\mathcal{J}}(b)$ for blocks $b \in \mathcal{T}_{\mathcal{I}\times\mathcal{J}}$ are denoted by $\mathcal{T}_b$.*

The cluster trees $\mathcal{T}_\mathcal{I}$ and $\mathcal{T}_\mathcal{J}$ are also called row cluster tree *and* column cluster tree *of $\mathcal{T}_{\mathcal{I}\times\mathcal{J}}$, respectively. Clusters $t \in \mathcal{T}_\mathcal{I}$ and $s \in \mathcal{T}_\mathcal{J}$ are called* row clusters *and* column clusters*, respectively.*

As for clusters, we will use blocks b as identifiers for their labels $\widehat{b}$ (cf. Definition 4.6), and if the used block cluster tree $\mathcal{T}_{\mathcal{I}\times\mathcal{J}}$ is known from the context, we denote the set of leaves by $\mathcal{L}_{\mathcal{I}\times\mathcal{J}} := \mathcal{L}(\mathcal{T}_{\mathcal{I}\times\mathcal{J}})$.

Remark 4.9. 1. *A block cluster tree $\mathcal{T}_{\mathcal{I}\times\mathcal{J}}$ is also a cluster tree for $\mathcal{I} \times \mathcal{J}$. The third property yields*

$$\bigcup_{(t',s')=b'\in\operatorname{sons}(b)} t' \times s' = \bigcup_{\substack{t'\in\operatorname{sons}(t),\\ s'\in\operatorname{sons}(s)}} t' \times s' = \bigcup_{t'\in\operatorname{sons}(t)} t' \times s = t \times s$$

for all $(t, s) = b \in \mathcal{T}_{\mathcal{I}\times\mathcal{J}}$.

2. *Similar to cluster trees (cf. Remark 4.7) the subtree $\mathcal{T}_b$ for $b = (t, s) \in \mathcal{T}_{\mathcal{I}\times\mathcal{J}}$ is a block cluster tree with row cluster tree $\mathcal{T}_t$ and column cluster tree $\mathcal{T}_s$.*

Since a block cluster tree $\mathcal{T}_{\mathcal{I}\times\mathcal{J}}$ is also a cluster tree, we obtain a partition of the product index set $\mathcal{I} \times \mathcal{J}$ through the leaves $\mathcal{L}_{\mathcal{I}\times\mathcal{J}}$ as in Equation (4.1). This yields a hierarchical block structure, but later on, we will distinguish between blocks yielding a low-rank matrix representation and blocks yielding a dense matrix representation. This distinction is typically done with an admissibility condition.

Definition 4.10 (Admissibility condition, [28]). *Let $\mathcal{I}$ and $\mathcal{J}$ be index sets with cluster trees $\mathcal{T}_\mathcal{I}$ and $\mathcal{T}_\mathcal{J}$, respectively. Then an* admissibility condition *is a map*

$$\operatorname{adm} : \mathcal{T}_\mathcal{I} \times \mathcal{T}_\mathcal{J} \to \{\text{true}, \text{false}\}.$$

With such an admissibility condition, we can construct a block cluster tree $\mathcal{T}_{\mathcal{I}\times\mathcal{J}}$ as follows.

1. *Set the root as* $\operatorname{root}(\mathcal{T}_{\mathcal{I}\times\mathcal{J}}) = (\operatorname{root}(\mathcal{T}_\mathcal{I}), \operatorname{root}(\mathcal{T}_\mathcal{J}))$.
2. *For an already constructed block $b = (t, s) \in \mathcal{T}_{\mathcal{I}\times\mathcal{J}}$, we set*

$$\operatorname{sons}(b) = \begin{cases} \emptyset & \text{if } \operatorname{adm}(t,s) = \text{true}, \\ \emptyset & \text{if } \operatorname{sons}(t) = \emptyset \text{ or } \operatorname{sons}(s) = \emptyset \\ \{(t', s') : t' \in \operatorname{sons}(t), s' \in \operatorname{sons}(s)\} & \text{else.} \end{cases}$$

Then an admissibility condition imposes the division of the leaves of the constructed block tree $\mathcal{T}_{\mathcal{I}\times\mathcal{J}}$ into admissible and inadmissible leaves. Thus, we define the farfield *of $\mathcal{T}_{\mathcal{I}\times\mathcal{J}}$ as the set of admissible leaves*

$$\mathcal{L}^+(\mathcal{T}_{\mathcal{I}\times\mathcal{J}}, \operatorname{adm}) := \{b = (t,s) \in \mathcal{L}_{\mathcal{I}\times\mathcal{J}} : \operatorname{adm}(t,s) = \text{true}\},$$

and the nearfield *of $\mathcal{T}_{\mathcal{I}\times\mathcal{J}}$ as the set of inadmissible leaves*

$$\begin{aligned} \mathcal{L}^-(\mathcal{T}_{\mathcal{I}\times\mathcal{J}}, \operatorname{adm}) &:= \{b = (t,s) \in \mathcal{L}_{\mathcal{I}\times\mathcal{J}} : \operatorname{adm}(t,s) = \text{false}\} \\ &= \mathcal{L}_{\mathcal{I}\times\mathcal{J}} \setminus \mathcal{L}^+(\mathcal{T}_{\mathcal{I}\times\mathcal{J}}, \operatorname{adm}). \end{aligned}$$

If the block cluster tree and the admissibility condition are known from the context, we denote the far- and nearfield by $\mathcal{L}^+_{\mathcal{I}\times\mathcal{J}} := \mathcal{L}^+(\mathcal{T}_{\mathcal{I}\times\mathcal{J}}, \operatorname{adm})$ and $\mathcal{L}^-_{\mathcal{I}\times\mathcal{J}} := \mathcal{L}^-(\mathcal{T}_{\mathcal{I}\times\mathcal{J}}, \operatorname{adm})$, respectively.

The far- and nearfield blocks let us distinguish between dense and low-rank matrix blocks and allow us to define hierarchical matrices as follows.

Definition 4.11 ($\mathcal{H}$-matrix, [28]). *Let $\mathcal{I}$ and $\mathcal{J}$ be index sets with cluster trees $\mathcal{T}_\mathcal{I}$ and $\mathcal{T}_\mathcal{J}$ and a block cluster tree $\mathcal{T}_{\mathcal{I}\times\mathcal{J}}$ constructed with an admissibility condition* adm. *Then a matrix $\mathbf{H} \in \mathbb{R}^{\mathcal{I}\times\mathcal{J}}$ is called an $\mathcal{H}$-matrix with local rank $k \in \mathbb{N}$ if for each admissible block $b = (t,s) \in \mathcal{L}^+_{\mathcal{I}\times\mathcal{J}}$ there is a tuple $(\mathbf{A}_b, \mathbf{B}_b) \in \mathcal{R}(t,s,k)$ with*

$$\mathbf{H}|_{t\times s} = \mathbf{A}_b \mathbf{B}_b^\mathsf{T},$$

i.e., $\operatorname{rank}(\mathbf{H}|_{t\times s})$ *is at most k. We denote the set of $\mathcal{H}$-matrices with local rank at most k by $\mathcal{H}(\mathcal{T}_{\mathcal{I}\times\mathcal{J}}, k)$.*

Remark 4.12. *Let $\mathbf{H} \in \mathcal{H}(\mathcal{T}_{\mathcal{I}\times\mathcal{J}}, k)$ and $b \in \mathcal{T}_{\mathcal{I}\times\mathcal{J}}$. Then the restriction $\mathbf{H}|_b$ of $\mathbf{H}$ to b is also a hierarchical matrix with local rank k, i.e.,*

$$\mathbf{H}|_b \in \mathcal{H}(\mathcal{T}_b, k).$$

The analysis of the storage cost for $\mathcal{H}$-matrices necessitates two additional assumptions. Since we have to store dense matrices for nearfield blocks $(t,s) = b \in \mathcal{L}^-_{\mathcal{I}\times\mathcal{J}}$, for which either $t \in \mathcal{L}_\mathcal{I}$ or $s \in \mathcal{L}_\mathcal{J}$, we need a bound for the size of these blocks. Therefore, we assume that the sizes of the leaf clusters of the cluster trees $\mathcal{T}_\mathcal{I}$ and $\mathcal{T}_\mathcal{J}$ have an upper bound.

Definition 4.13. *Let $\mathcal{T}_\mathcal{I}$ be a cluster tree for an index set $\mathcal{I}$. We call $n_{\text{leaf}} \in \mathbb{N}$ a* leaf size bound *of $\mathcal{T}_\mathcal{I}$ if*

$$|t| \leq n_{\text{leaf}}$$

holds for all $t \in \mathcal{L}_\mathcal{I}$.

The second assumption concerns the data sparsity of the matrix. For a moment, consider a sparse matrix $(m_{ij}) = \mathbf{M} \in \mathbb{R}^{\mathcal{I}\times\mathcal{J}}$. Since only the non-zero entries have to be stored, the sparsity can be quantified by an upper bound for the number of non-zero entries per row/column.

$$C := \max\{\max_{i\in\mathcal{I}} |\{j : m_{ij} \neq 0\}|, \max_{j\in\mathcal{J}} |\{i : m_{ij} \neq 0\}|\} \tag{4.2}$$

For $\mathcal{H}$-matrices the stored data is determined by the leaf blocks. Therefore, we are interested in an estimate for the number of leaf blocks $(t, s) \in \mathcal{L}_{\mathcal{I}\times\mathcal{J}}$ for row clusters $t \in \mathcal{T}_\mathcal{I}$ or column clusters $s \in \mathcal{T}_\mathcal{J}$, respectively (cf. [28, Section 6.3],[21, Section 2.1.1]).

Definition 4.14 (Sparsity constant)**.** *Let $\mathcal{I}$ and $\mathcal{J}$ be index sets with cluster trees $\mathcal{T}_\mathcal{I}$ and $\mathcal{T}_\mathcal{J}$. Let $\mathcal{T}_{\mathcal{I}\times\mathcal{J}}$ be a block cluster tree for $\mathcal{T}_\mathcal{I}$ and $\mathcal{T}_\mathcal{J}$. Then we define the* row sparsity constant *of $\mathcal{T}_{\mathcal{I}\times\mathcal{J}}$ by*

$$C_{\text{sp,row}}(\mathcal{T}_{\mathcal{I}\times\mathcal{J}}) := \max_{t\in\mathcal{T}_\mathcal{I}} |\{s \in \mathcal{T}_\mathcal{J} : (t, s) \in \mathcal{L}_{\mathcal{I}\times\mathcal{J}}\}|.$$

Similarly, we define the column sparsity constant *of $\mathcal{T}_{\mathcal{I}\times\mathcal{J}}$ by*

$$C_{\text{sp,col}}(\mathcal{T}_{\mathcal{I}\times\mathcal{J}}) := \max_{s\in\mathcal{T}_\mathcal{J}} |\{t \in \mathcal{T}_\mathcal{I} : (t, s) \in \mathcal{L}_{\mathcal{I}\times\mathcal{J}}\}|.$$

With these constants we obtain the sparsity constant $C_{\text{sp}}(\mathcal{T}_{\mathcal{I}\times\mathcal{J}})$ *of $\mathcal{T}_{\mathcal{I}\times\mathcal{J}}$ via*

$$C_{\text{sp}}(\mathcal{T}_{\mathcal{I}\times\mathcal{J}}) = \max\{C_{\text{sp,row}}(\mathcal{T}_{\mathcal{I}\times\mathcal{J}}), C_{\text{sp,col}}[\mathcal{I} \times \mathcal{J}]\}$$

If the block cluster tree $\mathcal{T}_{\mathcal{I}\times\mathcal{J}}$ is known from the context, we denote $C_{\text{sp}}(\mathcal{T}_{\mathcal{I}\times\mathcal{J}})$ by C_{sp}.

With these constants, we can estimate the memory complexity for hierarchical matrices. As we will see in the next Lemma, a small memory complexity is obtained if we can assume both constants, the sparsity constant and the size of the leaves, to be small.

Lemma 4.15. *Let $\mathcal{I}$, $\mathcal{J}$ be index sets. Let $\mathcal{T}_\mathcal{I}$ and $\mathcal{T}_\mathcal{J}$, respectively, be corresponding cluster trees with a leaf size bound $n_{\text{leaf}} \in \mathbb{N}$, and $\mathcal{T}_{\mathcal{I}\times\mathcal{J}}$ a corresponding block cluster tree with sparsity constant C_{sp}. Then the storage complexity $S_\mathcal{H}$ of matrices $\mathbf{H} \in \mathcal{H}(\mathcal{T}_{\mathcal{I}\times\mathcal{J}}, k)$ is bounded by*

$$S_\mathcal{H}(\mathcal{T}_{\mathcal{I}\times\mathcal{J}}, k) \leq C_{\text{sp}} \max\{n_{\text{leaf}}, k\} \left((\rho_\mathcal{I} + 1)|\mathcal{I}| + (\rho_\mathcal{J} + 1)|\mathcal{J}|\right) \tag{4.3}$$

with

$$\rho_\mathcal{I} = \text{depth}(\mathcal{T}_\mathcal{I}), \quad \textit{and} \quad \rho_\mathcal{J} = \text{depth}(\mathcal{T}_\mathcal{J}).$$

Proof. See [28, Lemma 6.13]. □

4.2 Clustering

The definition of an $\mathcal{H}$-matrix (Definition 4.11) as well as the estimate for its memory complexity in Lemma 4.15 imply the importance of the clustering and the admissibility condition on the block structure and memory complexity of $\mathcal{H}$-matrices. The structure, i.e., the block structure or the sparsity pattern, of matrices from practical applications is typically problem dependent. For example, the sparsity pattern of FEM matrices depends on the decomposition of the domain into finite elements as well as analytic properties of the chosen basis functions (cf. Section 3.3). Due to these dependencies, it is often beneficial to consider problem-dependent cluster strategies. We will discuss two of these in this section.

4.2.1 Geometric bisection clustering

In most cases, we can describe an index set $\mathcal{I}$ with geometric information, e.g., points in a domain or supports of basis functions. Let $d \in \{2, 3\}$. Then, we assume that for each index $i \in \mathcal{I}$ there is a bounded set $X_i \subseteq \mathbb{R}^d$ assigned to this index. Then we also define

$$X_t := \bigcup_{i \in t} X_i \tag{4.4}$$

for subsets $t \subseteq \mathcal{I}$.

Remark 4.16. *The sets X_i are supposed to contain geometric information from the underlying problem. They can, for example, be singletons $X_i = \{\xi_i\}$ or supports of functions $X_i = \operatorname{supp} \varphi_i$ (e.g. for PDEs discretized with the finite element method).*

The geometric bisection clustering aims to utilize the geometric information obtained from the sets X_i to divide clusters into two subsets. Since it may be hard to handle the sets X_i directly because of complicated geometries, we replace them by bounding boxes.

Definition 4.17 (Bounding box)**.** *Let $X \subseteq \mathbb{R}^d$ be bounded. Then an (axis-parallel) bounding box of X is the smallest cuboid*

$$B := [a_1, b_1] \times \cdots \times [a_d, b_d] \subseteq \mathbb{R}^d$$

with $X \subseteq B$. We denote the bounding box of X by $B(X)$.

Remark 4.18. *For $i \in \mathcal{I}$ we denote the bounding box of X_i by*

$$B_i := B(X_i).$$

Correspondingly for all $t \subseteq \mathcal{I}$, we denote the bounding box of X_t by

$$B_t := B(X_t).$$

We start the construction of a cluster tree by setting the root labeled with $\mathcal{I}$. To construct sons, we have to divide $\mathcal{I}$ into subsets. This is done by using the bounding box

$$B_{\mathcal{I}} =: [a_1, b_1] \times \cdots \times [a_d, b_d]$$

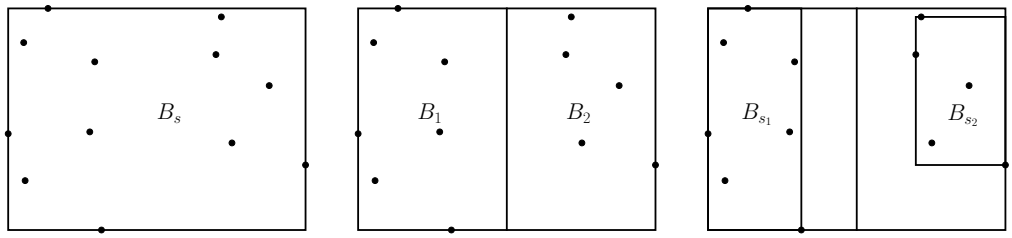

Figure 4.1: Example of the splitting of a bounding box B_s (left) into two parts B_1, B_2 (middle). The arising corresponding index subsets s_1, s_2 have bounding boxes B_{s_1}, B_{s_2} (right) differing from the boxes obtained via the splitting of the large bounding box.

for the index set $\mathcal{I}$. This bounding box is then split into the two distinct boxes B_1 and B_2 along the longest axis

$$k := \arg\max\{|b_i - a_i| : i \in \{1, \dots, d\}\}.$$

Using $m_k := \frac{a_k + b_k}{2}$, the boxes B_1 and B_2 are given by

$$B_1 = [a_1, b_1] \times \cdots \times [a_{k-1}, b_{k-1}] \times [a_k, m_k] \times [a_{k+1}, b_{k+1}] \times \cdots \times [a_d, b_d], \tag{4.5a}$$
$$B_2 = [a_1, b_1] \times \cdots \times [a_{k-1}, b_{k-1}] \times [m_k, b_k] \times [a_{k+1}, b_{k+1}] \times \cdots \times [a_d, b_d]. \tag{4.5b}$$

These boxes are utilized to construct two subsets of $\mathcal{I}$:

$$s_1 = \{i \in \mathcal{I} : X_i \cap B_1 \neq \emptyset\} \text{ and } s_2 = \mathcal{I} \setminus s_1.$$

The subsets s_1, s_2 are disjoint subsets of $\mathcal{I}$ with $s_1 \cup s_2 = \mathcal{I}$. Note that B_1 and B_2 are not bounding boxes for s_1 and s_2, respectively (cf. Figure 4.1).

The splitting described above is repeated subsequently with bounding boxes B_{s_1} for s_1 and B_{s_2} for s_2, respectively, until all subsets are small enough (cf. Definition 4.13). The constructed subsets are used to build a cluster tree, as summarized in Algorithm 4.1. Figure 4.2 illustrates the clustering for an example.

Remark 4.19. *The geometric bisection clustering implies a reordering of the index set $\mathcal{I}$ as follows: we order the indices from the subset s_1 before the indices from the subset s_2. The further clustering of s_1 and s_2 then implies a reordering within these subsets as well.*

Remark 4.20. *1. It is often easier to handle single points instead of supports of basis functions. Thus, it may be favorable to choose auxiliary points $\xi_i \in \mathbb{R}^d$ for all $i \in \mathcal{I}$ and replace X_i by*

$$\widehat{X}_i := \{\xi_i\}.$$

2. If we assume the sets X_i to contain only single points $\xi_i, i \in \mathcal{I}$, then the two subsets s_1, s_2 are given by

$$s_1 = \{i \in t : \xi_i \in B_1\} \text{ and } s_2 = \mathcal{I} \setminus s_1.$$

In this case the new bounding boxes B_{s_1} and B_{s_2} are contained in B_1 and B_2, respectively.

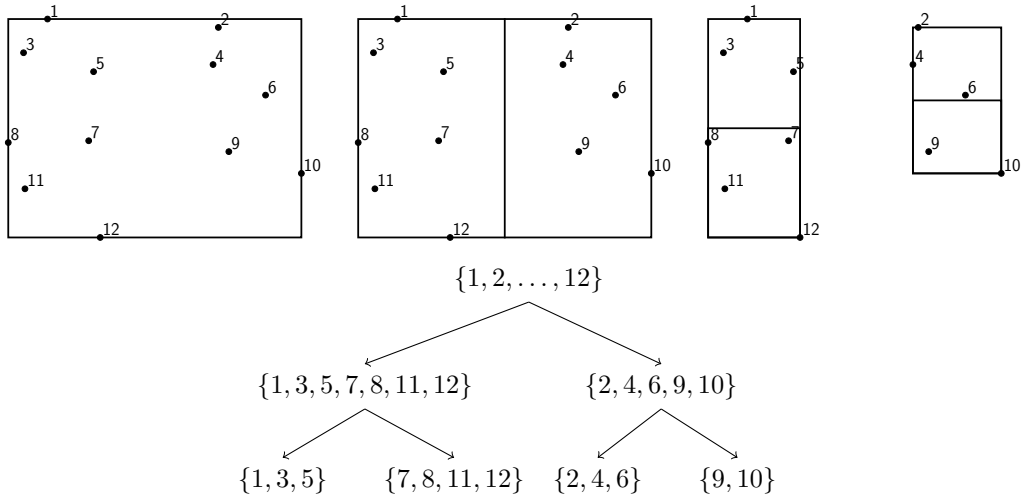

Figure 4.2: Example of the geometric bisection clustering with the index set $\mathcal{I} = \{1, 2, \ldots, 12\}$ and assigned points $\xi_1, \ldots, \xi_{12} \in \mathbb{R}^2$. The top row shows from left to right: the bounding box for all points ξ_i, $i = 1, \ldots, 12$, the first subdivision into two disjoint subsets with respect to the split bounding box along the x-axis, and the subdivision of both subsets with respect to the new bounding boxes. The resulting tree is illustrated in the bottom.

Remark 4.21. *The geometric bisection clustering as presented in Algorithm 4.1 results in a binary cluster tree, i.e.*

$$|\operatorname{sons}(s)| = 2$$

holds for all non-leaf clusters (cf. [28, Remark 5.20]).

Additional assumptions on the sets X_i allow us to further analyze the cluster tree constructed with Algorithm 4.1. We assume that we have

$$X_i = \operatorname{supp}(\varphi_i)$$

for a function φ_i and that we can choose an auxiliary point $\xi_i \in X_i$ for all $i \in \mathcal{I}$. Additionally, we assume that the supports X_i are locally separated as defined next (cf. [21, 22, 28]).

Definition 4.22. *Let $\mathcal{I}$ be a finite index set, and let $X_i \subseteq \mathbb{R}^d$ for all $i \in \mathcal{I}$. Then we call the sets X_i* locally separated *if there are $C_{\mathrm{sep}} > 0$ and $n_{\min} \in \mathbb{N}$ with*

$$\max_{i \in \mathcal{I}} \left| \left\{ j \in \mathcal{I} : \operatorname{dist}_2(X_i, X_j) \leq \frac{\operatorname{diam}_2(X_i)}{C_{\mathrm{sep}}} \right\} \right| \leq n_{\min}. \tag{4.6}$$

Remark 4.23. *In the case of finite elements, the sets X_i are supports $\operatorname{supp}(\varphi_i)$ of basis functions φ_i for all $i \in \mathcal{I}$. Furthermore, these supports are unions of simplices (cf. Section 3.3). For all $i, j \in \mathcal{I}$ with $\operatorname{dist}_2(X_i, X_j) > 0$ one can use that there is at least one simplex between X_i and X_j to show that there is a constant $\rho > 0$ depending on the*

Algorithm 4.1 Geometric bisection clustering algorithm

Input: Subset $s \subseteq \mathcal{I}$, leaf size bound n_{leaf}.
Output: Cluster tree with root t, $\widehat{t} = s$, and maximal leaf size n_{leaf}.

```
function BisectionClustering(s, n_leaf)
    Create a cluster t with t̂ = s and sons(t) = ∅          ▷ cf. Definition 4.6
    if |s| > n_leaf then                                    ▷ If t is too large, construct sons.
        Determine bounding box B_s                          ▷ cf. Definition 4.17
        Split B_s into B_1 and B_2                          ▷ cf. Equation (4.5)

        s_1 ← {i ∈ t : X_i ∩ B_1 ≠ ∅}
        s_2 ← s \ s_1

        t_1 ← BisectionClustering(s_1, n_leaf)
        t_2 ← BisectionClustering(s_2, n_leaf)

        sons(t) ← {t_1, t_2}
    end if
    return t
end function
```

triangulation such that $\operatorname{dist}_2(X_i, X_j) \geq \rho \operatorname{diam}_2(X_i)$. *Furthermore one can show that for all* $i \in \mathcal{I}$ *the number of intersecting supports*

$$|\{j \in \mathcal{I} : X_i \cap X_j \neq \emptyset\}|$$

is bounded by some $N \in \mathbb{N}$ *that also depends on the triangulation. Details about this can be found in [28, §6.4.3.2].*

With $C_{\text{sep}} > 1/\rho$ *and* $n_{\min} \geq N$ *the inequality from* (4.6) *holds, i.e., the sets* X_i *are locally separated.*

The assumption of locally separated X_i allows us for the following estimate of the depth of a cluster tree generated by geometric bisection.

Lemma 4.24. *Let* $\mathcal{I}$ *be a finite index set and* $X_i \subseteq \mathbb{R}^d$ *for all* $i \in \mathcal{I}$ *be locally separated. Let* $\mathcal{T}_{\mathcal{I}}$ *be a cluster tree constructed by Algorithm 4.1 with a leaf size bound* $n_{\text{leaf}} \geq n_{\min}$. *Let* $\underline{h} := \min_{i \in \mathcal{I}} \operatorname{diam}_2(X_i)$ *and* $\delta := \operatorname{diam}_2(B_{\mathcal{I}})$. *Then the depth of* $\mathcal{T}_{\mathcal{I}}$ *is bounded by*

$$\operatorname{depth}(\mathcal{T}_{\mathcal{I}}) \leq d \log_2 \left(2\sqrt{d} C_{\text{sep}} \delta \underline{h}^{-1} \right) = \mathcal{O}\left(\log_2 \left(\underline{h}^{-d} \right) \right).$$

Proof. (cf. [21, Lem. 4.5]) First note that each bounding box is divided along the longest axis (cf. Equation (4.5)). Thus, the diameter is halved after at most d steps. Let $t \in \mathcal{T}_{\mathcal{I}}$ and $\ell := \operatorname{level}(t)$. This observation yields

$$\operatorname{diam}_\infty (B_t) \leq 2^{-\lfloor \ell / d \rfloor} \operatorname{diam}_\infty (B_{\mathcal{I}}). \tag{4.7}$$

Since

$$\|x\|_\infty \leq \|x\|_2 \leq \sqrt{d} \|x\|_\infty \tag{4.8}$$

holds for all $x \in \mathbb{R}^d$, we obtain

$$\operatorname{diam}_2(B_t) \overset{(4.8)}{\leq} \sqrt{d}\operatorname{diam}_\infty(B_t) \overset{(4.7)}{\leq} \sqrt{d}2^{-\lfloor \ell/d \rfloor}\operatorname{diam}_\infty(B_{\mathcal{I}}) \overset{(4.8)}{\leq} \sqrt{d}2^{-\lfloor \ell/d \rfloor}\delta \tag{4.9}$$

Now we will use (4.6) to find a uniform bound for the level of t. If t is a non-leaf cluster, $|t| > n_{\text{leaf}} \geq n_{\min}$ holds due to Definition 4.13. From (4.6) and (4.9) we conclude that there are $i, j \in \widehat{t}$ with

$$\underline{h} \leq \operatorname{diam}_2(X_i) \overset{(4.6)}{<} C_{\text{sep}}\operatorname{dist}_2(X_i, X_j) \overset{\text{Definition 4.17}}{\leq} C_{\text{sep}}\operatorname{diam}_2(B_t) \overset{(4.9)}{\leq} C_{\text{sep}}\sqrt{d}2^{-\lfloor \ell/d \rfloor}\delta.$$

Therefore, we obtain

$$\ell/d < \lfloor \ell/d \rfloor + 1 \leq \log_2\left(\sqrt{d}C_{\text{sep}}\delta\underline{h}^{-1}\right) + 1 = \log_2\left(2\sqrt{d}C_{\text{sep}}\delta\underline{h}^{-1}\right). \tag{4.10}$$

Multiplying both sides of (4.10) with d finishes the proof. □

Remark 4.25. *Let $\mathcal{T}_{\mathcal{I}}$ be a cluster tree as in Lemma 4.24. Then inequality* (4.9) *also yields*

$$\operatorname{level}(t)/d < \lfloor \operatorname{level}(t)/d \rfloor + 1 \leq \log_2\left(\sqrt{d}\frac{\operatorname{diam}_\infty(B_{\mathcal{I}})}{\operatorname{diam}_\infty(B_t)}\right) + 1 = \log_2\left(2\sqrt{d}\frac{\operatorname{diam}_\infty(B_{\mathcal{I}})}{\operatorname{diam}_\infty(B_t)}\right).$$

Therefore, we get

$$\operatorname{level}(t) < d\log_2\left(2\sqrt{d}\frac{\operatorname{diam}_\infty(B_{\mathcal{I}})}{\operatorname{diam}_\infty(B_t)}\right).$$

4.2.2 Domain decomposition clustering

The geometric bisection cluster strategy clusters geometrically close points together. This strategy is based on the idea that points with a large distance have less influence on each other. While this strategy can be used for arbitrary types of matrices, it is typically used for $\mathcal{H}$-matrix representations of dense matrices, like, e.g., the inverse of finite element stiffness matrices (cf. [3, 16]). If applied to a stiffness matrix itself, which typically is sparse, the geometric bisection clustering in combination with the strong admissibility condition, discussed later in Section 4.3, yields rank 0 for all admissible blocks, while there may be inadmissible blocks with either no non-zero entries or at least a small number of non-zero entries. This imposes two approaches for the improvement of the block structure of these $\mathcal{H}$-matrix representations:

1. modifying the admissibility condition for an improved representation of the sparsity structure,
2. tailoring the clustering to the problem.

We will now consider a suitable clustering, which is tailored for an LU factorization of an $\mathcal{H}$-matrix representing a sparse matrix. For this application, we do not only desire a data-sparse representation of the matrix but also have to deal with potential fill-in (cf. Section 4.4.4). The modification of the admissibility condition based on the clustering discussed in this section will be part of Section 4.3.

In the context of sparse matrices, fill-in means that the LU factorization results in some non-zero entries in the two factors, which were zero entries before. Since the fill-in depends on the ordering of the index set a suitable reordering can reduce the fill-in (cf. [35]). A suitable ordering in this context is the nested dissection ordering discussed, e.g., in [35, Section 3.6.2].

In the context of $\mathcal{H}$-matrices, fill-in describes the increase of the local rank of admissible blocks. This increase in the local rank happens due to a successive update of blocks, as we will see later in Section 4.4.4. Therefore, [22, 23] introduced a cluster strategy based on the decomposition of a cluster into three parts similar to the nested dissection ordering. Two of the subclusters are disconnected in the matrix graph, hence the corresponding block is a (large) zero block. Note that there is also a black-box cluster strategy [23] relying only on the matrix graph. But we will consider the geometric variant introduced in [22] since we will only consider problems with underlying geometric information which allows for a simplified construction of the three subsets.

We assume that for each index $i \in \mathcal{I}$ there is a point ξ_i and a function φ_i associated with the index i. Then we set

$$X_i := \operatorname{supp}(\varphi_i) \qquad \text{for all } i \in \mathcal{I}, \tag{4.11}$$

$$X_t := \bigcup_{i \in \mathcal{I}} X_i \qquad \text{for all } t \subseteq \mathcal{I}. \tag{4.12}$$

Our goal is to subdivide the index set $\mathcal{I}$ into three disjoint subsets s_1, s_2, s_3, where the subsets s_1 and s_2 are separated in the sense of $X_{s_1} \cap X_{s_2} = \emptyset$. The subset s_3 then acts as an interface between s_1 and s_2.

Remark 4.26. *The condition $X_{s_1} \cap X_{s_2} = \emptyset$ may be too strong. For example in the case of FEM matrices, it can be relaxed to $X_{s_1} \cap X_{s_2}$ being a Lebesgue null set, since the entries in FEM matrices are determined via integrals.*

We start with a bounding box $B_{\mathcal{I}}$ for all points $\xi_i, i \in \mathcal{I}$, and subdivide it into B_1 and B_2 along the longest axis as in Equation (4.5). From these boxes B_1 and B_2, we construct the disjoint subsets

$$s_1 := \{i \in \mathcal{I} : \xi_i \in B_1\},\ s_2 := \{i \in \mathcal{I} : X_i \cap X_{s_1} = \emptyset\},\ s_3 := \mathcal{I} \setminus (s_1 \cup s_2).$$

The subsets s_1 and s_2 are (geometrically) separated and can be subdivided by the same approach. The subset s_3 on the other hand is handled differently with an adapted version of the geometric bisection clustering from Section 4.2.1. The construction of a corresponding cluster tree is summarized in Algorithm 4.2 and Algorithm 4.3.

Definition 4.27.

1. *The clusters constructed by Algorithm 4.2 are called* interface clusters.
2. *The clusters constructed by Algorithm 4.3 are called* domain clusters.
3. *The set of domain clusters of a cluster tree $\mathcal{T}_{\mathcal{I}}$ constructed by Algorithm 4.3 is denoted by $\mathcal{C}_{dom}(\mathcal{T}_{\mathcal{I}})$. If the cluster tree $\mathcal{T}_{\mathcal{I}}$ is known from the context, we also use $\mathcal{C}_{dom} := \mathcal{C}_{dom}(\mathcal{T}_{\mathcal{I}})$.*

Algorithm 4.2 Subdivision of interface clusters

Input: Subset $s \subseteq \mathcal{I}$, leaf size bound n_{leaf}, distance ℓ to the nearest domain predecessor
Output: Cluster tree with root t and $\widehat{t} = s$

```
function InterfaceClustering(s, n_leaf, ℓ)
    Create a cluster t with t̂ = s and sons(t) = ∅          ▷ cf. Definition 4.6
    if |s| > n_leaf then                                    ▷ If t is too large, construct sons.
        if ℓ mod d > 0 then
            Determine bounding box B_s                      ▷ cf. Definition 4.17
            Split B_s into B_1 and B_2                      ▷ cf. Equation (4.5)

            s_1 ← {i ∈ s : x_i ∈ B_1}
            s_2 ← s \ s_1

            t_1 ← InterfaceClustering(s_1, n_leaf, ℓ + 1)
            t_2 ← InterfaceClustering(s_2, n_leaf, ℓ + 1)
            sons(t) ← {t_1, t_2}
        else
            t' ← InterfaceClustering(s, n_leaf, ℓ + 1)
            sons(t) ← {t'}
        end if
    end if
    return t
end function
```

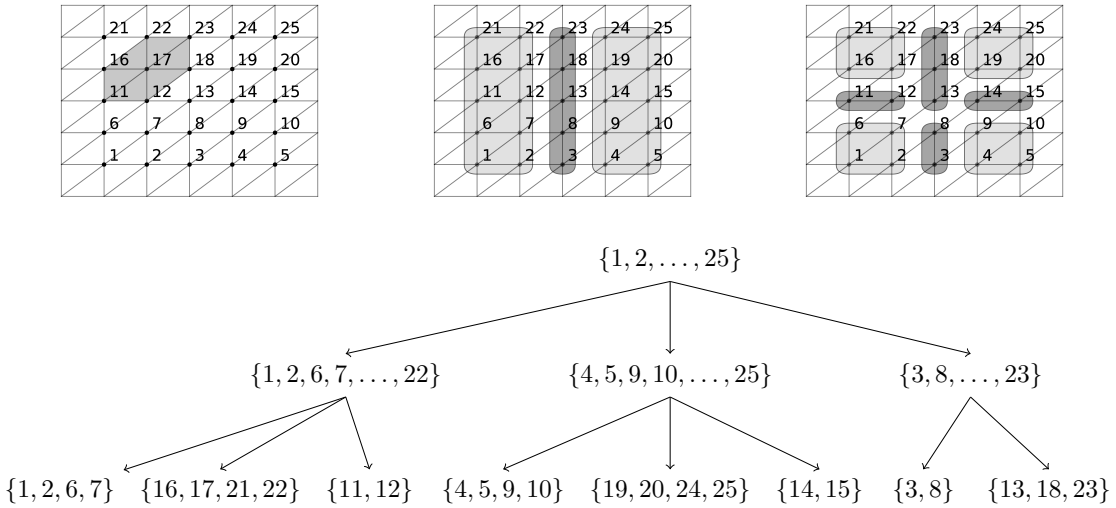

Figure 4.3: Example of the domain decomposition clustering for the index set $\mathcal{I} = \{1, \ldots, 25\}$ points $\xi_1, \ldots, \xi_{25} \in \mathbb{R}^2$ as shown in the top row. The sets X_i are the union of the shown triangles with ξ_i as vertex. The top row shows from left to right: The points ξ_i as well as X_{17} marked gray on the left, the subdivision into the two domain clusters (light gray) and the interface cluster (dark gray) in the middle, and the subdivision of these into corresponding domain clusters and interface clusters on the right. The bottom illustrates the resulting labeled cluster tree.

Remark 4.28. *As for the geometric bisection clustering (cf. Remark 4.19), this domain decomposition clustering implies a reordering of the index set $\mathcal{I}$. We order the indices of s_1 before the indices of s_2, and the indices of both s_1 and s_2 before the indices of s_3.*

Remark 4.29. *Let $h := \max_{i \in \mathcal{I}} \operatorname{diam}_2(X_i)$. Then the bounding box B_{s_3} has at most width h in the direction of partition. Thus, the diameter of the bounding box is halved after at most $d - 1$ bisections until all axes have at most width h, while the diameter of the bounding boxes of domain clusters is halved after at most d divisions. Additionally, s_3 is typically smaller than the two domain clusters s_1 and s_2. Therefore the clustering of s_3 with bisection would yield a block structure with long/tall rectangular matrix blocks for block clusters (t, s) where either t or s is a domain cluster, and the other one an interface cluster. This behavior can lead to a larger fill-in (cf. [22, Remark 11]). Hence, the subdivision of the interface clusters is delayed every d-th step.*

4.3 Admissibility conditions

An important part of the definition of $\mathcal{H}$-matrices is the distinction between dense and low-rank matrix blocks via admissibility conditions. Up to now, we only discussed methods for the clustering of index sets but not suitable admissibility conditions. Thus, we will discuss admissibility conditions suitable for the underlying data.

Let $\mathcal{I}$ and $\mathcal{J}$ be index sets with cluster trees $\mathcal{T}_\mathcal{I}$ and $\mathcal{T}_\mathcal{J}$, respectively (cf. Definition 4.6). For $i \in \mathcal{I}$ and $j \in \mathcal{J}$, assume there are sets $X_i \subseteq \mathbb{R}^d$ and $Y_j \subseteq \mathbb{R}^d$, respectively (cf.

Algorithm 4.3 Domain decomposition clustering algorithm

Input: Subset s, leaf size bound n_{leaf}
Output: Cluster tree with root t and $\widehat{t} = s$

```
function DomainClustering(s, n_leaf)
    Create a cluster t with t̂ = s and sons(t) = ∅          ▷ cf. Definition 4.6
    if |s| > n_leaf then                                    ▷ If s is too large, construct sons.
        Determine bounding box B_s                          ▷ cf. Definition 4.17
        Split B_s into B_1 and B_2                          ▷ cf. Equation (4.5)

        s_1 ← {i ∈ I : ξ_i ∈ B_1}
        s_2 ← {i ∈ I : X_i ∩ X_{s_1} = ∅}
        s_3 ← I \ (s_1 ∪ s_2)

        t_1 ← DomainClustering(s_1, n_leaf)
        t_2 ← DomainClustering(s_2, n_leaf)
        t_3 ← InterfaceClustering(s_3, n_leaf, 1)
        sons(t) ← {t_1, t_2, t_3}
    end if
    return t
end function
```

Remark 4.16). For $t \subset \mathcal{I}$ and $s \subseteq \mathcal{J}$ we then use X_t and Y_s as in Equation (4.4).

Strong admissibility

We first consider an example from [28] to motivate the admissibility condition. Consider the integral operator defined by

$$(\mathcal{K}u)(x) = \int_0^1 \log(|x-y|)u(y)\,dy \quad \text{for all } x \in [0,1] \tag{4.13}$$

for $u : [0,1] \to \mathbb{R}$. The Galerkin discretization of $\mathcal{K}$ with a basis $(\varphi_i)_{i\in\mathcal{I}}$ yields a matrix $(k_{ij}) = \mathbf{K} \in \mathbb{R}^{\mathcal{I}\times\mathcal{I}}$ with

$$k_{ij} = \int_0^1 \int_0^1 \log(|x-y|)\varphi_i(x)\varphi_j(y)\,dx\,dy \tag{4.14}$$

for all $i, j \in \mathcal{I}$. Let $X_i = \operatorname{supp}(\varphi_i) \subseteq \mathbb{R}$ for $i \in \mathcal{I}$. Let $t, s \subseteq \mathcal{I}$ and $\eta > 0$ with

$$\operatorname{diam}_2(X_t) \leq \eta \operatorname{dist}_2(X_t, X_s). \tag{4.15}$$

For simplicity, we will assume that $X_t = [\alpha_t, \beta_t]$ and $X_s = [\alpha_s, \beta_s]$ with $\alpha_t < \beta_t \leq \alpha_s < \beta_s$. As we will see, (4.15) is sufficient to obtain a low-rank approximation of $\mathbf{K}|_{t\times s}$ with an exponentially decreasing error.

Now, we consider the Taylor expansion of $\log(|\cdot - y|)$ in the barycenter $x^* = \frac{1}{2}(\beta_t + \alpha_t)$ of X_t, which yields the Taylor polynomial (cf. [28, Example 4.12])

$$T_k(x, y, x^*) = \log(|y - x^*|) + \sum_{\ell=1}^{k-1} (x - x^*)^\ell \frac{-1}{\ell(y - x^*)^\ell}. \tag{4.16}$$

of degree $k \in \mathbb{N}$ with the remainder

$$R_k(x,y,x^*) = \log(|x-y|) - T_k(x,y,x^*) = \sum_{\ell=k}^{\infty} (x-x^*)^\ell \frac{-1}{\ell(y-x^*)^\ell}. \tag{4.17}$$

Since

$$|x-x^*| \leq \frac{1}{2}(\beta_t - \alpha_t) = \frac{1}{2}\mathrm{diam}_2(X_t) \tag{4.18}$$

and

$$|y-x^*| \geq \frac{1}{2}(\beta_t - \alpha_t) + (\alpha_s - \beta_t) = \frac{1}{2}\mathrm{diam}_2(X_t) + \mathrm{dist}_2(X_t, X_s) \tag{4.19}$$

hold, we obtain

$$\begin{aligned} \frac{|x-x^*|}{|y-x^*|} &\overset{(4.19)}{\leq} \frac{|x-x^*|}{\mathrm{dist}_2(X_t,X_s) + \mathrm{diam}_2(X_t)/2} \\ &\overset{(4.18)}{\leq} \frac{\mathrm{diam}_2(X_t)/2}{\mathrm{dist}_2(X_t,X_s) + \mathrm{diam}_2(X_t)/2} \\ &= \frac{1}{2\frac{\mathrm{dist}_2(X_t,X_s)}{\mathrm{diam}_2(X_t)} + 1} \overset{(4.15)}{\leq} \frac{1}{\frac{2}{\eta} + 1} = \frac{\eta}{2+\eta} \end{aligned} \tag{4.20}$$

Therefore, the remainder can be estimated by (cf [28, Remark 4.13])

$$\begin{aligned} |R_k(x,y,x^*)| &= \sum_{\ell=k}^{\infty} |x-x^*|^\ell \frac{-1}{\ell |y-x^*|^\ell} \\ &\overset{(4.20)}{\leq} \sum_{\ell=k}^{\infty} \frac{1}{\ell} \left(\frac{\eta}{2+\eta}\right) \\ &\leq \frac{1}{k} \sum_{\ell=k}^{\infty} \left(\frac{\eta}{2+\eta}\right) \\ &= \frac{2+\eta}{2k} \left(\frac{\eta}{2+\eta}\right)^k. \end{aligned} \tag{4.21}$$

With

$$a_\ell(x) = (x-x^*)^\ell \text{ and } b_\ell(y) = \begin{cases} \log(|y-x^*|) & \text{if } \ell = 0, \\ \frac{-1}{\ell(y-x^*)^\ell} & \text{else,} \end{cases}$$

for $\ell = 0, \ldots, k-1$, (4.16) yields a so-called separable expansion

$$T_k(x,y,x^*) = \sum_{\ell=0}^{k-1} a_\ell(x) b_\ell(y). \tag{4.22}$$

From this separable expansion, we can construct the matrix $(\widetilde{k}_{ij}) = \widetilde{\mathbf{K}} \in \mathbb{R}^{t \times s}$ with

$$\widetilde{k}_{ij} = \int_0^1 \int_0^1 \sum_{\ell=0}^{k-1} a_\ell(x) b_\ell(y) \varphi_i(x) \varphi_j(y) \, dx \, dy \tag{4.23a}$$

$$= \sum_{\ell=0}^{k-1} \int_0^1 a_\ell(x)\varphi_i(x)\,dx \int_0^1 b_\ell(y)\varphi_j(y)\,dy. \tag{4.23b}$$

As a consequence of (4.23b), $\widetilde{\mathbf{K}}$ is a rank-k matrix. Additionally, it approximates the block $\mathbf{K}|_{t\times s}$ with

$$\|\mathbf{K}|_{t\times s} - \widetilde{\mathbf{K}}\|_F = \mathcal{O}\left(\|R_k(x,y,x^*)\|_{\infty,X_t\times X_s}\right) = \mathcal{O}\left(\left(\frac{\eta}{2+\eta}\right)^k\right) \tag{4.24}$$

due to (4.21) (cf. [28, Section 4.6]). Note, that we could have also used

$$\operatorname{diam}_2(X_s) \leq \eta \operatorname{dist}_2(X_t, X_s) \tag{4.25}$$

instead of (4.15) to obtain similar results.

The above can also be extended to other integral operators with suitable kernels. For example, [3] and [16] discuss the existence of an $\mathcal{H}$-matrix for the Galerkin discretization of integral operators with Green's function as kernel function. This kind of integral operator describes the inverse of elliptic differential operators and is therefore closely related to the inverse of FEM matrices.

The conditions from (4.15) and (4.25) motivate the following admissibility condition.

Definition 4.30. *Let $\eta > 0$. Then the admissibility condition*

$$\operatorname{adm}_\eta : \mathcal{T}_\mathcal{I} \times \mathcal{T}_\mathcal{J} \to \{\text{true}, \text{false}\}$$

with

$$\operatorname{adm}_\eta(t,s) := \begin{cases} \text{true} & \text{if } \min\{\operatorname{diam}_2(X_t), \operatorname{diam}_2(Y_s)\} \leq \eta \operatorname{dist}_2(X_t, Y_s), \\ \text{false} & \text{else.} \end{cases}$$

is called η-admissibility condition *or* strong admissibility condition.

Note that the computation of the supports X_t and X_s can be costly. But we can replace the supports by corresponding bounding boxes B_t and B_s, respectively. The η-admissibility of a block $t \times s$ then uses

$$\min\{\operatorname{diam}_2(B_t), \operatorname{diam}_2(B_s)\} \leq \eta \operatorname{dist}_2(B_t, B_s).$$

Weaker admissibility conditions

Although the η-admissibility condition yields an accurate $\mathcal{H}$-matrix approximation with low local ranks, e.g., for the inverses of certain FEM matrices, the constant C_{sp} describing the sparsity of the block cluster tree may be large (especially for small η). If we assume that the sets X_i are locally separated (cf. Definition 4.22) and the cluster tree $\mathcal{T}_\mathcal{I}$ is constructed with Algorithm 4.1, the constant $C_{\text{sp}}(\mathcal{T}_{\mathcal{I}\times\mathcal{I}})$ can be estimated by

$$C_{\text{sp}}(\mathcal{T}_{\mathcal{I}\times\mathcal{I}}) = \mathcal{O}\left(\eta^{-d}\right)$$

as shown in [28, Lemma 6.17].

Thus, several weaker admissibility conditions were introduced, e.g., in [20, 22, 27]. The admissibility condition from [27] was also considered in [25] but was rejected there since this admissibility does not guarantee exponential convergence of the low-rank approximations for weakly admissible blocks, as it is the case for strong admissible blocks (cf. Equation (4.24)). Due to the block structure induced by this admissibility condition being called hierarchically off-diagonal low-rank (HODLR) format, we will call it HODLR-admissibility.

Definition 4.31. *The admissibility condition*

$$\mathrm{adm}_{\mathrm{HODLR}} : \mathcal{T}_{\mathcal{I}} \times \mathcal{T}_{\mathcal{I}} \to \{\mathrm{true}, \mathrm{false}\}$$

with

$$\mathrm{adm}_{\mathrm{HODLR}}(t,s) = \begin{cases} \mathrm{true}, & \text{if } t \neq s \\ \mathrm{false} & \text{else} \end{cases}$$

is called HODLR admissibility condition.

Note that the HODLR admissibility condition is only defined for the same row and column cluster tree, since only in this case the clusters are comparable. For a binary cluster tree $\mathcal{T}_{\mathcal{I}}$, the block tree $\mathcal{T}_{\mathcal{I}\times\mathcal{I}}$ constructed with $\mathrm{adm}_{\mathrm{HODLR}}$ yields

$$C_{\mathrm{sp}} = 2.$$

The other two admissibility conditions from [22] and [20] are tailored for sparse matrices $(a_{ij}) = \mathbf{A} \in \mathbb{R}^{\mathcal{I}\times\mathcal{J}}$, e.g. FEM matrices. It is assumed that $a_{ij} \neq 0$ holds if $X_i \cap Y_j \neq \emptyset$.

Therefore, [22] proposed the following admissibility condition allowing for a few non-zero entries in admissible blocks.

Definition 4.32. *Let $n_{\max} \in \mathbb{N}$. Then we call the admissibility condition*

$$\mathrm{adm}_{\mathrm{sp},n_{\max}} : \mathcal{T}_{\mathcal{I}} \times \mathcal{T}_{\mathcal{J}} \to \{\mathrm{true}, \mathrm{false}\}$$

with

$$\mathrm{adm}_{\mathrm{sp},n_{\max}}(t,s) = \begin{cases} \mathrm{true} & \text{if } \ |\{i \in t : X_i \cap Y_s \neq \emptyset\}| \leq n_{\max}, \\ \mathrm{false} & \text{else} \end{cases}$$

sparse admissibility condition.

If the admissible blocks are required to be of rank $k = 0$ instead, the following admissibility condition yields as large zero blocks as possible.

Definition 4.33. *We call the admissibility condition*

$$\mathrm{adm}_{\mathrm{weak}} : \mathcal{T}_{\mathcal{I}} \times \mathcal{T}_{\mathcal{J}} \to \{\mathrm{true}, \mathrm{false}\}$$

with

$$\mathrm{adm}_{\mathrm{weak}}(t,s) = \begin{cases} \mathrm{true} & \text{if } \mathrm{dist}_2(X_t, Y_s) > 0, \\ \mathrm{false} & \text{else.} \end{cases}$$

weak admissibility condition.

Domain decomposition admissibility

Last we will consider an admissibility condition fitted for square matrices and the domain decomposition clustering from Section 4.2.2. Since different domain clusters on the same level are not connected in the matrix graph they represent large zero blocks in the system which remain zero when computing an LU factorization. Thus, it makes sense to declare these blocks as admissible, which results in the following admissibility condition.

Definition 4.34. *Let the cluster tree $\mathcal{T}_\mathcal{I}$ be generated with the domain decomposition clustering from Algorithm 4.3. Let $\eta > 0$. Then we define the* strong domain decomposition admissibility condition

$$\mathrm{adm}_\eta^{\mathrm{DD}} : \mathcal{T}_\mathcal{I} \times \mathcal{T}_\mathcal{I} \to \{\mathrm{true}, \mathrm{false}\}$$

with

$$\mathrm{adm}_\eta^{\mathrm{DD}}(t,s) = \begin{cases} \mathrm{true}, & \text{if } t \neq s \text{ for } t,s \in \mathcal{C}_{dom} \\ \mathrm{adm}_\eta(t,s), & \text{else.} \end{cases} \tag{4.26}$$

By replacing the strong admissibility condition adm_η *in* (4.26) *by the weaker admissibility conditions* $\mathrm{adm}_{\mathrm{weak}}$ *from Definition 4.33 or* $\mathrm{adm}_{\mathrm{sp},n_{\max}}$ *from Definition 4.32, respectively, we obtain the weaker admissibility conditions* $\mathrm{adm}_{\mathrm{weak}}^{\mathrm{DD}}$ *and* $\mathrm{adm}_{\mathrm{sp},n_{\max}}^{\mathrm{DD}}$.

4.4 Arithmetic

The hierarchical block structure of $\mathcal{H}$-matrices reduces the memory complexity via the dyadic representation of low-rank matrices and also allows for efficient arithmetic. While the matrix-vector product for hierarchical matrices can be implemented straightforwardly, the matrix-matrix sum and product, as well as the LU factorization, require the handling of sums of low-rank matrices. We will now discuss these arithmetic operations, including estimates for the computational complexity. In Section 4.4.1 we will describe the $\mathcal{H}$-matrix-vector multiplication. Then Section 4.4.2 is concerned with the sum and product of $\mathcal{H}$-matrices. The latter necessitates a discussion on the resulting block structure. With the $\mathcal{H}$-matrix multiplication we will be able to describe the construction of approximations to the inverses of $\mathcal{H}$-matrices in Section 4.4.3 and the $\mathcal{H}$-LU factorization in Section 4.4.4. Throughout this section let $\mathcal{K}, \mathcal{I}, \mathcal{J}$ be finite index sets with cluster trees $\mathcal{T}_\mathcal{K}, \mathcal{T}_\mathcal{I}, \mathcal{T}_\mathcal{J}$ (cf. Definition 4.6) with a leaf size bound n_{leaf} (cf. Definition 4.13). Furthermore, let $\mathcal{T}_{\mathcal{I}\times\mathcal{J}}, \mathcal{T}_{\mathcal{I}\times\mathcal{K}}$ and $\mathcal{T}_{\mathcal{K}\times\mathcal{J}}$ be corresponding block cluster trees (cf. Definition 4.8) constructed with an admissibility condition as in Definition 4.10.

4.4.1 $\mathcal{H}$-matrix-vector multiplication

Most applications do not only require the matrix-vector product but also updating a vector by this product. Hence, we consider the update

$$\mathbf{y} \leftarrow \mathbf{y} + \alpha \mathbf{H}\mathbf{x} \tag{4.27}$$

for a hierarchical matrix $\mathbf{H} \in \mathcal{H}(\mathcal{T}_{\mathcal{I}\times\mathcal{J}}, k)$, $k \in \mathbb{N}$, and vectors $\mathbf{x} \in \mathbb{R}^\mathcal{J}$ and $\mathbf{y} \in \mathbb{R}^\mathcal{I}$. This update can be handled recursively. We start with the entire matrix $\mathbf{H}$, i.e., with the root $r = \mathrm{root}(\mathcal{T}_{\mathcal{I}\times\mathcal{J}})$ of the block cluster tree $\mathcal{T}_{\mathcal{I}\times\mathcal{J}}$. For any block $b = t \times s \in \mathcal{T}_{\mathcal{I}\times\mathcal{J}}$ we proceed as follows:

- If b has sons, we perform the updates

$$\mathbf{y}|_{t'} \leftarrow \mathbf{y}|_{t'} + \mathbf{H}|_{t'\times s'}\mathbf{x}|_{s'}$$

  for all $(t', s') \in \operatorname{sons}(b)$ recursively.

- If $b \in \mathcal{L}^+_{\mathcal{I}\times\mathcal{J}}$ is an admissible leaf, there are matrices $\mathbf{A}_b \in \mathbb{R}^{t\times k}$ and $\mathbf{B}_b \in \mathbb{R}^{s\times k}$ with $\mathbf{H}|_{t\times s} = \mathbf{A}_b\mathbf{B}_b^\mathsf{T}$ (cf. Definition 4.11). Thus, the update can be performed through

$$\begin{aligned}\widehat{\mathbf{x}} &\leftarrow \mathbf{B}_b^\mathsf{T}\mathbf{x}|_s,\\ \mathbf{y}|_t &\leftarrow \mathbf{y}|_t + \mathbf{A}_b\widehat{\mathbf{x}}.\end{aligned}$$

- If $b \in \mathcal{L}^-_{\mathcal{I}\times\mathcal{J}}$ is an inadmissible leaf, the update has to be performed directly.

The update is summarized in Algorithm 4.4.

Algorithm 4.4 $\mathcal{H}$-matrix-vector update

Input: Scaling factor α, block $b = (t, s) \in \mathcal{T}_{\mathcal{I}\times\mathcal{J}}$, $\mathcal{H}$-matrix $\mathbf{H} \in \mathcal{H}(\mathcal{T}_{(t,s)}, k)$, source vector $\mathbf{x} \in \mathbb{R}^s$, target vector $\mathbf{y} \in \mathbb{R}^t$ that is updated (see (4.27))

```
function addmul_hmatrix_vector(α, b, H, x, y)
    if b ∈ L⁺_{I×J} then
        Determine A ∈ R^{t×k} and B ∈ R^{s×k} with H = AB^T        ▷ cf. Definition 4.11
        x̂ ← B^T x ∈ R^k
        y ← y + αAx̂
    else
        if b ∈ L⁻_{I×J} then
            y ← y + αHx
        else
            for b' = (t', s') ∈ sons(b) do
                addmul_hmatrix_vector(α, b', H|_{b'}, x|_{s'}, y|_{t'})
            end for
        end if
    end if
end function
```

Lemma 4.35. *Let $\mathbf{H} \in \mathcal{H}(\mathcal{T}_{\mathcal{I}\times\mathcal{J}}, k)$, $\mathbf{x} \in \mathbb{R}^{\mathcal{J}}$ and $\mathbf{y} \in \mathbb{R}^{\mathcal{I}}$. Then the computational complexity $W_{\mathrm{MV}}(\mathcal{T}_{\mathcal{I}\times\mathcal{J}}, k)$ of the $\mathcal{H}$-matrix-vector update from Algorithm 4.4 can be estimated by the memory complexity $S_{\mathcal{H}}$ from Lemma 4.15:*

$$W_{\mathrm{MV}}(\mathcal{T}_{\mathcal{I}\times\mathcal{J}}, k) \leq 2S_{\mathcal{H}}(\mathcal{T}_{\mathcal{I}\times\mathcal{J}}, k).$$

Proof. See [28, Lemma 7.17]. □

4.4.2 (Standard) $\mathcal{H}$-matrix multiplication

Now, we aim to discuss the update

$$\mathbf{Z} \leftarrow \mathbf{Z} + \mathbf{XY} \tag{4.28}$$

for $\mathcal{H}$-matrices $\mathbf{X} \in \mathcal{H}(\mathcal{T}_{\mathcal{I}\times\mathcal{K}}, k_1), \mathbf{Y} \in \mathcal{H}(\mathcal{T}_{\mathcal{K}\times\mathcal{J}}, k_2)$, and $\mathbf{Z} \in \mathcal{H}(\mathcal{T}_{\mathcal{I}\times\mathcal{J}}, k_3)$ with local ranks $k_1, k_2, k_3 \in \mathbb{N}$. We could just apply the standard dense matrix arithmetic block by block. But typically we are interested in a particular format for the result. Additionally, the rank of admissible blocks may increase significantly. The latter typically occurs when adding matrices to low-rank matrices. Therefore, we first discuss the addition of $\mathcal{H}$-matrices. Then we will determine a suitable block structure for the representation of $\mathbf{XY}$ as an $\mathcal{H}$-matrix. Typically a different block structure is required in combination with a particular local rank. Thus, we will furthermore examine approximations of the exact product in the required formats. All this finally accumulates into a formatted update denoted by

$$\mathbf{Z} \leftarrow \mathbf{Z} \oplus \mathbf{X} \odot \mathbf{Y}.$$

Note that, apart from the standard algorithm introduced in [25], there are other variants developed in [5] and [11]. In this section, however, we will consider the standard algorithm while the discussion of the other variants is part of Chapter 6.

Addition

First, we illustrate the (formatted) $\mathcal{H}$-matrix addition for summands with the same block structure. Thus, let $k, \ell \in \mathbb{N}$ and $\mathbf{Z}_1 \in \mathcal{H}(\mathcal{T}_{\mathcal{I}\times\mathcal{J}}, k)$ and $\mathbf{Z}_2 \in \mathcal{H}(\mathcal{T}_{\mathcal{I}\times\mathcal{J}}, \ell)$.

Since the classical matrix addition is defined element-wise, we can handle the sum block by block. Inadmissible blocks, i.e. blocks $b \in \mathcal{L}^-_{\mathcal{I}\times\mathcal{J}}$ are handled directly. For admissible blocks $b = t \times s \in \mathcal{L}^+_{\mathcal{I}\times\mathcal{J}}$ Definition 4.11 yields matrices $\mathbf{A}_{b,1} \in \mathbb{R}^{t\times k}$, $\mathbf{A}_{b,2} \in \mathbb{R}^{t\times \ell}$, $\mathbf{B}_{b,1} \in \mathbb{R}^{s\times k}$, and $\mathbf{B}_{b,2} \in \mathbb{R}^{s\times \ell}$ with

$$\mathbf{Z}_1|_b = \mathbf{A}_{b,1}\mathbf{B}_{b,1}^\mathsf{T} \quad \text{and} \quad \mathbf{Z}_2|_b = \mathbf{A}_{b,2}\mathbf{B}_{b,2}^\mathsf{T}.$$

In this case, the sum can be expressed as

$$\mathbf{Z}_1|_b + \mathbf{Z}_2|_b = \mathbf{A}_{b,1}\mathbf{B}_{b,1}^\mathsf{T} + \mathbf{A}_{b,2}\mathbf{B}_{b,2}^\mathsf{T} = \begin{pmatrix}\mathbf{A}_{b,1} & \mathbf{A}_{b,2}\end{pmatrix}\begin{pmatrix}\mathbf{B}_{b,1} & \mathbf{B}_{b,2}\end{pmatrix}^\mathsf{T} \tag{4.29}$$

Then this matrix block of the sum can be represented as a rank-$(k+\ell)$ matrix by setting

$$\mathbf{A}_b = \begin{pmatrix}\mathbf{A}_{b,1} & \mathbf{A}_{b,2}\end{pmatrix} \in \mathbb{R}^{t\times(k+\ell)} \quad \text{and} \quad \mathbf{B}_b = \begin{pmatrix}\mathbf{B}_{b,1} & \mathbf{B}_{b,2}\end{pmatrix} \in \mathbb{R}^{s\times(k+\ell)}$$

satisfying $\mathbf{Z}_1|_b + \mathbf{Z}_2|_b = \mathbf{A}_b\mathbf{B}_b^\mathsf{T}$. Therefore, the rank of $\mathbf{Z}_1|_b + \mathbf{Z}_2|_b$ is bounded by $k + \ell$.

Hence, adding two low-rank matrices does not require any arithmetical operations but only an agglomeration of the corresponding factors to a rank-$(k+\ell)$ representation. The rank increase on the other hand may be problematic or undesirable, especially when this operation is performed repeatedly. The increase of the rank can be handled by truncating it to a desired quantity, e.g., with the methods explored in Section 2.1. This yields the formatted $\mathcal{H}$-matrix addition, which is summarized in Algorithm 4.5. The rank of the sum of low-rank matrices is truncated via SVD (see Remark 2.2) in that algorithm.

Since the addition of low-rank matrices has no computational cost, and the sum of dense matrices is only computed for the (potentially small) inadmissible blocks, the computational complexity of the truncated $\mathcal{H}$-matrix sum is dominated by the truncation of the low-rank blocks.

Algorithm 4.5 Formatted $\mathcal{H}$-matrix addition

Input: Block $b = (t,s) \in \mathcal{T}_{\mathcal{I}\times\mathcal{J}}$, source matrix $\mathbf{Z}_1 \in \mathcal{H}(\mathcal{T}_{(t,s)}, k)$, target matrix $\mathbf{Z}_2 \in \mathcal{H}(\mathcal{T}_{(t,s)}, \ell)$ overwritten by $\mathbf{Z}_1 \oplus_{\mathcal{H}} \mathbf{Z}_2$, maximal rank r

```
function add_hmatrix(b, Z1, Z2, r)
    if b ∈ L+_{I×J} then
        Determine (A1, B1) ∈ R(t, s, k) with A1 B1^T = Z1.      ▷ cf. Definition 4.11
        Determine (A2, B2) ∈ R(t, s, ℓ) with A2 B2^T = Z2.
        A ← (A1  A2)
        B ← (B1  B2)
        Z2 ← T^{k+ℓ}_{thin,r}(A, B)                             ▷ cf. Remark 2.2
    else
        if b ∈ L-_{I×J} then
            Z2 ← Z2 + Z1
        else
            for b' ∈ sons(b) do
                add_hmatrix(b', (Z1)|_{b'}, (Z2)|_{b'}, r)
            end for
        end if
    end if
end function
```

Lemma 4.36. *Let* $\widehat{k} := \max\{k_1, k_2, n_{\mathrm{leaf}}\}$ *and* $\widehat{\rho} := \max\{\mathrm{depth}(\mathcal{T}_{\mathcal{I}}), \mathrm{depth}(\mathcal{T}_{\mathcal{J}})\}$. *Then the computational complexity* W_{Add} *of the (truncated)* $\mathcal{H}$*-matrix addition is bounded by*

$$W_{\mathrm{Add}}(\mathcal{T}_{\mathcal{I}\times\mathcal{J}}, k_1, k_2) \leq 24\widehat{k}^2 C_{\mathrm{sp}}(\widehat{\rho}+1)(|\mathcal{I}|+|\mathcal{J}|) + 176\widehat{k}^3 C_{\mathrm{sp}}(|\mathcal{I}|+|\mathcal{J}|)$$

where C_{sp} *is the sparsity constant from Definition 4.14.*

Proof. See [21, Remark 2.14]. □

Exact $\mathcal{H}$-matrix product and induced block cluster tree

As it will turn out later, the product $\mathbf{XY}$ can be represented blockwise with sums of low-rank matrices. The block structure of this representation is induced by the block cluster trees $\mathcal{T}_{\mathcal{I}\times\mathcal{K}}$ and $\mathcal{T}_{\mathcal{K}\times\mathcal{J}}$ of the factors $\mathbf{X} \in \mathcal{H}(\mathcal{T}_{\mathcal{I}\times\mathcal{K}}, k_1)$ and $\mathbf{Y} \in \mathcal{H}(\mathcal{T}_{\mathcal{K}\times\mathcal{J}}, k_2)$, respectively. Therefore, we consider the product

$$\mathbf{X}|_{b_1}\mathbf{Y}|_{b_2} = \mathbf{X}|_{t\times r}\mathbf{Y}|_{r\times s}$$

for arbitrary blocks $b_1 = (t,r) \in \mathcal{T}_{\mathcal{I}\times\mathcal{K}}$ and $b_2 = (r,s) \in \mathcal{T}_{\mathcal{K}\times\mathcal{J}}$. Then we have to distinguish the following cases.

1. If $b_1 \in \mathcal{L}^+_{\mathcal{I}\times\mathcal{K}}$ or $b_2 \in \mathcal{L}^+_{\mathcal{K}\times\mathcal{J}}$, i.e., either b_1 or b_2 is an admissible leaf block, then there are matrices $\mathbf{A}_{b_1} \in \mathbb{R}^{t\times k_1}, \mathbf{B}_{b_1} \in \mathbb{R}^{r\times k_1}$ or $\mathbf{A}_{b_2} \in \mathbb{R}^{r\times k_2}, \mathbf{B}_{b_2} \in \mathbb{R}^{s\times k_2}$ with

$$\mathbf{X}|_{b_1} = \mathbf{A}_{b_1}\mathbf{B}_{b_1}^{\mathsf{T}} \quad \text{or} \quad \mathbf{Y}|_{b_2} = \mathbf{A}_{b_2}\mathbf{B}_{b_2}^{\mathsf{T}},$$

respectively, due to Definition 4.11. Therefore, we obtain

$$\mathbf{X}|_{b_1}\mathbf{Y}|_{b_2} = \mathbf{A}_b\mathbf{B}_b^\mathsf{T}$$

with $b := (t, s)$ and

$$\mathbf{A}_b := \mathbf{A}_{b_1} \in \mathbb{R}^{t\times k_1},\ \mathbf{B}_b := \mathbf{Y}|_{b_2}^\mathsf{T}\mathbf{B}_{b_1} \in \mathbb{R}^{s\times k_1} \tag{4.30}$$

or

$$\mathbf{A}_b := \mathbf{X}|_{b_1}\mathbf{A}_{b_2} \in \mathbb{R}^{t\times k_2},\ \mathbf{B}_b := \mathbf{B}_{b_2} \in \mathbb{R}^{s\times k_2}, \tag{4.31}$$

respectively. Note, that if one of the blocks b_1 and b_2 is not a leaf, the corresponding product

$$\mathbf{X}|_{b_1}\mathbf{A}_{b_2} \quad \text{or} \quad \mathbf{Y}|_{b_2}^\mathsf{T}\mathbf{B}_{b_1}$$

can be computed with k_1 or k_2 $\mathcal{H}$-matrix-vector products, respectively.

2. If $b_1 \in \mathcal{L}^-_{\mathcal{I}\times\mathcal{K}}$ or $b_2 \in \mathcal{L}^-_{\mathcal{K}\times\mathcal{J}}$, then $\mathbf{X}|_{b_1}$ or $\mathbf{Y}|_{b_2}$ is a dense matrix and either $t \in \mathcal{L}_\mathcal{I}$, $r \in \mathcal{L}_\mathcal{K}$, or $s \in \mathcal{L}_\mathcal{J}$ (cf. Definition 4.10), i.e. $|t| \le n_{\text{leaf}}$, $|r| \le n_{\text{leaf}}$, or $|s| \le n_{\text{leaf}}$. Therefore, we have

$$\mathbf{X}|_{b_1}\mathbf{Y}|_{b_2} = \mathbf{A}_b\mathbf{B}_b,$$

where $(\mathbf{A}_b, \mathbf{B}_b) \in \mathcal{R}(t, s, n_{\text{leaf}})$ can be computed as follows:

   (a) If $|t| \le n_{\text{leaf}}$ then b_1 is an inadmissible leaf, and we set

$$\mathbf{A}_b := \mathbf{I}_t, \quad \mathbf{B}_b := \mathbf{X}|_{b_1}\mathbf{Y}|_{b_2}. \tag{4.32}$$

   (b) If $|r| \le n_{\text{leaf}}$ then both, b_1 and b_2, are inadmissible leafs, and we set

$$\mathbf{A}_b := \mathbf{X}|_{b_1}, \quad \mathbf{B}_b := \mathbf{Y}|_{b_2}. \tag{4.33}$$

   (c) If $|s| \le n_{\text{leaf}}$ then b_2 is an inadmissible leaf, and we set

$$\mathbf{A}_b := \mathbf{X}|_{b_1}\mathbf{Y}|_{b_2}, \quad \mathbf{B}_b := \mathbf{I}_s. \tag{4.34}$$

   The occurring products can be computed with at most n_{leaf} $\mathcal{H}$-matrix-vector multiplications.

3. If $b_1 \notin \mathcal{L}_{\mathcal{I}\times\mathcal{K}}$ and $b_2 \notin \mathcal{L}_{\mathcal{K}\times\mathcal{J}}$, then both blocks have sons with

$$\text{sons}(b_1) = \{(t', r') : t' \in \text{sons}(t), r' \in \text{sons}(r)\}$$

and

$$\text{sons}(b_2) = \{(r', s') : r' \in \text{sons}(r), s' \in \text{sons}(s)\}.$$

Therefore, the product $\mathbf{X}|_{t\times r}\mathbf{Y}|_{r\times s}$ can be handled block-wise with

$$(\mathbf{X}|_{t\times r}\mathbf{Y}|_{r\times s})|_{t'\times s'} = \sum_{r'\in\text{sons}(r)} \mathbf{X}|_{t'\times r'}\mathbf{Y}|_{r'\times s'}. \tag{4.35}$$

Remark 4.37. *Let* $r_{\mathcal{I}} := \operatorname{root}(\mathcal{T}_{\mathcal{I}})$, $r_{\mathcal{K}} := \operatorname{root}(\mathcal{T}_{\mathcal{K}})$, *and* $r_{\mathcal{J}} := \operatorname{root}(\mathcal{T}_{\mathcal{J}})$. *Then the discussion above induces the following block cluster tree* $\mathcal{T}^{\mathrm{ind}}_{\mathcal{I}\times\mathcal{J}}$ *defining a block structure for the exact product. The root is* $\operatorname{root}(\mathcal{T}^{\mathrm{ind}}_{\mathcal{I}\times\mathcal{J}}) = (r_{\mathcal{I}}, r_{\mathcal{J}})$. *For a block* $b = (t, s) \in \mathcal{T}^{\mathrm{ind}}_{\mathcal{I}\times\mathcal{J}}$, *the set of sons is*

$$\operatorname{sons}(b) = \{(t', s') : t' \in \operatorname{sons}(t), s' \in \operatorname{sons}(s)\}$$

if there is a cluster $r \in \mathcal{T}_{\mathcal{K}}$ *with* $(t, r) \in \mathcal{T}_{\mathcal{I}\times\mathcal{K}} \setminus \mathcal{L}_{\mathcal{I}\times\mathcal{K}}$ *and* $(r, s) \in \mathcal{T}_{\mathcal{K}\times\mathcal{J}} \setminus \mathcal{L}_{\mathcal{K}\times\mathcal{J}}$. *If, on the other hand, for all* $r \in \mathcal{T}_{\mathcal{K}}$ *with* $(t, r) \in \mathcal{T}_{\mathcal{I}\times\mathcal{K}}$ *and* $(r, s) \in \mathcal{T}_{\mathcal{K}\times\mathcal{J}}$ *the block* (t, r) *or* (r, s) *is a leaf, respectively, the set of sons is*

$$\operatorname{sons}(b) = \emptyset.$$

In the following, we will introduce some helpful definitions to express the result of the product on the leaf blocks of $\mathcal{T}^{\mathrm{ind}}_{\mathcal{I}\times\mathcal{J}}$. Due to the tree structure of (block-) cluster trees, each cluster (or block) has uniquely determined predecessors on the prior levels.

Definition 4.38. *Let* $\mathcal{I}$ *be an index set with cluster tree* $\mathcal{T}_{\mathcal{I}}$. *Let* $t \in \mathcal{T}_{\mathcal{I}}$ *with* $\ell := \operatorname{level}(t) > 0$. *Then there is a unique predecessor* $\bar{t} \in \mathcal{T}_{\mathcal{I}}$ *with* $t \in \operatorname{sons}(\bar{t})$. *For* $j \in \{0, \dots \ell\}$, *we define the* predecessor $\mathcal{P}^j(t)$ of t on level j *recursively via*

$$\mathcal{P}^j(t) = \begin{cases} t & , \text{if } j = \ell \\ \bar{t} & , \text{if } j = \ell - 1 \\ \mathcal{P}^j(\bar{t}) & , \text{else.} \end{cases}$$

This allows us to describe a block $(\mathbf{XY})|_b$ of the product for $b = (t, s) \in \mathcal{T}^{\mathrm{ind}}_{\mathcal{I}\times\mathcal{J}}$. The definition for sub-blocks in (4.35) implies that

$$(\mathbf{XY})|_b = \sum_{\substack{r\in\mathcal{T}_{\mathcal{K}} \\ \operatorname{level}(r)=\operatorname{level}(b)}} \mathbf{X}|_{t\times r}\mathbf{Y}|_{r\times s}, \tag{4.36}$$

where the summands $\mathbf{X}|_{t\times r}\mathbf{Y}|_{r\times s}$ may be a hierarchical matrix or a low-rank matrix. The latter case occurs, if there was a cluster $r^* \in \mathcal{T}_{\mathcal{K}}$ with $j := \operatorname{level}(r^*)$ such that $(\mathcal{P}^j(t), r^*) \in \mathcal{L}_{\mathcal{I}\times\mathcal{K}}$ or $(r^*, \mathcal{P}^j(s)) \in \mathcal{L}_{\mathcal{K}\times\mathcal{J}}$ (see Figure 4.4).

When b is not a leaf block, i.e., $b \notin \mathcal{L}(\mathcal{T}^{\mathrm{ind}}_{\mathcal{I}\times\mathcal{J}})$, there is at least one summand in (4.36) which is a hierarchical matrix. It is no problem to subdivide these to obtain a representation for all sub-blocks of b. But if there is a cluster $r \in \mathcal{T}_{\mathcal{K}}$ such that $(t, r) \in \mathcal{L}_{\mathcal{I}\times\mathcal{K}}$ or $(r, s) \in \mathcal{L}_{\mathcal{K}\times\mathcal{J}}$, there is one summand which is a low-rank matrix. This has to be subdivided as depicted in Figure 4.5 to obtain a representation similar to (4.36) for all sub-blocks of b.

The discussion above imposes that, on the other hand, if $b \in \mathcal{L}(\mathcal{T}^{\mathrm{ind}}_{\mathcal{I}\times\mathcal{J}})$ is a leaf block, we can represent this block as a sum of low-rank matrices. First, we define the following sets helping to describe low-rank summands occurring in representations of non-leaf blocks.

Definition 4.39. *Let* $b := (t, s) \in \mathcal{T}^{\mathrm{ind}}_{\mathcal{I}\times\mathcal{J}}$ *with* $\ell := \operatorname{level}(b) = \operatorname{level}(t) = \operatorname{level}(s)$. *For* $j \in \{0, \dots, \ell\}$, *we define the sets*

$$\mathcal{U}^j(t, s) := \left\{ r \in \mathcal{T}_{\mathcal{K}} : \begin{array}{c} \mathcal{P}^j(t) \times r \in \mathcal{T}_{\mathcal{I}\times\mathcal{K}} \text{ and } r \times \mathcal{P}^j(s) \in \mathcal{L}_{\mathcal{K}\times\mathcal{J}} \\ \text{or} \\ \mathcal{P}^j(t) \times r \in \mathcal{L}_{\mathcal{I}\times\mathcal{K}} \text{ and } r \times \mathcal{P}^j(s) \in \mathcal{T}_{\mathcal{K}\times\mathcal{J}} \end{array} \right\},$$

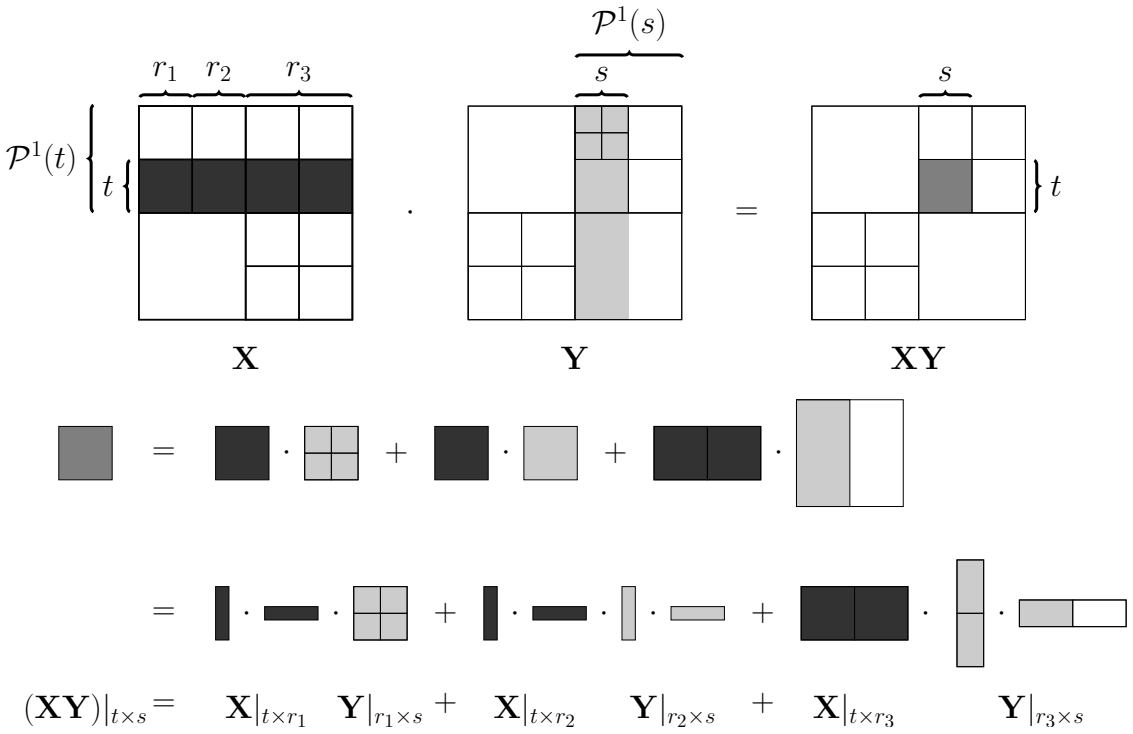

Figure 4.4: Example for the computation for a block of the product $\mathbf{XY}$. Here we have $(t, r_1) \in \mathcal{L}_{\mathcal{T}_{\mathcal{I}\times\mathcal{K}}}$, $(t, r_2) \in \mathcal{L}_{\mathcal{I}\times\mathcal{K}}$, and $(r_3, \mathcal{P}^1(s)) \in \mathcal{L}_{\mathcal{K}\times\mathcal{J}}$. Therefore, $\mathbf{X}|_{t\times r_1}\mathbf{Y}|_{r_1\times s}$, $\mathbf{X}|_{t\times r_2}\mathbf{Y}|_{r_2\times s}$ and $\mathbf{X}|_{\mathcal{P}^1(t)\times r_3}\mathbf{Y}|_{r_3\times\mathcal{P}^1(s)}$ are low-rank matrices, as depicted in Equations (4.30) to (4.34). They are restricted to $t \times s$ (gray shaded parts) and are summed up to obtain the result $(\mathbf{XY})|_{t\times s}$.

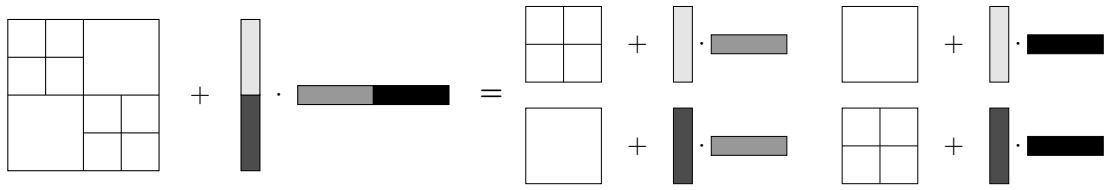

Figure 4.5: Block-wise computation of the sum of a hierarchical matrix and a low-rank matrix. The left (light gray/dark gray) and right (gray/black) low-rank factor is split, respectively, to obtain a block structure similar to the left summand.

and their union

$$\mathcal{U}(t,s) := \bigcup_{j=0}^{\ell} \mathcal{U}^j(t,s) \subseteq \mathcal{T}_{\mathcal{K}}.$$

Now, we utilize these sets to find a block-wise expression of the product as a sum of low-rank matrices (see also Figure 4.4).

Lemma 4.40. *Let $b := (t,s) \in \mathcal{T}^{\mathrm{ind}}_{\mathcal{I}\times\mathcal{J}}$ with $\ell := \operatorname{level}(b) = \operatorname{level}(t) = \operatorname{level}(s)$. Then*

$$\mathcal{K} = \bigcup_{j=0}^{\ell} \bigcup_{r\in\mathcal{U}^j(t,s)} r$$

is a disjoint union, and

$$(\mathbf{XY})|_{t\times s} = \sum_{j=0}^{\ell} \sum_{r\in\mathcal{U}^j(t,s)} \mathbf{X}|_{t\times r}\mathbf{Y}|_{r\times s} = \sum_{r\in\mathcal{U}(t,s)} \mathbf{X}|_{t\times r}\mathbf{Y}|_{r\times s}, \tag{4.37}$$

with all summands being low-rank matrices.

Proof. See [21, Lemma 2.19]. □

The analysis of the computational complexity necessitates an estimate for the number of summands in (4.37).

Proposition 4.41. *Let $b = (t,s) \in \mathcal{L}^{\mathrm{ind}}_{\mathcal{I}\times\mathcal{J}}$. Then*

$$|\mathcal{U}^j(t,s)| \leq \min\{C_{\mathrm{sp}}(\mathcal{T}_{\mathcal{I}\times\mathcal{K}}), C_{\mathrm{sp}}(\mathcal{T}_{\mathcal{K}\times\mathcal{J}})\}$$

holds for all $j \in \{0,\ldots,\operatorname{level}(b)\}$, and therefore

$$|\mathcal{U}(t,s)| \leq (\min\{\operatorname{depth}(\mathcal{T}_{\mathcal{I}\times\mathcal{K}}), \operatorname{depth}(\mathcal{T}_{\mathcal{K}\times\mathcal{J}})\}+1)\min\{C_{\mathrm{sp}}(\mathcal{T}_{\mathcal{I}\times\mathcal{K}}), C_{\mathrm{sp}}(\mathcal{T}_{\mathcal{K}\times\mathcal{J}})\}.$$

Proof. See [28, Remark 7.33]. □

With this upper bound, we can estimate the computational complexity for the exact $\mathcal{H}$-matrix product.

Theorem 4.42. *Let $\widehat{\rho} := \max\{\mathcal{T}_{\mathcal{I}}, \mathcal{T}_{\mathcal{K}}, \mathcal{T}_{\mathcal{J}}\}$, and let $\mathbf{W} := \mathbf{XY}$ be the result of the exact $\mathcal{H}$-matrix product. Let $C_{\mathrm{sp}} := \max\{C_{\mathrm{sp}}(\mathcal{T}_{\mathcal{I}\times\mathcal{K}}), C_{\mathrm{sp}}(\mathcal{T}_{\mathcal{K}\times\mathcal{J}})\}$ (cf. Definition 4.14) and $\widehat{k} := \max\{k_1, k_2, n_{\mathrm{leaf}}\}$. Then*

$$\mathbf{W} \in \mathcal{H}(\mathcal{T}^{\mathrm{ind}}_{\mathcal{I}\times\mathcal{J}}, \widetilde{k})$$

holds with

$$\widetilde{k} \leq C_{\mathrm{sp}}(\widehat{\rho}+1)\widehat{k}.$$

The computational complexity W_{exact} for the exact $\mathcal{H}$-matrix product is bounded by

$$W_{\mathrm{exact}}(\mathcal{T}_{\mathcal{I}\times\mathcal{K}}, \mathcal{T}_{\mathcal{K}\times\mathcal{J}}, k_1, k_2) \leq C_{\mathrm{exact}}(\widehat{\rho}+1)^2\widehat{k}^2(|\mathcal{I}|+|\mathcal{K}|+|\mathcal{J}|),$$

where

$$C_{\mathrm{exact}} = 4C_{\mathrm{sp}}^3$$

Proof. See [21, Theorem 2.20]. □

Truncated $\mathcal{H}$-matrix update

In the previous section, we have expressed the (exact) product block-wise as a sum of low-rank matrices. The block structure for this representation was obtained from an induced block cluster tree $\mathcal{T}^{\mathrm{ind}}_{\mathcal{I}\times\mathcal{J}}$ (cf. Remark 4.37) depending on the factors' $\mathbf{X}$ and $\mathbf{Y}$ block cluster trees $\mathcal{T}_{\mathcal{I}\times\mathcal{K}}$ and $\mathcal{T}_{\mathcal{K}\times\mathcal{J}}$. In most applications, on the other hand, the result is required to have a predefined block structure described by another block cluster tree $\mathcal{T}_{\mathcal{I}\times\mathcal{J}}$ as well as a desired maximal local rank $k \in \mathbb{N}$. This leads to the truncated product $\mathbf{X} \odot \mathbf{Y}$ which can be computed from the exact product. But first, we have to deal with the following two major problems:

1. The blocks of the exact product are sums of low-rank matrices while the $\mathcal{H}$-matrix format requires low-rank and dense matrix blocks.

2. The block structures defined by the block cluster trees $\mathcal{T}_{\mathcal{I}\times\mathcal{J}}$ and $\mathcal{T}^{\mathrm{ind}}_{\mathcal{I}\times\mathcal{J}}$ differ.

The first can be handled by computing these sums explicitly for inadmissible leaf blocks and truncating them to the required rank for admissible leaf blocks.

The second requires a detailed examination of the mentioned differences to convert the block structure. Since $\mathcal{T}_{\mathcal{I}\times\mathcal{J}}$ and $\mathcal{T}^{\mathrm{ind}}_{\mathcal{I}\times\mathcal{J}}$ are block cluster trees for the same cluster trees $\mathcal{T}_{\mathcal{I}}$ and $\mathcal{T}_{\mathcal{J}}$, they can only differ if there are leaf blocks in one of the block cluster trees which have sons in the other tree, i.e. blocks $b = (t, s)$ with $b \in \mathcal{T}_{\mathcal{I}\times\mathcal{J}}$ and $b \in \mathcal{T}^{\mathrm{ind}}_{\mathcal{I}\times\mathcal{J}}$ with either $b \in \mathcal{L}\left(\mathcal{T}_{\mathcal{I}\times\mathcal{J}}\right)$ or $b \in \mathcal{L}\left(\mathcal{T}^{\mathrm{ind}}_{\mathcal{I}\times\mathcal{J}}\right)$.

We first consider the case $b \in \mathcal{L}(\mathcal{T}_{\mathcal{I}\times\mathcal{J}})$ but $b \notin \mathcal{L}(\mathcal{T}^{\mathrm{ind}}_{\mathcal{I}\times\mathcal{J}})$. Then b is an admissible leaf of $\mathcal{T}_{\mathcal{I}\times\mathcal{J}}$ (cf. Definition 4.10). In $\mathcal{T}^{\mathrm{ind}}_{\mathcal{I}\times\mathcal{J}}$, on the other hand, b has sons. Then we compute the exact results, i.e., we determine $k_{b'} \in \mathbb{N}$ and $\mathbf{A}_{b'} \in \mathbb{R}^{t'\times k_{b'}}$ and $\mathbf{B}_{b'} \in \mathbb{R}^{s'\times k_{b'}}$ with $\mathbf{A}_{b'}\mathbf{B}_{b'}^{\mathsf{T}} = (\mathbf{XY})|_{b'}$ for all $b' = (t', s') \in \mathrm{sons}(b)$. Then we can truncate the agglomerations

$$\widehat{\mathbf{A}}_b := \sum_{b'\in\mathrm{sons}(b)} \mathbf{A}_{b'}|^{t\times k_{b'}}, \quad \widehat{\mathbf{B}}_b := \sum_{b'\in\mathrm{sons}(b)} \mathbf{B}_{b'}|^{s\times k_{b'}} \tag{4.38}$$

to the required rank, i.e. we set

$$(\mathbf{X} \odot \mathbf{Y})|_b := \mathbf{A}_b\mathbf{B}_b^{\mathsf{T}} \text{ with } (\mathbf{A}_b, \mathbf{B}_b) = \mathfrak{T}^{\widetilde{k}}_{\mathrm{thin},k}(\widehat{\mathbf{A}}_b, \widehat{\mathbf{B}}_b),$$

where $\widetilde{k} = \sum_{b'\in\mathrm{sons}(b)} k_{b'}$.

In the other case, i.e. $b \in \mathcal{L}(\mathcal{T}^{\mathrm{ind}}_{\mathcal{I}\times\mathcal{J}})$ but $b \notin \mathcal{L}(\mathcal{T}_{\mathcal{I}\times\mathcal{J}})$, the exact result $\mathbf{W}_b = (\mathbf{XY})|_b$ is a sum of low-rank matrices due to Lemma 4.40. This sum has to be computed as

$$\mathbf{W}_b = \mathbf{A}_{\mathbf{W}}\mathbf{B}_{\mathbf{W}}$$

with $\mathbf{A}_{\mathbf{W}} \in \mathbb{R}^{t\times k_b}, \mathbf{B}_{\mathbf{W}} \in \mathbb{R}^{s\times k_b}$, $k_b \in \mathbb{N}$. This result can then be truncated to the desired local rank to obtain the rank-k representation

$$\mathbf{A}_b\mathbf{B}_b^{\mathsf{T}} = \mathfrak{T}^{k_b}_{\mathrm{thin},k}(\mathbf{A}_{\mathbf{W}}, \mathbf{B}_{\mathbf{W}}). \tag{4.39}$$

In the next step, we split $\mathbf{A}_b$ and $\mathbf{B}_b$ to obtain the formatted result $(\mathbf{A}_b\mathbf{B}_b^{\mathsf{T}})|_{b'}$ for all sons $b' \in \mathrm{sons}(b)$ in $\mathcal{T}_{\mathcal{I}\times\mathcal{J}}$ (cf. Figure 4.5).

The truncated $\mathcal{H}$-matrix product requires, in addition to the exact multiplication, the truncation of low-rank matrices. This truncation can be done afterwards for the exact result to acquire the best approximation for the truncated low-rank blocks via an SVD or pairwise through truncating intermediate results (cf. Section 2.1) Note that the sum in (4.38) is an agglomeration of smaller, disjoint matrix blocks to a larger matrix block. This operation itself does not require any arithmetic operations (cf. (4.29)). These agglomerations occur whenever the block structure from the induced block cluster tree $\mathcal{T}_{\mathcal{I}\times\mathcal{J}}^{\mathrm{ind}}$ is finer than the block structure from $\mathcal{T}_{\mathcal{I}\times\mathcal{J}}$. Therefore, we have to quantify this difference between $\mathcal{T}_{\mathcal{I}\times\mathcal{J}}$ and $\mathcal{T}_{\mathcal{I}\times\mathcal{J}}^{\mathrm{ind}}$ (cf. [21, 28]).

Definition 4.43. *Let $b = (t, s) \in \mathcal{T}_{\mathcal{I}\times\mathcal{J}}$ with $\ell := \mathrm{level}(b)$. We define the elementwise idempotency constant $C_{\mathrm{id}}(t \times s)$ by*

$$C_{\mathrm{id}}(t, s, \mathcal{T}_{\mathcal{I}\times\mathcal{J}}^{\mathrm{ind}}) := |\{(t', s') \in \mathcal{T}_{\mathcal{I}\times\mathcal{J}}^{\mathrm{ind}} : t = \mathcal{P}^{\ell}(t'), s = \mathcal{P}^{\ell}(s')\}|,$$

and the block cluster tree idempotency constant $C_{\mathrm{id}}(\mathcal{T}_{\mathcal{I}\times\mathcal{J}})$ by

$$C_{\mathrm{id}}(\mathcal{T}_{\mathcal{I}\times\mathcal{J}}, \mathcal{T}_{\mathcal{I}\times\mathcal{J}}^{\mathrm{ind}}) := \max_{(t,s)\in\mathcal{L}_{\mathcal{I}\times\mathcal{J}}} C_{\mathrm{id}}(t, s, \mathcal{T}_{\mathcal{I}\times\mathcal{J}}^{\mathrm{ind}}).$$

If the block cluster tree $\mathcal{T}_{\mathcal{I}\times\mathcal{J}}$ is known from the context, we denote $C_{\mathrm{id}}(\mathcal{T}_{\mathcal{I}\times\mathcal{J}})$ by C_{id}.

If the induced tree $\mathcal{T}_{\mathcal{I}\times\mathcal{J}}^{\mathrm{ind}}$ is identical to $\mathcal{T}_{\mathcal{I}\times\mathcal{J}}$, the idempotency constant is $C_{\mathrm{id}} = 1$. The same holds if the induced block cluster tree is coarser than $\mathcal{T}_{\mathcal{I}\times\mathcal{J}}$. If, on the other hand, the induced block cluster tree $\mathcal{T}_{\mathcal{I}\times\mathcal{J}}^{\mathrm{ind}}$ is finer than $\mathcal{T}_{\mathcal{I}\times\mathcal{J}}$, i.e., there are blocks b with $b \in \mathcal{T}_{\mathcal{I}\times\mathcal{J}}^{\mathrm{ind}} \setminus \mathcal{L}_{\mathcal{I}\times\mathcal{J}}^{\mathrm{ind}}$ and $b \in \mathcal{L}_{\mathcal{I}\times\mathcal{J}}$, then the idempotency constant is bounded from below by $C_{\mathrm{id}} > 1$. In this case, the formatted multiplication requires additional truncated agglomerations.

The idempotency constant allows an estimation of the rank increase from agglomerations and therefore allows an estimate of the computational complexity of the truncated $\mathcal{H}$-matrix product.

Theorem 4.44. *Let $\widehat{\rho} := \max\{\mathrm{depth}(\mathcal{T}_{\mathcal{I}}), \mathrm{depth}(\mathcal{T}_{\mathcal{K}}), \mathrm{depth}(\mathcal{T}_{\mathcal{J}})\}$, and let $\mathbf{W} := \mathbf{XY}$ be the result of the exact $\mathcal{H}$-matrix product. Let $C_{\mathrm{sp}} := \max\{C_{\mathrm{sp}}(\mathcal{T}_{\mathcal{I}\times\mathcal{K}}), C_{\mathrm{sp}}(\mathcal{T}_{\mathcal{K}\times\mathcal{J}})\}$ and $\widehat{k} := \max\{k_1, k_2, n_{\mathrm{leaf}}\}$. Then*

$$\mathbf{W} \in \mathcal{H}(\mathcal{T}_{\mathcal{I}\times\mathcal{J}}, \widetilde{k})$$

with

$$\widetilde{k} \leq C_{\mathrm{id}} C_{\mathrm{sp}} (\widehat{\rho} + 1)\widehat{k}.$$

The truncated $\mathcal{H}$-matrix product with a single truncation for all admissible blocks that yields the best approximation for these blocks has a computational complexity W_{best} of

$$W_{\mathrm{best}}(\mathcal{T}_{\mathcal{I}\times\mathcal{K}}, \mathcal{T}_{\mathcal{K}\times\mathcal{J}}, k_1, k_2) \leq C_{\mathrm{best}}(\widehat{\rho} + 1)^3(|\mathcal{I}| + |\mathcal{K}| + |\mathcal{J}|),$$

where

$$C_{\mathrm{best}} = 32 C_{\mathrm{sp}}^4 C_{\mathrm{id}}^3.$$

The truncated $\mathcal{H}$-matrix product with pairwise truncation, on the other hand, has a computational complexity W_{pw} of

$$\begin{aligned} W_{\mathrm{pw}}(\mathcal{T}_{\mathcal{I}\times\mathcal{K}}, \mathcal{T}_{\mathcal{K}\times\mathcal{J}}, k_1, k_2) &\leq C_{\mathrm{pw}} \left(\widehat{k}^2(\widehat{\rho}+1)^2(|\mathcal{I}|+|\mathcal{K}|+|\mathcal{J}|) + \frac{22}{3}\widehat{k}^3(\widehat{\rho}+1)(|\mathcal{I}|+|\mathcal{J}|) \right) \\ &= \mathcal{O}\left((\widehat{\rho}+1)^2(|\mathcal{I}|+|\mathcal{K}|+|\mathcal{J}|) \right) \end{aligned}$$

with

$$C_{\mathrm{pw}} = 28 C_{\mathrm{sp}}^3 C_{\mathrm{id}}.$$

Proof. See [21, Thm. 2.24]. □

By combining the truncated addition and the truncated product, we obtain the truncated $\mathcal{H}$-matrix update

$$\mathbf{Z} \leftarrow \mathbf{Z} \oplus \alpha(\mathbf{X} \odot \mathbf{Y})$$

summarized in Algorithms 4.6 to 4.9 for pairwise truncations.

4.4.3 $\mathcal{H}$-matrix inversion

The (approximate) inversion of $\mathcal{H}$-matrices can be derived from the inversion of block matrices. Therefore, we first consider the following factorization of a 2×2 block matrix $\mathbf{M}$, where $\mathbf{M}_{11}$ has to be regular:

$$\mathbf{M} := \begin{pmatrix} \mathbf{M}_{11} & \mathbf{M}_{12} \\ \mathbf{M}_{21} & \mathbf{M}_{22}. \end{pmatrix} = \begin{pmatrix} \mathbf{M}_{11} & \\ \mathbf{M}_{21} & \mathbf{I} \end{pmatrix} \begin{pmatrix} \mathbf{I} & \mathbf{M}_{11}^{-1}\mathbf{M}_{12} \\ & \mathbf{S} \end{pmatrix}$$

with $\mathbf{S} = \mathbf{M}_{22} - \mathbf{M}_{21}\mathbf{M}_{11}^{-1}\mathbf{M}_{12}$. Hence, if in addition to $\mathbf{M}_{11}$ the Schur complement $\mathbf{S}$ is also regular, the inverse

$$\mathbf{M}^{-1} =: \mathbf{N} = \begin{pmatrix} \mathbf{N}_{11} & \mathbf{N}_{12} \\ \mathbf{N}_{21} & \mathbf{N}_{22} \end{pmatrix}$$

of $\mathbf{M}$ is given by

$$\begin{aligned} \mathbf{N} &= \begin{pmatrix} \mathbf{I} & \mathbf{M}_{11}^{-1}\mathbf{M}_{12} \\ & \mathbf{S} \end{pmatrix}^{-1} \begin{pmatrix} \mathbf{M}_{11} & \\ \mathbf{M}_{21} & \mathbf{I} \end{pmatrix}^{-1} \\ &= \begin{pmatrix} \mathbf{I} & -\mathbf{M}_{11}^{-1}\mathbf{M}_{12}\mathbf{S}^{-1} \\ & \mathbf{S}^{-1} \end{pmatrix} \begin{pmatrix} \mathbf{M}_{11}^{-1} & \\ -\mathbf{M}_{21}\mathbf{M}_{11}^{-1} & \mathbf{I} \end{pmatrix} \\ &= \begin{pmatrix} \mathbf{M}_{11}^{-1} + \mathbf{M}_{11}^{-1}\mathbf{M}_{12}\mathbf{S}^{-1}\mathbf{M}_{21}\mathbf{M}_{11}^{-1} & -\mathbf{M}_{11}^{-1}\mathbf{M}_{12}\mathbf{S}^{-1} \\ -\mathbf{S}^{-1}\mathbf{M}_{21}\mathbf{M}_{11}^{-1} & \mathbf{S}^{-1} \end{pmatrix}. \end{aligned}$$

Therefore, the inversion of the block matrix $\mathbf{M}$ can be reduced to matrix multiplications of the blocks and the recursive inversion of $\mathbf{M}_{11}$ and $\mathbf{S}$, respectively, by

- determine $\widehat{\mathbf{N}}_{11} = \mathbf{M}_{11}^{-1}$ (4.40a)

Algorithm 4.6 Truncated update of a rank-k' matrix by the product of two $\mathcal{H}$-matrices

Input: Scaling factor $\alpha \in \mathbb{R}$, clusters $t \in \mathcal{T}_{\mathcal{I}}$, $r \in \mathcal{T}_{\mathcal{K}}$ and $s \in \mathcal{T}_{\mathcal{J}}$ with $(t,r) \in \mathcal{T}_{\mathcal{I}\times\mathcal{K}}$ and $(r,s) \in \mathcal{T}_{\mathcal{K}\times\mathcal{J}}$, $\mathcal{H}$-matrices $\mathbf{X} \in \mathcal{H}(\mathcal{T}_{(t,r)}, k_1)$ and $\mathbf{Y} \in \mathcal{H}(\mathcal{T}_{(r,s)}, k_2)$, rank-$k'$ factors $(\mathbf{A},\mathbf{B}) \in \mathcal{R}(t,s,k')$ that are overwritten by rank-k factors of the sum $\mathbf{A}\mathbf{B}^{\mathsf{T}} + \alpha\mathbf{X}\mathbf{Y}$ truncated to a target local rank k.

function `addmul_rkmatrix_hmatrix`(α, t, r, s, $\mathbf{X}$, $\mathbf{Y}$, $\mathbf{A}$, $\mathbf{B}$, k)
  if $(t,r) \in \mathcal{L}_{\mathcal{I}\times\mathcal{K}}$ **then**
    Determine $(\mathbf{A}_\Pi, \mathbf{B}_\Pi) \in \mathcal{R}(t,s,k_1)$ with $\mathbf{X}\mathbf{Y} = \mathbf{A}_\Pi\mathbf{B}_\Pi^{\mathsf{T}}$ ▷ cf. (4.30) to (4.34)
    $\widehat{\mathbf{A}} \leftarrow \begin{pmatrix}\mathbf{A} & \alpha\mathbf{A}_\Pi\end{pmatrix}$, $\widehat{\mathbf{B}} \leftarrow \begin{pmatrix}\mathbf{B} & \alpha\mathbf{B}_\Pi\end{pmatrix}$
    $(\mathbf{A},\mathbf{B}) \leftarrow \mathfrak{T}^{k'+k_1}_{\mathrm{thin},k}(\widehat{\mathbf{A}},\widehat{\mathbf{B}})$ ▷ cf. Remark 2.2
  else
    if $(r,s) \in \mathcal{L}_{\mathcal{I}\times\mathcal{K}}$ **then**
      Determine $(\mathbf{A}_\Pi\mathbf{B}_\Pi) \in \mathcal{R}(t,s,k_2)$ with $\mathbf{X}\mathbf{Y} = \mathbf{A}_\Pi\mathbf{B}_\Pi^{\mathsf{T}}$ ▷ cf. (4.30) to (4.34)
      $\widehat{\mathbf{A}} \leftarrow \begin{pmatrix}\mathbf{A} & \alpha\mathbf{A}_\Pi\end{pmatrix}$, $\widehat{\mathbf{B}} \leftarrow \begin{pmatrix}\mathbf{B} & \alpha\mathbf{B}_\Pi\end{pmatrix}$
      $(\mathbf{A},\mathbf{B}) \leftarrow \mathfrak{T}^{k'+k_2}_{\mathrm{thin},k}(\widehat{\mathbf{A}},\widehat{\mathbf{B}})$ ▷ cf. Remark 2.2
    else
      for $t' \in \mathrm{sons}(t), s' \in \mathrm{sons}(s)$ **do**
        $\mathbf{A}' \leftarrow \mathbf{0}_{t'\times 1}$, $\mathbf{B}' \leftarrow \mathbf{0}_{s'\times 1}$
        for $r' \in \mathrm{sons}(r)$ **do**
          `addmul_rkmatrix_hmatrix`(α, t', r', s', $\mathbf{X}|_{t'\times r'}$, $\mathbf{Y}|_{r'\times s'}$, $\mathbf{A}'$, $\mathbf{B}'$, k)
        end for
        $\widehat{\mathbf{A}} \leftarrow \begin{pmatrix}\mathbf{A} & \mathbf{A}'|^{t\times k}\end{pmatrix}$, $\widehat{\mathbf{B}} \leftarrow \begin{pmatrix}\mathbf{B} & \mathbf{B}'|^{s\times k}\end{pmatrix}$
        $(\mathbf{A},\mathbf{B}) \leftarrow \mathfrak{T}^{k'+k}_{\mathrm{thin},k}(\widehat{\mathbf{A}},\widehat{\mathbf{B}})$ ▷ cf. Remark 2.2
      end for
    end if
  end if
end function

Algorithm 4.7 Update of a dense matrix by the product of $\mathcal{H}$-matrices

Input: Scaling factor $\alpha \in \mathbb{R}$, clusters $t \in \mathcal{T}_\mathcal{I}$, $r \in \mathcal{T}_\mathcal{K}$ and $s \in \mathcal{T}_\mathcal{J}$ with $(t,r) \in \mathcal{T}_{\mathcal{I}\times\mathcal{K}}$ and $(r,s) \in \mathcal{T}_{\mathcal{K}\times\mathcal{J}}$, $\mathcal{H}$-matrices $\mathbf{X} \in \mathcal{H}(\mathcal{T}_{(t,r)}, k_1)$ and $\mathbf{Y} \in \mathcal{H}(\mathcal{T}_{(r,s)}, k_2)$, matrix $\mathbf{Z} \in \mathbb{R}^{t\times s}$ that is overwritten by $\mathbf{Z} + \alpha\mathbf{XY}$.

```
function addmul_matrix_hmatrix(α, t, r, s, X, Y, Z)
    if (t,r) ∈ L_{I×K} then
        Determine (A,B) ∈ R(t,s,k1) with XY = AB^T        ▷ cf. (4.30) to (4.34)
        Z ← Z + αAB^T
    else
        if (r,s) ∈ L_{I×K} then
            Determine (A,B) ∈ R(t,s,k2) with XY = AB^T    ▷ cf. (4.30) to (4.34)
            Z ← Z + αAB^T
        else
            for t' ∈ sons(t), r' ∈ sons(r), s' ∈ sons(s) do
                addmul_matrix_hmatrix(α, t', r', s', X|_{t'×r'}, Y|_{r'×s'}, Z|_{t'×s'})
            end for
        end if
    end if
end function
```

Algorithm 4.8 Truncated addition of a rank-k' matrix to an $\mathcal{H}$-matrix

Input: Scaling factor $\alpha \in \mathbb{R}$, clusters $t \in \mathcal{T}_\mathcal{I}$ and $s \in \mathcal{T}_\mathcal{J}$ with $(t,s) \in \mathcal{T}_{\mathcal{I}\times\mathcal{J}}$, rank-$k'$ factors $(\mathbf{A},\mathbf{B}) \in \mathcal{R}(t,s,k')$, $\mathcal{H}$-matrix $\mathbf{Z} \in \mathcal{H}(\mathcal{T}_{(t,s)}, \ell)$, target rank $k \in \mathbb{N}$.

```
function add_hmatrix_rkmatrix(α, t, s, A, B, Z, k)
    if (t,s) ∈ L^+_{I×J} then
        Determine (A_(t,s), B_(t,s)) ∈ R(t,s,ℓ) with Z = A_(t,s) B_(t,s)^T   ▷ cf. Definition 4.11
        Â ← (A_(t,s)  αA),   B̂ ← (B_(t,s)  αB)
        (A_Z, B_Z) ← T^{k'+ℓ}_{thin,k}(Â, B̂)                                ▷ cf. Remark 2.2
        Z ← A|_Z B|_Z^T
    else
        if (t,s) ∈ L^-_{I×J} then
            Z ← Z + αAB^T
        else                                                                ▷ (t,s) ∉ L_{I×J}
            for t' ∈ sons(t), s' ∈ sons(s) do
                add_hmatrix_rkmatrix(α, t', s', A|_{t'×k'}, B|_{s'×k'}, Z|_{t'×s'}, k)
            end for
        end if
    end if
end function
```

Algorithm 4.9 Truncated $\mathcal{H}$-matrix update with pairwise SVD truncation

Input: Scaling factor $\alpha \in \mathbb{R}$, clusters $t \in \mathcal{T}_\mathcal{I}$, $r \in \mathcal{T}_\mathcal{K}$ and $s \in \mathcal{T}_\mathcal{J}$ with $(t,r) \in \mathcal{T}_{\mathcal{I}\times\mathcal{K}}$ and $(r,s) \in \mathcal{T}_{\mathcal{K}\times\mathcal{J}}$, $\mathcal{H}$-matrices $\mathbf{X} \in \mathcal{H}(\mathcal{T}_{(t,r)}, k_1)$, $\mathbf{Y} \in \mathcal{H}(\mathcal{T}_{(r,s)}, k_2)$, and $\mathbf{Z} \in \mathcal{H}(\mathcal{T}_{(t,s)}, k_3)$, desired maximal local rank $k \in \mathbb{N}$.

```
function addmul_hmatrix(α, t, r, s, X, Y, Z, k)
    if (t, r) ∈ L_{I×K} then
        Determine (A, B) ∈ R(t, s, k_1) with XY = AB^T        ▷ cf. (4.30) to (4.34)
        add_hmatrix_rkmatrix(α, t, s, A, B, Z, k)
    else
        if (r, s) ∈ L_{K×J} then
            Determine (A, B) ∈ R(t, s, k_2) with XY = AB^T    ▷ cf. (4.30) to (4.34)
            add_hmatrix_rkmatrix(α, t, s, A, B, Z, k)
        else                                                  ▷ (t, r) ∉ L_{I×J}, (r, s) ∉ L_{I×J}
            if (t, s) ∈ L^+_{I×J} then
                Determine (A, B) ∈ R(t, s, k_3) with Z = AB^T    ▷ cf. Definition 4.11
                addmul_rkmatrix_hmatrix(α, t, r, s, X, Y, A, B)
            else
                if (t, s) ∈ L^-_{I×J} then
                    addmul_matrix_hmatrix(α, t, r, s, X, Y, Z)
                else                                          ▷ (t, s) ∉ L_{I×J}
                    for t' ∈ sons(t), r' ∈ sons(r), s' ∈ sons(s) do
                        addmul_hmatrix(α, t', r', s', X|_{t'×r'}, Y|_{r'×s'}, Z|_{t'×s'})
                    end for
                end if
            end if
        end if
    end if
end function
```

- compute $\widehat{\mathbf{N}}_{21} = \mathbf{M}_{21}\widehat{\mathbf{N}}_{11}$ (4.40b)
- compute $\widehat{\mathbf{N}}_{12} = \widehat{\mathbf{N}}_{11}\mathbf{M}_{12}$ (4.40c)
- compute $\mathbf{S} = \mathbf{M}_{22} - \widehat{\mathbf{N}}_{21}\mathbf{M}_{12}$ (4.40d)
- determine $\mathbf{N}_{22} = \mathbf{S}^{-1}$ (4.40e)
- compute $\mathbf{N}_{21} = \mathbf{N}_{22}\widehat{\mathbf{N}}_{21}$ (4.40f)
- compute $\mathbf{N}_{12} = \widehat{\mathbf{N}}_{12}\mathbf{N}_{22}$ (4.40g)
- compute $\mathbf{N}_{11} = \widehat{\mathbf{N}}_{11} - \widehat{\mathbf{N}}_{12}\mathbf{N}_{21}$ (4.40h)

If we are only interested in the inverse, this computation can also be done in-place.

This 2×2 block inverse can be generalized for an arbitrary number of block rows and columns. For the $\mathcal{H}$-matrix inversion, we just replace all exact updates with truncated $\mathcal{H}$-matrix updates, as described in [21, 28]. But first, we have to fix an order for clusters on the same level (cf. Remarks 4.20 and 4.28), i.e., we assume that for each non-leaf cluster $t \in \mathcal{T}_\mathcal{I} \setminus \mathcal{L}_\mathcal{I}$ the sons are enumerated by $\operatorname{sons}(t) = \{t_1, \ldots, t_\nu\}$, where $\nu = |\operatorname{sons}(t)|$. The $\mathcal{H}$-matrix inversion is summarized in Algorithm 4.10.

Remark 4.45. *1. The regularity of* $\mathbf{M}$ *is not sufficient for the existence of a block-wise inverse of* $\mathbf{M}$*. Consider, for example,*

$$\mathbf{M} = \begin{pmatrix} \mathbf{M}_{11} & \mathbf{M}_{12} \\ \mathbf{M}_{21} & \mathbf{M}_{22} \end{pmatrix} = \left(\begin{array}{cc|cc} 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ \hline 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{array}\right).$$

Then $\mathbf{M}$ *is regular, but the block* $\mathbf{M}_{11}$ *is not. Hence, the matrix* $\mathbf{M}$ *has to satisfy stricter conditions. A sufficient condition is, for example, the regularity of all principal submatrices since in this case there is a unique (block) LU factorization (see, e.g., [18]).*

$$\mathbf{A} = \begin{pmatrix} \mathbf{M}_{11} & \mathbf{M}_{12} \\ \mathbf{M}_{21} & \mathbf{M}_{22} \end{pmatrix} = \begin{pmatrix} \mathbf{L}_{11} & \\ \mathbf{L}_{21} & \mathbf{L}_{22} \end{pmatrix} \begin{pmatrix} \mathbf{U}_{11} & \mathbf{U}_{12} \\ & \mathbf{U}_{22} \end{pmatrix}$$

Therefore, $\mathbf{M}_{11} = \mathbf{L}_{11}\mathbf{U}_{11}$ *and* $\mathbf{S} = \mathbf{L}_{22}\mathbf{U}_{22}$ *are regular.*

2. If the block inverse of an $\mathcal{H}$*-matrix exists, the regularity of the upper left block as well as the regularity of the Schur complement are equivalent to a full rank of these matrices. Therefore, the* $\mathcal{H}$*-matrix inverse can only be computed if all diagonal blocks are inadmissible. Although this is not required in Definition 4.10, all admissibility conditions from Section 4.3, and therefore all block cluster trees constructed with them, satisfy this condition.*

Lemma 4.46. *Let* $\mathbf{M} \in \mathcal{H}(\mathcal{T}_{\mathcal{I}\times\mathcal{I}}, k_1)$*. Then the computational complexity* W_{inv} *of the* $\mathcal{H}$*-matrix inversion is bounded by*

$$W_{\mathrm{inv}}(\mathcal{T}_{\mathcal{I}\times\mathcal{I}}, k_1) \leq W_{\mathrm{pw}}(\mathcal{T}_{\mathcal{I}\times\mathcal{I}}, \mathcal{T}_{\mathcal{I}\times\mathcal{I}}, k_1, k_1),$$

where W_{pw} *is the computational complexity of the truncated* $\mathcal{H}$*-matrix multiplication with pairwise truncations (cf. Theorem 4.44).*

Proof. See [21, Thm. 2.29]. □

Algorithm 4.10 $\mathcal{H}$-matrix inversion

Input: Cluster $t \in \mathcal{T}_\mathcal{I}$, $\mathcal{H}$-matrix $\mathbf{M} \in \mathcal{H}(\mathcal{T}_{(t,t)}, k_1)$ that is overwritten by intermediate results, auxiliary matrix $\widehat{\mathbf{N}} \in \mathcal{H}(\mathcal{T}_{(t,t)}, 0)$, target matrix $\mathbf{N} \in \mathcal{H}(\mathcal{T}_{(t,t)}, 0)$ that is overwritten by $\mathbf{M}^{-1}$, target local rank $k \in \mathbb{N}$

function `invert_hmatrix`(t, $\mathbf{M}$, $\widehat{\mathbf{N}}$, $\mathbf{N}$, k)
  if $(t,t) \in \mathcal{L}_{\mathcal{I}\times\mathcal{I}}$ **then** ▷ $(t,t) \in \mathcal{L}^-_{\mathcal{I}\times\mathcal{J}}$ due to Remark 4.45
    $\mathbf{N} \leftarrow \mathbf{M}^{-1}$
  else
    $\mathbf{N}|_{t\times t} \leftarrow \mathbf{0}$
    $\{t_1, \ldots, t_\nu\} \leftarrow \operatorname{sons}(t)$
    for $\ell = 1, \ldots, \nu$ **do**
      `invert_hmatrix`(t_ℓ, $\mathbf{M}|_{t_\ell\times t_\ell}$, $\widehat{\mathbf{N}}|_{t_\ell\times t_\ell}$, $\mathbf{N}|_{t_\ell\times t_\ell}$, k) ▷ See (4.40a)(4.40e)
      for $i = 1, \ldots, \ell - 1$ **do**
        $\widehat{\mathbf{N}}|_{t_\ell\times t_i} \leftarrow \mathbf{0}$
        `addmul_hmatrix`(1, t_ℓ, t_ℓ, t_i, $\mathbf{N}|_{t_\ell\times t_\ell}$, $\mathbf{N}|_{t_\ell\times t_i}$, $\widehat{\mathbf{N}}|_{t_\ell\times t_i}$, k) ▷ See (4.40f)
        $\mathbf{N}|_{t_\ell\times t_i} \leftarrow \widehat{\mathbf{N}}|_{t_\ell\times t_i}$
      end for
      for $i = \ell + 1, \ldots, \nu$ **do**
        $\widehat{\mathbf{N}}_{t_\ell\times t_i} \leftarrow \mathbf{0}$
        `addmul_hmatrix`(1, t_ℓ, t_ℓ, t_i, $\mathbf{N}|_{t_\ell\times t_\ell}$, $\mathbf{M}|_{t_\ell\times t_i}$, $\widehat{\mathbf{N}}|_{t_\ell\times t_i}$, k)
        $\mathbf{M}|_{t_\ell\times t_i} \leftarrow \widehat{\mathbf{N}}_{t_\ell\times t_i}$ ▷ See (4.40c)
      end for
      for $i = \ell + 1, \ldots \nu$ **do**
        for $j = 1, \ldots, \ell$ **do**
          $\widehat{\mathbf{N}}|_{t_i\times t_j} \leftarrow \mathbf{0}$
          `addmul_hmatrix`(-1, t_i, t_ℓ, t_j, $\mathbf{M}|_{t_i\times t_\ell}$, $\mathbf{N}|_{t_\ell\times t_j}$, $\widehat{\mathbf{N}}|_{t_i\times t_j}$, k)
          ▷ See (4.40b)
          $\mathbf{N}|_{t_i\times t_j} \leftarrow \widehat{\mathbf{N}}|_{t_i\times t_j}$
        end for
        for $j = \ell + 1, \ldots, \nu$ **do**
          `addmul_hmatrix`(-1, t_i, t_ℓ, t_j, $\mathbf{M}|_{t_i\times t_\ell}$, $\mathbf{M}|_{t_\ell\times t_j}$, $\mathbf{M}|_{t_i\times t_j}$, k)
          ▷ See (4.40d)
        end for
      end for
    end for
    for $\ell = \nu, \ldots, 1$ **do**
      for $i = \ell - 1, \ldots, 1$ **do**
        for $j = 1, \ldots, \nu$ **do**
          `addmul_hmatrix`(-1, t_i, t_ℓ, t_j, $\mathbf{M}|_{t_i\times t_\ell}$, $\mathbf{N}|_{t_\ell\times t_j}$, $\mathbf{N}|_{t_i\times t_j}$, k)
          ▷ See (4.40g),(4.40h)
        end for
      end for
    end for
  end if
end function

4.4.4 $\mathcal{H}$-matrix factorizations

In most applications, the inverse is not required directly but rather its action on vectors or matrices. A popular alternative to computing the inverse (or an approximation of the inverse) is to use a factorization, e.g. an LU factorization or Cholesky factorization. As for the inverse, we first consider the exact LU factorization for a 2×2 block matrix. For

$$\mathbf{M} = \begin{pmatrix} \mathbf{M}_{11} & \mathbf{M}_{12} \\ \mathbf{M}_{21} & \mathbf{M}_{22} \end{pmatrix},$$

the existence of the LU factorization $\mathbf{M} = \mathbf{LU}$ directly yields a representation in the block form

$$\begin{aligned} \mathbf{M} = \begin{pmatrix} \mathbf{M}_{11} & \mathbf{M}_{12} \\ \mathbf{M}_{21} & \mathbf{M}_{22} \end{pmatrix} &= \begin{pmatrix} \mathbf{L}_{11} & \\ \mathbf{L}_{21} & \mathbf{L}_{22} \end{pmatrix} \begin{pmatrix} \mathbf{U}_{11} & \mathbf{U}_{12} \\ & \mathbf{U}_{22} \end{pmatrix} \\ &= \begin{pmatrix} \mathbf{L}_{11}\mathbf{U}_{11} & \mathbf{L}_{11}\mathbf{U}_{12} \\ \mathbf{L}_{21}\mathbf{U}_{11} & \mathbf{L}_{22}\mathbf{U}_{22} + \mathbf{L}_{21}\mathbf{U}_{12} \end{pmatrix} = \mathbf{LU} \end{aligned}$$

Therefore the LU factorization of $\mathbf{M}$ can be computed block-wise through

1. Determine $\mathbf{L}_{11}$ and $\mathbf{U}_{11}$ with $\mathbf{M}_{11} = \mathbf{L}_{11}\mathbf{U}_{11}$.
2. Determine $\mathbf{U}_{12}$ by solving $\mathbf{L}_{11}\mathbf{U}_{12} = \mathbf{M}_{12}$.
3. Determine $\mathbf{L}_{21}$ by solving $\mathbf{U}_{11}^{\mathsf{T}}\mathbf{L}_{21}^{\mathsf{T}} = \mathbf{M}_{21}^{\mathsf{T}}$.
4. Determine $\mathbf{L}_{22}$ and $\mathbf{U}_{22}$ with $\mathbf{L}_{22}\mathbf{U}_{22} = \mathbf{M}_{22} - \mathbf{L}_{21}\mathbf{U}_{12}$.

Besides the computation of the LU factorization for sub-blocks, the computation of the LU factorization of $\mathbf{M}$ requires two forward solves, one with $\mathbf{L}_{11}$ and one with $\mathbf{U}_{11}^{\mathsf{T}}$, and an update of a subblock. The update can be computed with $\mathcal{H}$-matrix arithmetic as described in Section 4.4.2. Computing the LU factorization of the sub-blocks in $\mathcal{H}$-matrix arithmetics can be handled by a recursive call.

The remaining part is the block forward substitution (exact or with $\mathcal{H}$-matrix arithmetic). As before, we first consider the exact 2×2-block case

$$\begin{pmatrix} \mathbf{N}_1 \\ \mathbf{N}_2 \end{pmatrix} = \mathbf{N} = \mathbf{LX} = \begin{pmatrix} \mathbf{L}_{11} & \\ \mathbf{L}_{21} & \mathbf{L}_{22} \end{pmatrix} \begin{pmatrix} \mathbf{X}_1 \\ \mathbf{X}_2 \end{pmatrix} = \begin{pmatrix} \mathbf{L}_{11}\mathbf{X}_1 \\ \mathbf{L}_{21}\mathbf{X}_1 + \mathbf{L}_{22}\mathbf{X}_2 \end{pmatrix}.$$

Solving the block system $\mathbf{LX} = \mathbf{N}$ for $\mathbf{X}$ requires

1. solving the equation $\mathbf{L}_{11}\mathbf{X}_1 = \mathbf{N}_1$ for the respective sub-blocks,
2. computing $\widehat{\mathbf{N}}_2 := \mathbf{N}_2 - \mathbf{L}_{21}\mathbf{X}_1$,
3. solving the equation $\mathbf{L}_{22}\mathbf{X}_2 = \widehat{\mathbf{N}}_2$.

The $\mathcal{H}$-matrix variants of the operations discussed above require, as for the $\mathcal{H}$-matrix inversion, fixing an order for clusters on the same level, i.e. we assume that for each non-leaf cluster $t \in \mathcal{T}_{\mathcal{I}} \setminus \mathcal{L}_{\mathcal{I}}$, the sons are enumerated by $\operatorname{sons}(t) = \{t_1, \ldots, t_\nu\}$ where

Algorithm 4.11 $\mathcal{H}$-matrix forward substitution for $\mathcal{H}$-matrices right hand sides

Input: Clusters $t \in \mathcal{T}_\mathcal{I}$ and $s \in \mathcal{T}_\mathcal{J}$, block lower triangular $\mathcal{H}$-matrix $\mathbf{L} \in \mathcal{H}(\mathcal{T}_{(t,t)}, k_2)$, right hand side $\mathbf{X} \in \mathcal{H}(\mathcal{T}_{(t,s)}, k_2)$ (gets overwritten), desired maximal local rank $k \in \mathbb{N}$.

function `hmatrix_forward_substitution`(t, s, $\mathbf{L}$, $\mathbf{X}$, k)
  if $(t,s) \in \mathcal{L}^-_{\mathcal{I}\times\mathcal{J}}$ **then**
    for $i \in s$ **do**
      `hmatrix_vector_forward_substitution`(t, $\mathbf{L}$, $\mathbf{X}|_{t\times\{i\}}$)
    end for
  else if $(t,s) \in \mathcal{L}^+_{\mathcal{I}\times\mathcal{J}}$ **then**
    Determine $(\mathbf{A},\mathbf{B}) \in \mathcal{R}(t,s,k_1)$ with $\mathbf{AB}^\mathsf{T} = \mathbf{X}$ ▷ cf. Definition 4.11
    for $i = 1,\ldots,k_1$ **do**
      `hmatrix_vector_forward_substitution`(t, $\mathbf{L}$, $\mathbf{A}|_{t\times\{i\}}$)
    end for
    $\mathbf{X} \leftarrow \mathfrak{T}^{k_1}_{\mathrm{thin},k}(\mathbf{A},\mathbf{B})$ ▷ cf. Remark 2.2
  else
    $\{t_1,\ldots,t_n\} \leftarrow \mathrm{sons}(t)$
    for $i = 1,\ldots,n$ **do**
      for $s' \in \mathrm{sons}(s)$ **do**
        `hmatrix_forward_substitution`(t_i, s', $\mathbf{L}|_{t_i\times t_i}$, $\mathbf{X}|_{t_i\times s'}$, k)
        for $j = i+1,\ldots,n$ **do**
          `addmul_hmatrix`(-1, t_j, t_i, s', $\mathbf{L}|_{t_j\times t_i}$, $\mathbf{X}|_{t_i\times s'}$, $\mathbf{X}|_{t_j\times s'}$, k)
        end for
      end for
    end for
  end if
end function

Algorithm 4.12 $\mathcal{H}$-matrix forward substitution for vector right hand sides

Input: Cluster $t \in \mathcal{T}_\mathcal{I}$, block lower triangular $\mathcal{H}$-matrix $\mathbf{L} \in \mathcal{H}(\mathcal{T}_{(t,t)}, k)$, right hand side matrix $\mathbf{x} \in \mathbb{R}^t$.

function `hmatrix_vector_forward_substitution`(t, $\mathbf{L}$, $\mathbf{x}$)
  if $t \in \mathcal{L}_\mathcal{I}$ **then** ▷ $t \times t \in \mathcal{L}^-_{\mathcal{I}\times\mathcal{I}}$
    $\mathbf{x} \leftarrow \mathbf{L}^{-1}\mathbf{x}$
  else
    $\{t_1,\ldots,t_n\} \leftarrow \mathrm{sons}(t)$
    for $i = 1,\ldots,n$ **do**
      `hmatrix_vector_forward_substitution`(t_i, $\mathbf{L}|_{t_i\times t_i}$, $\mathbf{x}|_{t_i}$)
      for $j = i,\ldots,n$ **do**
        `addmul_hmatrix_vector`($-1_{\prime\prime}$, (t_j,t_i), $\mathbf{L}|_{t_j\times t_i}$, $\mathbf{x}|_{t_i}$, $\mathbf{x}|_{t_j}$)
      end for
    end for
  end if
end function

Algorithm 4.13 $\mathcal{H}$-matrix LU factorization

Input: Cluster $t \in \mathcal{T}_\mathcal{I}$, $\mathcal{H}$-matrix $\mathbf{M} \in \mathcal{H}(\mathcal{T}_{(t,t)}, k')$, target maximal local rank k.

```
function lu_decomposition_hmatrix(t, M)
    if t ∈ L_I then                                          ▷ t × t ∈ L⁻_{I×I}
        Compute the LU factorization of the dense matrix M
    else
        {t_1, ..., t_n} ← sons(t)
        for ℓ = 1, ..., n do
            lu_decomposition_hmatrix(t_ℓ, M|_{t_ℓ×t_ℓ})
            for i = ℓ + 1, ..., n do
                hmatrix_forward_substitution(t_ℓ, t_i, M|_{t_ℓ×t_ℓ}, M|_{t_ℓ×t_i})
                hmatrix_forward_substitution(t_ℓ, t_i, M|^T_{t_ℓ×t_ℓ}, M|^T_{t_i×t_ℓ})
                for j = ℓ + 1, ..., n do
                    addmul_hmatrix(t_i, t_ℓ, t_j, M|_{t_i×t_ℓ}, M|_{t_ℓ×t_j}, M|_{t_i×t_j})
                end for
            end for
        end for
    end if
end function
```

$\nu = |\operatorname{sons}(t)|$. Replacing all exact arithmetic operations with corresponding $\mathcal{H}$-matrix operations then yields Algorithm 4.11 for the forward substitution with $\mathcal{H}$-matrix right hand sides, Algorithm 4.12 for the forward substitution with dense matrix (or vector) right hand sides, and Algorithm 4.13 for the $\mathcal{H}$-LU factorization.

Without further analysis of the block structure, we can estimate the computational complexity with the complexity of the $\mathcal{H}$-matrix product.

Lemma 4.47. *Let $\mathbf{M} \in \mathcal{H}(\mathcal{T}_{\mathcal{I}\times\mathcal{I}}, k_1)$ such that there exists an LU-decomposition of $\mathbf{M}$. Then the computational complexity W_{LU} of the $\mathcal{H}$-LU factorization is bounded by*

$$W_{\mathrm{LU}}(\mathcal{T}_{\mathcal{I}\times\mathcal{I}}, k_1) \leq W_{\mathrm{pw}}(\mathcal{T}_{\mathcal{I}\times\mathcal{I}}, Cluster[\mathcal{I}\times\mathcal{I}], k_1, k_1),$$

where W_{pw} is the computational complexity of the truncated $\mathcal{H}$-matrix multiplication with pairwise truncation (cf. Theorem 4.44).

Proof. See [28, Lemma 7.39]. □

Chapter 5

$\mathcal{H}$-LU preconditioning of saddle point problems

We are interested in the application of $\mathcal{H}$-matrix factorizations to block preconditioners for saddle point problems of the form

$$\mathcal{M}\begin{pmatrix}\mathbf{x}\\ \mathbf{y}\end{pmatrix} = \begin{pmatrix}\mathbf{F} & \mathbf{B}^\mathsf{T}\\ \mathbf{B} & \mathbf{0}\end{pmatrix}\begin{pmatrix}\mathbf{x}\\ \mathbf{y}\end{pmatrix} = \begin{pmatrix}\mathbf{f}\\ \mathbf{g}\end{pmatrix}$$

as described in Section 2.3.2. The main difficulty of these preconditioning techniques is to find easily invertible approximations $\widetilde{\mathbf{F}}$ to $\mathbf{F}$ and $\widetilde{\mathbf{S}}$ to the Schur complement $\mathbf{S} = \mathbf{B}\mathbf{F}^{-1}\mathbf{B}^\mathsf{T}$. In this chapter, we will explore different approaches to the construction of $\mathcal{H}$-matrix approximations of $\mathbf{F}$ and $\mathbf{S}$ and their $\mathcal{H}$-LU factorizations. We will only consider saddle point problems originating from the discretization of Oseen problems with the FEM with the stable pair of finite element spaces described in Section 3.3, i.e., $\mathcal{M}$ is of the form (cf. (3.19))

$$\mathcal{M} = \begin{pmatrix}\mathbf{F} & \mathbf{B}^\mathsf{T}\\ \mathbf{B} & \mathbf{0}\end{pmatrix} = \begin{pmatrix}\check{\mathbf{F}} & & & (\mathbf{B}^1)^\mathsf{T}\\ & \ddots & & \vdots\\ & & \check{\mathbf{F}} & (\mathbf{B}^d)^\mathsf{T}\\ \mathbf{B}^1 & \cdots & \mathbf{B}^d & \mathbf{0}\end{pmatrix}. \tag{5.1}$$

Remark 5.1. *Since we consider systems from the discretization of Oseen problems with the stable pair of finite element spaces described in Section 3.3, we can assume the following to be given:*

- *The continuous Oseen equations (cf.* (3.3)*) on a polyhedral domain $\Omega \subseteq \mathbb{R}^d$ ($d \in \{2,3\}$) with boundary $\Gamma := \partial\Omega$. The boundary is divided into a part Γ_D with Dirichlet boundary conditions and Γ_N with Neumann boundary conditions.*
- *A triangulation $\mathcal{T}^h$ of Ω (cf. Definition 3.6), as well as a triangulation $\mathcal{T}^{h/2}$ constructed from $\mathcal{T}^h$ by connecting the edge midpoints (cf.* (3.23) *and Remark 3.7).*
- *The spaces (cf.* (3.18) *and* (3.23)*)*

$$Q^h = \mathrm{P}^1(\mathcal{T}^h), \quad V^h = \mathrm{P}^1(\mathcal{T}^{h/2}), \quad \text{and } \mathbf{V}^h = \prod_{i=1}^{d} V^h$$

- *Bases* $(\Psi_1, \dots, \Psi_M)$ *of* Q^h *and* $(\varphi_1, \dots, \varphi_n, \varphi_{n+1}, \dots, \varphi_{n+n_D})$ *of* V^h *such that*

$$(\varphi_1, \dots, \varphi_n) = V_0^h := V^h \cap \mathsf{H}_0^1(\Omega; \Gamma_D)$$

  (cf. (3.21) *and* (3.22)*).*

- *The two index sets* $\mathcal{I} := \{1, \dots, n\}$ *and* $\mathcal{J} \in \{1, \dots, M\}$ *for which* $\check{\mathbf{F}} \in \mathbb{R}^{\mathcal{I}\times\mathcal{I}}$, $\mathbf{B}^k \in \mathbb{R}^{\mathcal{J}\times\mathcal{I}}$ *for* $k = 1, \dots, d$, *and* $\mathbf{S} \in \mathbb{R}^{\mathcal{J}\times\mathcal{J}}$.

For $i \in \mathcal{I}$ *and* $j \in \mathcal{J}$, *we define* $X_i := \operatorname{supp}(\varphi_i)$ *and* $Y_j := \operatorname{supp}(\Psi_j)$, *respectively. Furthermore, we use*

$$X_s := \bigcup_{i\in s} X_i \text{ and } Y_t := \bigcup_{j\in t} Y_i$$

for $s \subseteq \mathcal{I}$ *and* $t \subseteq \mathcal{J}$, *respectively, as in Section 4.2. In addition, the functions* φ_i *are hat functions for some vertex* $\boldsymbol{\xi}_i \in \mathcal{V}(\mathcal{T}^{h/2})$ *for all* $i \in \mathcal{I}$ *(cf.* (3.21)*). Similarly,* Ψ_j *is a hat function for some vertex* $\boldsymbol{\chi}_j \in \mathcal{V}(\mathcal{T}^h)$ *for all* $j \in \mathcal{J}$.

Remark 5.2. *Let* $i \in \mathcal{I}$. *Since* $\mathcal{T}^{h/2}$ *is refined from* $\mathcal{T}^h$ *by connecting the edge midpoints, one of the following two cases apply to* $\boldsymbol{\xi}_i$*:*

1. *If* $\boldsymbol{\xi}_i \in \mathcal{V}(\mathcal{T}^h) \subseteq \mathcal{V}(\mathcal{T}^{h/2})$ *then there is a* $j \in \mathcal{J}$ *with* $\boldsymbol{\xi}_i = \boldsymbol{\chi}_j$. *Let* $K \in \mathcal{T}^{h/2}$ *with* $\boldsymbol{\xi}_i \in K$. *Then, there is a unique* $L \in \mathcal{T}^h$ *with* $\overline{K} \subseteq \overline{L}$. *Since* $\mathcal{T}^h$ *is a triangulation (cf. Definition 3.6), this implies that* $\boldsymbol{\xi}_i = \boldsymbol{\chi}_j$ *is a vertex of* L, *i.e.,* $\boldsymbol{\xi}_i \in L$. *This implies*

$$X_i = \operatorname{supp}(\varphi_i) = \bigcup_{\substack{K\in\mathcal{T}^{h/2}\\ \boldsymbol{\xi}_i\in K}} \overline{K} \subseteq \bigcup_{\substack{L\in\mathcal{T}^h\\ \boldsymbol{\chi}_j\in L}} \overline{L} = \operatorname{supp}(\Psi_j) = Y_j.$$

2. *If* $\xi_i \in \mathcal{V}(\mathcal{T}^{h/2}) \setminus \mathcal{V}(\mathcal{T}^h)$, *then it is a midpoint of an edge* $E \in \mathcal{E}(\mathcal{T}^h)$. *Hence, there are* $k, \ell \in \mathcal{J}$ *with* $E = \{\boldsymbol{\chi}_k, \boldsymbol{\chi}_\ell\}$ *and* $\boldsymbol{\xi}_i = \frac{1}{2}(\boldsymbol{\chi}_k + \boldsymbol{\chi}_\ell)$. *Now, let again* $K \in \mathcal{T}^{h/2}$ *with* $\boldsymbol{\xi}_i \in K$ *and* $L \in \mathcal{T}^h$ *the unique simplex with* $\overline{K} \subseteq \overline{L}$. *Then* E *has to be an edge of* L, *since* $\boldsymbol{\xi}_i \in \overline{L}$. *Therefore, we obtain*

$$X_i = \bigcup_{\substack{K\in\mathcal{T}^{h/2}\\ \boldsymbol{\xi}_i\in K}} \overline{K} \subseteq \bigcup_{\substack{L\in\mathcal{T}^h\\ \boldsymbol{\chi}_k,\boldsymbol{\chi}_\ell\in L}} \overline{L} = Y_k \cap Y_\ell.$$

To construct $\widetilde{\mathbf{F}}$ and $\widetilde{\mathbf{S}}$ with $\mathcal{H}$-LU factorizations, we have to

1. Represent $\mathbf{F}$ as $\mathcal{H}$-matrix $\mathbf{F}_\mathcal{H}$, i.e., its sub-blocks $\check{\mathbf{F}}$ as $\check{\mathbf{F}}_\mathcal{H}$.
2. Compute an (approximate) $\mathcal{H}$-LU factorization $\widetilde{\mathbf{F}} = \mathbf{L}_\mathbf{F}\mathbf{U}_\mathbf{F} \approx \mathbf{F}_\mathcal{H}$.
3. Represent $\mathbf{B}$ as $\mathcal{H}$-matrix $\mathbf{B}_\mathcal{H}$, i.e., $\mathbf{B}^1, \dots, \mathbf{B}^d$ as $\mathbf{B}^1_\mathcal{H}, \dots, \mathbf{B}^d_\mathcal{H}$.
4. Compute $\mathbf{S}_\mathcal{H} := (\mathbf{B}_\mathcal{H}\mathbf{U}_\mathbf{F}^{-1}) \odot (\mathbf{L}_\mathbf{F}^{-1}\mathbf{B}_\mathcal{H}^\mathsf{T})$ with $\mathcal{H}$-matrix arithmetic.
5. Compute an (approximate) $\mathcal{H}$-LU factorization $\widetilde{\mathbf{S}} = \mathbf{L}_\mathbf{S}\mathbf{U}_\mathbf{S} \approx \mathbf{S}_\mathcal{H}$.

Due to the block-diagonal structure of $\mathbf{F}$, its $\mathcal{H}$-LU factorization will be of the form

$$\widetilde{\mathbf{F}} = \begin{pmatrix} \check{\mathbf{L}}_\mathbf{F}\check{\mathbf{U}}_\mathbf{F} & & \\ & \ddots & \\ & & \check{\mathbf{L}}_\mathbf{F}\check{\mathbf{U}}_\mathbf{F} \end{pmatrix}.$$

Remark 5.3. *The discussion above implies that the computations of $\widetilde{\mathbf{F}}$ and $\widetilde{\mathbf{S}}$ can be expressed as follows:*

$$\begin{aligned}
\mathbf{S} = -\mathbf{B}_\mathcal{H}\mathbf{F}_\mathcal{H}^{-1}\mathbf{B}_\mathcal{H} &= -\sum_{k=1}^{d} \mathbf{B}_\mathcal{H}^k \check{\mathbf{F}}_\mathcal{H}^{-1}(\mathbf{B}_\mathcal{H}^k)^\mathsf{T} \\
&\approx -\sum_{k=1}^{d} \mathbf{B}_\mathcal{H}^k(\check{\mathbf{L}}_\mathbf{F}\check{\mathbf{U}}_\mathbf{F})^{-1}(\mathbf{B}_\mathcal{H}^k)^\mathsf{T} && \text{① Compute an } \mathcal{H}\text{-LU factorization } \check{\mathbf{L}}_\mathbf{F}\check{\mathbf{U}}_\mathbf{F} \approx \check{\mathbf{F}} \\
&\approx -\sum_{k=1}^{d} (\mathbf{B}_\mathcal{H}^k\check{\mathbf{U}}_\mathbf{F}^{-1})(\check{\mathbf{L}}_\mathbf{F}^{-1}(\mathbf{B}_\mathcal{H}^k)^\mathsf{T}) \\
& && \text{② Compute } \mathbf{V}^k = \mathbf{B}_\mathcal{H}^k\check{\mathbf{U}}_\mathbf{F}^{-1}\text{, ③ compute } \mathbf{W}^k = \check{\mathbf{L}}_\mathbf{F}^{-1}(\mathbf{B}_\mathcal{H}^k)^\mathsf{T} \\
&\approx -\bigoplus_{k=1}^{d} (\mathbf{B}_\mathcal{H}^k\check{\mathbf{U}}_\mathbf{F}^{-1}) \odot (\check{\mathbf{L}}_\mathbf{F}^{-1}(\mathbf{B}_\mathcal{H}^k)^\mathsf{T}) && \text{④ Compute } \mathbf{S}_\mathcal{H} = -\bigoplus_{k=1}^{d} \mathbf{V}^k \odot \mathbf{W}^k \\
&\approx \mathbf{L}_\mathbf{S}\mathbf{U}_\mathbf{S},, && \text{⑤ Compute an } \mathcal{H}\text{-LU factorization } \mathbf{L}_\mathbf{S}\mathbf{U}_\mathbf{S} \approx \mathbf{S}_\mathcal{H}
\end{aligned}$$

where $\oplus$ and $\odot$ are the truncated $\mathcal{H}$-matrix addition and product from Section 4.4.2.

The construction illustrated in Remark 5.3 necessitates three different block cluster trees for the occurring $\mathcal{H}$-matrices:

- A block cluster tree $\mathcal{T}_{\mathcal{I}\times\mathcal{I}}$ for $\check{\mathbf{F}}_\mathcal{H}$ and its $\mathcal{H}$-LU factorization.
- A block cluster tree $\mathcal{T}_{\mathcal{J}\times\mathcal{J}}$ for $\mathbf{S}_\mathcal{H}$ and its $\mathcal{H}$-LU factorization.
- A block cluster tree $\mathcal{T}_{\mathcal{J}\times\mathcal{I}}$ for $\mathbf{B}_\mathcal{H}^1, \ldots, \mathbf{B}_\mathcal{H}^d$ as well as for $\mathbf{V}^1, \ldots, \mathbf{V}^d$ and $\mathbf{W}^1, \ldots, \mathbf{W}^d$ from Remark 5.3.

To perform the necessary arithmetic operations, the block structures of the matrices have to be compatible, i.e., they have to be based on two cluster trees. A cluster tree $\mathcal{T}_\mathcal{I}$ for the index set $\mathcal{I}$ from the velocity discretization and a cluster tree $\mathcal{T}_\mathcal{J}$ for the index set $\mathcal{J}$ from the pressure discretization.

In the remainder of this chapter, we will first describe two different approaches for the construction of these two cluster trees. The first approach, described in Section 5.1, was introduced in [31]. The cluster tree $\mathcal{T}_\mathcal{I}$ is constructed with the domain decomposition clustering described in Section 4.2.2 to obtain a favorable block structure for the $\mathcal{H}$-LU factorization of $\check{\mathbf{F}}$. The cluster tree $\mathcal{T}_\mathcal{J}$, on the other hand, is constructed with the (geometric) bisection clustering explained in Section 4.2.1 to obtain a good representation of the (typically dense) Schur complement $\mathbf{S}$.

In Section 5.2, we will describe a slightly different approach first introduced in [19]. As before, the cluster tree $\mathcal{T}_\mathcal{J}$ is constructed with the geometric bisection clustering to

obtain a representation of the Schur complement $\mathbf{S}$. However, the cluster tree $\mathcal{T}_\mathcal{I}$ is then constructed for a representation of the matrices $\mathbf{B}^k$ offering a favorable block structure for the time-consuming multiplication (step ④). Then, we will discuss a modification of the second approach in Section 5.3. This modification aims to tackle some disadvantages of the clustering described in Section 5.2.

Finally, we will examine the results of numerical experiments in Section 5.4.

5.1 Uncoupled clustering

The first approach from [31] is to construct the cluster tree $\mathcal{T}_\mathcal{I}$ for representing the matrix $\check{\mathbf{F}}$ such that its block structure is favorable for computing an $\mathcal{H}$-LU factorization. The cluster tree $\mathcal{T}_\mathcal{J}$, on the other hand, is built for the representation of the Schur complement $\mathbf{S}$. Since the Schur complement is typically dense, we use (geometric) bisection clustering (cf. Algorithm 4.1) to build $\mathcal{T}_\mathcal{J}$. One cluster strategy allowing for an efficient computation of the $\mathcal{H}$-LU factorization of $\mathcal{H}$-matrices representing sparse matrices is the domain decomposition clustering from Section 4.2.2. It reduces the fill-in of the $\mathcal{H}$-LU factorizations, i.e., reduces the number and ranks of non-zero blocks emerging from the factorization. Additionally, it implies a reordering of the index set $\mathcal{I}$ resulting in a block-arrowhead structure that is favorable for the computation of an LU factorization. The two cluster strategies can be performed independently, which is why we call this approach *uncoupled clustering*.

Now, let $\mathcal{T}_\mathcal{I}$ and $\mathcal{T}_\mathcal{J}$ be cluster trees for the index sets $\mathcal{I}$ and $\mathcal{J}$ generated with the strategies discussed above, i.e., $\mathcal{T}_\mathcal{I}$ is built with the domain decomposition clustering and $\mathcal{T}_\mathcal{J}$ is built with the geometric bisection clustering. Although both cluster trees are built independently for (favorable) representations of $\check{\mathbf{F}}$ and $\mathbf{S}$, we also have to use them both for representing the matrices $\mathbf{B}^k_\mathcal{H}$ (and therefore also the matrices $\mathbf{V}^k$ and $\mathbf{W}^k$ occurring in the computation of the approximation $\mathbf{S}_\mathcal{H}$ of the Schur complement). These matrices describe a coupling of the discrete pressure space Q^h and the velocity space V^h (cf. Remark 5.1), and therefore between the index sets $\mathcal{J}$ and $\mathcal{I}$.

As we will see later in Section 5.4, the set-up of the preconditioner, i.e., the computation of the $\mathcal{H}$-LU factorizations $\widetilde{\mathbf{F}}$ and $\widetilde{\mathbf{S}}$ is dominated by the $\mathcal{H}$-matrix multiplication in step ④.

We will now consider an example for $d = 2$ as illustrated in Figure 5.1. We recall from (3.17) that the entries of the matrices $\mathbf{B}^k$, $k = 1, \dots, d$ are of the form

$$b^k_{ji} = \int_\Omega \Psi_j \frac{\partial \varphi_i}{\partial x_k}\, d\mathbf{x} \tag{5.2}$$

for all $i \in \mathcal{I}$ and $j \in \mathcal{J}$. Therefore, we may have $b^k_{ij} \neq 0$ only if the intersection of the supports $\operatorname{supp}(\Psi_j)$ and $\operatorname{supp}(\varphi_i)$ has a positive Lebesgue measure

$$\lambda(\operatorname{supp}(\Psi_j) \cap \operatorname{supp}(\varphi_i)) > 0.$$

On the first level, the cluster tree $\mathcal{T}_\mathcal{J}$ divides the root $r_\mathcal{J} = \operatorname{root}(\mathcal{T}_\mathcal{J})$ into two clusters $\{r_1, r_2\} = \operatorname{sons}(r_\mathcal{J})$ (see Figure 5.1(a)) while the cluster tree $\mathcal{T}_\mathcal{I}$ divides the root $r_\mathcal{I} = \operatorname{root}(\mathcal{T}_\mathcal{I})$ into three clusters $\{s_1, s_2, s_3\} = \operatorname{sons}(r_\mathcal{I})$ (see Figure 5.1(b)). The cluster trees also imply a reordering of $\mathcal{I}$ and $\mathcal{J}$, respectively, as noted in Remarks 4.19 and 4.28. The indices in r_1 are ordered before the indices in r_2. The indices in s_1 are ordered before the

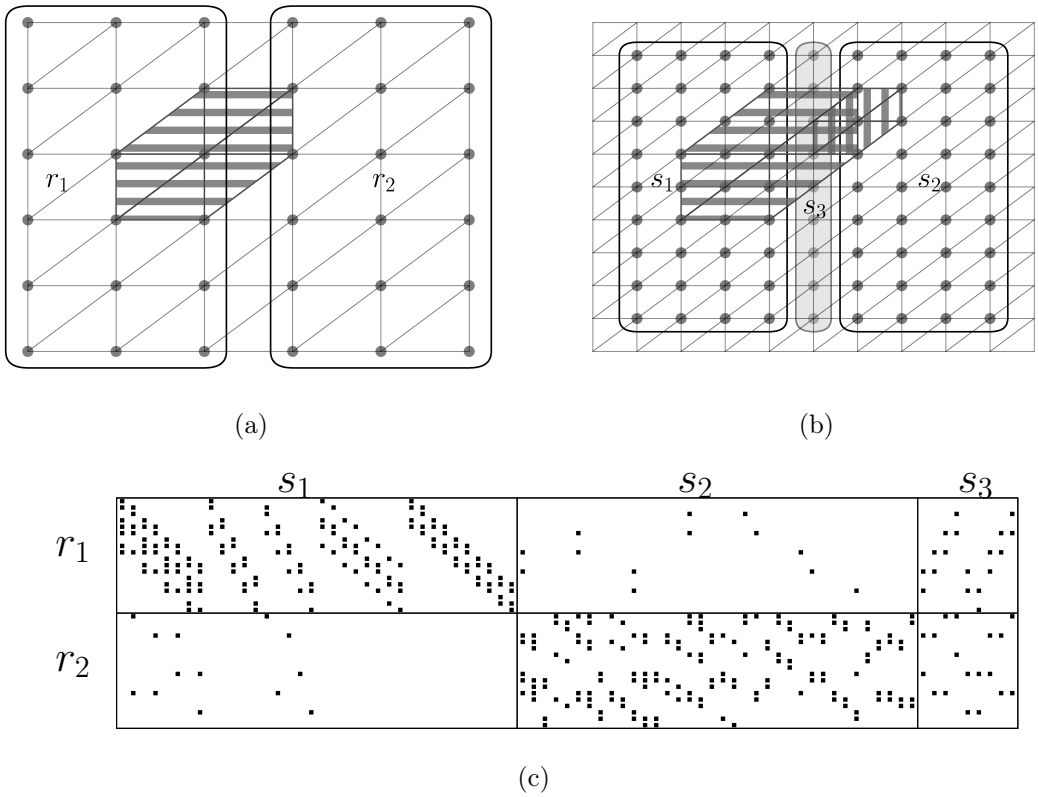

Figure 5.1: 2D example of the uncoupled clustering and the effect of its induced reordering of the index sets $\mathcal{I}$ and $\mathcal{J}$. Figure 5.1(a) in the top left shows the first division of $\mathcal{J}$ into r_1 and r_2 while Figure 5.1(b) shows the division of $\mathcal{I}$ into the domain clusters s_1 and s_2 and the interface cluster s_3 (marked light gray). These figures also show the support of a basis function Ψ_j with $j \in r_1$ (marked dark gray, hatched horizontally) and the support of a basis function φ_i with $i \in s_2$ (dark gray, hatched vertically). The bottom figure shows the sparsity pattern of the matrices $\mathbf{B}^k$, $k = 1, \ldots, d$ with reordered rows/columns and the block structure induced by the uncoupled clustering.

indices in s_2, and the indices in s_3 are ordered after the indices of both s_1 and s_2. The effect of the reordering is visualized in Figure 5.1(c).

As we can see in Figure 5.1(c), there are only few non-zero entries in $\mathbf{B}^1|_{r_1 \times s_2}$ and $\mathbf{B}^1|_{r_2 \times s_1}$, respectively. These non-zero entries can be explained by overlapping supports $\operatorname{supp}(\Psi_j)$ and $\operatorname{supp}(\varphi_i)$ for $i \in s_2$ and $j \in r_1$ or $i \in s_1$ and $j \in r_2$ (see Figure 5.1(b)). This yields

$$\operatorname{adm}_\eta(r_1, s_2) = \text{false}$$

and therefore the block $r_1 \times s_2 \in \mathcal{T}_{\mathcal{J} \times \mathcal{I}}$ is subdivided despite the small number of non-zero entries and therefore small rank.

5.2 Coupled clustering

In the previous section, we have discovered that the uncoupled clustering can result in a subdivision of blocks in the matrices $\mathbf{B}^k$ which only have a few non-zero entries and therefore have a small rank. In this section we describe how we can adapt the clustering of $\mathcal{I}$ as well as the admissibility condition for $\mathcal{T}_{\mathcal{J} \times \mathcal{I}}$ to avoid this behavior.

We have seen in Figure 5.1 that these few non-zero entries in the off-diagonal blocks of $\mathbf{B}^k$ $(k = 1, \dots, d)$ are the result of the overlap of the supports of few basis functions $\varphi_i \in V^h$ and $\Psi_j \in Q^h$. We start with a cluster tree $\mathcal{T}_{\mathcal{J}}$ already generated with (geometric) bisection. Then, we adapt the clustering of $\mathcal{I}$ so that indices of $\mathcal{I}$ corresponding to basis functions generating these non-zero entries are moved to the interface. This is done by coupling the division of $\mathcal{I}$ to the cluster tree $\mathcal{T}_{\mathcal{J}}$. We start with the index set $\mathcal{I}$ and decompose it with respect to the root $r_{\mathcal{J}} := \operatorname{root}(\mathcal{T}_{\mathcal{J}})$ of $\mathcal{T}_{\mathcal{J}}$. Later, we will use this by associating the root $r_{\mathcal{I}}$ with $r_{\mathcal{J}}$. The decomposition is done as follows:

- If $r_{\mathcal{J}}$ is already a leaf, $\mathcal{I}$ is not subdivided further.
- If $r_{\mathcal{J}}$ is not a leaf of $\mathcal{T}_{\mathcal{J}}$, it has exactly two sons r_1 and r_2 (cf. Remark 4.21). Then we divide $\mathcal{I}$ into the three distinct subsets

$$\begin{aligned} s_1 &:= \{i \in \mathcal{I} : X_i \cap Y_{r_2} = \emptyset\}, \\ s_2 &:= \{i \in \mathcal{I} : X_i \cap Y_{r_1} = \emptyset\}, \\ s_3 &:= \mathcal{I} \setminus (s_1 \cup s_2). \end{aligned} \tag{5.3}$$

  The subsets s_1 and s_2 are then (geometrically) separated from r_2 and r_1, respectively, since by construction $\operatorname{supp}(\varphi_i) \cap \operatorname{supp}(\Psi_j) = \emptyset$ holds for all $i \in s_1$ and $j \in r_2$ or $i \in s_2$ and $j \in r_1$ (cf. Figure 5.2). They can be subdivided with respect to r_1 and r_2, respectively, by the same approach. The subset s_3, on the other hand, is handled similarly to the interface clusters from the domain decomposition clustering, i.e., s_3 is subdivided with geometric bisection independently of $\mathcal{T}_{\mathcal{J}}$. The decomposition is also delayed every d-th step analogously to Remark 4.29 and as further explained in Remark 5.4.

Remark 5.4. *For the generation of $\mathcal{T}_{\mathcal{J}}$ with geometric bisection, the bounding box $B_{\mathcal{J}}$ was split along the longest axis. Let $k \in \{1, \dots, d\}$ be such that this is the k-th axis, i.e.,*

$$k = \arg \max_{i \in \{1, \dots, d\}} \{b_i - a_i\},$$

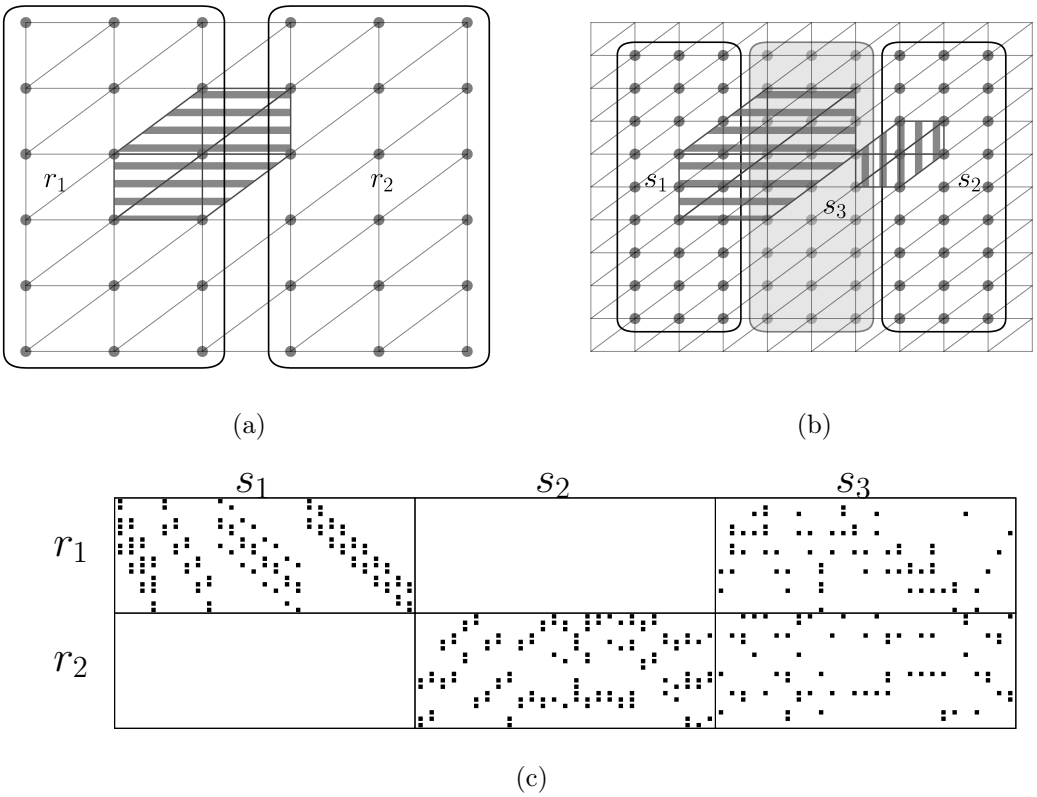

Figure 5.2: 2D example of the coupled clustering and the effect of the induced reordering of the index sets $\mathcal{I}$ and $\mathcal{J}$. Figure 5.2(a) on the top left shows the first division of $\mathcal{J}$ into r_1 and r_2 as in Figure 5.1(a). Figure 5.2(b) on the top right shows the division of $\mathcal{I}$ into the two domain clusters s_1 and s_2 as well as the interface s_3 (marked light gray). These figures also show the support of a basis function Ψ_j for $j \in r_1$ (dark gray, hatched horizontally) and the support of a basis function φ_i for $i \in s_2$ (dark gray, hatched vertically). The bottom figure shows the sparsity pattern of the matrices $\mathbf{B}^k$, $k = 1, \ldots, d$, with reordered rows and columns.

$B_{\mathcal{J}} = \prod_{i=1}^{d}[a_i, b_i]$. Then Remark 5.2 yields $B_{\mathcal{I}} \subseteq B_{\mathcal{J}}$. The clusters r_1 and r_2 were generated by splitting $B_{\mathcal{J}}$ along the k-th axis into

$$B_1 = [a_1, b_1] \times \cdots \times [a_{k-1}, b_{k-1}] \times [a_k, m_k] \times [a_{k+1}, b_{k+1}] \times \cdots \times [a_d, b_d]$$

and

$$B_2 = [a_1, b_1] \times \cdots \times [a_{k-1}, b_{k-1}] \times [m_k, b_k] \times [a_{k+1}, b_{k+1}] \times \cdots \times [a_d, b_d],$$

where $m_k = \frac{1}{2}(a_k + b_k)$.

Now, let $i \in s_3$. Then there are $j_1 \in r_1$ and $j_2 \in r_2$ such that $X_i \cap Y_{j_1} \neq \emptyset$ and $X_i \cap Y_{j_2} \neq \emptyset$. Note that we can assume that j_1 and j_2 are chosen such that the vertices $\boldsymbol{\chi}_{j_1}$ and $\boldsymbol{\chi}_{j_2}$ are both vertices of a simplex $L \in \mathcal{T}^h$. Then the vertex $\boldsymbol{\xi}_i$ is contained in the affine span of this simplex, i.e., $\boldsymbol{\xi}_i \in \overline{L}$. Note that since $\boldsymbol{\chi}_{j_1} \in r_1$ and $\boldsymbol{\chi}_{j_2} \in r_2$ the k-th components of $\boldsymbol{\chi}_{j_1}$ and $\boldsymbol{\chi}_{j_2}$ satisfy $\chi_{j_1,k} \leq m_k < \chi_{j_2,k}$ (cf. Equation (4.5)). If the k-th component of $\boldsymbol{\xi}_i$ satisfies $\xi_{i,k} \leq m_k$, we obtain

$$|\xi_{i,k} - m_k| = m_k - \xi_{i,k} \leq \chi_{j_2,k} - \chi_{i,k} \leq \|\boldsymbol{\chi}_{j_2} - \boldsymbol{\xi}_i\| \leq \operatorname{diam}_2(\overline{L}) \leq h.$$

If, on the other hand, $\xi_{i,k} > m_k$ holds, we obtain analogously

$$|\xi_{i,k} - m_k| \leq \|\chi_{j_1} - \xi_i\| \leq \operatorname{diam}_2(\overline{L}) \leq h.$$

This implies that

$$B_{s_3} \subseteq [a_1, b_1] \times \cdots \times [a_{k-1}, b_{k-1}] \times [m_k - h, m_k + h] \times [a_{k+1}, b_{k+1}] \times \cdots \times [a_d, b_d],$$

i.e., B_{s_3} has at most width $2h$ along the k-th axis.

Since s_3 is handled with geometric bisection, we can therefore assume that the bounding boxes are split along the other axes until all the axes have at most width $2h$. This justifies the delay of the subdivision of interface clusters every d-th step. The diameter of the interface clusters' bounding boxes and the diameters of the domain clusters' bounding boxes are calibrated, as for the domain decomposition clustering (cf. Remark 4.29).

Remark 5.5. *The subsets $s_1, s_2 \subset \mathcal{I}$ are not only separated (geometrically) from r_2 and r_1, respectively, but also from each other. By construction, we have*

$$X_{s_1} \cap Y_{r_2} = \emptyset \quad \text{and} \quad X_{s_2} \cap Y_{r_1} = \emptyset.$$

Now, let $i \in s_1$. Then one of the following two cases applies to the corresponding vertex $\boldsymbol{\xi}_i \in \mathcal{V}(\mathcal{T}^{h/2})$ (see Remark 5.1):

1. *$\boldsymbol{\xi}_i \in \mathcal{V}(\mathcal{T}^h) \subseteq \mathcal{V}(\mathcal{T}^{h/2})$. Then there is a $j \in \mathcal{J}$ with $\boldsymbol{\xi}_i = \boldsymbol{\chi}_j$ and*

$$X_i = \operatorname{supp}(\varphi_i) \subseteq \operatorname{supp}(\Psi_j) = Y_j$$

holds (cf. Remark 5.2). Due to the definition of s_1 in (5.3) this yields $j \in r_1$ and therefore $X_i \subseteq Y_{r_1}$.

2. $\boldsymbol{\xi}_i \notin \mathcal{V}(\mathcal{T}^h)$. *Then, there are* $j_1, j_2 \in \mathcal{J}$ *with* $\boldsymbol{\xi}_i = \frac{1}{2}(\boldsymbol{\chi}_{j_1} + \boldsymbol{\chi}_{j_2})$ *and*

$$X_i = \operatorname{supp}(\varphi_i) \subseteq \operatorname{supp}(\Psi_{j_1}) \cap \operatorname{supp}(\Psi_{j_2})$$

holds (cf. Remark 5.2). Due to the definition of s_1 *in* (5.3) *this yields* $j_1, j_2 \in r_1$ *and therefore* $X_i \subseteq Y_{r_1}$.

Thus, we conclude $X_{s_1} \subseteq Y_{r_1}$. *Analogously, we obtain* $X_{s_2} \subseteq Y_{r_2}$. *Hence, we have*

$$X_{s_1} \cap X_{s_2} = \emptyset$$

due to the definition of s_1 *and* s_2 *in* (5.3). *Therefore, we can also apply the domain decomposition admissibility conditions from Definition 4.34 for constructing the block cluster tree* $\mathcal{T}_{\mathcal{I}\times\mathcal{I}}$ *from* $\mathcal{T}_{\mathcal{I}}$.

The construction described above is summarized in Algorithm 5.1. The function `InterfaceClustering` called in line 10 of Algorithm 5.1 was defined in Algorithm 4.2.

Algorithm 5.1 Coupled clustering

Input: Subset $s \subseteq \mathcal{I}$, associated cluster $r \in \mathcal{T}_{\mathcal{J}}$, leaf size bound n_{leaf} for interface clusters
Output: Cluster tree with root t and $\widehat{t} = s$ ▷ see Definition 4.6

```
1: function CoupledClustering(s, r, n_leaf)
2:     Create new cluster t with t̂ = s and sons(t) = ∅
3:     if sons(r) ≠ ∅ then
4:         {r_1, r_2} ← sons(r)

5:         s_1 ← {i ∈ s : X_i ∩ Y_{r_2} = ∅}
6:         s_2 ← {i ∈ s : X_i ∩ Y_{r_1} = ∅}
7:         s_3 ← s \ (s_1 ∪ s_2)

8:         t_1 ← CoupledClustering(s_1, r_1, n_leaf)
9:         t_2 ← CoupledClustering(s_2, r_2, n_leaf)
10:        t_3 ← InterfaceClustering(s_3, n_leaf, 1)        ▷ see Algorithm 4.2
11:        sons(t) ← {t_1, t_2, t_3}
12:    end if
13:    return t
14: end function
```

In the following, we will use the same terminology as for the domain decomposition clustering (cf. Definition 4.27). Additionally, Algorithm 5.1 generates clusters $s \in \mathcal{T}_{\mathcal{I}}$ with respect to associated clusters $t_s \in \mathcal{T}_{\mathcal{J}}$. We formalize this association by introducing a corresponding mapping.

Definition 5.6. *Let* $\mathcal{T}_{\mathcal{I}}$ *be a cluster tree generated with Algorithm 5.1.*

1. *Like for the domain decomposition clustering, we call clusters* $s \in \mathcal{T}_{\mathcal{I}}$ *generated by* `InterfaceClustering` *from Algorithm 4.2* interface clusters. *Clusters generated by Algorithm 5.1, on the other hand, are called* domain clusters. *As in Definition 4.27, we denote the set of domain clusters of* $\mathcal{T}_{\mathcal{I}}$ *by* $\mathcal{C}_{\text{dom}}(\mathcal{T}_{\mathcal{I}})$ *or* $\mathcal{C}_{\text{dom}}$.

2. *As noted before, each domain cluster $s \in \mathcal{C}_{\text{dom}}$ is generated by Algorithm 5.1 with respect to an associated cluster $t_s \in \mathcal{T}_{\mathcal{J}}$. We define the mapping*

$$^{\text{a}} : \mathcal{C}_{\text{dom}} \to \mathcal{T}_{\mathcal{J}};\ s \mapsto t_s$$

formalizing this association.

Remark 5.7. *The coupled clustering, as before the domain decomposition clustering (see Remark 4.28), induces a reordering of the index set $\mathcal{I}$. We order s_1 before s_2 and both of them before s_3.*

In Figure 5.2 we can observe the effect of the reordering mentioned in Remark 5.7 on the matrices $\mathbf{B}^k$, $k = 1, \dots, d$. There, we can observe that the coupled clustering results in a three times larger interface s_3 compared to the domain decomposition cluster (see, e.g., Figure 5.1). But, on the other hand, the off-diagonal blocks $\mathbf{B}^k|_{r_2 \times s_1}$ and $\mathbf{B}^k|_{r_1 \times s_2}$ are now zero blocks. These zero blocks can be identified easily with the help of the mapping $^{\text{a}} : \mathcal{C}_{\text{dom}} \to \mathcal{T}_{\mathcal{J}}$ from Definition 5.6.

This results in the following admissibility condition.

Definition 5.8 (Coupled admissibility condition)**.** *Let $\mathcal{T}_{\mathcal{J}}$ be a cluster tree generated with geometric bisection. Let $\mathcal{T}_{\mathcal{I}}$ be a cluster tree generated with the coupled clustering from Algorithm 5.1 and with respect to $\mathcal{T}_{\mathcal{J}}$. Then we call the admissibility condition*

$$\text{adm}^{\text{c}}_{\eta} : \mathcal{T}_{\mathcal{J}} \times \mathcal{T}_{\mathcal{I}} \to \{\text{true}, \text{false}\}$$

with

$$\text{adm}^{\text{c}}_{\eta}(r, s) = \begin{cases} \text{true} & \text{if } s \in \mathcal{C}_{\text{dom}} \text{ and } r \neq s^{\text{a}}, \\ \text{adm}_{\eta}(r, s) & \text{else} \end{cases}$$

(strong) coupled admissibility condition. *As for the domain decomposition admissibility condition (cf. Definition 4.34), we can replace* adm_{η} *by the weaker admissibility conditions from Definitions 4.32 and 4.33 to obtain a* weak *or a* sparse coupled admissibility condition $\text{adm}^{\text{c}}_{\text{weak}}$ *or* $\text{adm}^{\text{c}}_{\text{sp},n_{\max}}$*, respectively.*

Two important concepts for the analyzation of the computational and the memory complexity were the sparsity constant C_{sp} from Definition 4.14 and the leaf size bound from Definition 4.13. The first is an upper bound for the number of leafs of a block cluster tree, the second an upper bound for the cardinality of leaf clusters of a cluster tree. If the block cluster tree was generated with an admissibility condition (cf. Definition 4.10), the sparsity constant is mainly determined by this admissibility condition. The leaf size bound, on the other hand, is typically defined when the cluster tree is generated (see, e.g., Algorithm 4.1). But this is not the case for cluster trees generated with the coupled clustering, as it is discussed in the following remark.

Remark 5.9. *Let $\mathcal{T}_{\mathcal{I}}$ be a cluster tree generated with the coupled clustering from Algorithm 5.1. Since we stopped the division of domain clusters $s \in \mathcal{C}_{\text{dom}}$ when the associated cluster $s^{\text{a}} \in \mathcal{L}_{\mathcal{J}}$ is a leaf cluster, we can not control the size of the leaves of $\mathcal{T}_{\mathcal{I}}$ directly. But, due to the finite element setting, we can estimate the size of the leaves as follows.*

Let $s \in \mathcal{L}_{\mathcal{I}}$. Then also $s^{\text{a}} \in \mathcal{L}_{\mathcal{J}}$. Let n_{leaf} be a leaf size bound for $\mathcal{T}_{\mathcal{J}}$. Then $|s^{\text{a}}| \leq n_{\text{leaf}}$. Now, let $i \in s$. Then, as in Remark 5.5, one of the following two cases applies to $\boldsymbol{\xi}_i \in \mathcal{V}(\mathcal{T}^{h/2})$.

1. $\boldsymbol{\xi}_i \in \mathcal{V}(\mathcal{T}^h)$. *Then there is a* $j \in s^{\mathrm{a}}$ *with* $\boldsymbol{\xi}_i = \boldsymbol{\chi}_j$.

2. $\boldsymbol{\xi}_i \notin \mathcal{V}(\mathcal{T}^h)$. *Then there are* $j_1, j_2 \in s^{\mathrm{a}}$ *with* $\boldsymbol{\xi}_i = \frac{1}{2}(\boldsymbol{\chi}_{j_1} + \boldsymbol{\chi}_{j_2})$, *i.e.,* $\boldsymbol{\xi}_i$ *is the midpoint of the edge between* $\boldsymbol{\chi}_{j_1}$ *and* $\boldsymbol{\chi}_{j_2}$. *Note that this edge has to be part of a simplex* $L \in \mathcal{T}^h$ *with vertices from* $\mathcal{V}(s^{\mathrm{a}}) := \{\chi_j : j \in s^{\mathrm{a}}\}$, *since* s *is separated from other clusters* $t \in \mathcal{T}_{\mathcal{J}}$ *on the same level.*

Therefore, we have

$$\begin{aligned} |s| &\leq |s^{\mathrm{a}}| + |\{E \in \mathcal{E}(\mathcal{T}^h) : E \subseteq \mathcal{V}(s^{\mathrm{a}})\}| \\ &\leq n_{\mathrm{leaf}} + |\{E \in \mathcal{E}(\mathcal{T}^h) : E \subseteq \mathcal{V}(s^{\mathrm{a}})\}|. \end{aligned}$$

Hence, the coupled clustering can result in significantly larger leaves compared to the (uncoupled) domain decomposition clustering. To tackle this, one can continue with domain decomposition clustering until the leaves are small enough. This would not affect the block cluster tree $\mathcal{T}_{\mathcal{J}\times\mathcal{I}}$ *but only* $\mathcal{T}_{\mathcal{I}\times\mathcal{I}}$ *(cf. Definition 4.10).*

Remark 5.10. *For the domain decomposition clustering, we argued that demanding* $X_{s_1} \cap X_{s_2} = \emptyset$ *may be too strict. Instead, it may suffice to demand* $X_{s_1} \cap X_{s_2}$ *to be a Lebesgue null set. This is, e.g., the case in the context of the FEM. The entries* f_{ij} *of the matrix* $\check{\mathbf{F}}$ *are zero not only if* $X_i \cap X_j = \emptyset$ *but already if* $X_i \cap X_j = \operatorname{supp}(\varphi_i) \cap \operatorname{supp}(\varphi_j)$ *is a Lebesgue null set (cf. Equations* (3.11) *and* (3.17)*). Therefore, we can also relax the conditions in the case of the coupled clustering by demanding the intersections to be Lebesgue null sets instead of being empty in all definitions above.*

In the (memory and computational) complexity estimates for $\mathcal{H}$-matrices, the depth of the row and column cluster trees has an important role (see, e.g., Lemma 4.15). Therefore, we are now interested in an estimate of the depth of the cluster trees generated with the coupled clustering. As for the estimate in the case of the bisection clustering in Lemma 4.24, we need additional assumptions.

Since the sets $Y_j, j \in \mathcal{J}$, are supports of FEM basis functions they are locally separated (cf. Definition 4.22 and Remark 4.23), i.e., there are $C_{\mathrm{sep}} > 0$ and $n_{\min} \in \mathbb{N}$ with

$$\max_{j\in\mathcal{J}} \left|\left\{k \in \mathcal{J} : \operatorname{dist}_2(Y_j, Y_k) \leq C_{\mathrm{sep}}^{-1} \operatorname{diam}_2(Y_j)\right\}\right| \leq n_{\min}. \tag{5.4}$$

Both C_{sep} and $n_{\min}$ depend on the triangulation $\mathcal{T}^h$ (cf. Remark 4.23). Since $\mathcal{T}^{h/2}$ is refined from $\mathcal{T}^h$ by connecting the edge midpoints, the sets X_i are also locally separated with

$$\max_{i\in\mathcal{I}} \left|\left\{\ell \in \mathcal{I} : \operatorname{dist}_2(X_i, X_\ell) \leq C_{\mathrm{sep}}^{-1} \operatorname{diam}_2(X_i)\right\}\right| \leq n_{\min}. \tag{5.5}$$

This follows since the properties of $\mathcal{T}^h$ determining the constant C_{sep} and $n_{\min}$ in (5.4) are preserved by the refinement. Details about this can be found, e.g., in [28, §6.4.3.2].

Additionally, we assume that $n_{\min}$ also satisfies

$$n_{\min} > 4^d \left(\frac{h}{h_{\min}}\right)^d, \tag{5.6}$$

where h is the mesh width of the triangulation $\mathcal{T}^h$ (cf. Definition 3.6) and

$$h_{\min} := \min_{\substack{\mathbf{p},\mathbf{q}\in\mathcal{V}(\mathcal{T}^h)\\ \mathbf{p}\neq\mathbf{q}}} \|\mathbf{p}-\mathbf{q}\|_2.$$

This allows the following estimate.

Theorem 5.11. *Let $\mathcal{T}_\mathcal{J}$ be a cluster tree generated with geometric bisection and leaf size bound $n_{\text{leaf}} \geq n_{\min}$ (cf. Definition 4.13). Let $\mathcal{T}_\mathcal{I}$ be a cluster tree generated with the coupled clustering, i.e., with Algorithm 5.1 and with respect to $\mathcal{T}_\mathcal{J}$. Let $\underline{h} := \min_{i\in\mathcal{I}} \operatorname{diam}_2(X_i)$ and $\delta := \operatorname{diam}_\infty(B_\mathcal{J})$. Then the depth of the cluster tree $\mathcal{T}_\mathcal{I}$ can be estimated by*

$$\operatorname{depth}(\mathcal{T}_\mathcal{I}) \leq d\log_2\left(4\sqrt{d}C_{\text{sep}}\delta\underline{h}^{-1}\right) = \mathcal{O}\left(\log_2(\underline{h}^{-d})\right).$$

Proof. We will prove the estimate by defining an upper bound for the level of the clusters in $\mathcal{T}_\mathcal{I}$. Therefore, let $s \in \mathcal{T}_\mathcal{I}$. Then s is either a domain cluster or an interface cluster (cf. Definition 5.6).

In the first case, i.e., if s is a domain cluster, Lemma 4.24 yields an estimate for the associated cluster s^{a}. Since $\mathcal{T}^{h/2}$ was refined from $\mathcal{T}^h$ by connecting the edge midpoints, $\min_{j\in\mathcal{J}} Y_j = 2\underline{\text{h}}$ holds. Therefore, we obtain

$$\begin{aligned}\operatorname{level}(s) = \operatorname{level}(s^{\text{a}}) &\overset{\text{Lemma 4.24}}{\leq} d\log_2\left(2\sqrt{d}C_{\text{sep}}\delta(2\underline{h})^{-1}\right)\\ &= d\log_2\left(\sqrt{d}C_{\text{sep}}\delta\underline{\text{h}}^{-1}\right)\\ &< d\log_2\left(4\sqrt{d}C_{\text{sep}}\underline{\text{h}}^{-1}\right).\end{aligned} \tag{5.7}$$

Now, assume that s is an interface cluster. Let r be the nearest predecessor of s to be a domain cluster. Then there is a path

$$p = (r_0, r_1, \ldots, r_\ell)$$

of length $\ell \in \mathbb{N}$ from $r_0 = r$ to $r_\ell = s$. Since r is the nearest predecessor of s to be a domain cluster, the clusters $r_1, \ldots, r_{k-1}$ are all interface clusters. This yields (cf. Definition 4.5)

$$\operatorname{level}(s) = \operatorname{level}(r) + \ell. \tag{5.8}$$

So, it remains to estimate $\operatorname{level}(r)$ and ℓ. The first can be estimated with Remark 4.25:

$$\operatorname{level}(r) = \operatorname{level}(r^{\text{a}}) < d\log_2\left(2\delta\frac{1}{\operatorname{diam}_\infty(B_{r^{\text{a}}})}\right). \tag{5.9}$$

To estimate ℓ, we will use the same techniques used for the proof of Lemma 4.24, i.e., the estimate for the depth of trees generated by geometric bisection. First note, that since $\mathcal{T}^{h/2}$ was refined from $\mathcal{T}^h$ by connecting the edge midpoints, we have

$$\min_{\substack{\mathbf{p},\mathbf{q}\in\mathcal{V}(\mathcal{T}^h)\\ \mathbf{p}\neq\mathbf{q}}} \|\mathbf{p}-\mathbf{q}\|_2 = \frac{1}{2}\min_{K\in\mathcal{T}^h}\min_{\substack{\mathbf{p},\mathbf{q}\in K\\ \mathbf{p}\neq\mathbf{q}}} \|\mathbf{p}-\mathbf{q}\| = \frac{1}{2}h_{\min}.$$

This implies that there is at least one axis of B_s with a larger width than $2h$, since otherwise (5.6) would yield

$$|s| \leq \left(\frac{\operatorname{diam}_\infty(B_s)}{\frac{1}{2} h_{\min}} \right)^d \leq \left(\frac{4h}{h_{\min}} \right)^d \overset{(5.6)}{<} n_{\min} \leq |s|.$$

Due to the delayed subdivision every d-th step, the diameter of the bounding boxes is at least halved after at most d steps, i.e.,

$$\operatorname{diam}_\infty(B_{r_{k+d}}) \leq \frac{1}{2} \operatorname{diam}_\infty(B_k)$$

holds for $k = 1, \ldots, \ell - d$. Hence, we have

$$\begin{aligned} \operatorname{diam}_\infty(B_s) = \operatorname{diam}_\infty(B_{r_\ell}) &\leq 2^{-\left\lfloor \frac{\ell}{d} \right\rfloor} \operatorname{diam}_\infty(B_{r_1}) \\ &\leq 2^{-\left\lfloor \frac{\ell}{d} \right\rfloor} \operatorname{diam}_\infty(B_r) \\ &\leq 2^{-\left\lfloor \frac{\ell}{d} \right\rfloor} \operatorname{diam}_\infty(B_{r^{\mathrm{a}}}). \end{aligned} \tag{5.10}$$

Now, we assume that s is not a leaf, i.e., $|s| > n_{\text{leaf}} \geq n_{\min}$. Since the sets X_i are locally separated, there have to be $i, j \in s$ with

$$\operatorname{dist}_2(X_i, X_j) > C_{\text{sep}} \operatorname{diam}_2(X_i). \tag{5.11}$$

This directly yields

$$\operatorname{diam}_2(B_s) \geq \operatorname{dist}_2(\boldsymbol{\xi}_i, \boldsymbol{\xi}_j) \geq \operatorname{dist}_2(X_i, X_j) > C_{\text{sep}}^{-1} \operatorname{diam}_2(X_i) \geq C_{\text{sep}}^{-1} \underline{\mathrm{h}}. \tag{5.12}$$

Together with (5.10), we therefore obtain

$$\begin{aligned} 2^{\left\lfloor \frac{\ell}{d} \right\rfloor} &\overset{(5.10)}{\leq} \frac{\operatorname{diam}_\infty(B_{r^{\mathrm{a}}})}{\operatorname{diam}_\infty s} \\ &\leq \frac{\operatorname{diam}_\infty B_{r^{\mathrm{a}}}}{\operatorname{diam}_2(B_s)} \\ &\overset{(5.12)}{<} C_{\text{sep}} \sqrt{d} \operatorname{diam}_\infty(B_{r^{\mathrm{a}}}) \underline{\mathrm{h}}^{-1}. \end{aligned} \tag{5.13}$$

By taking the logarithm on both sides, we finally obtain

$$\ell = d \frac{\ell}{d} \leq d \left(\left\lfloor \frac{\ell}{d} \right\rfloor + 1 \right) \overset{(5.13)}{<} d \log_2 \left(\sqrt{d} C_{\text{sep}} \operatorname{diam}_\infty(B_{r^{\mathrm{a}}}) \underline{\mathrm{h}}^{-1} \right) + d. \tag{5.14}$$

Combining (5.14) with (5.8) and (5.9) then finally yields

$$\begin{aligned} \operatorname{level}(s) &\overset{(5.8)}{=} \operatorname{level}(r) + \ell \overset{(5.9)}{<} d \log_2 \left(2\delta \frac{1}{\operatorname{diam}_\infty B_{r^{\mathrm{a}}}} \right) + \ell \\ &\overset{(5.14)}{<} d \log_2 \left(2\delta \frac{1}{\operatorname{diam}_\infty B_{r^{\mathrm{a}}}} \right) + d \log_2 \left(\sqrt{d} C_{\text{sep}} \operatorname{diam}_\infty(B_{r^{\mathrm{a}}}) \underline{\mathrm{h}}^{-1} \right) + d \\ &= d \log_2 \left(2\sqrt{d} C_{\text{sep}} \delta \underline{\mathrm{h}}^{-1} \right) + d \\ &= d \log_2 \left(4\sqrt{d} C_{\text{sep}} \delta \underline{\mathrm{h}}^{-1} \right). \end{aligned}$$

□

5.3 Coupled clustering with interface decomposition

In the previous section, we introduced the coupled clustering where we created the cluster tree $\mathcal{T}_\mathcal{I}$ coupled with the cluster tree $\mathcal{T}_\mathcal{J}$ generated with geometric bisection. This resulted in large zero blocks in the matrices $\mathbf{B}^k$ $(k = 1, \ldots, d)$ but also increased the size of the interface clusters (see, e.g., Figure 5.2) compared to the (uncoupled) domain decomposition clustering. In Figure 5.3, we can observe that there are just a few nodes from the interface which are connected to one of the domain clusters.

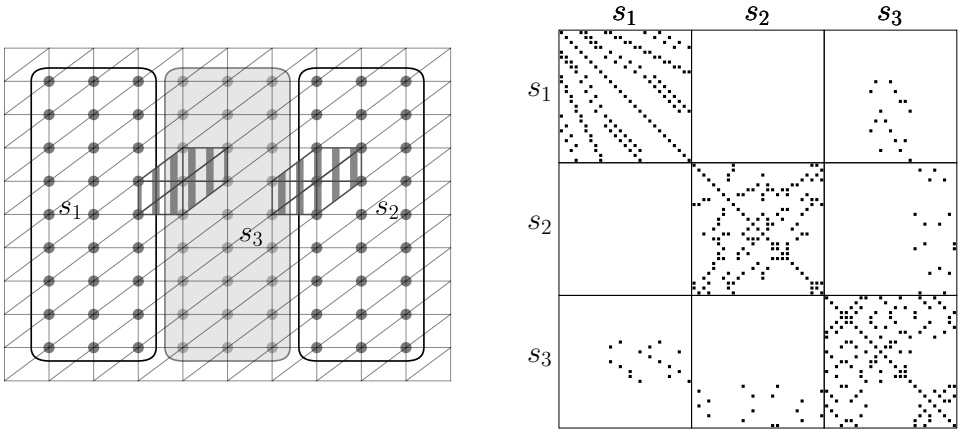

Figure 5.3: Effect of the reordering induced by the coupled clustering to the structure of $\check{\mathbf{F}}$ for a 2D example. The left figure shows the geometric decomposition of $\mathcal{I}$ into the two domain clusters s_1, s_2 and the interface cluster s_3 (marked light gray). Additionally, it shows the support of a basis function φ_i with $i \in s_2$ and the support of a basis function φ_j with $j \in s_3$ (both dark gray, hatched vertically). The right figure shows the effect of the reordering on the matrix $\check{\mathbf{F}}$. The interface s_3 was ordered internally so that all nodes in the middle are ordered first, then all nodes neighboring s_1 and then all nodes neighboring s_2.

The sparsity plot in Figure 5.3 was obtained by reordering the rows and columns such that s_1 is ordered before s_2 and both of them before s_3. Additionally, the indices in s_3 are ordered such that those neighboring neither s_1 nor s_2 are ordered first, while the others are ordered last beginning with those neighboring s_1. We can also observe in Figure 5.3 that the non-zero entries in $\check{\mathbf{F}}|_{s_3 \times s_1}$ and $\check{\mathbf{F}}|_{s_3 \times s_2}$ are in about a third of the rows of the corresponding blocks. Based on the observations we made, we will now describe a cluster strategy similar to the coupled clustering but with an additional decomposition of the interfaces which we call *coupled clustering with interface decomposition*. We start with the following statement to formalize the terms "connected" and "neighboring" used above.

Lemma 5.12. *Let $\mathcal{T}_\mathcal{I}$ be generated with the coupled clustering from Algorithm 5.1 coupled with $\mathcal{T}_\mathcal{J}$. Let $s \in \mathcal{T}_\mathcal{I}$ be a domain cluster with three sons. Let s_1, s_2, s_3 be these sons such that s_1, s_2 are the domain cluster sons of s and s_3 is the interface cluster son. Then*

$$X_i \cap X_{s_1} = \emptyset \text{ or } X_i \cap X_{s_2} = \emptyset$$

holds for all $i \in s_3$.

Proof. First, let r_1 and r_2 be the sons of the cluster $s^{\mathrm{a}} \in \mathcal{TJ}$ associated with s (cf. Definition 5.6). Now, let $i \in s_3$. Since $\mathcal{T}^{h/2}$ was refined from $\mathcal{T}^h$ by connecting the edge midpoints, we have to distinguish the following two cases:

1. If $\boldsymbol{\xi}_i \in \mathcal{V}(\mathcal{T}^h)$, then there is a $j \in s^{\mathrm{a}}$ with $\boldsymbol{\chi}_j = \boldsymbol{\xi}_i$ (see Remark 5.5). Let r_1, r_2 be the sons of s^{a}. Then either $j \in r_1$ or $j \in r_2$. We first assume that $j \in r_1$. Note that then (cf. Remark 5.2)

$$Y_{r_1} \overset{j \in r_1}{\supseteq} Y_j \overset{\textit{Remark 5.2}}{\supseteq} X_i \tag{5.15}$$

   holds. Since (cf. Algorithm 5.1)

$$\widehat{s}_2 = \{\ell \in s : X_\ell \cap Y_{r_1} = \emptyset\},$$

   we obtain

$$X_i \cap X_{s_2} \overset{(5.15)}{\subseteq} Y_{r_1} \cap X_{s_2} = \emptyset.$$

   For $j \in r_2$, we obtain analogously $X_i \cap X_{s_1} = \emptyset$.

2. If $\boldsymbol{\xi}_i \notin \mathcal{V}(\mathcal{T}^h)$ then $\boldsymbol{\xi}_i$ is a midpoint of an edge of $\mathcal{T}^h$, i.e., there are $j_1, j_2 \in s^{\mathrm{a}}$ with $\boldsymbol{\xi}_i = \frac{1}{2}(\boldsymbol{\chi}_{j_1} + \boldsymbol{\chi}_{j_2})$. Hence, Remark 5.2 yields

$$Y_{j_1} \supseteq X_i \text{ and } Y_{j_2} \supseteq X_i. \tag{5.16}$$

   We recall from Algorithm 5.1 that

$$\begin{aligned} s_1 &= \{\ell \in s : X_\ell \cap Y_{r_2} = \emptyset\}, \\ s_2 &= \{\ell \in s : X_\ell \cap Y_{r_1} = \emptyset\}. \end{aligned} \tag{5.17}$$

   (a) If $j_1, j_2 \in r_1$, then we obtain

$$X_i \cap X_{s_2} \overset{(5.16)}{\subseteq} Y_{r_1} \cap X_{s_2} \overset{(5.17)}{=} \emptyset.$$

   (b) If $j_1, j_2 \in r_2$, then $X_i \cap X_{s_1} = \emptyset$ follows analogously.

   (c) If $j_1 \in r_1$ and $j_2 \in r_2$ or vice versa, then we obtain

$$X_i \cap X_{s_1} \overset{(5.16)}{\subseteq} Y_{r_2} \cap X_{s_1} \overset{(5.17)}{=} \emptyset$$

   and

$$X_i \cap X_{s_2} \overset{(5.16)}{\subseteq} Y_{r_1} \cap X_{s_2} \overset{(5.17)}{=} \emptyset.$$

□

The generation of the cluster tree $\mathcal{T}_{\mathcal{I}}$ is again started with the decomposition of $\mathcal{I}$ with respect to the root $r_{\mathcal{J}} := \mathrm{root}(\mathcal{T}_{\mathcal{J}})$ of $\mathcal{T}_{\mathcal{J}}$ into now up to five subsets:

- If $r_{\mathcal{J}}$ is a leaf, $\mathcal{I}$ is neither subdivided.

- If $r_{\mathcal{J}}$ is not a leaf, it has exactly two sons r_1 and r_2 (cf. Remark 4.21). Then, we divide $\mathcal{I}$ into the five (distinct) subsets (where s_1 and s_2 are as before in (5.3)) as illustrated in Figure 5.4:

$$\begin{aligned} s_1 &:= \{i \in \mathcal{I} : X_i \cap Y_{r_2} = \emptyset\}, \\ s_2 &:= \{i \in \mathcal{I} : X_i \cap Y_{r_1} = \emptyset\}, \\ s_3 &:= \{i \in \mathcal{I} \setminus (s_1 \cup s_2) : X_i \cap X_{s_1} = \emptyset \text{ and } X_i \cap X_{s_2} = \emptyset\}, \\ s_4 &:= \{i \in \mathcal{I} \setminus (s_1 \cup s_2) : X_i \cap X_{s_1} \neq \emptyset \text{ and } X_i \cap X_{s_2} = \emptyset\}, \\ s_5 &:= \{i \in \mathcal{I} \setminus (s_1 \cup s_2) : X_i \cap X_{s_1} = \emptyset \text{ and} X_i \cap X_{s_2} \neq \emptyset\}. \end{aligned} \tag{5.18}$$

  Note that Lemma 5.12 yields $\bigcup_{i=1}^{5} s_i = \mathcal{I}$. The subsets s_1 and s_2 are still (geometrically) separated from r_2 and r_1, respectively. They can be subdivided with respect to r_1 and r_2, respectively, by the same approach. The set s_3 is (geometrically) separated from both, s_1 and s_2, while s_4 and s_5 are (geometrically) separated from either s_1 or s_2, respectively. We handle all of them as before the single interface clusters: s_3, s_4, and s_5 are subdivided with geometric bisection. The subdivision is delayed every d-th step for a similar reason this was done for the coupled clustering and the domain decomposition clustering (see Remarks 4.29 and 5.4).

Algorithm 5.2 Coupled clustering with interface decomposition

Input: Subset $s \subseteq \mathcal{I}$, associated cluster $r \in \mathcal{T}_{\mathcal{J}}$, leaf size bound n_{leaf} for interface clusters
Output: Cluster tree with root t and $\widehat{t} = s$

```
1:  function CoupledIDClustering(s, r, n_leaf)
2:      Create new cluster t with t̂ = s and sons(t) = ∅
3:      if sons(r) ≠ ∅ then
4:          {r_1, r_2} ← sons(r)

5:          s_1 ← {i ∈ s : X_i ∩ Y_{r_2} = ∅}
6:          s_2 ← {i ∈ s : X_i ∩ Y_{r_1} = ∅}
7:          s_3 ← {i ∈ s \ (s_1 ∪ s_2) : X_i ∩ X_{s_1} = ∅ and X_i ∩ X_{s_2} = ∅}
8:          s_4 ← {i ∈ s \ (s_1 ∪ s_2 ∪ s_3) : X_i ∩ X_{s_2} = ∅}
9:          s_5 ← s \ ⋃_{i=1}^{4} s_i

10:         t_1 ← CoupledIDClustering(s_1, r_1, n_leaf)
11:         t_2 ← CoupledIDClustering(s_2, r_2, n_leaf)
12:         t_3 ← InterfaceClustering(s_3, n_leaf, 1)        ▷ see Algorithm 4.2
13:         t_4 ← InterfaceClustering(s_4, n_leaf, 1)
14:         t_5 ← InterfaceClustering(s_5, n_leaf, 1)
15:         sons(t) ← {t_1, t_2, t_3, t_4, t_5}
16:     end if
17:     return t
18: end function
```

The strategy described above is summarized in Algorithm 5.2. The function called in lines 12–14 of Algorithm 5.2, `InterfaceClustering`, is again from Algorithm 4.2. Fig-

ure 5.4 illustrates the coupled clustering with interface decomposition and its effect on the block structure of the matrices $\mathbf{B}^k$ $(k = 1, \ldots, d)$ as well as the matrix $\check{\mathbf{F}}$.

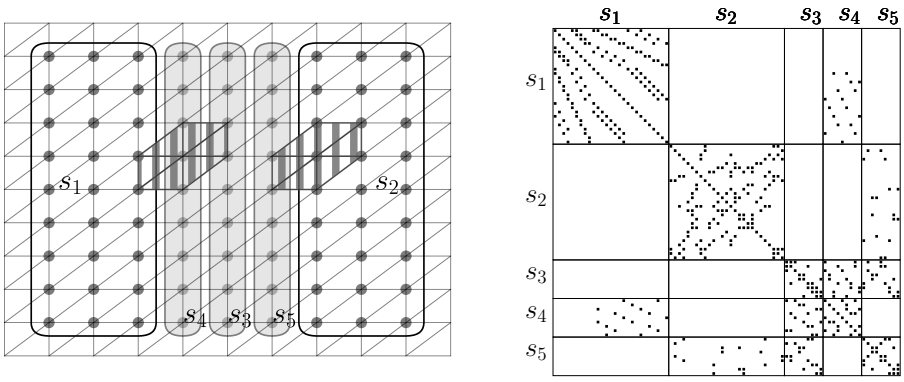

Figure 5.4: Effect of the reordering induced by the coupled clustering with interface decomposition for a 2D example. The left figure shows the geometric decomposition of $\mathcal{I}$ into the two domain clusters s_1 and s_2 as well as the three interface clusters s_3, s_4, and s_5 (marked light gray). Additionally, it shows the support of a basis function φ_i with $i \in s_2$ and the support of a basis function φ_j with $j \in s_4$ (both dark gray, hatched vertically). The right figure shows the effect of the reordering to the matrix $\check{\mathbf{F}}$.

In the previous section, we used the same terminology for the different clusters generated by Algorithms 4.2 and 5.1 as for the domain decomposition clustering. Furthermore, we introduced a mapping ${}^{\mathrm{a}} : \mathcal{C}_{\mathrm{dom}} \to \mathcal{T}_{\mathcal{J}}$ assigning associated clusters from $\mathcal{T}_{\mathcal{J}}$ to domain clusters from $\mathcal{T}_{\mathcal{I}}$ (see Definition 5.6). For the coupled clustering with interface decomposition, we expand this terminology.

Definition 5.13. *Let $\mathcal{T}_{\mathcal{I}}$ be a cluster tree generated by Algorithm 5.2, coupled with the cluster tree $\mathcal{T}_{\mathcal{J}}$ generated with geometric bisection (cf. Section 4.2.1).*

1. *As in Definition 5.6, we call clusters from $\mathcal{T}_{\mathcal{I}}$ generated by Algorithm 5.2* domain clusters *and clusters from $\mathcal{T}_{\mathcal{I}}$ generated by Algorithm 4.2* interface clusters. *We denote the set of domain clusters of $\mathcal{T}_{\mathcal{I}}$ by $\mathcal{C}_{\mathrm{dom}}(\mathcal{T}_{\mathcal{I}})$ or just $\mathcal{C}_{\mathrm{dom}}$.*

2. *As in Definition 5.6, we use the mapping*

$$^{\mathrm{a}} : \mathcal{C}_{\mathrm{dom}} \to \mathcal{T}_{\mathcal{J}};\ s \mapsto s^{\mathrm{a}}$$

   assigning associated clusters $s^{\mathrm{a}} \in \mathcal{T}_{\mathcal{J}}$ to the corresponding domain clusters $s \in \mathcal{C}_{\mathrm{dom}}$.

3. *Let $s \in \mathcal{T}_{\mathcal{I}}$ be a domain cluster with*

$$\mathrm{sons}(s) = \{s_1, \ldots, s_5\}$$

   where $s_1, \ldots, s_5$ are as defined in Algorithm 5.2. Then s_3 is separated from both domain clusters s_1 and s_2. We call s_3 separated interface cluster. *The cluster s_4 is separated from s_2 but not from s_1, i.e., $X_{s_4} \cap X_{s_2} = \emptyset$ and $X_{s_4} \cap X_{s_1} \neq \emptyset$ holds. The cluster s_5, on the other hand, is separated from s_1 but not from s_2. We say that s_4*

is connected *to* s_1 *and* s_5 *is connected to* s_2 *and call* s_4 *and* s_5 connected interface clusters.

We denote the set of connected interface clusters of $\mathcal{T}_\mathcal{I}$ *with* $\mathcal{C}_{\text{cint}}(\mathcal{T}_\mathcal{I})$ *or just* $\mathcal{C}_{\text{cint}}$ *and the set of separated interface clusters of* $\mathcal{T}_\mathcal{I}$ *with* $\mathcal{C}_{\text{sint}}(\mathcal{T}_\mathcal{I})$ *or just* $\mathcal{C}_{\text{sint}}$.

4. *Let* $s \in \mathcal{T}_\mathcal{I}$ *be a connected interface cluster. Then it is connected to a (unique) domain cluster* $t_s \in \mathcal{C}_{\text{dom}}$ *with the same father as* s. *We define the mapping*

$$^{\text{c}} : \mathcal{C}_{\text{cint}} \to \mathcal{C}_{\text{dom}}$$

assigning the corresponding connected domain cluster $s^{\text{c}} = t_s$ *to each connected cluster* s.

This allows us to define a new admissibility condition for the block cluster tree $\mathcal{T}_{\mathcal{I}\times\mathcal{I}}$ marking blocks with geometrically separated clusters as admissible.

Definition 5.14. *Let* $\mathcal{T}_\mathcal{I}$ *be a cluster tree generated with the coupled clustering with interface decomposition. Then we define the* (strong) coupled interface decomposition admissibility condition

$$\text{adm}_\eta^{\text{cid}} : \mathcal{T}_\mathcal{I} \times \mathcal{T}_\mathcal{I} \to \{\text{true}, \text{false}\}$$

with

$$\text{adm}_\eta^{\text{cid}}(t,s) = \begin{cases} \text{true} & \text{if } t,s \in \mathcal{C}_{\text{dom}} \text{ and } t \neq s, \\ \text{true} & \text{if } t \in \mathcal{C}_{\text{dom}}, s \in \mathcal{C}_{\text{cint}} \text{ and } t \neq s^{\text{c}}, \\ \text{true} & \text{if } t \in \mathcal{C}_{\text{cint}}, s \in \mathcal{C}_{\text{dom}} \text{ and } t^{\text{c}} \neq s, \\ \text{true} & \text{if } t \in \mathcal{C}_{\text{dom}}, s \in \mathcal{C}_{\text{sint}} \text{ or } t \in \mathcal{C}_{\text{sint}}, s \in \mathcal{C}_{\text{dom}}, \\ \text{adm}_\eta(t,s) & \text{else.} \end{cases}$$

By replacing adm_η *in the last case with the weaker admissibility conditions* $\text{adm}_{\text{sp},n_{\min}}$ *from Definition 4.32 or* adm_{weak} *from Definition 4.33, we obtain a* sparse *or* weak coupled interface decomposition admissibility condition $\text{adm}_{\text{sp},n_{\min}}^{\text{cid}}$ *or* $\text{adm}_{\text{weak}}^{\text{cid}}$, *respectively.*

Remark 5.15. 1. *As for the domain decomposition clustering (cf. Remark 4.26) and the coupled clustering (cf. Remark 5.10), we can replace the stricter conditions like* $X_i \cap Y_{r_2} = \emptyset$ *by* $X_i \cap Y_{r_2}$ *being a Lebesgue null set. This can be determined, e.g., by the sparsity pattern of the matrices* $\mathbf{B}^k$ *(*$k = 1, \ldots, d$*). The corresponding conditions for the sets* s_3, s_4, *and* s_5 *can then be determined by the sparsity pattern of the matrix* $\check{\mathbf{F}}$.

2. *In Figure 5.4 we can also observe that the blocks* $\mathbf{F}|_{s_4\times s_5}$ *and* $\mathbf{F}|_{s_5\times s_4}$ *are zero blocks. This can be explained as follows. An entry* f_{ij} *of* $\mathbf{F}$ *is non-zero not only if* $X_i \cap X_j = \emptyset$, *but already if* $X_i \cap X_j$ *has Lebesgue measure zero, i.e., is a Lebesgue null set.*

   Since the sets X_i *are of the form*

$$X_i = \text{supp}(\varphi_i) = \bigcup_{\substack{K \in \mathcal{T}^{h/2} \\ \boldsymbol{\xi}_i \in K}} \overline{K},$$

the intersection is of the form

$$X_i \cap X_j \bigcup_{\substack{K \in \mathcal{T}^{h/2} \\ \boldsymbol{\xi}_i \in K}} \bigcup_{\substack{L \in \mathcal{T}^{h/2} \\ \boldsymbol{\xi}_j \in L}} \overline{K} \cap \overline{L}.$$

Since $\mathcal{T}^{h/2}$ is a triangulation (cf. Definition 3.6), the intersection $X_i \cap X_j$ is therefore a union of points, edges, faces (for $d = 3$) and simplices. Only the latter have a positive measure and have to contain both, $\boldsymbol{\xi}_i$ and $\boldsymbol{\xi}_j$ as vertices. So $X_i \cap X_j$ has a positive measure only if there is a $K \in \mathcal{T}^{h/2}$ with $\boldsymbol{\xi}_i, \boldsymbol{\xi}_j \in K$. But for $i \in s_4$ and $j \in s_5$ there is no simplex $K \in \mathcal{T}^h$ with $\boldsymbol{\xi}_i, \boldsymbol{\xi}_j \in K$ (see, e.g., Figure 5.4). Hence, we could mark the blocks $s_4 \times s_5$ and $s_5 \times s_4$ as admissible blocks. But, due to the update of these blocks throughout the computation of the LU factorization, we may get significant fill-in in these blocks.

5.4 Numerical results

In this section, we will evaluate the results of numerical experiments comparing the coupled clustering from Section 5.2, the coupled clustering with interface decomposition from Section 5.3 and the uncoupled clustering from Section 5.1.

We will start with a brief description of the model problem used in the numerical experiments in Section 5.4.1. The evaluation in Section 5.4.2 is then structured as follows.

1. Results for the cluster strategies described in Sections 5.1 to 5.3 for a varying system size and fixed truncation accuracies.
2. Results for the uncoupled clustering and the coupled clustering for a fixed system size and varying truncation accuracies.
3. Results for the uncoupled clustering and the coupled clustering with different admissibility conditions and a varying system size.

For Oseen equations, the stability of the linear systems obtained from the discretization depends primarily on the viscosity parameter ν (see the discussion in Section 3.4). The smaller ν gets, the less stable the system is. Hence, we will examine the behavior of the different preconditioners for decreasing ν last.

5.4.1 Model problem

As a model problem for our numerical experiments, we will use the Oseen equations (cf. (3.3)) with the recirculating convection

$$\mathbf{w}_{\mathrm{re}} = \begin{pmatrix} -\sin(\pi x_1)(\cos(\pi x_2)\sin(\pi x_3) + \sin(\pi x_2)\cos(\pi x_3)) \\ \sin(\pi x_2)(\cos(\pi x_1)\sin(\pi x_3) - \sin(\pi x_1)\cos(\pi x_3)) \\ \sin(\pi x_3)(\cos(\pi x_1)\sin(\pi x_2) + \sin(\pi x_1)\cos(\pi x_2)) \end{pmatrix}$$

on the cube $\Omega := (-1, 1)^3$. The boundary conditions are homogeneous Dirichlet boundary conditions, i.e., the Dirichlet part Γ_D of the boundary is the whole boundary $\Gamma_D = \partial\Omega$ and

the boundary values are given by $\mathbf{g}_D = \mathbf{0}$. The viscosity parameter ν is set to $\nu = 10^{-2}$. The continuous problem is discretized using the FEM with piecewise linear basis functions (cf. Section 3.3) and the upwind method described in Section 3.4. The triangulation $\mathcal{T}^h$ is built from a coarse triangulation $\mathcal{T}^H$ by repeated refinement via connecting the edge midpoints (cf. Remark 3.7). This halves the mesh width with each refinement. The coarse triangulation $\mathcal{T}^H$ decomposes Ω into 6 tetrahedra (see, e.g., [15, Fig. 7.8]). This determines $n = |\mathcal{I}|$, $M = |\mathcal{J}|$ (cf. Remark 5.1) as well as the total system size $3n + M$. The values of n, M and $3n + M$ for different numbers of refinements are shown in Table 5.1.

Refinements	3	4	5	6
M	728	4,912	35,936	274,624
n	3,375	29,791	250,047	2,048,383
$3n + M$	10,853	94,285	786,077	6,419,773

Table 5.1: System sizes of the model problem for different numbers of repeated refinements

Throughout the remainder of this chapter, we will use the following notations for different parameters for the set-up of the required $\mathcal{H}$-matrices:

- We will denote the truncation accuracy for the $\mathcal{H}$-LU factorization of $\check{\mathbf{F}}$ (step ①) with $\delta^{\mathcal{H}}_{\text{vel}}$, the truncation accuracy for the explicit computation of the Schur complement (steps ②–④) with $\delta^{\mathcal{H}}_{\text{mul}}$, and the truncation accuracy for the $\mathcal{H}$-LU factorization of the Schur complement (step ⑤) with $\delta^{\mathcal{H}}_{\text{Schur}}$. If the same truncation accuracy is used for all steps, we will also denote it by

$$\delta^{\mathcal{H}} := \delta^{\mathcal{H}}_{\text{vel}} = \delta^{\mathcal{H}}_{\text{mul}} = \delta^{\mathcal{H}}_{\text{Schur}}. \tag{5.19}$$

- The block cluster trees $\mathcal{T}_{\mathcal{I}\times\mathcal{I}}$, $\mathcal{T}_{\mathcal{J}\times\mathcal{I}}$, and $\mathcal{T}_{\mathcal{J}\times\mathcal{J}}$ are constructed from the cluster trees $\mathcal{T}_{\mathcal{I}}$ and $\mathcal{T}_{\mathcal{J}}$ with an admissibility condition (cf. Definition 4.10). We will denote these admissibility conditions with $\text{adm}_{\mathcal{I}\times\mathcal{I}}$, $\text{adm}_{\mathcal{J}\times\mathcal{I}}$, and $\text{adm}_{\mathcal{J}\times\mathcal{J}}$, respectively. Note that we will use $\text{adm}_{\mathcal{J}\times\mathcal{J}} = \text{adm}_{\eta}$ in all cases since the weaker admissibility conditions are not suitable for the representation of the (dense) Schur complement, i.e., would result in significantly larger ranks for the (admissible) low-rank blocks.

Furthermore, we will use the preconditioned BiCGStab method (PBiCGStab method) to solve the systems with the block triangular preconditioner (cf. Equation (2.12))

$$\widetilde{\mathcal{P}}_{\text{lt}} = \begin{pmatrix} \mathbf{L_F U_F} & \\ \mathbf{B} & \mathbf{L_S U_S} \end{pmatrix}.$$

The PBiCGStab method started with $\mathbf{x}_0 = \mathbf{0}$ and is stopped when a relative residual norm of at most 10^{-12} is achieved.

Remark 5.16. Choice of the solver *We restricted our tests to preconditioned Krylov subspace methods since those methods are among the most popular iterative solvers for linear systems. However, the PBiCGStab method and the preconditioned GMRes method method performed rather similar in our tests so that we only present the results for the PBiCGStab method here.*

Stopping criterion *The choice of* 10^{-12} *as maximal relative residual norm is rather small. In practice, one may consider aligning this value to the discretization error. However, this parameter does not affect the preconditioner set-up, in which we are primarily interested. However, a larger number of iteration steps, which is the result of a smaller value for the maximal residual norm, may allow for a better estimation of the application cost (per iteration step) of the preconditioners.*

The system solved is the saddle point system

$$\mathcal{M}\begin{pmatrix}\mathbf{x}\\ \mathbf{y}\end{pmatrix} = \begin{pmatrix}\mathbf{F} & \mathbf{B}^\mathsf{T}\\ \mathbf{B} & \mathbf{0}\end{pmatrix}\begin{pmatrix}\mathbf{x}\\ \mathbf{y}\end{pmatrix} = \begin{pmatrix}\mathbf{r}\\ \mathbf{s}\end{pmatrix}$$

obtained from the discretization of the saddle point problem with the finite element method, and with the right hand side

$$\begin{pmatrix}\mathbf{r}\\ \mathbf{s}\end{pmatrix} = \mathcal{M}\begin{pmatrix}1\\ \vdots\\ 1\end{pmatrix}.$$

Software All numerical tests presented here were performed using the H2Lib [6], a software library for hierarchical matrices written in C. The H2Lib library also provides a simple FEM discretization for the Poisson equation, but not for the Oseen equations. Hence, this was added based on the existing implementation.

Hardware All numerical tests were performed sequentially, i.e., with one thread on one processor core, on the HPC cluster of the TUHH[1], using an AMD Epyc 9354 processor with 32 cores and 3.8 GHz.

5.4.2 Results

In the following, we will present the results of our numerical experiments as well as conclusions we draw from the results. The discussion is divided into subsections following the test settings mentioned in the introduction of this Section 5.4:

1. A varying system size and fixed truncation accuracies.
2. A fixed system size and varying truncation accuracies.
3. Different admissibility conditions with a varying system size.

System size

We fix the truncation accuracies for all (truncated) $\mathcal{H}$-matrix operations of the preconditioner set-up to $\delta^{\mathcal{H}} = 10^{-1}$ (cf. Equation (5.19)).

Remark 5.17. *The choice of* $\delta^{\mathcal{H}}$ *balanced between a faster set-up of a less accurate preconditioner and a slower set-up for a more accurate preconditioner leading to a smaller number of iteration steps. Although* $\delta^{\mathcal{H}} = 10^{-1}$ *is a rather large choice for the truncation accuracy, it results in an accurate preconditioner. A smaller choice of* $\delta^{\mathcal{H}}$ *only leads to an increase of the total time required to solve the system for all tested cluster strategies.*

[1] `https://www.tuhh.de/rzt/services/hpc/hardware`, last accessed August 8th, 2024

The system sizes depend on the level of refinement for the triangulation $\mathcal{T}^h$ (cf. Table 5.1) which is varied between 3 and 6.

In the case of the uncoupled clustering, we used the admissibility conditions

$$\mathrm{adm}_{\mathcal{I}\times\mathcal{I}} = \mathrm{adm}_{\eta}^{\mathrm{DD}}, \quad \mathrm{adm}_{\mathcal{J}\times\mathcal{I}} = \mathrm{adm}_{\eta}, \quad \text{and } \mathrm{adm}_{\mathcal{J}\times\mathcal{J}} = \mathrm{adm}_{\eta}\,. \tag{5.20}$$

For the coupled clustering, we can utilize the separation of domain clusters from their associated clusters which is why we used the admissibility conditions

$$\mathrm{adm}_{\mathcal{I}\times\mathcal{I}} = \mathrm{adm}_{\eta}^{\mathrm{DD}}, \quad \mathrm{adm}_{\mathcal{J}\times\mathcal{I}} = \mathrm{adm}_{\eta}^{\mathrm{c}}, \quad \text{and } \mathrm{adm}_{\mathcal{J}\times\mathcal{J}} = \mathrm{adm}_{\eta}\,. \tag{5.21}$$

For the coupled clustering with interface decomposition, on the other hand, we used

$$\mathrm{adm}_{\mathcal{I}\times\mathcal{I}} = \mathrm{adm}_{\eta}^{\mathrm{cid}}, \quad \mathrm{adm}_{\mathcal{J}\times\mathcal{I}} = \mathrm{adm}_{\eta}^{\mathrm{c}}, \quad \text{and } \mathrm{adm}_{\mathcal{J}\times\mathcal{J}} = \mathrm{adm}_{\eta}\,. \tag{5.22}$$

Figure 5.5 shows the computation times per degree of freedom for the coupled clustering and the uncoupled clustering, while Table 5.2 shows the absolute computation times.

Uncoupled clustering

#DOF	①	②	④	⑤	PBiCGStab (Iter)	Total
10,853	0.2s	0.7s	1.6s	0.1s	0.05s (9)	3.2s
94,285	5.9s	16.4s	33.8s	2.3s	1.2s (13)	74.8s
786,077	78.0s	214.9s	461.6s	41.3s	19.0s (18)	1,025.9s
6,419,773	782.4s	2,252.6s	5,322.6s	580.1s	268.2s (26)	11,492.6s

Coupled clustering

#DOF	①	②	④	⑤	PBiCGStab (Iter)	Total
10,853	0.6s	0.4s	0.3s	0.1s	0.06s (9)	1.8s
94,285	10.4s	7.8s	8.2s	2.4s	1.8s (13)	38.3s
786,077	129.3s	106.2s	149.6s	43.5s	29.3s (21)	567.4s
6,419,773	1,286.2s	1,152.8s	2,073.6s	646.2s	445.4s (33)	6,825.1s

Table 5.2: Absolute computation times for the preconditioner set-up and the solve with the BiCGStab method with the uncoupled clustering (top) and the coupled clustering (bottom). The time required to solve the system is supplemented by the number of iteration steps required to achieve a residual error of 10^{-12}.

Both show a significant speed-up for the computation of the matrices $\mathbf{V}^k$ (step ②) of the preconditioner set-up) with the coupled clustering. The computation of the matrices $\mathbf{W}^k$ (step ③)) requires a similar effort, i.e., this also holds for the computation of $\mathbf{W}^k$, $k = 1, \ldots, d$. The $\mathcal{H}$-matrix update for the explicit computation of an (approximate) Schur complement $\mathbf{S}_{\mathcal{H}}$ (step ④)) is significantly faster with the coupled clustering than with the uncoupled clustering as well. However, we can also observe that the enlarged interface that

is generated by the coupled clustering affects the time required for the computation of an $\mathcal{H}$-LU factorization $\mathbf{L_F U_F} \approx \check{\mathbf{F}}$. Additionally, we can observe that the time required for solving the system with the PBiCGStab method also slightly increases with the coupled clustering. This coincides with an increased number of iterations for larger system sizes. We can also observe that the computation of an $\mathcal{H}$-LU factorization of the (approximate) Schur complement $\mathbf{L_S U_S} \approx \mathbf{S}_\mathcal{H}$ requires a similar amount of time in both cases. This is to be expected since the block structure of the (approximate) Schur complement is the same in both cases.

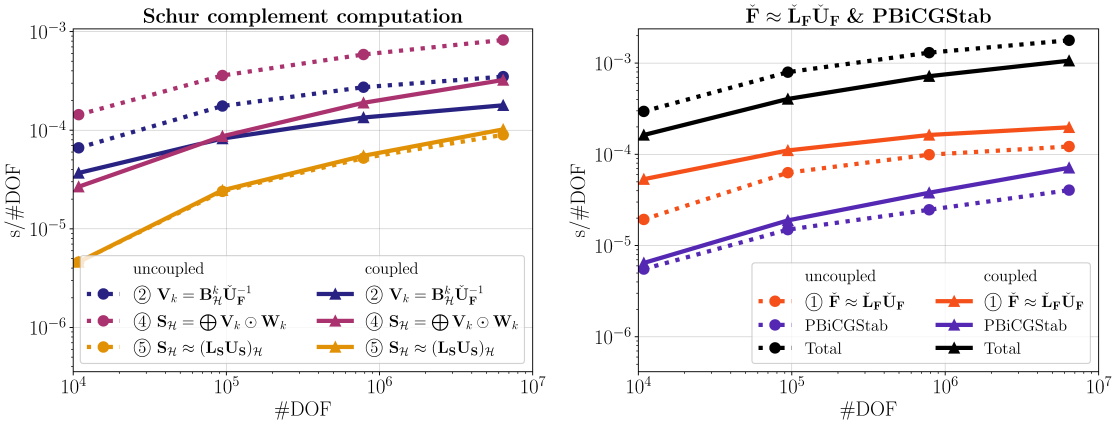

Figure 5.5: Computation times per degree of freedom for the preconditioner set-up and the solve with the BiCGStab method with the uncoupled clustering (dotted lines, circles) and the coupled clustering (solid lines, triangles), truncation accuracy $\delta^\mathcal{H} = 10^{-1}$.

Conclusion. Our numerical experiments suggest that the coupled clustering results in a significant speed-up of the time-consuming task of computing the (approximate) Schur complement $\mathbf{S}_\mathcal{H}$ explicitly. In Figure 5.5 we were able to observe a speed-up of about 50% for the computation of the matrices $\mathbf{V}^1$, $\mathbf{V}^2$, and $\mathbf{V}^3$ (step ② of the preconditioner set-up). For the $\mathcal{H}$-matrix updates in step ④, the speed-up is even greater.

However, the coupled clustering has also disadvantages. In Figure 5.5 and Table 5.2, we observed that the coupled clustering resulted in a significantly slower computation of an $\mathcal{H}$-LU factorization for the matrix $\check{\mathbf{F}}$. Additionally, the time and number of iterations required to achieve a residual error of 10^{-12} increased slightly. But since the preconditioner set-up is dominated by the time-consuming explicit computation of the (approximate) Schur complement $\mathbf{S}_\mathcal{H}$ (steps ②–④ of the preconditioner set-up), the coupled clustering still yields a speed-up of about 40% – 50% for the total time required to solve the system.

Next, Figure 5.6 and Table 5.3 show the computation times per degree of freedom and the absolute computation times, respectively, for the uncoupled clustering and the coupled clustering with interface decomposition.

There, we can observe a different behavior as for the coupled clustering. Especially for large system sizes the time required for the $\mathcal{H}$-matrix updates in step ④ of the preconditioner set-up with the coupled clustering with interface decomposition is about as expensive as with the uncoupled clustering.

Conclusion. Although the coupled clustering with interface decomposition provides a similar, advantageous, block structure of the (reordered) matrices $\mathbf{B}^k$, $k = 1, 2, 3$, the precondi-

Uncoupled clustering

#DOF	①	②	④	⑤	PBiCGStab (Iter)	Total
10,853	0.2s	0.7s	1.6s	0.1s	0.05s (9)	3.2s
94,285	5.9s	16.4s	33.8s	2.3s	1.2s (13)	74.8s
786,077	78.0s	214.9s	461.6s	41.3s	19.0s (18)	1,025.9s
6,419,773	782.4s	2,252.6s	5,322.6s	580.1s	268.2s (26)	11,492.6s

Coupled clustering with interface decomposition

#DOF	①	②	④	⑤	PBiCGStab (Iter)	Total
10,853	0.2s	0.6s	0.7s	0.1s	0.04s (8)	2.1s
94,285	6.4s	14.1s	20.3s	2.3s	1.1s (12)	58.1s
786,077	92.3s	197.3s	333.9s	41.2s	17.7s (16)	886.8s
6,419,773	973.0s	2,160.7s	4,306.4s	574.1s	267.0s (24)	10,581.1s

Table 5.3: Absolute computation times for the preconditioner set-up and the solve with the BiCGStab method with the uncoupled clustering (top) and the coupled clustering with interface decomposition (bottom). The time required to solve the system is supplemented by the number of iteration steps required to achieve a residual error of 10^{-12}.

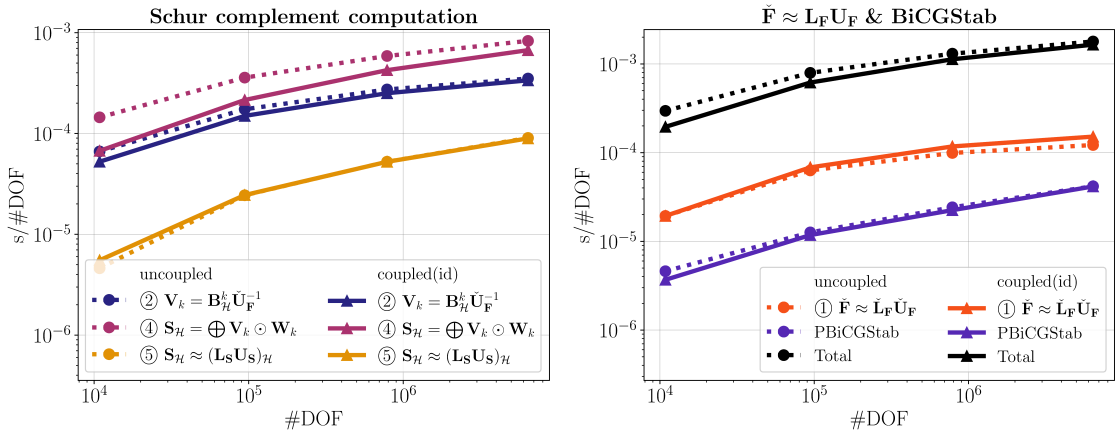

Figure 5.6: Computation times per degree of freedom for the preconditioner set-up and the solve with the BiCGStab method with the uncoupled clustering (dotted lines, circles) and the coupled clustering with interface decomposition (solid lines, triangles), truncation accuracy $\delta^{\mathcal{H}} = 10^{-1}$.

tioner set-up seems to be slower. We observed almost no speed-up of the total computation time compared to the uncoupled clustering. We suspect this to be the result of the additional interface decomposition leading to a larger number of (smaller) blocks in the block cluster tree $\mathcal{T}_{\mathcal{J}\times\mathcal{I}}$ in general and a larger sparsity constant $C_{\text{sp}}(\mathcal{T}_{\mathcal{J}\times\mathcal{I}})$ in particular. The results of the numerical experiments suggest that there is no advantage of using the coupled clustering with interface decomposition for the generation of $\mathcal{T}_{\mathcal{I}}$ in this setting.

Next, Figure 5.7 depicts the memory requirements of the matrices that are computed for the preconditioner set-up, for the uncoupled clustering and the coupled clustering as well as the uncoupled clustering and the coupled clustering with interface decomposition.

Remark 5.18. *The matrices $\mathbf{B}_{\mathcal{H}}^k$, $\mathbf{S}_{\mathcal{H}}$, and an $\mathcal{H}$-matrix representation $\check{\mathbf{F}}_{\mathcal{H}}$ of $\check{\mathbf{F}}$ also need to be stored. But $\check{\mathbf{F}}_{\mathcal{H}}$ and $\mathbf{S}_{\mathcal{H}}$ are overwritten by their corresponding $\mathcal{H}$-LU factorizations, while $\mathbf{B}_{\mathcal{H}}^k$ is overwritten by $\mathbf{V}^k$ or $\mathbf{W}^k$ for $k = 1, 2, 3$. In all cases, the matrices that are overwritten require less memory. Furthermore, the matrices $\mathbf{V}^k$, $k = 1, 2, 3$, as well as the matrices $\mathbf{W}^k$, $k = 1, 2, 3$, have similar memory requirements. Therefore, we only consider the memory requirements for the $\mathcal{H}$-LU factorizations $\check{\mathbf{L}}_{\mathbf{F}}\check{\mathbf{U}}_{\mathbf{F}}$ and $\mathbf{L}_{\mathbf{S}}\mathbf{U}_{\mathbf{S}}$, both factorizations stored in a single matrix, respectively, and the memory requirements for the matrix $\mathbf{V}^1$.*

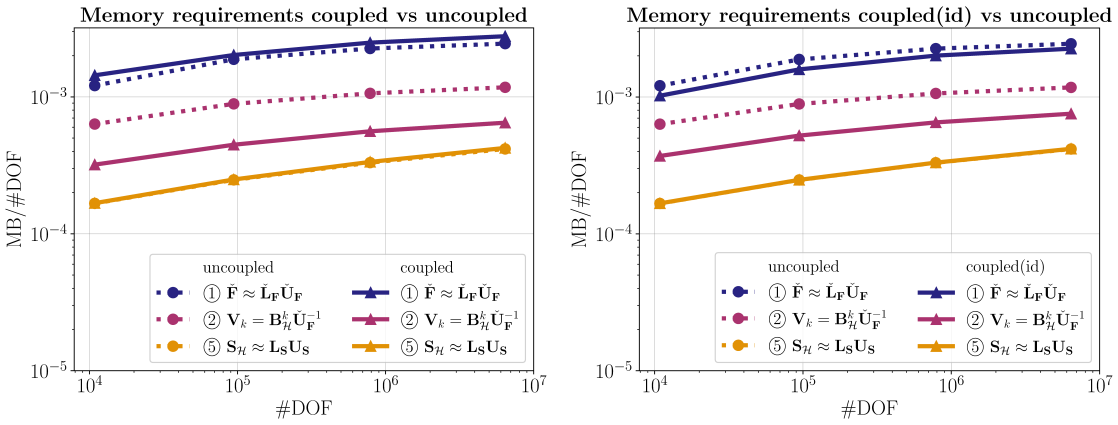

Figure 5.7: Memory requirements in Megabyte per degree of freedom for the uncoupled clustering and the coupled clustering (left) as well as the uncoupled clustering and the coupled clustering with interface decomposition (right).

In Figure 5.7, we can observe that the difference in the memory requirements for the $\mathcal{H}$-LU factorization $\check{\mathbf{L}}_{\mathbf{F}}\check{\mathbf{U}}_{\mathbf{F}}$ with the uncoupled clustering and the coupled clustering is only negligible, despite the larger interface generated by the coupled clustering. But the matrix $\mathbf{V}^1$, on the other hand, requires significantly less memory (about 50%) with the coupled clustering than with the uncoupled clustering.

The coupled clustering with interface decomposition requires less memory for both, the $\mathcal{H}$-LU factorization $\check{\mathbf{L}}_{\mathbf{F}}\check{\mathbf{U}}_{\mathbf{F}}$ and $\mathbf{V}^1$, compared to the uncoupled clustering in both cases. But, we note that the difference between both cluster strategies for the $\mathcal{H}$-LU factorization $\check{\mathbf{L}}_{\mathbf{F}}\check{\mathbf{U}}_{\mathbf{F}}$ is also only marginal.

The memory required for the $\mathcal{H}$-LU factorization $\mathbf{L}_{\mathbf{S}}\mathbf{U}_{\mathbf{S}}$ is similar in all cases which is to be expected due to the equal block-structure in all cases (The cluster strategy for $\mathcal{T}_{\mathcal{J}}$

and the admissibility condition for $\mathcal{T}_{\mathcal{J}\times\mathcal{J}}$ are the same).

Conclusion. In Figure 5.7 we were able to observe that both the coupled clustering and the coupled clustering with interface decomposition reduce the memory required for the matrix $\mathbf{V}^1$ significantly. The memory required for the $\mathcal{H}$-LU factorization $\check{\mathbf{L}}_\mathbf{F}\check{\mathbf{U}}_\mathbf{F}$, on the other hand, is increased only slightly with the coupled clustering and even reduced with the coupled clustering with interface decomposition. Hence, both reduce the required memory for intermediate results effectively.

Truncation accuracy

Following the tests for different system sizes, we will now consider varying truncation accuracies with a fixed system size. For our tests, we used $n = 250,047$ and $M = 35,937$ resulting in $3n + M = 786,078$ total degrees of freedom (cf. Table 5.1).

The results shown in the previous subsection indicated that the coupled clustering with interface decomposition does not improve the time required for the set-up of the preconditioner. Therefore, we will only compare the coupled clustering and the uncoupled clustering from now on.

As for the previous tests, we constructed the block cluster trees with the admissibility conditions

$$\mathrm{adm}_{\mathcal{I}\times\mathcal{I}} = \mathrm{adm}_\eta^{\mathrm{DD}}, \quad \mathrm{adm}_{\mathcal{J}\times\mathcal{I}} = \mathrm{adm}_\eta, \quad \text{and } \mathrm{adm}_{\mathcal{J}\times\mathcal{J}} = \mathrm{adm}_\eta \ (\eta = 16)$$

in case of the uncoupled clustering and

$$\mathrm{adm}_{\mathcal{I}\times\mathcal{I}} = \mathrm{adm}_\eta^{\mathrm{DD}}, \quad \mathrm{adm}_{\mathcal{J}\times\mathcal{I}} = \mathrm{adm}_\eta^{\mathrm{c}}, \quad \text{and } \mathrm{adm}_{\mathcal{J}\times\mathcal{J}} = \mathrm{adm}_\eta \ (\eta = 16)$$

in case of the coupled clustering. The truncation accuracies $\delta_{\mathrm{vel}}^{\mathcal{H}}$ and $\delta_{\mathrm{mul}}^{\mathcal{H}}$ are varied one at the time starting with $\delta_{\mathrm{vel}}^{\mathcal{H}}$.

Remark 5.19. *We do not consider varying $\delta_{\mathrm{Schur}}^{\mathcal{H}}$, since it only affects the factorization of the (approximate) Schur complement $\mathbf{S}_\mathcal{H}$. Here we can expect the same behavior for the coupled and the uncoupled clustering since the block structure of $\mathbf{S}_\mathcal{H}$ is the same in both cases.*

We vary $\delta_{\mathrm{vel}}^{\mathcal{H}} \in \{10^{-1}, \ldots, 10^{-4}\}$ and fix the other truncation accuracies to $\delta_{\mathrm{mul}}^{\mathcal{H}} = \delta_{\mathrm{Schur}}^{\mathcal{H}} = 10^{-1}$. The results are visualized in Figure 5.8 depicting the computation times per degree of freedom, as before, as well as the memory requirements for the matrices computed for the preconditioner set-up (cf. Remark 5.18).

We can observe that increasing the truncation accuracy, i.e., decreasing $\delta_{\mathrm{vel}}^{\mathcal{H}}$, in general leads to increased computation times for the $\mathcal{H}$-LU factorization of $\check{\mathbf{F}}$, which is to be expected. But we can also observe an increasing difference between the coupled and the uncoupled clustering in the computation times as well as the memory requirements. This can be explained by the fill-in the $\mathcal{H}$-LU factorization introduces. In the case of the uncoupled clustering, the block structure was chosen to minimize this fill-in. For the coupled clustering the situation is different. We introduced a larger interface to optimize the block structure for the $\mathcal{H}$-matrix updates for the explicit computation of an (approximate) Schur complement $\mathbf{S}_\mathcal{H}$. But with the larger interface, we have more potential for fill-in, which we can also observe in Figure 5.9. Although the ranks of the blocks filled in by the $\mathcal{H}$-LU factorization are smaller for the coupled clustering, there are more of these blocks compared to the uncoupled clustering, many of them tall or wide rectangular blocks.

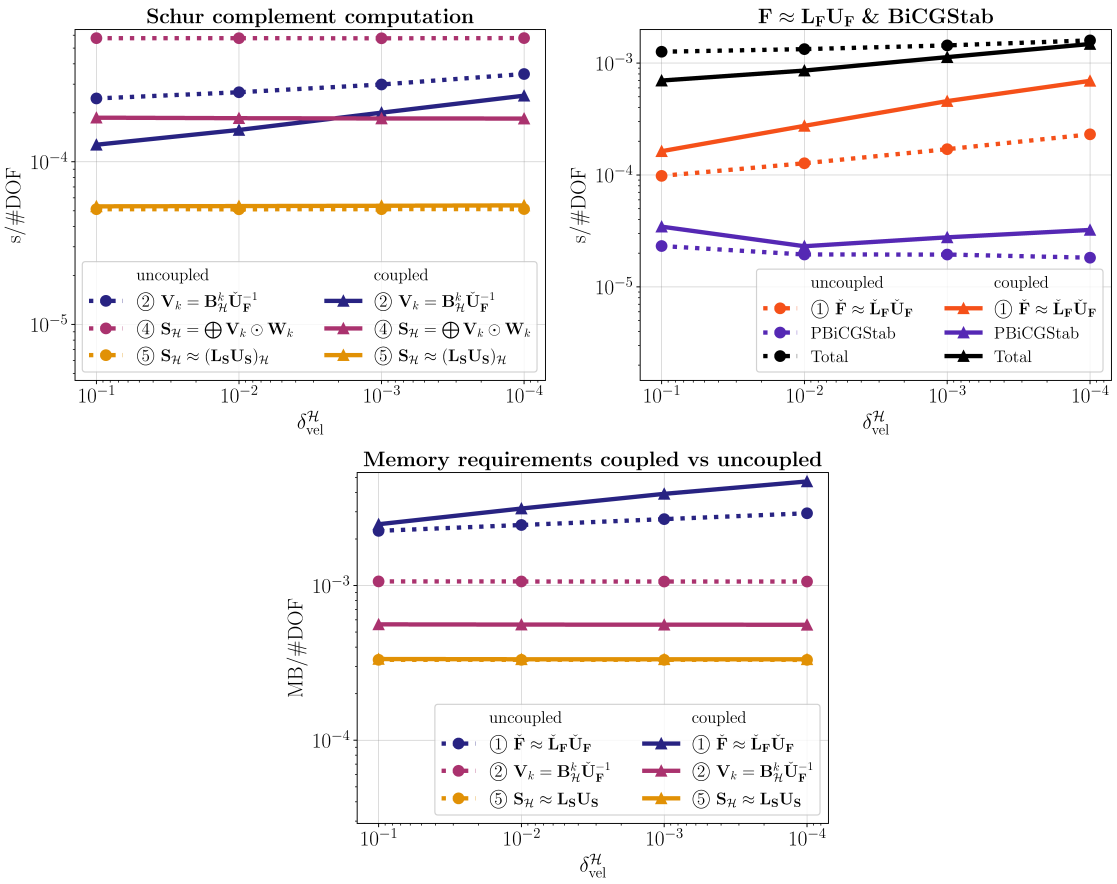

Figure 5.8: Top: computation times per degree of freedom for the preconditioner set-up and the solve with the BiCGStab method with the uncoupled clustering (dotted lines, circles) and the coupled clustering (solid lines, triangles) for a varying truncation accuracy $\delta_{\text{vel}}^{\mathcal{H}}$. Bottom: memory requirements per degree of freedom for matrices computed for the preconditioner set-up, $3n + M = 3 \cdot 250,047 + 35,937 = 786,078$ total degrees of freedom.

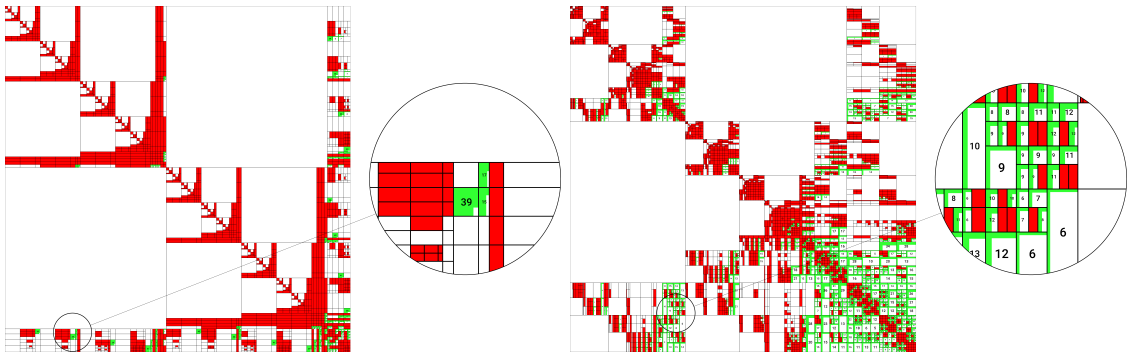

Figure 5.9: Hierarchical block structure of the $\mathcal{H}$-LU factorization of the matrix $\check{\mathbf{F}}$ ($\check{\mathbf{F}} \in \mathbb{R}^{n \times n}, n = 3,375$) stored in a single matrix. (red: dense matrix blocks, white/green: low-rank blocks). The left figure shows the block structure obtained with the uncoupled clustering and the right figure the block structure obtained with the coupled clustering. Both figures show magnified parts with occurring fill-in.

Conclusion. With a decreased truncation accuracy $\delta^{\mathcal{H}}_{\text{vel}}$, the rank of the low-rank blocks increases. This results in increased computation times, which we were also able to observe here. But the block structure induced by the coupled clustering, which generates larger interface clusters compared to the (uncoupled) domain decomposition clustering, amplifies this behavior. The estimates for the computational complexity of the $\mathcal{H}$-matrix arithmetic (see Section 4.4) as well as the block structure we observed in Figure 5.9 suggest that this is due to a larger number wide/tall rectangular non-zero low-rank blocks in the $\mathcal{H}$-LU factorization $\check{\mathbf{L}}_{\mathbf{F}}\check{\mathbf{U}}_{\mathbf{F}}$.

For the total computation time we observed a (slightly) larger increase in case of the coupled clustering. If $\delta^{\mathcal{H}}_{\text{vel}}$ is decreased further we expect this trend to continue so that the uncoupled clustering may be faster and therefore favorable for very small truncation accuracies.

Now, we will consider a varying truncation accuracy $\delta^{\mathcal{H}}_{\text{mul}} \in \{10^{-1}, \dots, 10^{-4}\}$ for the explicit computation of an (approximate) Schur complement $\mathbf{S}_{\mathcal{H}}$ and fix the other truncation accuracies to $\delta^{\mathcal{H}}_{\text{vel}} = \delta^{\mathcal{H}}_{\text{Schur}} = 10^{-1}$. The system size is not changed. The results are presented in Figure 5.10, showing the computation times and memory requirements, respectively, per degree of freedom.

As it can be expected, varying $\delta^{\mathcal{H}}_{\text{mul}}$ does not affect the computation time of the $\mathcal{H}$-LU factorizations of $\check{\mathbf{F}}$ and the (approximate) Schur complement $\mathbf{S}_{\mathcal{H}}$. For the time required to compute the $\mathcal{H}$-matrix updates in step ④ of the preconditioner set-up, on the other hand, we can notice an increase with decreasing $\delta^{\mathcal{H}}_{\text{mul}}$ for the uncoupled clustering and the coupled clustering, respectively. But we can also observe that the difference between both approaches diminishes, which is also the case for the memory required for the matrix $\mathbf{V}^1$.

As before, we take a closer look at the block structure of the involved matrices depicted in Figure 5.11. The matrices were obtained with truncation accuracies set to $\delta^{\mathcal{H}}_{\text{vel}} = 10^{-1}$ and $\delta^{\mathcal{H}}_{\text{mul}} = 10^{-4}$. The total number of degrees of freedom was

$$3n + M = 3 \cdot 3,375 + 729 = 10,854.$$

In Figure 5.11, we can observe a similar rank for the non-zero blocks in the interface. But, due to the larger interface, the low-rank blocks are wide rectangular blocks instead of nearly quadratic. This increases the computation time as well as the memory requirements and diminishes the advantage of the large rank-0 off-diagonal blocks.

Conclusion. As before an increase of the computation times for the steps ② – ④ is to be expected when $\delta^{\mathcal{H}}_{\text{mul}}$ is decreased. But we also observed that this increase is larger in case of the coupled clustering. This suggests that the coupled clustering is also less effective for a decreased $\delta^{\mathcal{H}}_{\text{mul}}$ since we also expect this trend to continue for even smaller values. The block structure depicted in Figure 5.11 suggests that this is due to many wide rectangular non-zero low-rank blocks in the matrices $\mathbf{V}^k$ and $\mathbf{W}^k$, $k = 1, 2, 3$, respectively.

Admissibility conditions

Besides the clustering, the block structure of hierarchical matrices is determined by the admissibility condition used for the construction of the block tree (see Definition 4.10). Up to now, we only considered the strong admissibility condition, although there exist weaker variants allowing for a coarser block structure (cf. Section 4.3). These weaker variants can also be utilized as a basis for the domain decomposition admissibility (cf. Definition 4.34)

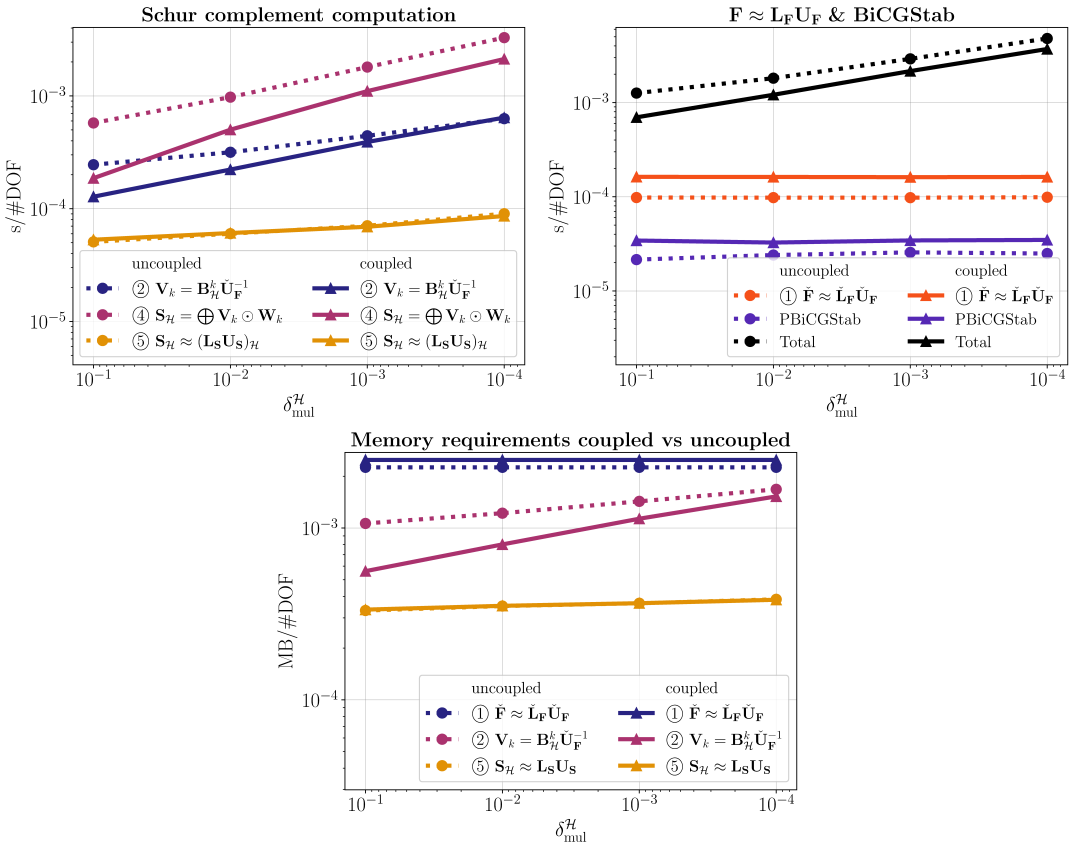

Figure 5.10: Top: computation times per degree of freedom for the preconditioner set-up and the solve with the BiCGStab method with the uncoupled clustering (dotted lines, circles) and the coupled clustering (solid lines, triangles) for a varying truncation accuracy $\delta^{\mathcal{H}}_{\text{mul}}$. Bottom: memory requirements per degree of freedom for matrices computed for the preconditioner set-up, $3n + M = 3 \cdot 250,047 + 35,937 = 786,078$ total degrees of freedom.

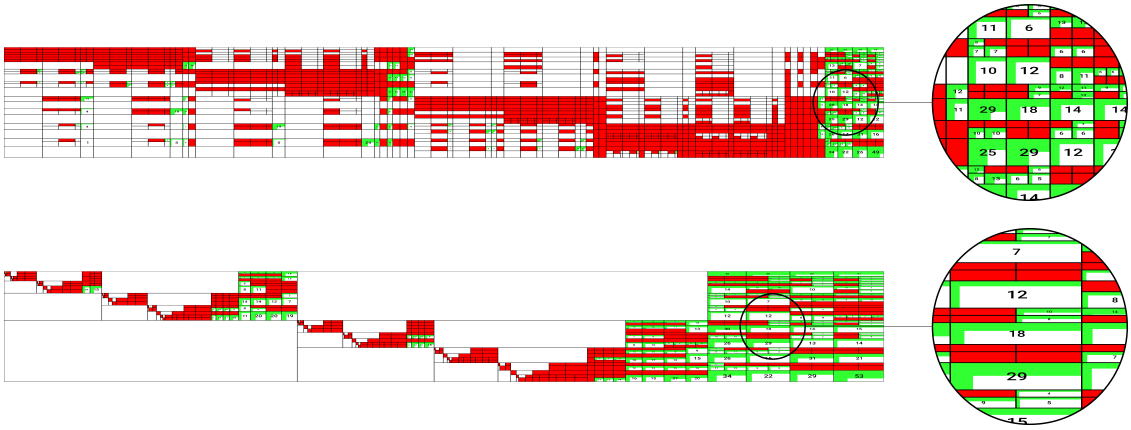

Figure 5.11: Hierarchical block structure of the matrix $\mathbf{V}^1 \in \mathbb{R}^{M \times n}$, $M = 728, n = 3375$ (red: dense matrix blocks, white/green: low-rank blocks). The top figure shows the block structure obtained with the uncoupled clustering and the bottom figure the block structure obtained with the coupled clustering.

and the coupled admissibility condition (cf. Definition 5.8). In the following, we will compare the coupled and the uncoupled clustering with different admissibility conditions used. We note that we still use the strong admissibility condition for the block structure of the Schur complement, i.e., $\mathrm{adm}_{\mathcal{J}\times\mathcal{J}} = \mathrm{adm}_\eta$ with $\eta = 16$ in all cases. When we refer to the strong admissibility, we used

$$\mathrm{adm}_{\mathcal{I}\times\mathcal{I}} = \mathrm{adm}_\eta^{\mathrm{DD}} \quad \text{as well as } \mathrm{adm}_{\mathcal{J}\times\mathcal{I}} = \mathrm{adm}_\eta \text{ or } \mathrm{adm}_{\mathcal{J}\times\mathcal{I}} = \mathrm{adm}_\eta^{\mathrm{c}},$$

where $\mathrm{adm}_{\mathcal{J}\times\mathcal{I}} = \mathrm{adm}_\eta$ is used in the case of the uncoupled clustering and $\mathrm{adm}_{\mathcal{J}\times\mathcal{I}} = \mathrm{adm}_\eta^{\mathrm{c}}$ in the case of the coupled clustering. When we refer to the weak or sparse admissibility, we used

$$\mathrm{adm}_{\mathcal{I}\times\mathcal{I}} = \mathrm{adm}_{\mathrm{weak}}^{\mathrm{DD}} \quad \text{as well as } \mathrm{adm}_{\mathcal{J}\times\mathcal{I}} = \mathrm{adm}_{\mathrm{weak}} \text{ or } \mathrm{adm}_{\mathcal{J}\times\mathcal{I}} = \mathrm{adm}_{\mathrm{weak}}^{\mathrm{c}}$$

or

$$\mathrm{adm}_{\mathcal{I}\times\mathcal{I}} = \mathrm{adm}_{\mathrm{sp},n_{\min}}^{\mathrm{DD}} \quad \text{as well as } \mathrm{adm}_{\mathcal{J}\times\mathcal{I}} = \mathrm{adm}_{\mathrm{sp},n_{\min}} \text{ or } \mathrm{adm}_{\mathcal{J}\times\mathcal{I}} = \mathrm{adm}_{\mathrm{sp},n_{\min}}^{\mathrm{c}}$$

with $n_{\min} = 10$, respectively.

Remark 5.20. *We also tested other values for $n_{\min}$. But, we observed similar results in all cases.*

We fix the truncation accuracies for all (truncated) $\mathcal{H}$-matrix operations of the preconditioner set-up to $\delta^{\mathcal{H}} = 10^{-1}$ (cf. Equation (5.19)). The system sizes depend on the level of refinement for the triangulation $\mathcal{T}^h$ (cf. Table 5.1) which is varied between 3 and 6. The computation times per degree of freedom for both operations are shown in Figure 5.12.

First note that we omitted the computation time per degree of freedom for $10,854$ degrees of freedom for the strong admissibility plots for distinguishability. What we can observe there is that there is almost no difference between the weak admissibility and the strong admissibility with $\eta = 16$. Without the omitted first point, the lines would not be distinguishable. This can be explained by the rather large value chosen for η. For the sparse admissibility we can observe that it leads to a speed-up of the preconditioner set-up in out tests. Note that from the complexity estimate for the $\mathcal{H}$-matrix updates in Theorem 4.44 and the tree depth estimates, e.g., in Lemma 4.24 and Theorem 5.11, we would expect the multiplication to require $\mathcal{O}(k\log_2(p)^2 p)$ operations where $p = 3n + M$ and k is the maximal local rank of the matrices $\mathbf{V}^k$ and $\mathbf{W}^k$ for $k = 1, 2, 3$, respectively. Hence, the plot for the explicit computation of the Schur complement (step ④) suggests that the local ranks of the matrices $\mathbf{V}^k, \mathbf{W}^k$ $(k = 1, 2, 3)$ increase significantly with an increasing number of degrees of freedom. The $\mathcal{H}$-LU factorization of $\breve{\mathbf{F}}$ and the triangular solves for the computation of $\mathbf{V}^k$, $k = 1, 2, 3$, however, seem to meet this expectation of log-linear complexity.

Conclusion. The results depicted in Figure 5.12 suggest that the sparse admissibility condition is a suitable and effective choice as admissibility condition for the construction of $\mathcal{T}_{\mathcal{I}\times\mathcal{I}}$ with both the uncoupled clustering and the coupled clustering. For the cluster tree $\mathcal{T}_{\mathcal{J}\times\mathcal{I}}$, however, it is not. Especially for a larger number of degrees of freedom, the $\mathcal{H}$-matrix updates for the explicit computation of the (approximate) Schur complement $\mathbf{S}_{\mathcal{H}}$ fails to achieve log-linear complexity as it is the case with the other admissibility conditions. Another thing to note here is that there is almost no difference between the weak and the strong admissibility condition in all cases. This can be explained by the rather large choice for η ($\eta = 16$).

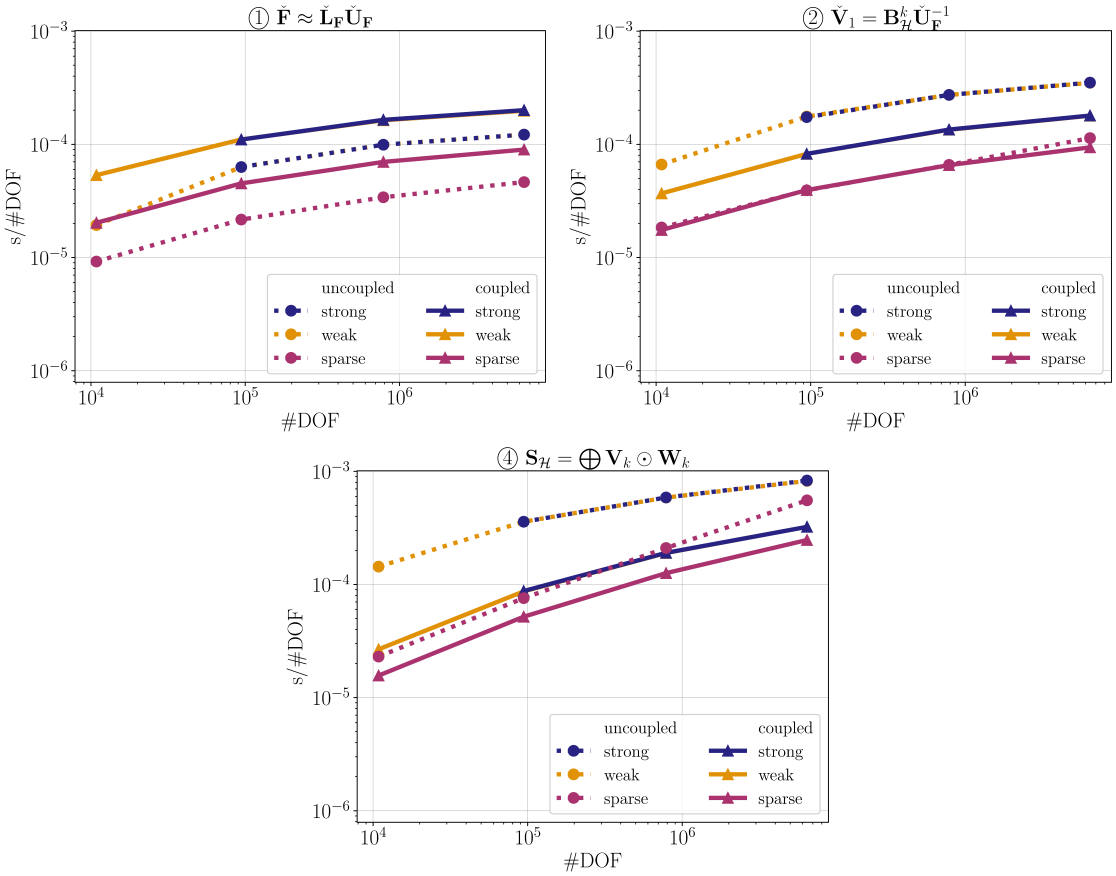

Figure 5.12: Computation times per degree of freedom for steps of the preconditioner set-up with different admissibility conditions and the uncoupled clustering (dotted lines, circles) as well as the coupled clustering (solid lines, triangles), truncation accuracy $\delta^\mathcal{H} = 10^{-1}$.

Remark 5.21. *Our key conclusions drawn from the presented numerical experiments are summarized as follows:*

- *For larger truncation accuracies* $\delta_{\text{vel}}^{\mathcal{H}}$ *and* $\delta_{\text{mul}}^{\mathcal{H}}$*, e.g.,* $\delta_{\text{vel}}^{\mathcal{H}} = \delta_{\text{mul}}^{\mathcal{H}} = 10^{-1}$*, the coupled clustering yields a favorable block structure of the matrices* $\mathbf{V}^k$ *and* $\mathbf{W}^k$*,* $k = 1, 2, 3$*, for the time-consuming explicit computation of an (approximate) Schur complement* $\mathbf{S}_{\mathcal{H}}$*.*

- *For smaller truncation accuracies, e.g.,* $\delta_{\text{vel}}^{\mathcal{H}}, \delta_{\text{mul}}^{\mathcal{H}} < 10^{-4}$*, the uncoupled clustering yields a more effective block structure. The larger interfaces generated by the coupled clustering are disadvantageous in this case.*

- *A rather large value for the truncation accuracies is sufficient for the construction of effective block triangular preconditioners for our model problem.*

- *While the sparse admissibility condition is a suitable choice for the construction of* $\mathcal{T}_{\mathcal{I}\times\mathcal{I}}$*, it is not for the construction of* $\mathcal{T}_{\mathcal{J}\times\mathcal{I}}$ *since the* $\mathcal{H}$*-matrix updates for the computation of an (approximate) Schur complement do not meet the expected log-linear complexity estimates. For the construction of* $\mathcal{T}_{\mathcal{J}\times\mathcal{I}}$ *the weak or the strong admissibility conditions (or variants based on these) are a better choice.*

- *While the coupled clustering with interface decomposition improves the time required for the computation of an* $\mathcal{H}$*-LU factorization* $\check{\mathbf{L}}_{\mathbf{F}}\check{\mathbf{U}}_{\mathbf{F}} \approx \check{\mathbf{F}}$*, as intended, it also results in significantly more time required for the other steps,* ② – ④*, of the preconditioner set-up compared to the coupled clustering. The total time required for the preconditioner set-up and the solve with the PBiCGStab method is comparable to the one with the uncoupled clustering.*

Chapter 6

Variants of the $\mathcal{H}$-matrix multiplication

The multiplication of hierarchical matrices has an important role in the (block) preconditioning with $\mathcal{H}$-matrix factorizations like the $\mathcal{H}$-LU factorization presented in Section 4.4.4. (Block) $\mathcal{H}$-matrix preconditioners for saddle point problems also require the explicit computation of an (approximate) Schur complement which involves $\mathcal{H}$-matrix updates directly (cf. Remark 5.3).

In the previous Chapter 5, we described approaches to improve the set-up time of block preconditioners for saddle point systems by different construction techniques for the hierarchical block structure of the involved matrices. Now, we will examine the multiplication algorithm itself.

While we described the multiplication algorithm in Section 4.4.2 as it was first introduced in [25], alternative approaches were developed recently in [5] and [11].

The first alternative approach, the multiplication with accumulated updates, is inspired by arithmetic operations of $\mathcal{H}^2$-matrices, a subclass of $\mathcal{H}$-matrices (see, e.g., [26]). The second approach organizes the summation of low-rank matrices differently with so-called sum-expressions. This allows for an easy truncation of the whole occurring sums of low-rank matrices using only matrix-vector operations and without computing these sums explicitly.

In the remainder of this chapter, we will describe both alternative versions briefly, and then explore their applicability to the block preconditioning of saddle point problems with numerical experiments. We start with the $\mathcal{H}$-matrix multiplication with accumulated updates from [5] in Section 6.1. Then, we will continue with the $\mathcal{H}$-matrix multiplication with sum-expressions from [11] in Section 6.2.

Throughout this chapter, let $\mathcal{I}$, $\mathcal{J}$, and $\mathcal{K}$ be finite index sets with cluster trees $\mathcal{T}_\mathcal{I}$, $\mathcal{T}_\mathcal{J}$, and $\mathcal{T}_\mathcal{K}$, respectively (see, Definition 4.6), with leaf size bound n_{leaf} (cf. Definition 4.13). Furthermore, let $\mathcal{T}_{\mathcal{I}\times\mathcal{K}}$, $\mathcal{T}_{\mathcal{K}\times\mathcal{J}}$, and $\mathcal{T}_{\mathcal{I}\times\mathcal{J}}$ be corresponding block cluster trees constructed with an admissibility condition (see Definitions 4.8 and 4.10) and $\mathcal{T}^{\text{ind}}_{\mathcal{I}\times\mathcal{J}}$ the induced product tree (see Remark 4.37). Then, we consider the update

$$\mathbf{Z} \leftarrow \mathbf{Z} \oplus \mathbf{X} \odot \mathbf{Y} \tag{6.1}$$

for hierarchical matrices $\mathbf{X} \in \mathcal{H}(\mathcal{T}_{\mathcal{I}\times\mathcal{K}}, k_1)$, $\mathbf{Y} \in \mathcal{H}(\mathcal{T}_{\mathcal{K}\times\mathcal{J}}, k_2)$, and $\mathbf{Z} \in \mathcal{H}(\mathcal{T}_{\mathcal{I}\times\mathcal{J}}, k_3)$ with local ranks $k_1, k_2, k_3 \in \mathbb{N}$ (see Definition 4.11).

6.1 $\mathcal{H}$-matrix multiplication with accumulated updates

6.1.1 The algorithm

The original algorithm for the update (6.1) is based on the induced product tree $\mathcal{T}^{\mathrm{ind}}_{\mathcal{I}\times\mathcal{J}}$. For leaves $b = (t, s) \in \mathcal{L}(\mathcal{T}^{\mathrm{ind}}_{\mathcal{I}\times\mathcal{J}})$ the product $(\mathbf{XY})|_{t\times s}$ is of the form (cf. (4.23b))

$$(\mathbf{XY})|_{t\times s} = \sum_{j=0}^{\mathrm{level}(b)} \sum_{r\in U^j(t,s)} \mathbf{X}|_{t\times r}\mathbf{Y}|_{r\times s}, \tag{6.2}$$

where the sets $U^j(t, s)$, $0 \le j \le \mathrm{level}(b)$ are from Definition 4.39. If $(t, s) \notin \mathcal{T}_{\mathcal{I}\times\mathcal{J}}$, each summand is added to the corresponding predecessor block $\mathbf{Z}|_{\widehat{b}}$ with $\widehat{b} \in \mathcal{L}_{\mathcal{I}\times\mathcal{J}}$. This requires a significant number of truncations for each successor of $\widehat{b}$ in the induced block cluster tree $\mathcal{T}^{\mathrm{ind}}_{\mathcal{I}\times\mathcal{J}}$.

The idea of the $\mathcal{H}$-matrix multiplication with accumulated updates is to compute the (truncated) sum

$$\bigoplus_{j=0}^{\mathrm{level}(b)} \bigoplus_{r\in U^j(t,s)} \mathbf{X}|_{t\times r}\mathbf{Y}|_{r\times s}$$

first, and then add the result to $\mathbf{Z}|_{\widehat{b}}$ instead of adding each summand directly to $\mathbf{Z}|_b$. If additionally $b \notin \mathrm{sons}(\widehat{b})$, i.e., $\widehat{b}$ is not the father of b but a predecessor multiple levels above b, the result is computed for all intermediate levels first. This approach ensures that each block $\widehat{b} \in \mathcal{L}_{\mathcal{I}\times\mathcal{J}}$ requires truncated additions only for direct successors and therefore allows for a decreased computational complexity.

The product is stored in auxiliary matrices $R_{t,s} \in \mathcal{R}(t, s, k)$ for $t \in \mathcal{T}_{\mathcal{I}}, s \in \mathcal{T}_{\mathcal{J}}$. Whenever a product $\mathbf{X}|_{t\times r}\mathbf{Y}|_{r\times s}$ for $r \in \mathcal{T}_{\mathcal{K}}$ with $(t, r) \in \mathcal{T}_{\mathcal{I}\times\mathcal{K}}$ and $(r, s) \in \mathcal{T}_{\mathcal{K}\times\mathcal{J}}$ can be represented as low-rank matrix (see (4.30) to (4.34)), it is added to $R_{t,s}$ followed by a truncation to rank k. Otherwise, we compute the product on the sub-blocks. To keep track of the already computed results and the results we have to handle on the sub-blocks, we introduce accumulators following [5].

Definition 6.1 (Accumulator)**.** *Let $b = (t, s) \in \mathcal{T}^{\mathrm{ind}}_{\mathcal{I}\times\mathcal{J}}$ and $k \in \mathbb{N}$. Then, an* accumulator *for the block b is a tuple*

$$(R_{t,s}, P_{t,s})$$

with $R_{t,s} \in \mathcal{R}(t, s, k)$ and

$$P_{t,s} \subseteq \{(\alpha, r, \mathbf{G}, \mathbf{H}) : \alpha \in \mathbb{R},\ r \in \mathcal{T}_{\mathcal{K}},\ \mathbf{G} \in \mathcal{H}(\mathcal{T}_{(t,r)}, k_1),\ \text{and } \mathbf{H} \in \mathcal{H}(\mathcal{T}_{(r,s)}, k_2)\},$$

where $\mathcal{T}_{(t,r)}$ and $\mathcal{T}_{(r,s)}$ are the sub block cluster trees of $\mathcal{T}_{\mathcal{I}\times\mathcal{K}}$ and $\mathcal{T}_{\mathcal{K}\times\mathcal{J}}$ with roots (t, r) and (r, s), respectively (cf. Definition 4.8). The set $P_{t,s}$ is called the set of pending products.

The $\mathcal{H}$-matrix multiplication with accumulated updates relies on three operations handling accumulators, summarized in Algorithms 6.1 to 6.3:

- `add_product` which adds a product $\alpha\mathbf{X}|_{t\times r}\mathbf{Y}_{r\times s}$ to an accumulator $(R_{t,s}, P_{t,s})$ for $(t,r) \in \mathcal{T}_{\mathcal{I}\times\mathcal{K}}$ and $(r,s) \in \mathcal{T}_{\mathcal{K}\times\mathcal{J}}$. If $\alpha\mathbf{X}|_{t\times r}\mathbf{Y}|_{r\times s}$ can be evaluated, i.e., if $(t,r) \in \mathcal{L}_{\mathcal{I}\times\mathcal{K}}$ or $(r,s) \in \mathcal{L}_{\mathcal{K}\times\mathcal{J}}$, then it is added to $R_{t,s}$ and the result is truncated. Otherwise, it is added to the set of pending products $P_{t,s}$.

- `split` which creates accumulators $(R_{t',s'}, P_{t',s'})$ for all $b' = (t',s') \in \operatorname{sons}(b)$, from an accumulator $(R_{t,s}, P_{t,s})$ for $b = (t,s) \in \mathcal{T}^{\mathrm{ind}}_{\mathcal{I}\times\mathcal{J}}$.

- `flush` which performs the summation of an accumulator $(R_{t,s}, P_{t,s})$ to the matrix $\mathbf{Z}|_{t\times s}$ for $(t,s) \in \mathcal{T}_{\mathcal{I}\times\mathcal{J}}$. If $P_{t,s} = \emptyset$, the matrix represented by the rank-k factors $R_{t,s} \in \mathcal{R}(t,s,k)$ is added to $\mathbf{Z}|_{t\times s}$. Otherwise, the accumulator is split and the sum is handled for all sons of $b = (t,s)$. If (t,s) was a leaf block $(t,s) \in \mathcal{L}_{\mathcal{I}\times\mathcal{J}}$, the result on the sub-blocks are merged into one large rank-k matrix with `merge_hmatrix` from Algorithm 6.4.

The update (6.1) is performed by initializing the accumulator $(R_{r_\mathcal{I},r_\mathcal{J}}, P_{r_\mathcal{I},r_\mathcal{J}})$ with $P_{r_\mathcal{I},r_\mathcal{J}} = \emptyset$ and $R_{r_\mathcal{I},r_\mathcal{J}} = (\mathbf{0}_{\mathcal{I}\times k}, \mathbf{0}_{\mathcal{J}\times k}) \in \mathcal{R}(\mathcal{I},\mathcal{J},k)$. Then the product $\mathbf{XY}$ is added to the accumulator with a call of `addproduct` followed by a call of `flush` with the updated accumulator and $\mathbf{Z}$.

Algorithm 6.1 Add a product to an accumulator (cf. [5, Fig. 5])

Input: Scaling factor α, clusters $t \in \mathcal{T}_\mathcal{I}$, $r \in \mathcal{T}_\mathcal{K}$, and $s \in \mathcal{T}_\mathcal{J}$, factors $\mathbf{X} \in \mathcal{H}(\mathcal{T}_{(t,r)}, k_1)$ and $\mathbf{Y} \in \mathcal{H}(\mathcal{T}_{(r,s)}, k_2)$, accumulator $(R_{t,s}, P_{t,s})$ with $R_{t,s} = (\mathbf{A}_R, \mathbf{B}_R) \in \mathcal{R}(t,s,k)$ to which the product $\mathbf{XY}$ is added.

function `add_product`(α, t, r, s, $\mathbf{X}$, $\mathbf{Y}$, $R_{t,s}$, $P_{t,s}$)
  if $(t,s) \in \mathcal{L}_{\mathcal{I}\times\mathcal{K}}$ or $(s,r) \in \mathcal{L}_{\mathcal{K}\times\mathcal{J}}$ **then**
    $\widetilde{k} \leftarrow \max\{k_1, k_2, n_{\mathrm{leaf}}\}$
    Determine $(\mathbf{A},\mathbf{B}) \in \mathcal{R}(t,s,k)$ with $\mathbf{XY} = \mathbf{AB}$ ▷ See (4.30) to (4.34)
    $\widetilde{\mathbf{A}} \leftarrow \begin{pmatrix}\mathbf{A}_R & \alpha\mathbf{A}\end{pmatrix}$, $\widetilde{\mathbf{B}} \leftarrow \begin{pmatrix}\mathbf{B}_R & \mathbf{B}\end{pmatrix}$
    $R_{t,s} \leftarrow \mathfrak{T}^{k+\widetilde{k}}_{\mathrm{thin},k}(\widetilde{\mathbf{A}}, \widetilde{\mathbf{B}})$ ▷ cf. 2.3.
  else
    $P_{t,s} \leftarrow P_{t,s} \cup \{(\alpha, r, \mathbf{X}, \mathbf{Y})\}$
  end if
end function

Theorem 6.2. *Let*

$$\widehat{\rho} := \max\{\operatorname{depth}(\mathcal{T}_\mathcal{I}), \operatorname{depth}(\mathcal{T}_\mathcal{J}), \operatorname{depth}(\mathcal{T}_\mathcal{K})\},$$

$$C_{\mathrm{sp}} := \max\{C_{\mathrm{sp}}(\mathcal{T}_{\mathcal{I}\times\mathcal{K}}), C_{\mathrm{sp}}(\mathcal{T}_{\mathcal{K}\times\mathcal{J}}), C_{\mathrm{sp}}(\mathcal{T}_{\mathcal{I}\times\mathcal{J}})\},$$

and $\widehat{k} := \max\{k_1, k_2, k_3, n_{\mathrm{leaf}}\}$. *Then the computational cost* W_{au} *of the* $\mathcal{H}$*-matrix update with*

$$\mathbf{Z} \leftarrow \mathbf{Z} \oplus \mathbf{X} \odot \mathbf{Y}$$

Algorithm 6.2 Splitting of an accumulator (cf. [5, Fig. 6])

Input: Non-leaf clusters $t \in \mathcal{T}_\mathcal{I} \setminus \mathcal{L}_\mathcal{I}$ and $s \in \mathcal{T}_\mathcal{J} \setminus \mathcal{L}_\mathcal{J}$, accumulator $(R_{t,s}, P_{t,s})$ with $R_{t,s} = (\mathbf{A}_R, \mathbf{B}_R) \in \mathcal{R}(t, s, k)$
Output: Accumulators $(R_{t',s'}, P_{t',s'})$ for all sons of t' of t and s' of s.

```
function split(t, s, R_{t,s}, P_{t,s})
    for t' ∈ sons(t), s' ∈ sons(s) do
        R_{t',s'} ← (A_R|_{t'×k}, B_R|_{s'×k})
        P_{t',s'} ← ∅
        for (α, r, X, Y) ∈ P_{t,s} do
            for r' ∈ sons(r) do
                add_product(t', r', s', X|_{t'×r'}, Y|_{r'×s'}, R_{t',s'}, P_{t',s'})   ▷ cf. Algorithm 6.1.
            end for
        end for
    end for
    return ((R_{t',s'}, P_{t',s'}))_{t',s'}
end function
```

Algorithm 6.3 Flush an accumulator into an $\mathcal{H}$-matrix

Input: Clusters $t \in \mathcal{T}_\mathcal{I}$ and $s \in \mathcal{T}_\mathcal{J}$, accumulator $(R_{t,s}, P_{t,s})$ with $R_{t,s} = (\mathbf{A}_R, \mathbf{B}_R) \in \mathcal{R}(t, s, k')$, target matrix $\mathbf{Z} \in \mathcal{H}(\mathcal{T}_{(t,s)}, k')$ the accumulator is added to, target rank k.

```
function flush(t, s, R_{t,s}, P_{t,s}, Z, k)
    if P_{t,s} = ∅ then
        add_hmatrix_rkmatrix(1, t, s, A_R, B_R, Z, k)                          ▷ cf. Algorithm 4.8
    else
        (R_{t',s'}, P_{t',s'})_{t',s'} ← split(t, s, R_{t,s}, P_{t,s})         ▷ cf. Algorithm 6.2
        if (t, s) ∈ T_{I×J} \ L_{I×J} then
            for t' ∈ sons(t), s' ∈ sons(s) do
                flush(t', s', R_{t,s}, P_{t,s}, Z|_{t'×s'}, k)
            end for
        else
            Create Z̃ ∈ H(T^ind_{(t,s)}, k') with Z = Z̃
            for t' ∈ sons(t), s' ∈ sons(s) do          ▷ (t, s) ∉ L(T^ind_{I×J}) since P_{t,s} ≠ ∅
                flush(t', s', R_{t',s'}, P_{t',s'}, Z̃)
            end for
            if (t, s) ∈ L^-_{I×J} then
                Z|_b ← Z̃|_b
            else                                                               ▷ (t, s) ∈ L^+_{I×J}
                Z ← merge_hmatrix(t, s, Z̃)                                     ▷ cf. Algorithm 6.4
            end if
        end if
    end if
end function
```

Algorithm 6.4 Merge a hierarchical matrix into a rank-k matrix

Input: Clusters $t \in \mathcal{T}_{\mathcal{I}}$ and $s \in \mathcal{T}_{\mathcal{J}}$, $\mathcal{H}$-matrix $\mathbf{Z} \in \mathcal{H}(\mathcal{T}^{\mathrm{ind}}_{(t,s)}, k')$, target rank k
Output: Rank-k factors $(\mathbf{A}, \mathbf{B}) \in \mathcal{R}(t, s, k)$ representing $\mathbf{Z}$ merged into one low-rank matrix truncated to rank k

```
function merge_hmatrix(t, s, Z)
    if then(t, s) ∈ L(T^ind_{I×J})
        Determine A|_Z, B_Z ∈ R(t, s, k') with Z = A_Z B_Z^T
        (A, B) ← T^{k'}_{thin,k}                          ▷ cf. Remark 2.3
    else
        A ← 0_{t×k},   B ← 0_{s×k}
        for t' ∈ sons(t), s' ∈ sons(s) do
            (A', B') ← merge_hmatrix(t', s', Z|_{t'×s'}, k)
            Ã ← (A  A'|^{t×k}),   B̃ ← (B  B'|^{t×k})
            (A, B) ← T^{k+k}_{thin,k}                     ▷ cf. Remark 2.3
        end for
    end if
end function
```

with accumulated updates can be estimated by

$$W_{\mathrm{au}}(\mathcal{T}_{\mathcal{I}\times\mathcal{K}}, \mathcal{T}_{\mathcal{K}\times\mathcal{J}}, \mathcal{T}_{\mathcal{I}\times\mathcal{J}}, k_1, k_2, k_3) \leq C_{\mathrm{au}} C_{\mathrm{sp}}^2 \widehat{k}^2 (\widehat{\rho}+1)^2 (|\mathcal{I}| + |\mathcal{K}| + |\mathcal{J}|),$$

where $C_{\mathrm{au}} > 0$ is a constant depending on the block cluster trees $\mathcal{T}_{\mathcal{I}\times\mathcal{K}}$, $\mathcal{T}_{\mathcal{K}\times\mathcal{J}}$, and $\mathcal{T}_{\mathcal{I}\times\mathcal{J}}$.

Proof. See [5, Theorem 6]. □

6.1.2 Numerical experiments

In the following, we will present and evaluate the results of numerical experiments comparing the use of the standard $\mathcal{H}$-matrix arithmetic and the $\mathcal{H}$-matrix arithmetic with accumulated updates for the set-up of the $\mathcal{H}$-LU factorizations $\check{\mathbf{F}} \approx \check{\mathbf{L}}_{\mathbf{F}} \check{\mathbf{U}}_{\mathbf{F}}$ and $\mathbf{S}_{\mathcal{H}} \approx \mathbf{L}_{\mathbf{S}} \mathbf{U}_{\mathbf{S}}$ (cf. Remark 5.3). The model problem and the test setting is the same as described in Section 5.4.1. We note that the H2Lib library already offers an implementation of the $\mathcal{H}$-matrix arithmetic with accumulated updates, which we used for our tests. The cluster trees $\mathcal{T}_{\mathcal{I}}$ and $\mathcal{T}_{\mathcal{J}}$ were generated with the coupled clustering introduced in Section 5.2, i.e., $\mathcal{T}_{\mathcal{J}}$ was generated with geometric bisection and $\mathcal{T}_{\mathcal{I}}$ was generated coupled with $\mathcal{T}_{\mathcal{J}}$ (see, e.g., Algorithm 5.1). The block cluster trees $\mathcal{T}_{\mathcal{I}\times\mathcal{I}}$, $\mathcal{T}_{\mathcal{J}\times\mathcal{I}}$, and $\mathcal{T}_{\mathcal{J}\times\mathcal{J}}$ were generated with the admissibility conditions (cf. Definitions 4.30, 4.34 and 5.8)

$$\mathrm{adm}_{\mathcal{I}\times\mathcal{I}} = \mathrm{adm}^{\mathrm{DD}}_{\mathrm{weak}}, \quad \mathrm{adm}_{\mathcal{J}\times\mathcal{I}} = \mathrm{adm}^{\mathrm{c}}_{\mathrm{weak}}, \quad \text{and } \mathrm{adm}_{\mathcal{J}\times\mathcal{J}} = \mathrm{adm}_{\eta} \quad (\eta = 16),$$

respectively.

As before in Section 5.4, we denote the truncation accuracies by $\delta^{\mathcal{H}}_{\mathrm{vel}}$, $\delta^{\mathcal{H}}_{\mathrm{mul}}$, and $\delta^{\mathcal{H}}_{\mathrm{Schur}}$, respectively (cf. (5.19)).

Remark 6.3. *From Remark 5.3, we recall the 5 steps of the preconditioner set-up:*

Step ① *Compute an $\mathcal{H}$-LU factorization $\check{\mathbf{L}}_{\mathbf{F}} \check{\mathbf{U}}_{\mathbf{F}} \approx \check{\mathbf{F}}$.*

Step ② *Compute* $\mathbf{V}^k \approx \mathbf{B}_{\mathcal{H}}^k \check{\mathbf{U}}^{-1}$ *for* $k = 1, 2, 3$.

Step ③ *Compute* $\mathbf{W}^k \approx \check{\mathbf{L}}^{-1} (\mathbf{B}_{\mathcal{H}}^k)^{\mathsf{T}}$ *for* $k = 1, 2, 3$.

Step ④ *Compute* $\mathbf{S}_{\mathcal{H}} = \bigoplus_{k=1}^{3} (\mathbf{V}^k \odot \mathbf{W}^k)$.

Step ⑤ *Compute an* $\mathcal{H}$*-LU factorization* $\mathbf{L}_{\mathbf{S}}\mathbf{U}_{\mathbf{S}} \approx \mathbf{S}_{\mathcal{H}}$.

First, we compare both $\mathcal{H}$-matrix arithmetic variants for a varying system size and fix $\delta_{\text{vel}}^{\mathcal{H}} = \delta_{\text{mul}}^{\mathcal{H}} = \delta_{\text{Schur}}^{\mathcal{H}} = 10^{-1}$. The results, i.e., the computation times per degree of freedom, are depicted in Figure 6.1.

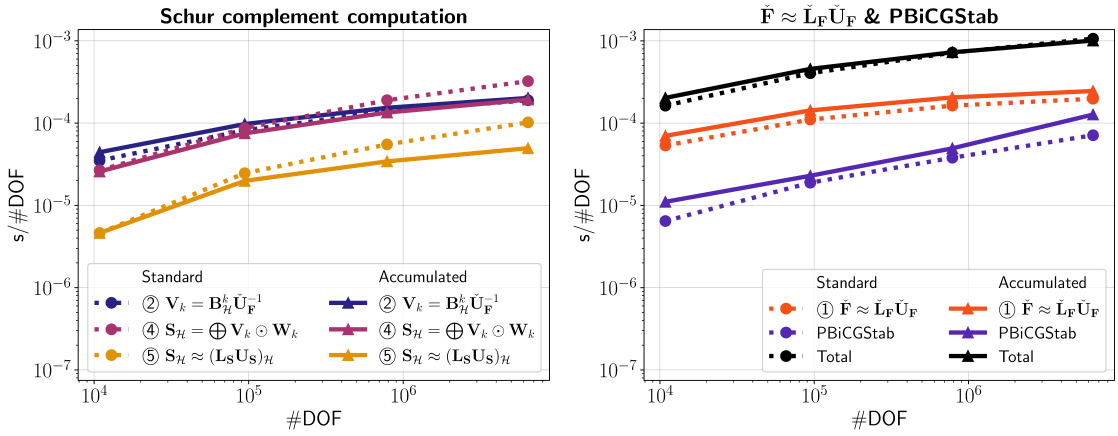

Figure 6.1: Computation times per degree of freedom for the preconditioner set-up and the solve with the PBiCGStab method with the standard $\mathcal{H}$-matrix arithmetic (dotted lines, circles) and the $\mathcal{H}$-matrix arithmetic with accumulated updates (solid lines, triangles) for a varying system size.

There, we can observe that the $\mathcal{H}$-matrix arithmetic with accumulated updates yields a speed-up for the $\mathcal{H}$-matrix update required for the computation of an (approximate) Schur complement $\mathbf{S}_{\mathcal{H}}$ (step ④ of the preconditioner set-up) and for the computation of an $\mathcal{H}$-LU factorization of $\mathbf{S}_{\mathcal{H}}$ (step ⑤ of the preconditioner set-up). The $\mathcal{H}$-LU factorization of $\check{\mathbf{F}}$ (step ①) and the triangular solves (step ② and ③) are finished faster with the standard $\mathcal{H}$-matrix arithmetic. Another thing to note here is that the approximations computed with the $\mathcal{H}$-matrix arithmetic with accumulated updates seem to be less accurate, since the solving the system requires more time due to a larger number of iteration steps.

Depicted in Figure 6.2 are the computation times per degree of freedom for a mix of the $\mathcal{H}$-matrix arithmetic with accumulated updates and the standard $\mathcal{H}$-matrix arithmetic, as well as a set-up only with the standard $\mathcal{H}$-matrix arithmetic. Table 6.2 presents the absolute times.

For the mixed preconditioner set-up, we used the standard $\mathcal{H}$-matrix arithmetic for the $\mathcal{H}$-LU factorization of $\check{\mathbf{F}}$ (step ①) as well as the triangular solves in steps ② and ③ of the preconditioner set-up. For the other operations (steps ④ and ⑤), we used the $\mathcal{H}$-matrix arithmetic with accumulated updates.

In Figure 6.2 and Table 6.2 we observed (almost) no difference between the two approaches for smaller system sizes. For larger system sizes, on the other hand, we can observe a small speed-up of about 10% – 20% in favor of the mixed approach.

Standard $\mathcal{H}$-matrix arithmetic

#DOF	①	②	④	⑤	PBiCGStab (Iter)	Total
10,853	0.6s	0.4s	0.3s	0.1s	0.07s (9)	1.8s
94,285	10.4s	7.8s	8.2s	2.3s	1.8s (13)	38.2s
786,077	128.2s	105.8s	149.1s	43.3s	29.9s (21)	565.4s
6,419,773	1,271.0s	1,149.2s	2,074.7s	654.7s	457.6s (34)	6,823.7s

$\mathcal{H}$-matrix arithmetic with accumulated updates

#DOF	①	②	④	⑤	PBiCGStab (Iter)	Total
10,853	0.8s	0.5s	0.3s	0.1s	0.1s (15)	2.2s
94,285	13.5s	9.3s	7.1s	1.9s	2.2s (18)	43.0s
786,077	161.5s	117.6s	105.7s	26.9s	38.8s (32)	571.2s
6,419,773	1,577.3s	1,231.0s	1,240.1s	318.0s	816.3s (71)	6,480.5s

Table 6.1: Absolute computation times for the preconditioner set-up and the solve with the BiCGStab method with the standard $\mathcal{H}$-matrix arithmetic (top) and with the $\mathcal{H}$-matrix arithmetic with accumulated updates (bottom). The time required to solve the system is supplemented by the number of iteration steps required to achieve a residual error of 10^{-12}.

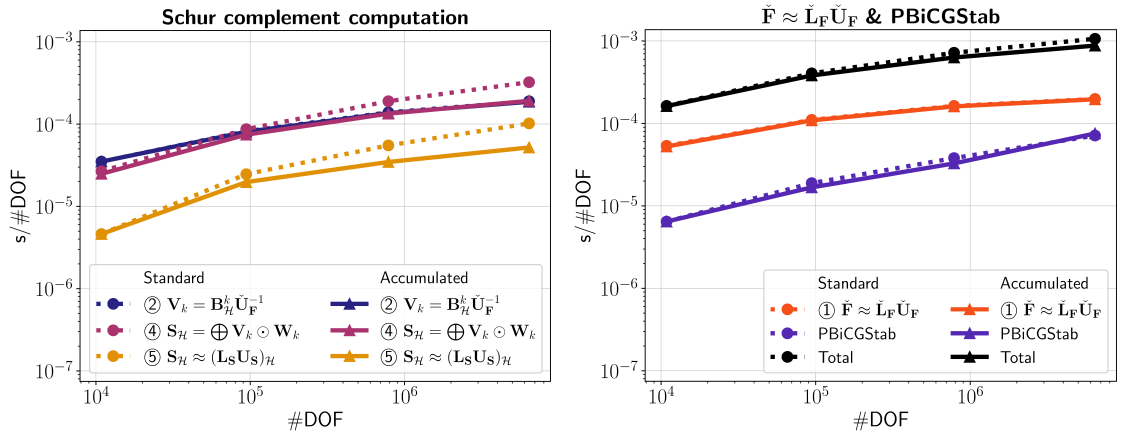

Figure 6.2: Computation times per degree of freedom for the preconditioner set-up and the solve with the PBiCGStab method with the standard $\mathcal{H}$-matrix arithmetic (dotted lines, circles) and the mixed approach (solid lines, triangles) for a varying system size.

Standard $\mathcal{H}$-matrix arithmetic

#DOF	①	②	④	⑤	PBiCGStab (Iter)	Total
10,853	0.6s	0.4s	0.3s	0.1s	0.07s (9)	1.8s
94,285	10.4s	7.8s	8.2s	2.3s	1.8s (13)	38.2s
786,077	128.2s	105.8s	149.1s	43.3s	29.9s (21)	565.4s
6,419,773	1,271.0s	1,149.2s	2,074.7s	654.7s	457.6s (34)	6,823.7s

Mixed $\mathcal{H}$-matrix arithmetic

#DOF	①	②	④	⑤	PBiCGStab (Iter)	Total
10,853	0.6s	0.4s	0.3s	0.1s	0.07s (10)	1.8s
94,285	10.3s	7.7s	7.0s	1.9s	1.6s (14)	36.0s
786,077	126.4s	104.0s	105.3s	27.2s	25.7s (21)	495.6s
6,419,773	1,259.8s	1,134.2s	1,230.7s	333.6s	485.1s (38)	5,642.2s

Table 6.2: Absolute computation times for the preconditioner set-up and the solve with the BiCGStab method with the standard $\mathcal{H}$-matrix arithmetic (top) and with the mixed approach (bottom). The time required to solve the system is supplemented by the number of iteration steps required to achieve a residual error of 10^{-12}.

Conclusion. In Figure 6.1 we observed that the arithmetics with accumulated updates performs slightly better than the standard $\mathcal{H}$-matrix arithmetic for the multiplication in step ④ and for the $\mathcal{H}$-LU factorization of the (approximate) Schur complement $\mathbf{S}_\mathcal{H}$ in step ⑤ of the preconditioner set-up. With a larger system size this advantage grows. However, we also observed that the $\mathcal{H}$-matrix arithmetic with accumulated updates requires more time for the computation of an $\mathcal{H}$-LU factorization of $\check{\mathbf{F}}$. Additionally, the whole preconditioner seems to be less accurate resulting in more iteration steps required to achieve the desired accuracy. Considering the total computation time it seems to be neither beneficial nor adverse to use the $\mathcal{H}$-matrix arithmetic with accumulated updates instead of the standard $\mathcal{H}$-matrix arithmetic in this case. But we note, that one may consider the usage of a mix of both for a small speed-up. In our tests, we observed a small speed-up of about 10%–20% for a larger system size.

Next, we consider a fixed (total) system size of $3n + M = 3 \cdot 250,047 + 35936 = 786,077$ and varying truncation accuracies. First, we vary $\delta^\mathcal{H}_{\text{vel}} \in \{10^{-1}, \dots, 10^{-4}\}$ and fix $\delta^\mathcal{H}_{\text{mul}} = \delta^\mathcal{H}_{\text{Schur}} = 10^{-1}$. The computation times per degree of freedom are depicted in Figure 6.3. Here, we can observe that varying $\delta^\mathcal{H}_{\text{vel}}$ does not have any effect on the

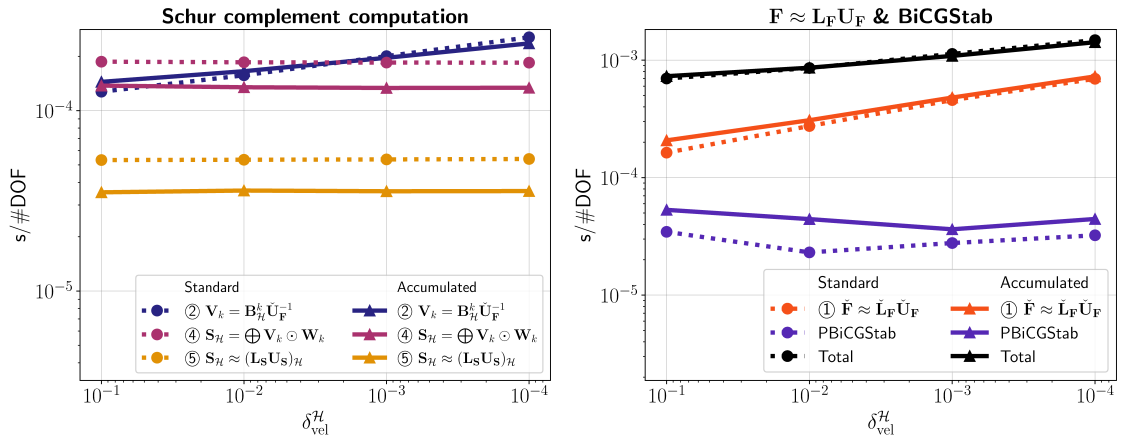

Figure 6.3: Computation times per degree of freedom for the preconditioner set-up and the solve with the PBiCGStab method with the standard $\mathcal{H}$-matrix arithmetic (dotted lines, circles) and the $\mathcal{H}$-matrix arithmetic with accumulated updates (solid lines, triangles) for a varying truncation accuracy $\delta^\mathcal{H}_{\text{vel}} \in \{10^{-1}, \dots, 10^{-4}\}$ and fixed $\delta^\mathcal{H}_{\text{mul}} = \delta^\mathcal{H}_{\text{Schur}} = 10^{-1}$.

multiplication in step ④ and the computation of the $\mathcal{H}$-LU factorization of $\mathbf{S}_\mathcal{H}$ in step ⑤ of the preconditioner set-up. This is to be expected since the computation time of these operations depends on $\delta^\mathcal{H}_{\text{mul}}$ and $\delta^\mathcal{H}_{\text{Schur}}$, respectively. For all operations involving the matrix $\check{\mathbf{F}}$ or its (approximate) $\mathcal{H}$-LU factorization $\check{\mathbf{F}} \approx \check{\mathbf{L}}_\mathbf{F}\mathbf{U}_\mathbf{F}$, i.e., steps ① – ③ of the preconditioner set-up, the computation times increase with a decreasing $\delta^\mathcal{H}_{\text{vel}}$. We also note that the time required to solve the system slightly decreases with both $\mathcal{H}$-matrix arithmetic variants. But the total computation time is still dominated by the preconditioner set-up.

Next, we consider a varying truncation accuracy $\delta^\mathcal{H}_{\text{mul}} \in \{10^{-1}, \dots, 10^{-4}\}$ for the explicit computation of the (approximate) Schur complement, i.e., steps ②–④ of the preconditioner set-up. The other truncation accuracies are fixed with $\delta^\mathcal{H}_{\text{vel}} = \delta^\mathcal{H}_{\text{Schur}} = 10^{-4}$. The computation times per degree of freedom are depicted in Figure 6.4. In this figure, we can observe that the computation times for the multiplications in step ④ increase with a

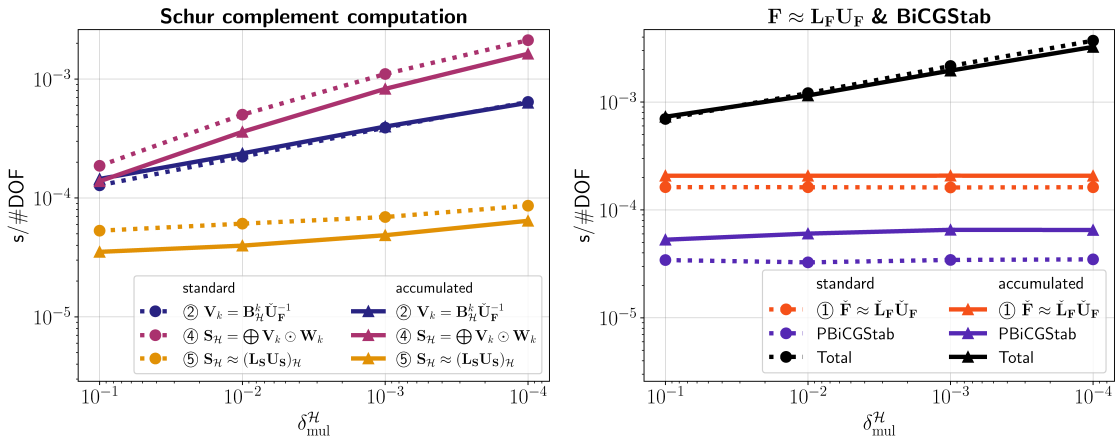

Figure 6.4: Computation times per degree of freedom for the preconditioner set-up and the solve with the PBiCGStab method with the standard $\mathcal{H}$-matrix arithmetic (dotted lines, circles) and the $\mathcal{H}$-matrix arithmetic with accumulated updates (solid lines, triangles) for a varying truncation accuracy $\delta^{\mathcal{H}}_{\text{mul}} \in \{10^{-1}, \ldots, 10^{-4}\}$ and fixed $\delta^{\mathcal{H}}_{\text{vel}} = \delta^{\mathcal{H}}_{\text{Schur}} = 10^{-4}$.

decreasing $\delta^{\mathcal{H}}_{\text{mul}}$ for both variants of the $\mathcal{H}$-matrix arithmetic. The difference between both variants of the $\mathcal{H}$-matrix arithmetic seems not to vary for all operations.

Conclusion. The observations described above suggest that the effectiveness of the $\mathcal{H}$-matrix arithmetic with accumulated updates does not depend on the truncation accuracies but rather on the block structure of the matrices. The nested-dissection like structure of the $\mathcal{H}$-matrix representation of $\check{\mathbf{F}}$ seems to be disadvantageous for the $\mathcal{H}$-matrix arithmetic with accumulated updates.

6.2 $\mathcal{H}$-matrix multiplication with sum-expressions

In this section, we will discuss the $\mathcal{H}$-matrix multiplication with sum-expressions. This approach was first introduced in [11] and is based on the idea of utilizing matrix-vector operations for the truncation of the occurring sums of matrices.

6.2.1 The algorithm

The idea of the $\mathcal{H}$-matrix arithmetic with accumulated updates was to reduce the number of rank truncations for multiple merges of low-rank matrices to the same larger block by assembling the result of the product $\mathbf{X} \odot \mathbf{Y}$ block-wise in auxiliary matrices $\mathbf{R}_{t,s}$. This idea stems from arithmetic operations of $\mathcal{H}^2$-matrices. While the motivation for the $\mathcal{H}$-matrix arithmetic with sum-expressions described in [11] was different, both variants are based on similar ideas (cf. [11, §3.2.4]).

The motivation for the $\mathcal{H}$-matrix arithmetic with sum-expressions was a reordering of the operations to achieve an easy applicability of matrix-vector operation based truncation methods. This is achieved by assembling all intermediate low-rank matrices in a set or list instead of adding them directly to the result or an auxiliary matrix. They are stored until a final (single) low-rank truncation is applied for the current block.

When the rank truncation is based on matrix-vector operations, we do not have to evaluate the stored sum of (intermediate) low-rank matrices but can use vector updates applied summand by summand.

In the following, we will use a structure similar to the accumulator from Definition 6.1 to keep track of all matrices to handle. The low-rank matrix of an accumulator is replaced by a so-called sum-expression of low-rank matrices containing all (intermediate) low-rank matrices to handle. Additionally, we keep track of $\mathcal{H}$-matrix products to handle with a set similar to the set of pending products of an accumulator. But in this context, we call it sum-expression of $\mathcal{H}$-matrices.

Definition 6.4 (Sum-expression)**.** *Let $b = (t, s) \in \mathcal{T}_{\mathcal{I}\times\mathcal{J}}$, and let $\mathcal{K}_{\mathcal{R}} \subseteq \mathcal{T}_{\mathcal{K}}$. Then, we call*

$$S_{\mathcal{R}}(t, s) = \{(r, \mathbf{A}_r, \mathbf{B}_r) : r \in \mathcal{K}_{\mathcal{R}} \text{ and } (\mathbf{A}_r, \mathbf{B}_r) \in \mathcal{R}(t, s, k) \text{ for some } k \leq |t|, |s|\}$$

sum-expression of low-rank matrices *for the block (t, s). Similarly, we call*

$$S_{\mathcal{H}}(t, s) \subseteq \{(\alpha_r, r, \mathbf{X}|_{t\times r}, \mathbf{Y}|_{r\times s}) : r \in \mathcal{TJ}, \mathbf{X} \in \mathcal{H}(\mathcal{T}_{\mathcal{I}\times\mathcal{K}}, k_1), \mathbf{Y} \in \mathcal{H}(\mathcal{T}_{\mathcal{K}\times\mathcal{J}}, k_2)\}$$

sum-expression of hierarchical matrices *for the block (t, s). Furthermore, we call the tuple*

$$(S_{\mathcal{R}}(t, s), S_{\mathcal{H}}(t, s))$$

sum-expression *for the block (t, s).*

Now, we use these sum-expressions to express the blocks of the product $\alpha\mathbf{XY}$ for $\alpha \in \mathbb{R}$, $\mathbf{X} \in \mathcal{H}(\mathcal{T}_{\mathcal{I}\times\mathcal{K}}, k_1)$, and $\mathbf{Y} \in \mathcal{H}(\mathcal{T}_{\mathcal{K}\times\mathcal{J}}, k_2)$. We start with $r_{\mathcal{I}}$ and $r_{\mathcal{J}}$ and set

$$S_{\mathcal{R}}(r_{\mathcal{I}}, r_{\mathcal{J}}) := \emptyset$$

and

$$S_{\mathcal{H}}(r_{\mathcal{I}}, r_{\mathcal{J}}) := \{(\alpha, r_{\mathcal{K}}, \mathbf{X}, \mathbf{Y})\}.$$

Now, if $\operatorname{root}(\mathcal{T}_{\mathcal{I}\times\mathcal{J}}) =: r_{\mathcal{I}\times\mathcal{J}} = (r_{\mathcal{I}}, r_{\mathcal{J}}) \notin \mathcal{L}_{\mathcal{I}\times\mathcal{J}}$ and $r_{\mathcal{I}\times\mathcal{J}} \notin \mathcal{L}(\mathcal{T}^{\text{ind}}_{\mathcal{I}\times\mathcal{J}})$, we can express the product on the sub-blocks as (cf. (4.35))

$$(\mathbf{XY})|_{t\times s} = \sum_{r\in\operatorname{sons}(r_{\mathcal{K}})} \mathbf{X}|_{t\times r}\mathbf{Y}_{r\times s}.$$

Then one of the following cases applies to all $r \in \operatorname{sons}(r_{\mathcal{K}})$:

1. If $(t, r) \in \mathcal{L}^+_{\mathcal{I}\times\mathcal{K}}$ or $(r, s) \in \mathcal{L}_{\mathcal{K}\times\mathcal{J}}$, then we can compute $(\mathbf{A}_r, \mathbf{B}_r) \in \mathcal{R}(t, s, k_1)$ or $(\mathbf{A}_r, \mathbf{B}_r) \in \mathcal{R}(t, s, k_2)$ with $\mathbf{AB} = \mathbf{X}|_{t\times r}\mathbf{Y}|_{r\times s}$ (cf. (4.30) and (4.31)). Hence, we add $(r, \mathbf{A}_r, \mathbf{B}_r)$ to the sum-expression of low-rank matrices for (t, s).

2. If $(t, r) \in \mathcal{L}^-_{\mathcal{I}\times\mathcal{K}}$ or $(r, s) \in \mathcal{L}^-_{\mathcal{K}\times\mathcal{J}}$, we could compute $(\mathbf{A}_r, \mathbf{B}_r) \in \mathcal{R}(t, s, n_{\text{leaf}})$ with $\mathbf{A}_r\mathbf{B}_r = \mathbf{X}|_{t\times r}\mathbf{Y}|_{r\times s}$. But instead, we add $(1, r, \mathbf{X}|_{t\times r}, \mathbf{Y}|_{r\times s})$ to the sum-expression of hierarchical matrices for (t, s).

3. If $(t, r) \notin \mathcal{L}_{\mathcal{I}\times\mathcal{K}}$ and $(r, s) \notin \mathcal{L}_{\mathcal{K}\times\mathcal{J}}$, the product has to be computed on further sub-blocks. Hence, we add $(1, r, \mathbf{X}|_{t\times r}, \mathbf{Y}|_{r\times s})$ to the sum-expression of hierarchical matrices for (t, s).

Therefore, we set

$$S_{\mathcal{R}}(t,s) := \{(r, \mathbf{A}_r, \mathbf{B}_r) : r \in \operatorname{sons}(r_{\mathcal{K}}), (t,r) \in \mathcal{L}^+_{\mathcal{I}\times\mathcal{K}} \text{ or } (r,s) \in \mathcal{L}^+_{\mathcal{K}\times\mathcal{J}}\}$$

and

$$\begin{aligned} S_{\mathcal{H}}(t,s) := \{(\alpha, r, \mathbf{X}|_{t\times r}, \mathbf{X}|_{r\times s}) :&(t,r) \in \mathcal{T}_{\mathcal{I}\times\mathcal{K}} \setminus \mathcal{L}^+_{\mathcal{I}\times\mathcal{K}} \text{ and} \\ &(r,s) \in \mathcal{T}_{\mathcal{K}\times\mathcal{J}} \setminus \mathcal{L}^+_{\mathcal{K}\times\mathcal{J}}, r \in \operatorname{sons}(r_{\mathcal{K}})\}, \end{aligned}$$

so that

$$(\alpha\mathbf{XY})|_{t\times s} = \sum_{(r,\mathbf{A},\mathbf{B})\in S_{\mathcal{R}}(t,s)} \mathbf{AB}^{\mathsf{T}} + \sum_{(\beta,r,\mathbf{V},\mathbf{W})\in S_{\mathcal{H}}(t,s)} \beta\mathbf{VW} \tag{6.3}$$

holds for all $(t,s) \in \operatorname{sons}(r_{\mathcal{I}\times\mathcal{J}})$. On the next level, we also have to handle all tuples contained in the corresponding sum-expressions of low-rank matrices. This can be done by simply restricting the rows of the given factors to the subclusters. The restriction of sum-expressions to sub-blocks as described above is summarized in Algorithm 6.5. The multiplication is then performed by restricting sum-expressions of non-leaf blocks to the sub-blocks, and evaluating the sum-expressions on the leaf blocks with a truncation if needed. This is summarized in Algorithm 6.6 The routine `addtrunc_sumexpression` used

Algorithm 6.5 Construct sum-expressions for sub-blocks

Input: Block $b = (t,s) \in \mathcal{T}_{\mathcal{I}\times\mathcal{J}} \setminus \mathcal{L}_{\mathcal{I}\times\mathcal{J}}$, sum-expression $S(t,s) := (S_{\mathcal{R}}(t,s), S_{\mathcal{H}}(t,s))$ for the block b, successor $b' = (t',s') \in \operatorname{sons}(b)$.

Output: Sum-expression $S(t',s') = (S_{\mathcal{R}}(t',s'), S_{\mathcal{H}}(t',s'))$ for the block b', constructed from the sum-expression $S(t,s)$.

1: **function** `restrict`(b, b', $S(t,s)$)
2: $S_{\mathcal{R}}(t',s') \leftarrow \{(r, \mathbf{A}|_{t'\times k}, \mathbf{B}|_{s'\times k}) : (r,\mathbf{A},\mathbf{B}) \in S_{\mathcal{R}}(t,s), k \in \mathbb{N} \text{ s.t.} (\mathbf{A},\mathbf{B}) \in \mathcal{R}(t,s,k)\}$
3: $S_{\mathcal{H}}(t',s') \leftarrow \emptyset$
4: **for** $(\alpha, r, \mathbf{V}, \mathbf{W}) \in S_{\mathcal{H}}(t,s)$ **do**
5: **for** $r' \in \operatorname{sons}(r)$ **do**
6: **if** $(t',r') \in \mathcal{L}^+_{\mathcal{I}\times\mathcal{K}}$ or $(r',s') \in \mathcal{L}^+_{\mathcal{K}\times\mathcal{J}}$ **then**
7: Determine $(\mathbf{A}_{r'}, \mathbf{B}_{r'}) \in \mathcal{R}(t,s,k)$ with $\mathbf{V}|_{t'\times r'}\mathbf{W}|_{r'\times s'} = \mathbf{A}_{r'}\mathbf{B}_{r'} (k \in \{k_1, k_2\})$
8: $S_{\mathcal{R}}(t',s') \leftarrow S_{\mathcal{R}}(t',s') \cup (r', \mathbf{A}_{r'}, \mathbf{B}_{r'})$
9: **else**
10: $S_{\mathcal{H}}(t',s') \leftarrow S_{\mathcal{H}}(t',s') \cup (\alpha, r', \mathbf{V}|_{t'\times r'}, \mathbf{W}|_{r'\times s'})$
11: **end if**
12: **end for**
13: **end for**
14: $S(t',s') \leftarrow (S_{\mathcal{R}}(t',s'), S_{\mathcal{H}}(t',s'))$
15: **return** $S(t',s')$
16: **end function**

in line 10 of Algorithm 6.6 adds a sum-expression to a low-rank matrix, and truncates the result to a target rank. This can be done, e.g., by:

Algorithm 6.6 $\mathcal{H}$-matrix update with sum-expressions

Input: Block $b = (t, s) \in \mathcal{T}_{\mathcal{I}\times\mathcal{J}}$, sum-expression $S(t, s) = (S_{\mathcal{R}}(t, s), S_{\mathcal{H}}(t, s))$ representing the product $\mathbf{XY}|_{t\times s}$ for $\mathbf{X} \in \mathcal{H}(\mathcal{T}_{\mathcal{I}\times\mathcal{K}}, k_1)$ and $\mathbf{Y} \in \mathcal{H}(\mathcal{T}_{\mathcal{K}\times\mathcal{J}}, k_2)$, target $\mathcal{H}$-matrix $\mathbf{Z} \in \mathcal{H}(\mathcal{T}_b, k_3)$ that is overwritten by $\mathbf{Z} \oplus (\mathbf{X} \odot \mathbf{Y})|_{t\times s}$, target rank $k \in \mathbb{N}$

function `addmul_hmatrix_sumexpression`(b, $S(t, s)$, $\mathbf{Z}$, k)
  if $(t, s) \notin \mathcal{L}_{\mathcal{I}\times\mathcal{J}}$ **then**
    for $b' = (t', s') \in \operatorname{sons}(b)$ **do**
      $S(t', s') \leftarrow$ `restrict`(b, b', $S(t, s)$)
      `addmul_hmatrix_sumexpression`(b', $S(t', s')$, $\mathbf{Z}|_{t'\times s'}$, k)
    end for
  else
    if $(t, s) \in \mathcal{L}^+_{\mathcal{I}\times\mathcal{J}}$ **then**
      Determine $(\mathbf{A}_b, \mathbf{B}_b) \in \mathcal{R}(t, s, k_3)$ with $\mathbf{Z} = \mathbf{A}_b\mathbf{B}_b^{\mathsf{T}}$ ▷ see Definition 4.11
      $(\mathbf{A}, \mathbf{B}) \leftarrow$ `addtrunc_sumexpression`($S(t, s)$, $\mathbf{A}_b$, $\mathbf{B}_b$)
      $\mathbf{Z} \leftarrow \mathbf{AB}^{\mathsf{T}}$
    else ▷ $(t, s) \in \mathcal{L}^-_{\mathcal{I}\times\mathcal{J}}$
      $$\mathbf{Z} \leftarrow \mathbf{Z} + \sum_{(r,\mathbf{A},\mathbf{B})\in S_{\mathcal{R}}(t,s)} \mathbf{AB}^{\mathsf{T}} + \sum_{(\alpha,r,\mathbf{V},\mathbf{W})\in S_{\mathcal{H}}(t,s)} \alpha\mathbf{VW}$$
    end if
  end if
end function

1. Computing the result explicitly and using a single truncation via SVD (cf. Remark 2.2). In this case, the multiplication with sum-expressions is equivalent to the original multiplication with single truncations via SVD (see [11, §3.2.2]) and the corresponding result from Theorem 4.44 can be applied.

2. Computing the result with intermediate truncations via SVD, i.e., pairwise truncation of the sums. In this case, the multiplication with sum-expressions is equivalent to the original multiplication with pairwise truncations (see [11, §3.2.3]) and the corresponding result from Theorem 4.44 can be applied.

3. Computing the result with approximation techniques utilizing matrix-vector operations such as the randomized low-rank approximation from Section 2.1.3 or the low-rank approximation via Lanczos bidiagonalization from Section 2.1.2.

In the last case, we estimate the computational complexity with the following result from [11].

Theorem 6.5. *Let* $\mathbf{X} \in \mathcal{H}(\mathcal{T}_{\mathcal{I}\times\mathcal{K}}, k_1)$ *and* $\mathbf{Y} \in \mathcal{H}(\mathcal{T}_{\mathcal{K}\times\mathcal{J}}, k_2)$. *Let* $\widehat{k} := \max\{k_1, k_2, n_{\text{leaf}}\}$ *and*

$$\widehat{\rho} = \max\{\operatorname{depth}(\mathcal{T}_{\mathcal{I}}), \operatorname{depth}(\mathcal{T}_{\mathcal{K}}), \operatorname{depth}(\mathcal{T}_{\mathcal{J}})\}$$

where n_{leaf} *is a leafsize bound for* $\mathcal{T}_{\mathcal{I}}$, $\mathcal{T}_{\mathcal{K}}$, *and* $\mathcal{T}_{\mathcal{J}}$ *(cf. Definition 4.13). Let* $\mathfrak{T}(\mathbf{M}, k)$ *be a truncation method which computes a rank-*k *approximation* $\mathbf{A}_{\mathbf{M}}\mathbf{B}_{\mathbf{M}} \in \mathcal{R}(m, n, k)$ *for a matrix* $\mathbf{M} \in \mathbb{R}^{m\times n}$ *with* ℓk *matrix-vector operations and* $N_{\mathfrak{T}}(k)$ *additional operations. If*

this truncation method is used for the routine `addtrunc_sumexpression`*, then the computational complexity* W_{se} *for the multiplication of* $\mathbf{X}$ *and* $\mathbf{Y}$ *with sum-expressions can be estimated by*

$$W_{\mathrm{se}}(\mathcal{T}_{\mathcal{I}\times\mathcal{K}}, \mathcal{T}_{\mathcal{K}\times\mathcal{J}}, \mathcal{T}_{\mathcal{I}\times\mathcal{J}}, k_1, k_2, k) \leq 25C_{\mathrm{sp}}^3(2\ell+5)\widehat{k}^2\widehat{\rho}^2(|\mathcal{I}|+|\mathcal{K}|+|\mathcal{J}|).$$

Proof. See [11, Theorem 5.4]. □

Remark 6.6. *The proof presented in [11, Theorem 5.4] uses* $\mathcal{I} = \mathcal{K} = \mathcal{J}$*. But it can be extended easily to arbitrary finite index sets* $\mathcal{I}$*,* $\mathcal{K}$*, and* $\mathcal{J}$*. Whenever the estimate*

$$\sum_{\substack{(t,r)\in\mathcal{T}_{\mathcal{I}\times\mathcal{I}}\\(r,s)\in\mathcal{T}_{\mathcal{I}\times\mathcal{I}}}} (|t|+a|r|+|s|) \leq C_{\mathrm{sp}}^2(2+a)(\rho+1)|\mathcal{I}|$$

is used for $a \in \mathbb{N}$ *(cf. [11, Lemma A.1]), it has to be replaced by the similarly derived estimate*

$$\sum_{\substack{(t,r)\in\mathcal{T}_{\mathcal{I}\times\mathcal{K}}\\(r,s)\in\mathcal{T}_{\mathcal{K}\times\mathcal{J}}}} (|t|+a|r|+|s|) \leq C_{\mathrm{sp}}^2(\rho+1)(|\mathcal{I}|+a|\mathcal{K}|+|\mathcal{J}|).$$

Remark 6.7. *The standard* $\mathcal{H}$*-matrix multiplication described in Section 4.4.2 adds intermediate low-rank matrices directly to the result. Therefore, we only have to store one such matrix at the same time. The* $\mathcal{H}$*-matrix multiplication with sum-expressions, on the other hand, is structured differently. We have to store all intermediate low-rank matrices in order to add and truncate them all at once. Although this allows to use fast truncation methods, this may also increase the storage required for intermediate results significantly, especially when the factors, and therefore the intermediate matrices, have a large (local) rank.*

6.2.2 Numerical experiments

In this section, we will again present and evaluate results of numerical experiments. This time, the experiments aim to compare the standard $\mathcal{H}$-matrix arithmetic with the $\mathcal{H}$-matrix arithmetic with sum-expression. Since a major advantage of the latter is the ability to use matrix-vector operation based truncation methods to compute blocks of the result with a single truncation, we also compare different truncation methods we have introduced in Section 2.1.

The model problem and the test setting are, again, the same as described in Section 5.4.1. Note that, in contrast to the $\mathcal{H}$-matrix arithmetic with accumulated updates, the H2Lib does not offer an implementation of the $\mathcal{H}$-matrix arithmetic with sum-expressions which therefore we implemented.

The cluster trees $\mathcal{T}_{\mathcal{I}}$ and $\mathcal{J}$ were generated with the coupled clustering introduced in Section 5.2, i.e., $\mathcal{T}_{\mathcal{J}}$ was generated with geometric bisection and $\mathcal{T}_{\mathcal{I}}$ was generated coupled with $\mathcal{T}_{\mathcal{J}}$ (see, e.g., Algorithm 5.1). The block cluster trees $\mathcal{T}_{\mathcal{I}\times\mathcal{I}}$, $\mathcal{T}_{\mathcal{J}\times\mathcal{I}}$, and $\mathcal{T}_{\mathcal{J}\times\mathcal{J}}$ were generated with the admissibility conditions (cf., Definitions 4.30, 4.34 and 5.8)

$$\mathrm{adm}_{\mathcal{I}\times\mathcal{I}} = \mathrm{adm}_{\mathrm{weak}}^{\mathrm{DD}}, \quad \mathrm{adm}_{\mathcal{J}\times\mathcal{I}} = \mathrm{adm}_{\mathrm{weak}}^{\mathrm{c}}, \quad \text{and } \mathrm{adm}_{\mathcal{J}\times\mathcal{J}} = \mathrm{adm}_{\eta} \quad (\eta = 16),$$

respectively. As before in Section 5.4, we denote the truncation accuracies by $\delta^{\mathcal{H}}_{\text{vel}}$, $\delta^{\mathcal{H}}_{\text{mul}}$, and $\delta^{\mathcal{H}}_{\text{Schur}}$, respectively

Figure 6.5 shows the computation times per degree of freedom for steps ①, ②, ④, and ⑤ of the preconditioner set-up (cf. Remark 5.3). Table 6.3 shows the absolute computation times for the steps of the preconditioner set-up with the standard $\mathcal{H}$-matrix arithmetic and with the $\mathcal{H}$-matrix arithmetic with sum-expressions and randomized low-rank truncations. Step ③ is omitted due to the similarities of the operation with the one in step ②. Therefore, we can expect similar results for both.

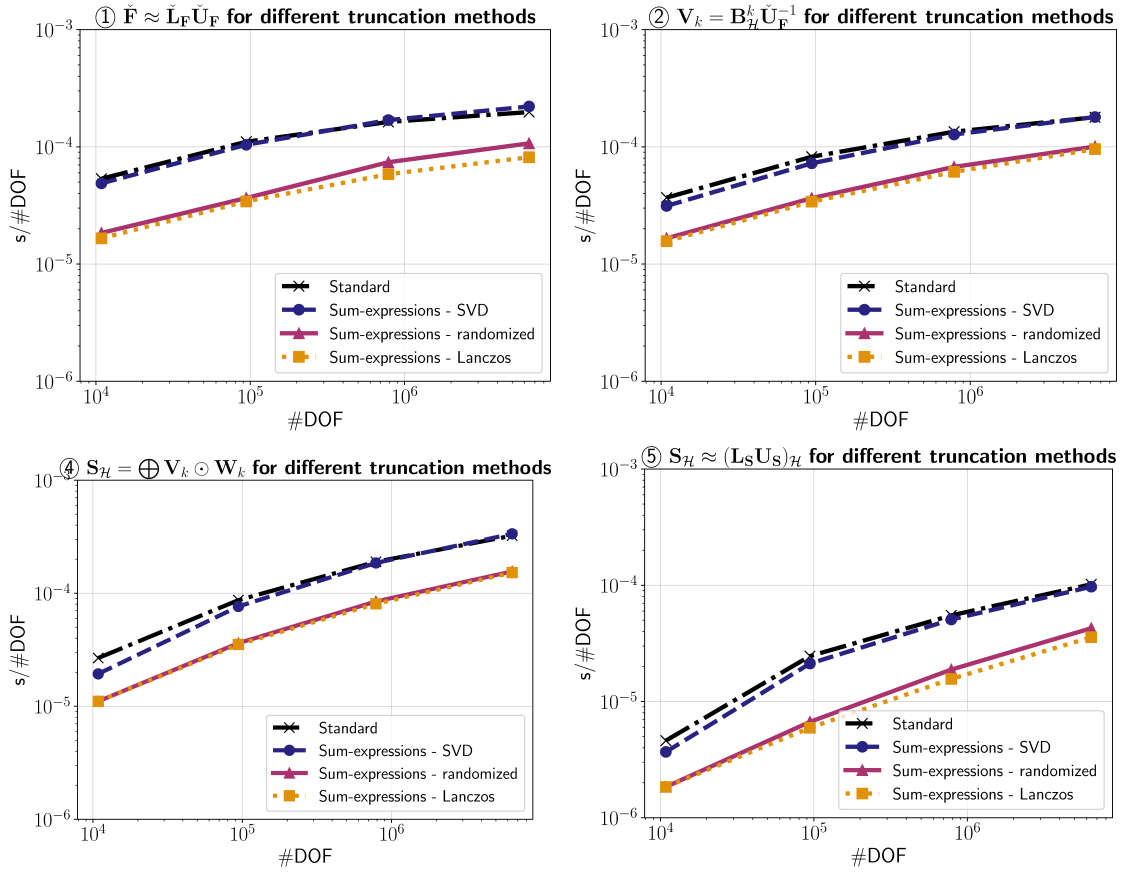

Figure 6.5: Computation times per degree of freedom for the preconditioner set-up with the standard $\mathcal{H}$-matrix arithmetic and the $\mathcal{H}$-matrix arithmetic with sum-expression with different truncation methods.

First, we can observe in Figure 6.5 that there is only a negligible difference between the standard arithmetic and the $\mathcal{H}$-matrix arithmetic with sum-expressions and rank truncations via SVD. This is to be expected since in this case both approaches are equivalent (cf. [11, §3.2.3]). Next, we can also observe that the matrix-vector operation based truncation methods both offer a significant speed-up for all steps of the preconditioner set-up. However, we can also observe that the difference between the truncation via Lanczos biorthogonalization and the randomized low-rank truncation is negligible. The time required to solve the system, which is depicted in Figure 6.6, is similar for all approaches. This suggests that the all truncation methods result in (spectrally) similar preconditioners.

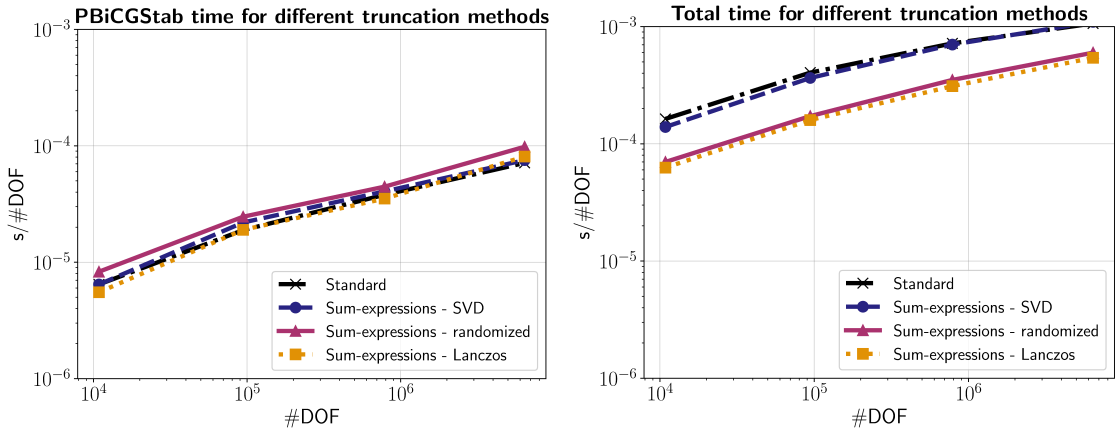

Figure 6.6: Computation times per degree of freedom for the solve with the PBiCGStab method and total time per degree of freedom with the standard $\mathcal{H}$-matrix arithmetic and the $\mathcal{H}$-matrix arithmetic with sum-expression with different truncation methods.

Standard $\mathcal{H}$-matrix arithmetic

#DOF	①	②	④	⑤	PBiCGStab (Iter)	Total
10,853	0.6s	0.4s	0.3s	0.1s	0.07s (9)	1.8s
94,285	10.4s	7.8s	8.2s	2.3s	1.8s (13)	38.2s
786,077	128.2s	105.8s	149.1s	43.3s	29.9s (21)	565.4s
6,419,773	1,271.0s	1,149.2s	2,074.7s	654.7s	457.6s (34)	6,823.7s

$\mathcal{H}$-matrix arithmetic with sum-expressions and randomized truncation

#DOF	①	②	④	⑤	PBiCGStab (Iter)	Total
10,853	0.2s	0.2s	0.1s	0.0s	0.09s (10)	0.8s
94,285	3.5s	3.4s	3.4s	0.6s	2.3s (16)	16.2s
786,077	58.1s	53.0s	66.5s	14.8s	35.0s (21)	275.8s
6,419,773	687.3s	646.0s	1,001.2s	274.0s	630.4s (39)	3,846.0s

Table 6.3: Absolute computation times for the preconditioner set-up and the solve with the BiCGStab method with the standard $\mathcal{H}$-matrix arithmetic (top) and with the $\mathcal{H}$-matrix arithmetic with sum-expressions (bottom). The time required to solve the system is supplemented by the number of iteration steps required to achieve a residual error of 10^{-12}.

Last, Figure 6.7 depicts the memory requirements for the $\mathcal{H}$-LU factorization $\check{\mathbf{L}}_\mathbf{F}\mathbf{U}_\mathbf{F} \approx \check{\mathbf{F}}$, the matrix $\mathbf{V}_1 = \mathbf{B}_\mathcal{H}\check{\mathbf{U}}_\mathbf{F}^{-1}$, and the $\mathcal{H}$-LU factorization $\mathbf{L}_\mathbf{S}\mathbf{U}_\mathbf{S}$ (cf. Remark 5.18).

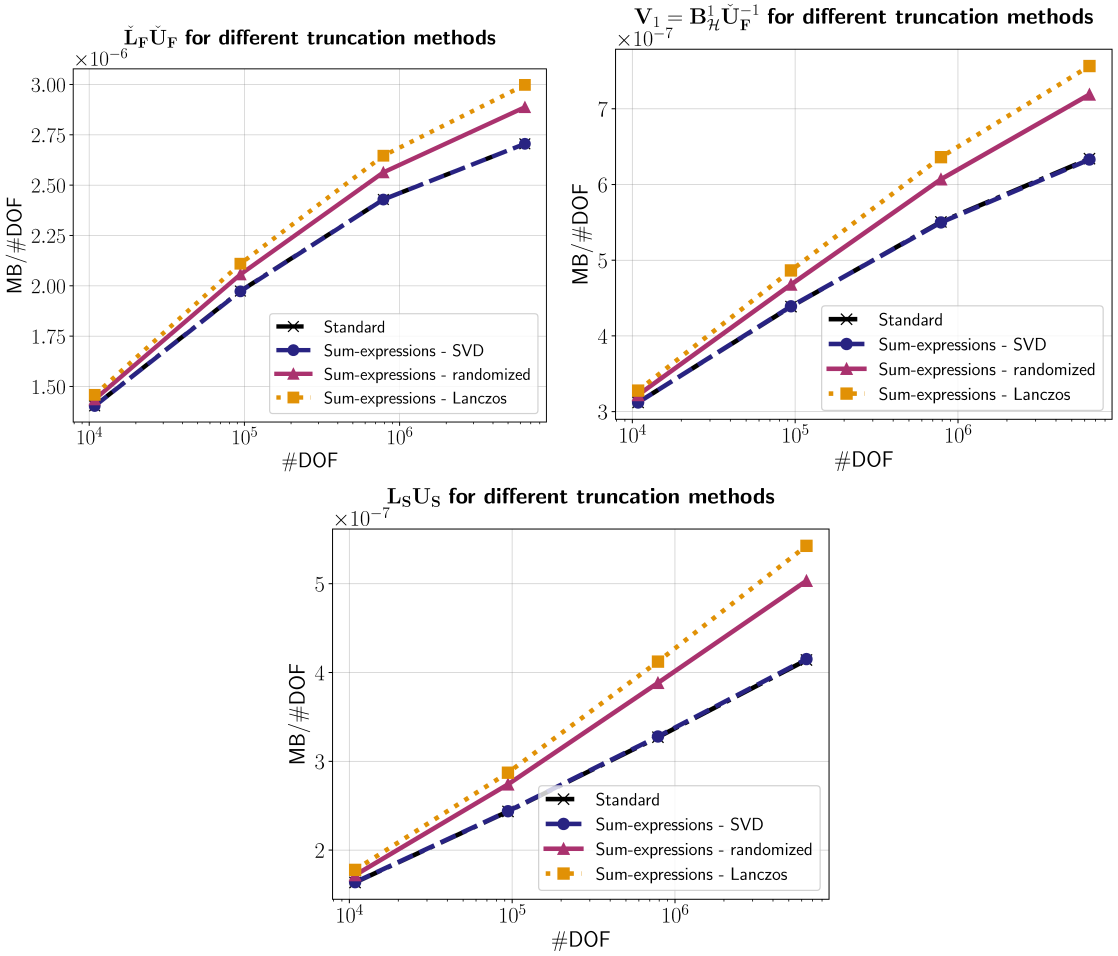

Figure 6.7: Memory required for the matrices computed for the preconditioner set-up with the standard $\mathcal{H}$-matrix arithmetic and with the $\mathcal{H}$-matrix arithmetic with sum-expressions with different truncation methods in Megabyte per degree of freedom.

Again, we can observe that the results with the standard $\mathcal{H}$-matrix arithmetic and the results with the $\mathcal{H}$-matrix arithmetic with sum-expressions and pairwise SVD truncations are similar, as it is to be expected since the approach via sum-expressions only delays the truncations.

However, we can also observe that the two matrix-vector operation based truncation methods result in computed matrices with a larger memory requirement which suggests that they result in larger local ranks of the (admissible) low-rank blocks. Additionally, there is also a difference between the randomized truncation method and the truncation via Lanczos biorthogonalization. The latter requires more memory for all computed matrices, i.e., these matrices have an even larger local rank.

Conclusion. Both approaches, the standard $\mathcal{H}$-matrix arithmetic and the $\mathcal{H}$-matrix arithmetic with sum-expressions, have their advantages and disadvantages. The computation

times observed in Figures 6.5 and 6.6 suggest that the $\mathcal{H}$-matrix arithmetic with sum-expressions in combination with matrix-vector operation based rank truncation methods can result in a significant speed-up of about 40%–50% compared to the standard $\mathcal{H}$-matrix arithmetic.

However, the standard $\mathcal{H}$-matrix arithmetic requires less intermediate storage (cf. Remark 6.7) and results in more accurate low-rank approximations, i.e., in less memory required for the computed matrices.

Hence, we conclude that the $\mathcal{H}$-matrix with sum-expressions in combination with matrix-vector operation based truncation methods are preferable if one does not have to worry about the additinonally required memory, i.e., if there is enough memory to store the intermediate low-rank matrices and handling the higher local rank of the computed matrices. In this case, we prefer the rank truncation via Lanczos biorthogonalization. Although this method results in larger memory requirements than the randomized rank truncation, the difference is not huge. But an advantage of the rank truncation with Lanczos biorthogonalization is that it is deterministic. If, on the other hand, memory is a limited resource, the standard $\mathcal{H}$-matrix arithmetic is preferable.

Chapter 7

Conclusion and outlook

We used two different approaches to improve the set-up time of $\mathcal{H}$-LU factorizations for (block) preconditioners of saddle point problems

$$\mathcal{M} \begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix} \begin{pmatrix} \mathbf{F} & \mathbf{B}^\mathsf{T} \\ \mathbf{B} & \mathbf{0} \end{pmatrix} \begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix} = \begin{pmatrix} \mathbf{r} \\ \mathbf{s} \end{pmatrix}$$

from a mixed finite element discretization of Oseen problems, as first introduced in [31].

1. An optimization of the block structure of the involved $\mathcal{H}$-matrices.
2. The application of two variants of the $\mathcal{H}$-matrix multiplication.

The first approach was part of Chapter 5. We introduced a new cluster strategy, the coupled clustering. This new strategy aimed to optimize the block structure of the matrix $\mathbf{B}$ for the time-consuming $\mathcal{H}$-matrix updates necessary for the explicit computation of an (approximate) Schur complement. Since $\mathbf{B}$ describes a coupling of the pressure and the velocity discretization, the idea was to couple the clustering of the velocity index set with an already generated cluster tree for the pressure index set. This resulted in a geometric decomposition of the finite element grid similar to the one generated by the domain decomposition clustering but with larger interfaces. To compensate for these disadvantageous larger interfaces, we additionally introduced an adapted version of the coupled clustering, the coupled clustering with interface decomposition.

From Section 5.4, we summarize the main conclusions drawn from the presented results of numerical experiments.

- The optimized block structure of the matrix $\mathbf{B}$ resulting from the coupled clustering allowed for a speed-up of about 50% for the computation of an (approximate) Schur complement. However, it also leads to a slower computation of an $\mathcal{H}$-LU factorization of $\mathbf{F}$. Nonetheless, the total speed-up for the preconditioner set-up was still about 40% for a truncation accuracy of 10^{-1}.
- While the coupled clustering with interface decomposition resulted, as intended, in a faster computation of an $\mathcal{H}$-LU factorization of $\mathbf{F}$, it had a negative effect on the time required to compute an (approximate) Schur complement. In total, the set-up time was comparable to the one obtained with the uncoupled clustering.

- Smaller truncation accuracies for the $\mathcal{H}$-matrix operations of the preconditioner set-up lead to larger computation times for the set-up without a significant speed-up for the time required to solve the system with the PBiCGStab method. Additionally, we observed that the coupled clustering got less effective due to the block structure of the involved matrices. For smaller truncation accuracies, we can expect the uncoupled clustering leading to a faster set-up. However, in our tests a truncation accuracy of 10^{-1} for all operations already lead to effective preconditioners.

In Chapter 6, we analyzed the second approach, the application of variants of the $\mathcal{H}$-matrix arithmetic to our problem.

In case of the $\mathcal{H}$-matrix arithmetic with accumulated updates, we presented results of numerical experiments that suggested that there is no benefit from using this variant for all operations of the preconditioner set-up. Although there was an observable minor speed-up for the $\mathcal{H}$-matrix updates required for the computation of an (approximate) Schur complement, we also observed a slower computation of an $\mathcal{H}$-LU factorization of $\mathbf{F}$. In total, there was only a negligible difference in favor of the standard $\mathcal{H}$-matrix arithmetic for smaller system sizes and in favor of the $\mathcal{H}$-matrix arithmetic for larger system sizes. However, one may consider using a mix of both variants, the standard $\mathcal{H}$-matrix arithmetic and the $\mathcal{H}$-matrix arithmetic with accumulated updates. In our tests, we observed a speed-up of about 10%–20% with this approach.

In case of the $\mathcal{H}$-matrix arithmetic with sum-expressions, we presented results illustrating the effectiveness of matrix-vector operation based truncation methods. Although these methods tend to yield less accurate approximations, observable in our results by larger memory requirements indicating larger local ranks, they also yield a significant speed-up of about 40%–50% for all operations of the preconditioner set-up. Although the larger local ranks, and therefore larger memory requirements, may be problematic if the amount of available memory is limited, we note that the $\mathcal{H}$-matrix arithmetic with sum-expressions itself may introduce a significant increase of memory required for intermediate results (cf. Remark 6.7). Hence, the $\mathcal{H}$-matrix arithmetic with sum-expressions may not be suitable in this case.

We close this thesis with a brief outlook on possible future work based on the results presented in this thesis.

- In our numerical experiments we used an inf-sup stable pair of finite element spaces. Another option would have been to stabilize an unstable pair of finite element space. This results in a saddle point problem of the form

$$\begin{pmatrix} \mathbf{F} & \mathbf{B}^T \\ \mathbf{B} & -\mathbf{C} \end{pmatrix} \begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix} = \begin{pmatrix} \mathbf{r} \\ \mathbf{s} \end{pmatrix}$$

  with a non-zero stabilization matrix $\mathbf{C}$. In this case, the discretization of the pressure and the velocity space can be chosen based on the same triangulation (cf. Section 3.3). The coupled clustering as we introduced it in Section 5.2, can be adapted for this case easily. Although we expect the coupled clustering to be also effective, further numerical experiments are required to evaluate the effectiveness of the coupled clustering in this case.

- A popular approximation to the Schur complement in fluid dynamical applications is the least squares commutator (see, e.g., [12, 13]). An advantage of this approximation is that it can be used easily for a sequence of linear systems as they occur, e.g., when solving the Navier-Stokes equations. However, the application of the inverse of the least square commutator requires two solves with a matrix $\mathbf{B}\mathbf{Q}^{-1}\mathbf{B}^{\mathsf{T}}$ where $\mathbf{Q}$ is either a diagonal matrix or a mass matrix for the velocity discretization. Computing $\mathbf{B}\mathbf{Q}^{-1}\mathbf{B}^{\mathsf{T}}$ (or its inverse) directly may not be an option for a larger number of degrees of freedom since it may be dense. Instead, we could use an $\mathcal{H}$-matrix factorization which has to be computed once for the whole sequence of linear systems. Since the computation of $\mathbf{B}\mathbf{Q}^{-1}\mathbf{B}^{\mathsf{T}}$ is similar to the computation of an (approximate) Schur complement, the coupled clustering may again yield a favorable block structure for $\mathcal{H}$-matrix representations of $\mathbf{B}$. It remains to be seen how this approach performs compared to the explicit computation and factorization of an (approximate) Schur complement as described in Chapter 5.

Bibliography

[1] M. Bebendorf. "Hierarchical LU decomposition-based preconditioners for BEM". In: *Computing* 74.3 (2005), pp.225–247. ISSN: 0010-485X,1436-5057. URL: https://doi.org/10.1007/s00607-004-0099-6.

[2] M. Bebendorf. "Why finite element discretizations can be factored by triangular hierarchical matrices". In: *SIAM J. Numer. Anal.* 45.4 (2007), pp.1472–1494. ISSN: 0036-1429,1095-7170. URL: https://doi.org/10.1137/060669747.

[3] M. Bebendorf and W. Hackbusch. "Existence of $\mathcal{H}$-matrix approximants to the inverse FE -matrix of elliptic operators with L^∞-coefficients". In: *Numer. Math.* 95.1 (2003), pp.1–28. ISSN: 0029-599X,0945-3245. URL: https://doi.org/10.1007/s00211-002-0445-6.

[4] M. Benzi, G. H. Golub, and J. Liesen. "Numerical solution of saddle point problems". In: *Acta Numer.* 14 (2005), pp.1–137. ISSN: 0962-4929. URL: https://doi.org/10.1017/S0962492904000212.

[5] S. Börm. "Hierarchical matrix arithmetic with accumulated updates". In: *Comput. Vis. Sci.* 20.3-6 (2019), pp.71–84. ISSN: 1432-9360. URL: https://doi.org/10.1007/s00791-019-00311-3.

[6] S. Börm, K. Reimer, D. Boysen, S. Christophersen, N. Albrecht, J. Burmeister, and C. Boerst. *H2Lib*. Version 3.0. URL: https://www.h2lib.org (visited on 08/12/2024).

[7] S. C. Brenner and L. R. Scott. *The mathematical theory of finite element methods*. Third. Vol. 15. Texts in Applied Mathematics. Springer, New York, 2008, pp.xviii+397. ISBN: 978-0-387-75933-3. URL: https://doi.org/10.1007/978-0-387-75934-0.

[8] C. Brezinski and M. Redivo-Zaglia. "Look-ahead in Bi-CGSTAB and other product methods for linear systems". In: *BIT* 35.2 (1995), pp.169–201. ISSN: 0006-3835. URL: https://doi.org/10.1007/BF01737161.

[9] E. Carson, J. Liesen, and Z. Strakoš. "Towards understanding CG and GMRES through examples". In: *Linear Algebra Appl.* 692 (2024), pp.241–291. ISSN: 0024-3795,1873-1856. URL: https://doi.org/10.1016/j.laa.2024.04.003.

[10] C. R. Dohrmann and P. B. Bochev. "A stabilized finite element method for the Stokes problem based on polynomial pressure projections". In: *Internat. J. Numer. Methods Fluids* 46.2 (2004), pp.183–201. ISSN: 0271-2091,1097-0363. URL: https://doi.org/10.1002/fld.752.

[11] J. Dölz, H. Harbrecht, and M. D. Multerer. "On the best approximation of the hierarchical matrix product". In: *SIAM J. Matrix Anal. Appl.* 40.1 (2019), pp.147–174. ISSN: 0895-4798. URL: https://doi.org/10.1137/18M1189373.

[12] H. Elman, V. E. Howle, J. Shadid, R. Shuttleworth, and R. Tuminaro. "Block preconditioners based on approximate commutators". In: *SIAM J. Sci. Comput.* 27.5 (2006), pp.1651–1668. ISSN: 1064-8275,1095-7197. URL: https://doi.org/10.1137/040608817.

[13] H. C. Elman, D. J. Silvester, and A. J. Wathen. *Finite elements and fast iterative solvers: with applications in incompressible fluid dynamics*. Second. Numer. Math. Sci. Comput. Oxford University Press, Oxford, 2014, pp.xiv+479. ISBN: 978-0-19-967880-8. URL: https://doi.org/10.1093/acprof:oso/9780199678792.001.0001.

[14] M. Embree. "The tortoise and the hare restart GMRES". In: *SIAM Rev.* 45.2 (2003), pp.259–266. ISSN: 0036-1445,1095-7200. URL: https://doi.org/10.1137/S003614450139961.

[15] A. Ern and J.-L. Guermond. *Theory and practice of finite elements*. Vol. 159. Applied Mathematical Sciences. Springer-Verlag, New York, 2004, pp.xiv+524. ISBN: 0-387-20574-8. URL: https://doi.org/10.1007/978-1-4757-4355-5.

[16] M. Faustmann, J. M. Melenk, and D. Praetorius. "$\mathcal{H}$-matrix approximability of the inverses of FEM matrices". In: *Numer. Math.* 131.4 (2015), pp.615–642. ISSN: 0029-599X,0945-3245. URL: https://doi.org/10.1007/s00211-015-0706-9.

[17] G. Golub and W. Kahan. "Calculating the singular values and pseudo-inverse of a matrix". In: *J. Soc. Indust. Appl. Math. Ser. B Numer. Anal.* 2 (1965), pp.205–224. ISSN: 0887-459X,2168-3581.

[18] G. H. Golub and C. F. Van Loan. *Matrix computations*. Fourth. Johns Hopkins Studies in the Mathematical Sciences. Johns Hopkins University Press, Baltimore, MD, 2013, pp.xiv+756. ISBN: 978-1-4214-0794-4; 1-4214-0794-9; 978-1-4214-0859-0.

[19] J. Grams and S. Le Borne. "Coupled clustering strategies for hierarchical matrix preconditioners in saddle point problems". In: *PAMM* 23.2 (2023), e202300077. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/pamm.202300077.

[20] L. Grasedyck, W. Hackbusch, and R. Kriemann. "Performance of $\mathcal{H}$-LU preconditioning for sparse matrices". In: *Comput. Methods Appl. Math.* 8.4 (2008), pp.336–349. ISSN: 1609-4840,1609-9389. URL: https://doi.org/10.2478/cmam-2008-0024.

[21] L. Grasedyck and W. Hackbusch. "Construction and arithmetics of $\mathcal{H}$-matrices". In: *Computing* 70.4 (2003), pp.295–334. ISSN: 0010-485X. URL: https://doi.org/10.1007/s00607-003-0019-1.

[22] L. Grasedyck, R. Kriemann, and S. Le Borne. "Domain decomposition based $\mathcal{H}$-LU preconditioning". In: *Numer. Math.* 112.4 (2009), pp.565–600. ISSN: 0029-599X. URL: https://doi.org/10.1007/s00211-009-0218-6.

[23] L. Grasedyck, R. Kriemann, and S. Le Borne. "Parallel black box $\mathcal{H}$-LU preconditioning for elliptic boundary value problems". In: *Comput. Vis. Sci.* 11.4-6 (2008), pp.273–291. ISSN: 1432-9360,1433-0369. URL: https://doi.org/10.1007/s00791-008-0098-9.

[24] P. R. Graves-Morris. "The breakdowns of BiCGStab". In: vol. 29. 1-3. Matrix iterative analysis and biorthogonality (Luminy, 2000). 2002, pp.97–105. URL: https://doi.org/10.1023/A:1014864007293.

[25] W. Hackbusch. "A sparse matrix arithmetic based on $\mathcal{H}$-matrices. I. Introduction to $\mathcal{H}$-matrices". In: *Computing* 62.2 (1999), pp.89–108. ISSN: 0010-485X,1436-5057. URL: https://doi.org/10.1007/s006070050015.

[26] W. Hackbusch, B. Khoromskij, and S. A. Sauter. "On $\mathcal{H}^2$-matrices". In: *Lectures on Applied Mathematics (Munich, 1999)*. Springer, Berlin, 2000, pp.9–29. ISBN: 3-540-66734-2.

[27] W. Hackbusch, B. N. Khoromskij, and R. Kriemann. "Hierarchical matrices based on a weak admissibility criterion". In: *Computing* 73.3 (2004), pp.207–243. ISSN: 0010-485X. URL: https://doi.org/10.1007/s00607-004-0080-4.

[28] W. Hackbusch. *Hierarchical matrices: algorithms and analysis*. Vol. 49. Springer Ser. Comput. Math. Springer, Heidelberg, 2015, pp.xxv+511. ISBN: 978-3-662-47323-8; 978-3-662-47324-5. URL: https://doi.org/10.1007/978-3-662-47324-5.

[29] N. Halko, P. G. Martinsson, and J. A. Tropp. "Finding structure with randomness: probabilistic algorithms for constructing approximate matrix decompositions". In: *SIAM Rev.* 53.2 (2011), pp.217–288. ISSN: 0036-1445. URL: https://doi.org/10.1137/090771806.

[30] V. John. *Finite element methods for incompressible flow problems*. Vol. 51. Springer Ser. Comput. Math. Springer, Cham, 2016, pp.xiii+812. ISBN: 978-3-319-45749-9; 978-3-319-45750-5. URL: https://doi.org/10.1007/978-3-319-45750-5.

[31] S. Le Borne. "Hierarchical matrix preconditioners for the Oseen equations". In: *Comput. Vis. Sci.* 11.3 (2008), pp.147–157. ISSN: 1432-9360. URL: https://doi.org/10.1007/s00791-007-0065-x.

[32] P.-G. Martinsson and J. A. Tropp. "Randomized numerical linear algebra: foundations and algorithms". In: *Acta Numer.* 29 (2020), pp.403–572. ISSN: 0962-4929,1474-0508. URL: https://doi.org/10.1017/s0962492920000021.

[33] M. F. Murphy, G. H. Golub, and A. J. Wathen. "A note on preconditioning for indefinite linear systems". In: *SIAM J. Sci. Comput.* 21.6 (2000), pp.1969–1972. ISSN: 1064-8275,1095-7197. URL: https://doi.org/10.1137/S1064827599355153.

[34] H.-G. Roos, M. Stynes, and L. Tobiska. *Robust numerical methods for singularly perturbed differential equations*. Second. Vol. 24. Springer Ser. Comput. Math. Convection-diffusion-reaction and flow problems. Springer-Verlag, Berlin, 2008, pp.xiv+604. ISBN: 978-3-540-34466-7.

[35] Y. Saad. *Iterative methods for sparse linear systems*. Second. Society for Industrial and Applied Mathematics, Philadelphia, PA, 2003, pp.xviii+528. ISBN: 0-89871-534-2. URL: https://doi.org/10.1137/1.9780898718003.

[36] H. D. Simon and H. Zha. "Low-rank matrix approximation using the Lanczos bidiagonalization process with applications". In: *SIAM J. Sci. Comput.* 21.6 (2000), pp.2257–2274. ISSN: 1064-8275,1095-7197. URL: https://doi.org/10.1137/S1064827597327309.

[37] M. Tabata. "A finite element approximation corresponding to the upwind finite differencing". In: *Mem. Numer. Math.* 4 (1977), pp.47–63.

[38] H. A. van der Vorst. "Bi-CGSTAB: a fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems". In: *SIAM J. Sci. Statist. Comput.* 13.2 (1992), pp.631–644. ISSN: 0196-5204. URL: https://doi.org/10.1137/0913035.