TECHNISCHE UNIVERSITÄT MÜNCHEN

Lehrstuhl für Informatik mit
Schwerpunkt Wissenschaftliches Rechnen

Sparse Approximate Inverses for Preconditioning, Smoothing, and Regularization

Matous Sedlacek

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des Akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:		Univ.-Prof. Dr. Daniel Cremers
Prüfer der Dissertation:	1.	Univ.-Prof. Dr. Thomas Huckle
	2.	Prof. Ing. Miroslav Tůma, CSc., Technical University of Liberec, Tschechien

Die Dissertation wurde am 27.6.2012 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 12.9.2012 angenommen.

Abstract

Many applications in scientific computing including the discretization of partial differential equations lead to large and sparse linear systems of equations which require to be solved in reasonable time and space. However, the dimension of the systems is usually too large for being handled efficiently by direct methods. Here, the usage of iterative solution methods becomes mandatory because they have considerably lower memory requirements and entail fewer operations. Yet, their convergence depends on the spectral properties of the underlying matrix. If it is ill-conditioned, it is often indispensable to include an efficient preconditioner in order to obtain fast convergence.

For high-dimensional problems, parallel environments and implementations often remain the only resort to compute the solution in acceptable time these days. While iterative solvers are simpler to parallelize than direct ones, preconditioners which are efficient for a large class of problems virtually always are strongly sequential. In this context, parallel preconditioners become of increasing importance. Sparse approximate inverses constructed by Frobenius norm minimization (SPAI and MSPAI) or K-condition number minimization (FSPAI) satisfy the preconditioner requirements for such large-scale computations: they are inherently parallel, robust, and flexible.

The thesis introduces variants of the sparse approximate inverse technique maintaining its advantages. Based on permutations and recursive local refinement, we present a multistep MSPAI strategy (MMSPAI) which turns out to be less time and memory consuming than a comparable multistep successive preconditioning (MSP) approach. In the symmetric positive definite case, we focus on FSPAI. Theoretically, it is possible that although the preconditioner may satisfy the given tolerance criterion, it may still have an arbitrary large K-condition number. Therefore, we derive an exact version of FSPAI (EFSPAI) whose index choosing criterion avoids unbounded K-condition values. Moreover, we generalize FSPAI to its block case (BFSPAI) where our preliminary results indicate similar preconditioner quality at lower setup costs.

Concerning technical realizations—forming the basis for future packages which may include the mentioned variants—the thesis presents an enhanced (M)SPAI implementation and a new parallel implementation of FSPAI. Both generic codes provide various algorithmic realizations including the usage of different linear algebra libraries. We investigate the impact of using dictionaries and sparse Cholesky factorizations in order to speedup the preconditioner setup. Moreover, we focus on optimizations related to memory allocations and critical algorithmic parts which—in terms of runtime—make both implementations competitive or even superior to state of the art implementations. First massively parallel setup (and solving) experiments give insight into the scalability of the implementations and into the inherent parallelism of the algorithms when using a large number of processing elements.

We refine the matrix based probing ansatz in the modified SPAI (MSPAI) formulation by introducing mask probing which allows to impose probing conditions on each column of the preconditioner. Multigrid and iterative regularization methods benefit from smoothing and regularizing preconditioners which have a different behavior on the low and high frequency subspace, respectively. In this application-oriented context, we analytically derive optimal preconditioners for structured 1D and 2D operators and illustrate the effect when translating the occurring minimization conditions into mask and global probing conditions for MSPAI. Leading to nearly optimal results, we apply these MSPAIs also to problems with varying coefficients and problems resulting from Fredholm integral equations of the first kind. In case

of regularizing preconditioners, we additionally analyze the effect of adjusting the probing subspace to the underlying signal structure.

For discrete ill-posed and deconvolution problems we provide further approaches to improve the reconstruction of common direct and iterative regularization methods. Taking the signal structure into account, we introduce data based regularization which for a certain class of problems yields strong improvement. Having a more theoretical character, we also consider results for a piecewise reconstruction. Moreover, we introduce operator dependent seminorms which incorporate spectral data. In contrast to smoothing norms, this approach is not restricted to a certain structure of the signal and provides robust behavior, also with respect to the regularization parameter. Along the way, we propose an L-curve algorithm based on smoothing and B-splines for the estimation of discrete regularization parameters. Its results indicate competitive behavior in comparison to other state of the art L-curve algorithms.

Zusammenfassung

Eine Vielzahl an Anwendungen im Bereich des wissenschaftlichen Rechnens, darunter auch die Diskretisierung partieller Differentialgleichungen, führt zu großen dünnbesetzten linearen Gleichungssystemen, die sowohl in vertretbarer Zeit als auch vertretbarem Speicherbedarf gelöst werden sollen. Die häufig hohe Dimensionalität solcher Systeme verhindert eine effiziente Verarbeitung mittels direkter Verfahren. In diesen Fällen ist der Einsatz iterativer Lösungsverfahren obligatorisch, da diese entscheidend weniger Speicher und Rechenoperationen benötigen. Deren Konvergenzrate hängt jedoch von den Spektraleigenschaften der zugrunde liegenden Matrix ab. Ist diese schlecht konditioniert, so ist es meistens unerlässlich einen effizienten Präkonditionierer anzuwenden, um schnelle Konvergenz zu erzielen.

Für hochdimensionale Probleme stellen parallele Umgebungen und damit auch parallele Implementierungen oft den einzigen Ausweg dar, um eine Lösung in praktikabler Zeit zu berechnen. Während iterative Löser im Vergleich zu direkten Verfahren einfacher zu parallelisieren sind, ist die Berechnung von Präkonditionierern, die für große Probleme effektiv sind, dagegen fast immer stark sequentiell. In diesem Zusammenhang werden parallele Präkonditionierungstechniken immer bedeutsamer. Dünnbesetzte approximative Inverse, konstruiert über Minimierung der Frobeniusnorm (SPAI und MSPAI) oder der K-Konditionszahl (FSPAI), erfüllen die Bedingungen an Präkonditionierer zur Lösung solcher großen Systeme: Sie sind inhärent parallel, robust und flexibel.

Die vorliegende Arbeit stellt Varianten von dünnbesetzten approximativen Inversen vor. Basierend auf Permutationen und rekursiven Verfeinerns wird eine mehrstufige Strategie (MMSPAI) präsentiert, die Zeit- und Speicherperformanter ist als ein vergleichbarer global verfeinernder mehrstufiger Präkonditionierungsansatz (MSP). Im symmetrisch positiv definiten Fall liegt der Fokus auf FSPAI. Hierbei ist es theoretisch möglich, dass das präkonditionierte Gleichungssystem eine beliebig große K-Konditionszahl besitzt, obwohl der Präkonditionierer das geforderte Toleranzkriterium erfüllt. Deshalb wird eine exakte Version des FSPAI (EFSPAI) abgeleitet, die unbeschränkte K-Konditionszahlen vermeidet. Die zusätzlich vorgestellte Blockvariante von FSPAI (BFSPAI) deutet mit ersten Resultaten auf einen qualitativ gleichwertigen Präkonditionierer hin, der in kürzerer Zeit berechnet werden kann.

Hinsichtlich technischer Umsetzung dünnbesetzter approximativer Inverser wird eine erweiterte (M)SPAI Implementierung und eine neue FSPAI Implementierung eingeführt. Diese können als Basis für künftige Softwarepakete dienen, welche die genannten Modifikationen bereitstellen. Beide Quellcodes sind generisch und implementieren verschiedene Berechnungsmodi, die unter anderem auf die Verwendung hochperformanter Kernroutinen, realisiert in unterschiedlichen Bibliotheken, zurückgreifen. Hierbei wird sowohl die Wirkung unterschiedlicher Caching-Strategien, als auch eine dünnbesetzte Realisierung der Cholesky Zerlegung zur Beschleunigung der Berechnung untersucht. Darüberhinaus wird die Laufzeitoptimierung hinsichtlich Speicherallokation und berechnungsintensiver Algorithmenabschnitte analysiert, die die Implementierungen konkurrenzfähig oder gar überlegen zum Stand der Technik machen. Erste massiv parallele Experimente mit einer Vielzahl an Prozessoren geben Einblick in die Skalierbarkeit der vorgestellten Implementierungen sowie in die inhärente Parallelität der zugrunde liegenden Algorithmen.

Der klassische matrixbasierte Probing Ansatz des MSPAI (modified SPAI) wird erweitert um ein lokales, sogenanntes Mask Probing, welches es erlaubt dem Präkonditionierer spaltenabhängige Probing Bedingungen aufzuerlegen. Mehrgitter und iterative Regularisierungsmethoden profitieren von glättenden beziehungsweise regularisierenden Präkonditionierern, die

unterschiedliches Verhalten auf dem niederfrequenten respektive dem hochfrequenten Unterraum besitzen. In diesem anwendungsorientierten Kontext werden optimale Präkonditionierer für strukturierte 1D und 2D Probleme analytisch hergeleitet. Die dabei auftretenden Minimierungsbedingungen werden in lokale und globale Probing Bedingungen für MSPAI übersetzt. Diese, auf optimale Resultate führenden MSPAIs, werden auch auf Probleme mit wechselnden Koeffizienten sowie Probleme resultierend aus der Diskretisierung von Fredholm Integralgleichungen erster Art angewandt. Bezüglich regularisierender Präkonditionierer wird zusätzlich ein Anpassen des Probing Unterraums an die Signalstruktur analysiert.

Des Weiteren werden für diskrete schlecht gestellte und Dekonvolutionsprobleme Ansätze zur verbesserten Rekonstruktion direkter und iterativer Regularisierungsverfahren präsentiert. Unter Berücksichtigung der Singalstruktur wird eine datenbasierte Regularisierung eingeführt, die für eine bestimmte Problemklasse zu starker Verbesserung führt. Vor theoretischem Hintergrund wird zusätzlich der Effekt einer stückweisen Rekonstruktion gezeigt. Es werden Seminormen vorgestellt, die die spektrale Information der vorliegenden Operatoren ausnutzen. Im Gegensatz zu Glättungsnormen ist dieser Ansatz nicht auf eine bestimmte Struktur des Signals eingeschränkt und zeigt robustes Verhalten, auch im Hinblick auf die Wahl des Regularisierungsparameters. Zur Bestimmung diskreter Regularisierungsparameter wird ein L-Kurven Algorithmus basierend auf Glättung und B-splines präsentiert, dessen Resultate auf konkurrenzfähiges Verhalten im Vergleich zu gebräuchlichen L-Kurven Algorithmen hindeuten.

Contents

Chapter 1

Introduction

1.1. Scientific Environment of the Thesis

Many scientific and in these days already industrial areas require the solution of computationally intensive problems in reasonable time. Among these we find climate or earthquake prediction, molecular dynamics simulations, aerospace computations, economic models, or the design of circuits. One of the most challenging parts in these fields is the solution of the occurring large and sparse linear systems. These also emerge whenever computational processes on physical reactions are involved, which typically are modelled by integral or differential equations. In particular, such processes are based on conservation laws for mass, momentum, and energy, or on extremal energy principles of the system. These physical laws as well as the determination of extremal values induce a rate of change of the involved physical quantities. Thus, the underlying system can be described by partial differential equations which establish a connection between functions and their derivatives. Any computerized treatment of such differential systems requires discretizations which again lead to large and sparse linear systems.

However, the size of such problems—consisting of several millions, even hundreds or thousands of millions of unknowns—anticipates their solution by direct methods due to impractical time and storage costs. Here, the usage of iterative methods is mandatory: they have significantly lower memory requirements and entail fewer operations, especially when an approximate solution of low accuracy is sufficient. Moreover, for sparse systems certain iterative (Krylov subspace) solvers achieve optimal complexity. For these, however, it is well-known that their convergence strongly depends on the spectral properties of the underlying system, i.e., if it is ill-conditioned, the method may take a large number of iterations or even diverge. Hence, in order to obtain (fast) convergence, it is essential to transform the system into a well-conditioned one. This technique is known as preconditioning. Over the past two decades, plenty of diverse preconditioning approaches were published. To choose the appropriate of is often problem dependent and more important than the choice of the solver. The following closing note from Trefethen and Bau [137] (also cited by Benzi [16]) highlights this point:

> "*In ending this book with the subject of preconditioners, we find ourselves at the philosophical center of the scientific computing of the future. [...] Nothing will be more central to computational science in the next century than the art of transforming a problem that appears intractable into another whose solution can be approximated rapidly. For Krylov subspace matrix iterations, this is preconditioning.*"

1.1.1. Sparse Approximate Inverses for Preconditioning, ...

The vicious circle between increasing problem size (see, e.g., [44]) and increasing performance of systems has been accelerated with the advent of vector computers and parallel architectures. Today, the mentioned extremely large-scale problems are only solvable by applying iterative methods on high performance clusters. By splitting up the overall problem into thousands or hundreds of thousands of pieces, makes it manageable both in time and space. However, the parallel efficiency of any algorithm depends on its sequential fraction, i.e., on the part which cannot be parallelized but must be performed successively. This proportion is responsible for how well the method scales, i.e., from which number of processing elements an intolerable imbalance occurs between increasing communication and decreasing computational speedup. Sophisticated implementations of iterative solvers which are able to solve large (and sparse) linear systems in parallel are available. The problem is on the preconditioner's side. Those which are efficient for a large number of problems are strongly sequential. Hence, they can hardly be used for large-scale computations in massively parallel settings because they represent the limiting factor of the scalability. On the path to exascale computing, where new parallel clusters with millions of cores come into existence and new parallel programming paradigms are necessary, inherently parallel preconditioners become of increasing importance.

Sparse approximate inverses constructed by Frobenius norm minimization are preconditioners which fit into the requirements for large-scale computations. However, their setup requires an a priori chosen sparsity pattern which consequently is responsible for the quality of the approximation. As the structure of the inverse is unpredictable, the original static approach (SAI) gained little attention for almost two decades. It was in the late 1990s when Grote and Huckle [58] proposed an adaptive technique (SPAI) which automatically (dynamically) captures most profitable indices for the pattern of the preconditioner. Since then, this approach draws attention and led to several modifications and parallel implementations. In 2003, Huckle [81, 82] transferred the approach to the symmetric positive definite (SPD) case providing a factorized variant (FSPAI). A generalized, modified version of SPAI (MSPAI) was proposed in 2007 by Huckle and Kallischko [85]. It allows to impose additional properties on the preconditioner while preserving the main advantages of SPAI: inherent parallelism, robustness, and adaptive capturing of a sparse pattern of the inverse.

In connection to this, the thesis provides an accurate description on sparse approximate inverses and introduces new variants which remain inherently parallel. Furthermore, we exploit and enhance MSPAI's probing concept to analyze its effect when applying the resulting preconditioners in applications demanding for different properties on different subspaces. We introduce parallel implementations on (M)SPAI and FSPAI for which we also provide scalability results for massively parallel settings in order to deliver insight both into the implementational and algorithmic behavior for large-scale problems.

1.1.2. ... Smoothing, and Regularization

Multigrid methods are efficient iterative methods for the solution of large (and sparse) linear systems. For certain classes of problems they are known to be among the fastest algorithms. They are based on the approach of combining smoothing and coarse grid correction recursively. The high frequency Fourier modes of the error—related to the solution—can be dampened by applying smoothers such as Gauss-Seidel iteration. By projection onto a

coarser grid in a following step, its remaining smooth components become oscillatory and can again be dampened using a smoothing step. Hence, in order to reduce the high frequency error part in very few iterations, an efficient smoother is essential. However, the inherent smoothing effect of classical iterative methods is often unsatisfactory. Moreover, methods which set up smoothers in a global frequency sense have difficulties to model the required behavior. Therefore, we analyze the smoothing effect of MSAI preconditioners where we can impose frequency subspace-specific behavior by incorporating probing conditions. Related to SAI, we denote MSAI as the static part of MSPAI.

As a second application, we consider the solution of inverse problems, i.e., the recovery of original information from observed noisy data. Such problems arise, for example, during tomographic and astronomic image reconstruction, in geophysics, or remote sensing. Large-scale inverse problems such as deblurring of images are usually only manageable by iterative regularization methods. However, constructing preconditioners for these methods is even more challenging than for the noise-free case. The preconditioner should treat the low frequency subspace (signal part) efficiently having no effect on the high frequency subspace (noise part). Again, by using MSAI, we derive regularizing preconditioners with a different action on these different frequency subspaces modelled by probing conditions. We additionally propose further approaches for improved reconstructions based on common direct and iterative regularization techniques.

1.2. Outline of the Thesis

We facilitate an overview of the main parts of the thesis which fluently lead to new contributions: both sections of Chapter 3 end with new algorithmic variants while the remaining chapters on the whole thematically address new approaches.

Chapter 2 provides the basics and the background for the covered topics. We start with direct methods which are generally used for the solution of dense and/or small linear systems. We focus on the LU, the Cholesky, and the QR factorization while the latter is introduced in connection to least squares problems. For the construction of sparse approximate inverses, these factorizations are usually used in order to solve the occurring lower dimensional subproblems. Passing on iterative methods for the solution of large and sparse systems, we first identify the term *sparse matrix* and storage schemes typically used by implementations operating on this type of matrices. Two well-known Krylov subspace methods—which we will use as solvers in our numerical experiments—are described: the conjugate gradient (CG) method for the SPD case and the biconjugate gradient stabilized (BiCGSTAB) method for the general case. In connection to CG, we highlight the dependency between its convergence rate and the spectral properties of the underlying system. Thus, we point out the indispensability of incorporating preconditioners in order to obtain (fast) convergence in iterative methods. The first part ends with an overview on various types of preconditioners, also with respect to the possibility of a parallel setup. In Chapter 5, we analyze the smoothing property of MSAI. Therefore, we provide basic knowledge of using smoothers in multigrid methods and introduce the local Fourier analysis as well as generating functions. These represent tools which allow to accomplish a smoothing analysis. The third part is concerned with the solution of discrete ill-posed inverse problems to recover blurred and with noise affected signals. To begin with, we identify the request for regularization techniques due to the presence of noisy data. We introduce the Tikhonov-Phillips regularization (TPR)

in general form, the truncated singular value decomposition (TSVD), and the conjugate gradient least squares (CGLS) method. These direct and iterative regularization methods, respectively, are typically used in practice. The quality of the solution obtained from any of these methods depends on the choice of the regularization parameter which represents a balancing factor between fitting the model and imposing regularity on the solution. As its optimal value is problem dependent and not known in advance, we finally present common techniques for its estimation.

Chapter 3 elaborates on sparse approximate inverses. The first part considers the Frobenius norm minimization approach for general matrices. We start with the basic idea behind this minimization ansatz in order to clarify the involved inherent parallelism. At the same time, we point out the challenge of providing an a priori sparsity structure $\mathcal{J}$ (pattern) which is required for the setup of the preconditioner. In this context, we summarize typical pattern choices and advert to a robust possibility avoiding zero columns in the preconditioner. The following part accurately describes the SPAI algorithm as it is a central component of the thesis. Its adaptive capturing of promising pattern candidates overcomes the drawback that an a priori choice of $\mathcal{J}$ may yield arbitrarily poor approximations. By incorporating targeting and probing in the SPAI formulation, we end up with the generalized modified SPAI (MSPAI) ansatz which allows to impose certain properties on the sparse approximate inverse. We enhance its classical global probing approach by introducing mask probing which enables us to impose column-specific probing conditions. Both probing approaches are exploited in order to derive smoothing preconditioners in Chapter 5 and regularizing preconditioners in Chapter 6. As the involved applications demand for preconditioners with different behavior on the low and high frequency subspace, respectively, we provide heuristic modelling of these. A different approach for the setup of sparse approximate inverses is based on a series of matrices to avoid the rather expensive construction of SPAI. After describing the idea known as multistep successive preconditioning (MSP), we introduce a new variant based on permutations and recursive refinement, the multistep MSPAI (MMSPAI). We end up with a comparison between these two algorithms. The second part considers the SPD case where factorized sparse approximate inverses based on K-condition number minimization (FSPAI) are elaborated. As far as possible, this section is structurally identical to the previous one. Using a different but mathematically equivalent minimization approach, the strong relationship between SPAI and FSPAI becomes (and remains) clear. A following artificial example illustrates that FSPAI's index choosing criterion represents a heuristic: although the preconditioner may satisfy the given tolerance criterion, the preconditioned system may still have an arbitrary large K-condition number. This motivates for deriving an exact version of FSPAI (EFSPAI) which overcomes this drawback. Choosing the pattern candidates according to their exact reduction avoids unbounded values of the K-condition number. Being more expensive—both in terms of computation and memory—we end up with notes on possible Schur complement update mechanisms in EFSPAI in order to minimize its computational costs. A final comparison between FSPAI and EFSPAI closes this discussion. Note that results arising from parallel environments are only provided in Chapter 7.

Chapter 4 generalizes FSPAI to its block variant denoted as BFSPAI. Recent research on block sparse approximate inverse preconditioners and the related results, compared to the scalar case, motivate to close the open gap for FSPAI, as well. After describing the blockwise SPAI (BSPAI)—which only appeared in one technical report so far—we show that the derivation for BFSPAI resembles the formulae of FSPAI. Consequently, BFSPAI inherits all advantages of FSPAI: it remains inherently parallel and robust, and captures promising (block) entries adaptively. Our preliminary comparative results between a scalar

and blockwise factorized version indicate the efficiency of block sparse approximate inverse preconditioners and recommend for further research.

Chapter 5 analyzes MSAI's smoothing property for the purpose of being applied as a smoother in multigrid methods. The chapter considers 1-dimensional (1D) and 2-dimensional (2D) problems separately but structurally in the same way. By focusing on discretizations of the Laplace operator, we are able to use generating functions in order to derive optimal smoothers. In this context, we translate the occurring minimization conditions into local mask probing conditions for MSAI. These, we transfer to (classical) global probing conditions afterward. Besides providing numerical results in the mentioned cases, we additionally show the effect of global probing vectors with action on the underlying system as well as on problems with varying coefficients.

Chapter 6 considers the (improved) reconstruction of discrete ill-posed problems using common regularization techniques. In the first section, we incorporate MS(P)AI's probing ansatz to derive regularizing preconditioners and apply them in CG and CGLS. As a transition from smoothing preconditioning to this second field of application, we use a similar methodology as in Chapter 5: based on generating functions of structured 1D and 2D problems we derive optimal regularizing preconditioners and translate the occurring minimization conditions into local and global probing conditions. In this context, we also analyze the effect of adjusting MSAI's probing subspace to the structure of the underlying signal. Again, our numerical experiments with the resulting MSAIs also involve operators with varying coefficients and problems resulting from discretizations of Fredholm integral equations of the first kind. Afterward, we focus on the estimation of the optimal discrete regularization parameter which is required, for instance, in iterative regularization methods such as CGLS. Based on a smoothed point set, we propose to use a B-spline interpolant to construct the (discrete) L-curve. Comparisons with state of the art estimation techniques illustrate the effect of our approach. Motivated by taking the signal structure into account in MSAI's probing subspace, the next part introduces data based preconditioners for discrete deconvolution problems. In this context, we provide theoretical considerations and data based regularization methods which can be embedded in an outer iteration to further improve the reconstruction. A section of numerous different 1D and 2D experiments follows in which we also apply our B-spline approach. Moreover, we provide results for other iterative regularization methods and image reconstruction problems. Although there is no technique available to estimate the regularization parameter for subparts of the solution, the next section deals with a piecewise reconstruction of the signal. This part has more theoretical character providing reconstruction results under the assumption that the exact signal is available. In our last part, we focus on the generalized form of TPR and derive seminorms which incorporate the spectral data of the underlying operator. We compare this new approach with standard TPR and TPR using smoothing norms in several numerical experiments. In this context, we also illustrate the effect of combining data based regularization with smoothing norms.

Chapter 7 deals with (parallel) implementations on (M)SPAI and FSPAI. Our enhanced and new implementations are based on the parallelization concept introduced by Barnard et al. in their `spai-3.2` implementation. Therefore, we describe this concept in terms of data distribution, communication between the processing elements, and load balancing. The following section presents a new `mspai-1.2` implementation and the enhancements compared to older versions. We elaborate on its dictionary approach, realized both as a hash table as well as cache of fixed size. Matrices with a structured sparsity pattern entail the solution of many identical least squares problems throughout the setup. Hence, incorporating

a dictionary avoids redundant factorizations and solutions of the subproblems, speeding up the construction of the preconditioner. Moreover, we illustrate the effect of runtime optimizations based on memory initializations: an optimized shadow and τ_{jk} computation. We close this part by analyzing `mspai-1.2`'s strong scalability—and thus also the underlying communication concept—for massively parallel preconditioner setups of large matrices. In the second part, we pass on a new implementation of FSPAI. We present the underlying code design and give insight into the realization of the τ_{jk} computation being a critical part in the algorithm, similar to `mspai-1.2`. Subsequently, we transfer both the dictionary and the sparse decomposition approach of `mspai-1.2` to `fspai-1.1` and provide first results for SPD systems using these technical improvements. A quality and performance comparison with MATLAB and Chow's PARASAILS implementation—which computes a factorized sparse approximate inverse via a different pattern approach—illustrates the efficiency of `fspai-1.1` and the FSPAI algorithm. Again, we end up with massively parallel scalability results of our new code.

Chapter 8 closes the discussion. As the thesis addresses different topics, a summary facilitates to classify the achieved results. Promising approaches and possible enhancements motivate for further research.

Several parts of this thesis are contained in already published or submitted articles:

1. Some very few parts of Chapter 3 can be found in [92].
2. Chapter 5 and Section 6.1 are original research published in the article [90].
3. Parts of Section 6.3 are contained in [90] and [91].
4. The theory and many results of Section 6.5 can be found in [94].
5. Section 6.6 is published in [93].
6. Some parts of Chapter 7 are related to the approaches presented in [86].

1.3. Nomenclature

The notation throughout the thesis is normalized. We therefore provide some general remarks on the used symbols and identifiers.

- While capital letters such as A denote rectangular matrices of specific dimension $m \times n$, lowercase letters denote scalar values or vectors of certain dimension n.
- Calligraphy letters are mainly used for sets containing indices, e.g., $\mathcal{I}$ or $\mathcal{J}$. In connection with the sparsity structure or the pattern $\mathcal{J}$ of a matrix A—see Definition 2.1—$\mathcal{J}_k$ corresponds to the sparsity pattern of the k^{th} column of A, i.e., the k^{th} index set of $\mathcal{J}$.
- Concerning block matrices, A_k denotes the k^{th} block column of A and A_{ij} the j^{th} block in the i^{th} block row of A.
- General terms are written in *italic letters* when introduced for the first time. Identifiers used in implementations are marked in `typewriter style`. Non-mathematical terms such as software packages are denoted in SMALL CAPITAL LETTERS.

See the following tables for the fundamental abbreviations.

Symbol	Description
$\mathbb{C}^n$, $\mathbb{C}^{m\times n}$	space of all complex vectors of dimension n and complex matrices of dimension $m \times n$, respectively
$\mathbb{R}^n$, $\mathbb{R}^{m\times n}$	space of all real vectors of dimension n and real matrices of dimension $m \times n$, respectively; note that $\mathbb{R}^n \subset \mathbb{C}^n$ and $\mathbb{R}^{m\times n} \subset \mathbb{C}^{m\times n}$
$\mathbb{B}^n$, $\mathbb{B}^{m\times n}$	space of Boolean vectors of dimension n and Boolean matrices of dimension $m \times n$; note that $\mathbb{B}^n \subset \mathbb{R}^n$ and $\mathbb{B}^{m\times n} \subset \mathbb{R}^{m\times n}$
A	coefficient matrix
b	right-hand side vector
A_k	k^{th} column of A
$A_{k:n,k}$	coefficients $k,\ldots,n$ in the k^{th} column of A
$A_{i,:}$	i^{th} row vector of A, i.e., $(a_{i,1},a_{i,2},\ldots,a_{i,n}) = (A_i^T)^T$
A_{ij}, a_{ij}	element of a matrix A in its i^{th} row and j^{th} column
A^T	transpose of A
A^H	Hermitian matrix of A, i.e., the transposed and complex conjugated matrix of A
A_k^T	k^{th} row of A^T, i.e., $\big((A^T)_{k,1},(A^T)_{k,2},\ldots,(A^T)_{k,n}\big)$
$\lvert A\rvert$	matrix of absolute values of A, i.e., $(\lvert A\rvert)_{ij} = \lvert A_{ij}\rvert$
$\rho(A)$	spectral radius of A
I_n	identity matrix of dimension $n \times n$
I	identity matrix, we omit the subscript if it is clear from context
e_k	k^{th} column of I
$\otimes$	Kronecker product
$\operatorname{diag}(A)$	diagonal matrix with the diagonal elements of A
$\operatorname{diag}(a_1,\ldots,a_n)$	diagonal matrix with coefficients $a_1,\ldots,a_n$
$\det(A)$	determinant of A
$\operatorname{rank}(A)$	rank of A
$\operatorname{adj}(A)$	adjoint of A
M_{ij}^A	the (i,j) minor of A
$\operatorname{cof}(A_{ij})$	the (i,j) cofactor of A defined as $(-1)^{i+j}M_{ij}^A$
$\operatorname{cof}(A)$	matrix of cofactors of A, i.e., $[\operatorname{cof}(A)]_{ij} = \operatorname{cof}(A_{ij})$
$\operatorname{chol}(A)$	Cholesky factor of a symmetric positive definite matrix A, i.e., $A = \operatorname{chol}(A)^T \operatorname{chol}(A)$
$\operatorname{low}(A)$	lower triangular part of a matrix A
$\lVert A\rVert_2$	Euclidean norm of A
$\lVert A\rVert_F$	Frobenius norm of A
$\lVert x\rVert_1$	1-norm of a vector x, i.e., $\sum_i \lvert x_i\rvert$
$\lVert x\rVert_2$	Euclidean norm of a vector x
$\lVert x\rVert_\infty$	maximum norm of a vector x, i.e., $\max(\lvert x_1\rvert,\ldots,\lvert x_n\rvert)$
$\lVert x\rVert_p$	p-norm (Hölder norm) of a vector x
$\lVert A\rVert$, $\lVert x\rVert$	if not mentioned otherwise, $\lVert A\rVert$ and $\lVert x\rVert$ correspond to the Euclidean norm of a matrix A and a vector x, respectively
$x^{(k)}$	k^{th} iterate in iterative solution methods
α	number of pattern updates for sparse approximate inverses, Tikhonov-Phillips regularization parameter in discrete ill-posed problems
ρ	MS(P)AI probing weight
H	blur operator matrix, ill-conditioned matrix in ill-posed problems
ξ	standard deviation of a Gaussian distribution, represents magnitude of white noise

η	Gaussian white noise vector
$\tilde{x}$	approximate solution vector in regularization methods, i.e., the reconstructed signal
$U\Sigma V^T$	singular value decomposition of a matrix
$\mathcal{N}(A)$	nullspace of A
$\mathcal{R}(A)$	range of A
x_α	regularized solution with respect to the regularization parameter α
$\{x_1, \ldots, x_n\}$	sequence of vectors $x_1, \ldots, x_n$
$(x_1, \ldots, x_n)$	matrix with columns $x_1, \ldots, x_n$
$\langle x, y \rangle$	Euclidean inner product of two real vectors x, y, i.e., $x^T y$
$\lambda_{\min}$, $\lambda_{\max}$	smallest and largest eigenvalue, respectively
$\sigma_{\min}$, $\sigma_{\max}$	smallest and largest singular value, respectively
$a \leftarrow b$	assign b to a
e_N, e_{N1}	vector representing the high frequency subspace $(1,-1,1,-1,\ldots)^T$
e_{N2}	vector representing the high frequency subspace $(1,0,-1,0,1,\ldots)^T$
e_{N3}	vector representing the high frequency subspace $(0,1,0,-1,0,1,\ldots)^T$
e_S	vector representing a low frequency subspace, i.e., $(1,1,\ldots,1)^T$
κ	condition number of a linear system
ε_*	tolerance for stopping criterion in an iterative method, the asterisk is the placeholder of the method
$k_{\max}$	maximum number of iterations in iterative methods
$\vec{1}$	ones vector $(1,1,\ldots,1)^T$
$f'(x)$, $f''(x)$	Lagrange's notation for the first and second derivative of a function $f(x)$ with respect to x
$\frac{\mathrm{d}}{\mathrm{d}x} f(x)$	Leibniz's notation for the derivative of a function $f(x)$ with respect to x
$\frac{\partial}{\partial x_k} f(x_1, \ldots, x_n)$	partial derivative of a function $f(x_1, \ldots, x_n)$ with respect to x_k
∇f	gradient of a function f, i.e., $(\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n})^T$
$\mathfrak{Re}[c]$	real part of a complex number $c \in \mathbb{C}$
$\mathfrak{Im}[c]$	imaginary part of a complex number $c \in \mathbb{C}$

Acronym	Full name
1D	1-dimensional
2D	2-dimensional
AINV	approximate inverse
BSPAI	block sparse approximate inverse, dynamic setup
BFSPAI	block factorized sparse approximate inverse, dynamic setup
(P)BiCGSTAB	(preconditioned) biconjugate gradient stabilized
CSC	compressed sparse column
(P)CG	(preconditioned) conjugate gradient
(P)CGLS	(preconditioned) conjugate gradient applied to normal equations
D(s)OF	degree(s) of freedom
EFSPAI	exact factorized sparse approximate inverse, dynamic setup
EBFSPAI	exact block factorized sparse approximate inverse, dynamic setup
flops	floating point operations
FSAI	factorized sparse approximate inverse, static setup
FSPAI	factorized sparse approximate inverse, dynamic setup
GPU	graphics processing unit

HPD	Hermitian positive definite
IC	incomplete Cholesky
ILU	incomplete LU
LRU	last recently used
LS	least squares
MG	multigrid
MILU	modified ILU
MINRES	minimum residual
MPI	message passing interface
MSP	multistep successive preconditioner
MSAI	modified sparse approximate inverse, static setup
MSPAI	modified sparse approximate inverse, dynamic setup
MMSPAI	multistep modified sparse approximate inverse, dynamic setup
nnz	number of nonzeros
PDE	partial differential equation
PE	processing element
ROI	regularization with outer iterations
ROIP	regularization with outer iterations and partitioning
RRE	relative reconstruction error
SAI	sparse approximate inverse, static setup
SAINV	stabilized approximate inverse
SMP	symmetric multi-processing mode, i.e., using one core per node
SPAI	sparse approximate inverse, dynamic setup
SPD	symmetric positive definite
SVD	singular value decomposition
TPR	Tikhonov-Phillips regularization
TSVD	truncated singular value decomposition
VN	virtual node mode, i.e., using four cores per node

Chapter 2

Solution of Linear Systems and Discrete Inverse Problems

This chapter facilitates the fundamentals for the solution of linear systems and discrete ill-posed inverse problems. Historically discovered earlier, direct methods are often the basis for more sophisticated algorithms and are mainly used to solve dense linear systems nowadays. After starting with a short introduction on common approaches used in this work, we pass on to iterative methods for the solution of large sparse systems of equations. The classification of a linear system to be sparse is followed by sparse storage schemes, which are typically used for the implementation of sparse linear algebra algorithms. We present classical stationary and nonstationary methods focusing on CG and BiCGSTAB for the symmetric positive definite and the general case, respectively. In connection with CG we highlight the dependency between the convergence rate of iterative methods and the spectral condition number of the system matrix. This establishes our motivation for preconditioning as an acceleration technique by modifying the spectrum of the underlying system. We elaborate the main aims of preconditioners and give a short survey on splitting, incomplete factorization, and approximate inverse preconditioners.

As we will investigate the smoothing property of MSAI in Chapter 5, we briefly describe the sense of applying smoothers in multigrid methods. We introduce the local Fourier analysis and generating functions which provide a strong tool to analyze the smoothing property of operators.

In the last part, we address linear model problems in which the system matrix is severely ill-conditioned or even singular and the right-hand side, i.e., the observed signal, is affected with noise. Here, regularization techniques are essential in order to solve these so-called ill-posed inverse problems and to reconstruct the signals with proper quality. After briefly describing the discrete Picard condition, which is necessary to be satisfied to obtain a meaningful solution, we present regularization methods which are typically used in practice and within this work. In the class of direct regularization methods, we focus on the Tikhonov-Phillips regularization and the TSVD. As iterative regularization method we choose CGLS. The quality of the solution obtained from any regularization method depends on the choice of the regularization parameter. As the optimal parameter is problem dependent and not known in advance, robust techniques for its estimation are necessary and therefore presented as well.

A linear equation expresses a relationship between given quantities and quantities yet to be determined. In contrast to nonlinear equations, it satisfies the properties of superposition and homogeneity, and can mathematically be expressed via

$$a_1x_1 + a_2x_2 + \ldots + a_nx_n = \sum_{i=1}^{n} a_ix_i = b,$$

where the linear combination of the known n coefficients a_i and the unknown n variables x_i has to yield a scalar right-hand side b. As the solution of many physical problems not only solves one but a set of m linear equations simultaneously, the problem is usually characterized in the compact matrix notation

$$\begin{pmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \ldots & a_{mn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = Ax = b, \tag{2.1}$$

with $A \in \mathbb{R}^{m\times n}$ being the coefficient matrix, $b \in \mathbb{R}^m$ the given right-hand side, and $x \in \mathbb{R}^n$ the unknown solution vector. In this thesis, we will mainly deal with problems where A is quadratic, i.e., $A \in \mathbb{R}^{n\times n}$, and where the solution of (2.1) exists and is unique.

2.1. Direct Methods for Dense Linear Systems

Linear systems where all or at least a high amount of coefficients a_{ij} are nonzero, i.e., $a_{ij} \neq 0$ for most i, j, can be classified to be dense. The counterpart is represented by sparse linear systems which will be introduced in Section 2.2. The solution of (dense) linear systems can be obtained using direct methods which, in a finite number of steps, transform A to some special form, making (2.1) easily solvable:

- If A is a regular diagonal matrix, i.e., $a_{ij} = 0$ for $i \neq j$ and $a_{ij} \neq 0$ for $i = j$, the solution is trivially gained by $x_i = \frac{b_i}{a_{ii}}$ for $i = 1, \ldots, n$.
- If A is an upper triangular matrix, i.e., $a_{ij} = 0$ for $i > j$ and $a_{ii} \neq 0$, we receive the solution by applying a *backward substitution*. Starting with $x_n = \frac{b_n}{a_{nn}}$ the remaining components of x can be computed stepwise by

$$x_i = \frac{b_i - \sum_{j=i+1}^{n} a_{ij}x_j}{a_{ii}} \qquad \text{for} \qquad i = n-1, n-2, \ldots, 1. \tag{2.2}$$

- For lower triangular A, x is obtained analogously to (2.2) by using a *forward substitution*.

2.1.1. LU factorization

An upper triangular matrix U can be obtained by applying the *Gaussian elimination* to A. Following [35, 53, 88, 127], we use the recursion $U := A^{(n)} = L^{(n-1)}A^{(n-1)}$ with $A^{(1)} := A$, which in expanded form can be written as

$$U = L^{(n-1)}L^{(n-2)} \cdots L^{(1)}A^{(1)} =: \tilde{L}A.$$

Here, the k^{th} strictly lower triangular matrix

$$L^{(k)} := I - l^{(k)} e_k^T \qquad \text{with} \qquad l^{(k)} := \left(0, \ldots, 0, \frac{a_{k+1,k}^{(k)}}{a_{kk}^{(k)}}, \ldots, \frac{a_{nk}^{(k)}}{a_{kk}^{(k)}}\right)^T$$

eliminates all entries below the main diagonal in the k^{th} column of A, i.e., performs the k^{th} elimination step. As the inverse is given by

$$\left(L^{(k)}\right)^{-1} = I + l^{(k)} e_k^T \qquad \text{and} \qquad \tilde{L}^{-1} = \prod_{k=1}^{n-1} \left(L^{(k)}\right)^{-1} = I + \sum_{k=1}^{n-1} l^{(k)} e_k^T =: L, \tag{2.3}$$

the Gaussian elimination yields the factorization $A = LU$. For proofs of (2.3) see, e.g., [127]. Once an LU factorization is available, the solution of $LUx = b$ is obtainable by applying the forward substitution $Ly = b$ followed by the backward substitution $Ux = y$ for different right-hand sides b. In such cases, where L has to be available explicitly, the coefficients of L are stored below the main diagonal in A itself.

In its basic version, the Gaussian elimination is only applicable to matrices where the pivotal elements $a_{kk} \neq 0$. However, this is not guaranteed to be satisfied for all nonsingular matrices. Moreover, for very small a_{kk}, there may occur large intermediate results amplifying round-off errors. A resort to these problems is to apply the so-called *pivoting* technique during the elimination process.

- Partial pivoting performs a 1D search along the rows or columns of A in order to find a nonzero pivotal element and to apply a row or column permutation of A. To ensure better stability, the permutation should yield diagonal elements which satisfy

$$|\tilde{a}_{kk}^{(k)}| = \max_{k \leq i \leq n} |a_{ik}^{(k)}|.$$

- Complete pivoting determines a two-sided permutation such that

$$|\tilde{a}_{kk}^{(k)}| = \max_{\substack{k \leq i \leq n \\ k \leq j \leq n}} |a_{ij}^{(k)}|,$$

  i.e., a 2D search is performed to determine the largest entry within the whole submatrix.

Once the permutation matrices $P := P^{(n-1)} P^{(n-2)} \cdots P^{(1)}$ and $Q := Q^{(n-1)} Q^{(n-2)} \cdots Q^{(1)}$ are determined, the factorization reads as $PA = LU$ for partial pivoting and $PAQ = LU$ for complete pivoting. Applying the approaches requires additional costs of $\mathcal{O}(n^2)$ and $\mathcal{O}(n^3)$ comparisons, respectively. For further reading, see [53, 121, 127].

The costs of performing an LU factorization of a matrix A of dimension n without pivoting are

$$\#\text{flops}_{\text{LU}} = \frac{2}{3} n^3 \in \mathcal{O}(n^3). \tag{2.4}$$

This high amount of work to compute a solution makes the LU factorization unfeasible for large problems. Even modern CPUs with a peak performance of around 100 Gigaflops such as the Intel® Core™ i7-980X or i7-990X [136] Extreme Edition, would need around 77 days of solely computational time for a matrix of size $n =$ 1E+06. In contrast to this, iterative methods—see Section 2.2.2—can solve dense matrix problems in $\mathcal{O}(n^2)$ flops. When using the *fast Fourier transform* (FFT) or fast wavelet methods, special problems become even

solvable in a nearly optimal amount of flops, i.e., in $\mathcal{O}(n)$ or $\mathcal{O}(n\log(n))$ flops [35]. Note that for sparse matrices iterative methods become of linear order $\mathcal{O}(n)$.

2.1.2. Cholesky Factorization

If computing the LU factorization via Gaussian elimination, the factorization $A = LDM^T$ can be determined from $U = DM^T$, where $D := \text{diag}(U) = \text{diag}(u_{11},\ldots,u_{nn})$. This factorization exists if all leading principal minors[1] of A are nonsingular. For a proof see, e.g., [53, 121]. When applying this factorization in order to specify x in (2.1), we have to solve the three linear systems $Ly = b$, $Dz = y$, and $M^Tx = z$.

For symmetric A, the factorization simplifies to $A = LDL^T$ because $M = L$. If A is additionally positive definite (SPD), i.e., $x^TAx > 0\ \forall x \in \mathbb{R}^n \setminus \{0\}$, it can be proven that the diagonal entries $d_{kk} > 0$ [53, 121]. Hence, with $D^{\frac{1}{2}} = \text{diag}(\sqrt{d_{kk}})$ the resulting factorization

$$A = LD^{\frac{1}{2}}D^{\frac{1}{2}}L^T = \tilde{L}\tilde{L}^T$$

is called *Cholesky factorization* with lower triangular Cholesky factor $\tilde{L} =: \text{chol}(A)$.

Concerning the computational costs, there are various algorithms to solve SPD systems directly. Many of them differ in the constant factor. Usually, the computation of the Cholesky factor is twice as fast as the computation of the LU factorization, i.e.,

$$\#\text{flops}_{\text{Cholesky}} = \frac{1}{3}n^3 \in \mathcal{O}(n^3). \tag{2.5}$$

Remark 2.1 *In literature sometimes only multiplications and divisions are taken into account for the complexity analysis. This is because additions and multiplications usually occur in pairs and are considered as a whole. In this case the costs for an LU and Cholesky factorization result in*

$$\#flops_{LU} = \frac{1}{3}n^3 \in \mathcal{O}(n^3) \qquad \textit{and} \qquad \#flops_{Cholesky} = \frac{1}{6}n^3 \in \mathcal{O}(n^3),$$

respectively [53, 109]. In the following, we consider additions and multiplications separately, similar to (2.4) and (2.5).

2.1.3. Least Squares and QR Factorization

We briefly introduce two terms required for the motivation of the following part. First, the *condition* of a problem is the sensitivity of the solution with respect to errors in the input data. A problem is *well-conditioned* if small errors in the data produce small errors in the solution. A problem is *ill-conditioned* if small errors in the data may produce large errors in the solution. We will refer to the condition number κ in the context of linear systems in

1 The k^{th} order leading principal minor of an $n \times n$ matrix is the determinant of the submatrix obtained by deleting the last $n-k$ rows and columns. An $n \times n$ matrix has n leading principal minors.

Section 2.3. Second, a method of computation is characterized being (numerically) *stable*[2] if it does not amplify the relative input errors of a well-conditioned problem, otherwise it is considered *unstable*. These two properties have an effect on the numerical solution of linear systems. Consider, for instance, the following cases:

- The computation of an LU factorization leads to a successively transformation of the system A when applying the elimination matrices $L^{(k)}$. As their condition numbers are not bounded from above [46], we have to expect $\kappa(L) > 1$ and that the condition number of the transformed system will evidently change to

$$\kappa(A) < \kappa(LA) \leq \kappa(L)\kappa(A).$$

  Although in practice the resulting $L^{(k)}$ are usually not ill-conditioned, it is not guaranteed that the LU factorization behaves always stable.

- In case that A is singular or (2.1) is overdetermined with $A \in \mathbb{R}^{m\times n}$, $m \geq n$, and $b \in \mathbb{R}^m$, the solution can be obtained by minimizing the Euclidean length of the *residual* $r = Ax - b$, i.e., to solve the *least squares* (LS) problem

$$\min_x \|Ax - b\|_2^2. \tag{2.6}$$

  Following [121, 127], the solution of (2.6) is the solution of the system of *normal equations* $A^TAx = A^Tb$. If A has full rank, the solution exists, is unique, and the matrix A^TA is SPD. However, the product A^TA may lose its positive definiteness or nonsingularity due to round-off errors. Consequently, it has a larger condition number $\kappa(A^TA) = \kappa^2(A)$ [46] and the usage of a Cholesky factorization to solve the problem becomes unstable or even useless.

A remedy is the QR factorization

$$A = QR = (\hat{Q},\tilde{Q})\begin{pmatrix}\hat{R}\\0\end{pmatrix} = \hat{Q}\hat{R},$$

where Q is orthogonal, $\hat{Q}\hat{R}$ is unique, fulfilling $\hat{Q}^T\hat{Q} = I$ with $\hat{Q} \in \mathbb{R}^{m\times n}$, and $\hat{R} \in \mathbb{R}^{n\times n}$ is upper triangular. By definition, an orthogonal matrix Q is a square real valued matrix whose columns and rows are orthogonal unit vectors, i.e., it satisfies the properties $Q^T = Q^{-1}$, $\|Q\|_2 = 1$, and $\kappa(Q) = 1$. Its condition number invariance yields the desired stable transformation. Furthermore, the invariance of the Euclidean norm with respect to orthogonal transformations allows for the stable solution of LS problems. Using $c := Q^Tb = (\hat{c},\tilde{c})^T$ with $\hat{c} \in \mathbb{R}^n$ and $\tilde{c} \in \mathbb{R}^{m-n}$, the solution of (2.6) can be obtained via

$$\min_x \|QRx - b\|_2^2 = \min_x \|Rx - Q^Tb\|_2^2 = \min_x \left\|\begin{pmatrix}\hat{R}\\0\end{pmatrix}x - \begin{pmatrix}\hat{c}\\\tilde{c}\end{pmatrix}\right\|_2^2 \quad \Leftrightarrow \quad x = \hat{R}^{-1}\hat{c}, \tag{2.7}$$

i.e., by performing the QR factorization of A, a matrix-vector multiplication $\hat{c} := \hat{Q}^Tb$, and a backward substitution $x = \hat{R}^{-1}\hat{c}$. Note that in practice the latter analytic expression is not calculated, i.e., instead of inverting $\hat{R}$ explicitly, the corresponding linear system $\hat{R}x = \hat{c}$ is solved.

2 We refer to stability in the sense of forward analysis. See, e.g., [46] for a formal description and the difference to backward stability.

Q can be constructed by *Givens rotations*, *(modified) Gram-Schmidt orthogonalization*, or by *Householder reflections* [35]. Here, we shortly describe the latter according to [100, 121]. Given a vector $a \in \mathbb{R}^n$ and

$$u := \frac{a - \rho e_1}{\|a - \rho e_1\|_2} \qquad \text{with} \qquad \rho := -\operatorname{sign}(a_1)\|a\|_2,$$

then $Ha = \rho e_1$ when using the Householder reflection $H := I - 2uu^T$ with $\|u\|_2 = 1$. We choose $e_1 \in \mathbb{R}^n$ to be of equal dimension with respect to a; here $e_1 = (I_n)_1$. I.e., by reflection of a onto e_1, all components of a are set to zero except the first one. Note that ρ is chosen to be $-\operatorname{sign}(a_1)\|a\|_2$ to avoid numerical cancellation during the computation of u_1. H satisfies the properties of symmetry, orthogonality, and idempotence. In order to transform A to upper triangular form R, the first $k-1$ components of each column A_k have to be left unaltered, while the components $k+1,\ldots,n$ are to be set to zero. Consequently, with $\tilde{a} := A^{(k)}_{k:n,k} \in \mathbb{R}^{n-k+1}$, the k^{th} Householder matrix takes the form

$$H^{(k)} := \begin{pmatrix} I_{k-1} & 0 \\ 0 & \tilde{H}^{(k)} \end{pmatrix} \qquad \text{with} \qquad \tilde{H}^{(k)} := I_{n-k+1} - \frac{2(\tilde{a} - \rho e_1)(\tilde{a} - \rho e_1)^T}{\|\tilde{a} - \rho e_1\|_2^2}$$

and $\rho = -\operatorname{sign}(\tilde{a}_1)\,\|\tilde{a}\|_2$. With $l := \min\{m-1, n\}$, we obtain a QR factorization of A using Householder reflections by

$$H^{(l)}H^{(l-1)}\cdots H^{(1)}A = R = \begin{pmatrix} \hat{R} \\ 0 \end{pmatrix} \qquad \text{where} \qquad Q := \prod_{k=1}^{l} H^{(k)}.$$

Summarizing, the Householder QR factorization is similar to the Gaussian elimination for the LU factorization: the construction of the Householder vectors $u^{(k)} := \tilde{a} - \rho e_1$ corresponds to the computation of the multipliers in the Gaussian elimination and both methods transform A to an upper triangular matrix by subsequent updates of the remaining unreduced portions. The computational costs of common algorithms, e.g., presented in [53], which compute merely the QR factorization and additionally the solution of the linear system are

$$\#\text{flops}_{\text{QR}} = 2n^2\left(m - \frac{n}{3}\right) \in \mathcal{O}(n^3) \qquad \text{and} \qquad \#\text{flops}_{\text{QR-solve}} = \frac{4}{3}n^3 \in \mathcal{O}(n^3),$$

respectively. Hence, this direct approach is more expensive than an LU factorization, but it is stable.

2.2. Iterative Methods for Sparse Linear Systems

Many practical numerical applications involve the solution of large linear systems of equations. In order to handle high-dimensional settings in terms of computation effort and memory requirements, it is essential to use problem formulations which produce sparse matrices. A typical scenario occurs in the finite approximation of *partial differential equations* (PDEs) which are used to characterize physical models. Here, the need for highly accurate results usually requires domain discretizations with small mesh size leading to large and sparse matrices.

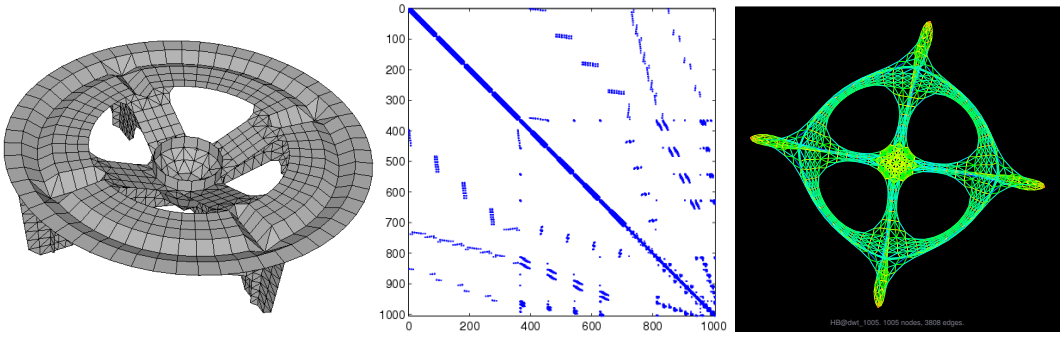
Although the boundaries between a matrix to be classified as dense or sparse often overlap, common formal descriptions [35] specify a matrix A to be sparse if

- the algorithms and applications have advantages in terms of storage and computation time when exploiting the sparsity and the structure of the matrix,
- the *number of nonzeros* (nnz) is of the order of the underlying model problem, i.e., $n^2 \gg \text{nnz}(A) \approx cn \in \mathcal{O}(n)$, where $c \in \mathbb{R}$ is often predictable, e.g., depending on the element function type of the *finite element method* (FEM), the discretization stencil, etc.,
- $\exists p \ll n$ such that every row and column of A has a maximum of p nonzero entries.

Furthermore, a sparse matrix is called *structured* if its nonzeros form a regularly structured, possibly symmetric, pattern.

Definition 2.1 (Sparsity pattern of a matrix) *We constitute a pattern as $\mathcal{J} \in \mathbb{B}^{n \times n}$. Hence, we refer to the sparsity pattern of a matrix $A \in \mathbb{R}^{n \times n}$ with $\mathcal{J}(A)$ which is a Boolean matrix with $\mathcal{J}(A)_{ij} = 1$ for $a_{ij} \neq 0$ and $\mathcal{J}(A)_{ij} = 0$ otherwise.*

In structured sparse matrices, for instance, the elements can be arranged along few subdiagonals or be clustered in (dense) blocks of equal size. Typical examples are the finite difference discretizations of regular grids using certain stencils, e.g., the 1D or 2D Laplacians. Figure 2.1 illustrates an example of a finite element model and the pattern of a corresponding sparse matrix. Matrices with an irregular pattern are denoted to be *unstructured*.



(a) Finite element model. (b) Pattern of sparse matrix. (c) Undirected graph.

Figure 2.1.: Exemplary finite element model of a baseplate (taken from [116]), its corresponding sparse matrix structure (generated with MATLAB [108]), and the corresponding undirected graph (taken from [44]). The matrix DWT_1005 belongs to the Harwell-Boeing collection from [116].

2.2.1. Sparse Storage Schemes and Graph Theory

On a 32-bit system with a memory consumption of 4 bytes per floating point number in single precision, a tridiagonal system of order $n = 1\text{E}{+}07$ would require 363.8 Terabytes when stored in a naive dense format. Omitting the zero entries and storing the same matrix using, for instance, the *compressed sparse column* (CSC) storage, only 267.0 Megabytes become necessary, which saves a vast amount of memory. Furthermore, in iterative solution

methods (see Section 2.2.2) the matrix-vector product involving the system matrix represents the most time consuming part. Hence, the usage of an efficient data structure to allow fast computations and reasonable memory management becomes indispensable. Last but not least, a compressed storage saves a lot of communication between *processing elements* (PE) in a parallel environment as only the nonzeros have to be exchanged.

Following [3, 123], we briefly introduce the CSC format which is often used in numerical linear algebra software, e.g., in CXSparse by Davis [42, 43] or `spai-3.2` by Barnard et al. [10, 11]. We use it with additional information for our implementations on sparse approximate inverses as well (see Chapter 7). The information is compressed by eliminating all zeros and storing it in three arrays: `val[]` holding the matrix values, `row[]` containing the corresponding row indices, and `col[]` whose indices point to the beginning of each column in the arrays `val[]` and `row[]`. The last element `col[`n`]` $= \mathrm{nnz}(A)+1$ points to a fictitious $(n+1)^{\text{th}}$ column. See Figure 2.2 for an example.

$$A = \begin{pmatrix} 1.0 & & & 7.0 \\ & 2.0 & -5.0 & \\ 4.0 & & 6.0 & \\ & & & 3.0 \end{pmatrix} \xrightarrow{\text{CSC storage}} \begin{aligned} \texttt{val[]} &= \{1.0, 4.0, 2.0, 6.0, -5.0, 7.0, 3.0\} \\ \texttt{row[]} &= \{1,3,2,3,2,1,4\} \\ \texttt{col[]} &= \{1,3,4,6,8\} \end{aligned}$$

Figure 2.2.: Example for the compressed sparse column (CSC) storage of a matrix A.

A second natural sparse matrix format is the *coordinate* or *triplet format* which contains the values and their corresponding row and column indices, respectively. Matrix Market [116] and the UF Sparse Matrix Collection [44] are exemplary repositories containing sparse test matrices which are available in such a format [27]. Many numerical results within this thesis resort to these collections. While the CSC format allocates $2\cdot\mathrm{nnz}(A)+n+1$ of elements, the coordinate format requires $3\cdot\mathrm{nnz}(A)$ storage costs. Note that the transposed variant of CSC is the *compressed sparse row* (CSR) format and that no explicit order is required for the aforementioned formats. There are many other storage schemes—see [3, 123]—all mainly differing in access times due to indirect addressing. The choice of the best format depends on the application. Implementations of sparse approximate inverses based on Frobenius norm minimization mainly use the CSC/CSR format—or modifications of it—because the preconditioner is computed column/row-wise.

It is worth mentioning *adjacency graphs* which conveniently express the structure of sparse matrices. The adjacency graph $\mathcal{G}(A)$ of a sparse matrix A is defined as a tuple $\mathcal{G} := (\mathcal{V}, \mathcal{E})$ of vertices $\mathcal{V} := \{1, \ldots, n\}$ and edges $\mathcal{E} := \{(i,j) : a_{ij} \neq 0\}$ [123]. Graph based representations are invariant under permutations of A and make it easier to deduce properties from the graph than from the pattern of A. A graph based representation is versatile:

- In order to avoid prohibitive growth in memory consumption, and in turn, of the number of operations needed, factorization methods require preprocessing steps to circumvent dense factors. By graph reordering the unknowns in the matrix are permuted such that the arising *fill-in* is minimized. See, for example, Figure 2.3 where an LU factorization of the left arrow matrix would cause fill-in—at positions ○—yielding a dense result while the permutation by simply relabeling the vertices of the adjacency graph will avoid any fill-in yielding a sparse result.

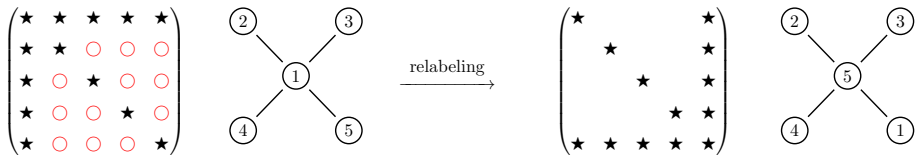


Figure 2.3.: Example of avoiding fill-in during an LU factorization by graph reordering.

- To make fast bandwidth-based methods applicable, graph algorithms like *Cuthill-McKee* [40], *reverse Cuthill-McKee* [121, 123], *minimum degree* variants [43], or others such as *Gibbs-Poole-Stockmeyer*, aim to reduce the bandwidth or the profile.
- To enable the stability or solvability of certain algorithms special permutations become necessary. Desirable pivots can be ensured by permuting large entries towards the diagonal, e.g., by maximum weight matching algorithms [59]. This technique can be also used to obtain a certain block structure with regular blocks.
- To facilitate solving the linear system in a parallel environment, independent sets are identified by graph partitioning.

Refer to Figure 2.1 (c) for a representation of the undirected graph of the symmetric matrix DWT_1005. Note that modern software packages such as CXSparse offer implementations of direct methods and fill-reducing orderings catered to sparse linear systems. Operating merely on the sparse representation of the underlying problem, they often outperform libraries working on dense array formats such as Lapack [1]. Our implementations on sparse approximate inverses provide the possibility to choose between both representations; see Chapter 7.

2.2.2. Classical Iterative Methods

Direct methods, as presented in Section 2.1, are a reasonable choice for small-size problems because they are less sensitive to perturbations in the input data, are affordable, and very reliable [3]. However, for high-dimensional problems, as they typically arise in the discretization of PDEs, the costs to compute a solution, both in time and space, prevent them from being a practical choice. Iterative methods come along with a list of advantages motivating for their application in large and sparse linear systems:

- They are suitable whenever it is possible to take the sparsity and structure of A into account, as factorization methods destroy this matrix property.
- There is no need for A being given explicitly, because the elementary operations within the iterations are matrix-vector products. One only needs a function which returns the action of the matrix on a given vector. This makes iterative methods—to some extend—comparatively simple to parallelize.
- A priori knowledge about the solution can be integrated into the solution process by an initial approximation, i.e., an initial guess.
- They can terminate when a desired accuracy is achieved. It makes no sense to solve the system with higher accuracy than the discretization error introduced by the discretized underlying model.

- They are optimal in the sense that they solve sparse systems with $\text{nnz}(A) \in \mathcal{O}(n)$ in $\mathcal{O}(n)$ flops per iteration, i.e., $\mathcal{O}(1)$ flops per nonzero.

The disadvantage with iterative methods is that they depend on the spectral properties of A and the condition number, causing them to converge very slowly or even diverge when A is ill-conditioned. This is where preconditioning becomes essential. We will motivate for this technique in Section 2.3.

Following [3, 121], the fundamental idea behind iterative methods is to use a mapping $\Phi : \mathbb{R}^n \to \mathbb{R}^n$ which constructs a sequence of solutions $x^{(k)}$ via a linear fixed point iteration

$$\Phi(x^{(k)}) = x^{(k+1)} \in \mathbb{R}^n,\ k \geq 0 \quad \text{with} \quad x^{(k+1)} = Rx^{(k)} + b \quad \text{and} \quad R := I - A \tag{2.8}$$

satisfying $x = Rx + b$, i.e., $Ax = b$ in terms of linear systems corresponding to (2.1). Such an iterative method is called to be *consistent*. It is defined to converge to the desired exact solution, i.e.,

$$\lim_{k\to\infty} x^{(k)} = x = A^{-1}b \qquad \text{with the fixed point} \qquad \Phi(x) = x.$$

In practice, the iteration is stopped after a finite number of steps such that $\|x^{(k)} - x\| < \varepsilon$, for a given tolerance $\varepsilon > 0$, in a specific vector norm, usually the Euclidean norm. However, as the exact solution x is not available, the iteration is usually continued until $\|r^{(k)}\|_2 \leq \varepsilon$. Theorem 2.1 gives an important property which has to be satisfied for the iteration matrix R when applying iterative methods.

Theorem 2.1 (Convergence of iterative methods) *For a linear and consistent fixed point iteration with any given initial guess x_0 holds*

$$\lim_{k\to\infty} x^{(k)} = x \qquad \Leftrightarrow \qquad \rho(R) < 1$$

where $\rho(R) := |\lambda_{max}(R)|$ is the spectral radius of the iteration matrix R.

Stationary Methods

Based on additive splittings of $A = M - N$, classical iterative methods can largely be considered as principles to improve the fixed point iteration (2.8) by

$$x^{(k+1)} = M^{-1}Nx^{(k)} + M^{-1}b \qquad \text{which is equivalent to} \qquad M^{-1}Ax = M^{-1}b \tag{2.9}$$

and denoted as *preconditioned* system with *preconditioner* matrix M (see Section 2.3.3). As convergence is achieved for $\rho(M^{-1}N) = \rho(I - M^{-1}A) < 1$, the transformed system $M^{-1}A$ should be better conditioned than A itself. We refer to [3, 35, 109, 121, 123] for the derivations of the various iteration schemes for stationary methods and briefly summarize them in Table 2.1 instead. Note that there is a block version of each scheme.

Unfortunately, stationary methods may suffer under poor convergence. To overcome this drawback, acceleration can be typically obtained by using a relaxation parameter ω. For certain problems its optimal value can be analytically derived, e.g., for the *successive over*

Table 2.1.: Stationary methods and their splittings.

Method	Splitting	M	N
Richardson	$A - I + I$	I	$I - A$
Jacobi	$D - L - U$	D	$L + U$
Gauss-Seidel	$D - L - U$	$D - L$	U
Damped Jacobi	$\omega(D - L - U)$	$\omega^{-1}D$	$(\omega^{-1} - 1)D + L + U$
SOR	$\omega(D - L - U)$	$\omega^{-1}D - L$	$(\omega^{-1} - 1)D + U$
Symmetric SOR	$\omega(D - L - U)$	$\frac{D(I-\omega L)(I-\omega U)}{(2-\omega)\omega}$	$\frac{D[(1-\omega)I+\omega L][(1-\omega)I+\omega U]}{(2-\omega)\omega}$

relaxation (SOR) in case of *consistently ordered*[3] systems with *property* $\mathscr{A}$[4] [5, 123]. In complicated model problems, however, a detailed eigenvalue analysis may be necessary to determine an efficient relaxation [53]. The most famous stationary relaxation schemes are the damped Jacobi and the SOR. The theory of classical iterative methods based on matrix splittings is well-known [5, 109, 123]. Although being simple, they are not used any more for their original purpose of solving linear systems. Nevertheless, they represent standard smoothers in modern multigrid methods (see Section 2.4).

2.2.3. Nonstationary Methods: Krylov Subspace Methods

Another approach to solve $Ax = b$ is to use projection methods which compute an approximate solution

$$x^{(k)} \in x^{(0)} + \mathcal{K}^{(k)} \qquad \text{considering} \qquad (b - Ax^{(k)}) \perp \mathcal{L}^{(k)},$$

where $\mathcal{K}^{(k)} \subseteq \mathbb{R}^n$ and $\mathcal{L}^{(k)} \subseteq \mathbb{R}^n$, both of dimension $m \leq n$. This can be reformulated as a minimization of a quadratic functional in the SPD case such as

$$\Phi_1(x) := \frac{1}{2}x^T Ax - b^T x \quad \text{or the residual functional} \quad \Phi_2(x) := (Ax - b)^T(Ax - b) \tag{2.10}$$

in the general case. Here, the minimization is performed on a sequence of subspaces $\mathcal{K}^{(k)}$ of increasing dimension, denoted as *Krylov subspaces*. $\mathcal{L}^{(k)}$ is another subspace of dimension k. By recursion, a new basis vector $A^{(k-1)}r^{(0)}$ is added to the previous subspace $\mathcal{K}^{(k-1)}$ constructing $\mathcal{K}^{(k)}$, i.e.,

$$\mathcal{K}^{(k)} := \mathcal{K}^{(k-1)} \otimes \{A^{(k-1)}r^{(0)}\} \qquad \text{with} \qquad \mathcal{K}^{(1)} := \{r^{(0)}\} \qquad \text{for} \qquad k \geq 1.$$

Among the class of Krylov subspace methods, i.e., methods which generate a basis of $\mathcal{K}^{(k)}$ and seek the solution in this space, we can find iterative methods which outperform the stationary splitting based methods both in time and a faster convergence rate. Note that the Krylov subspace can be generated at no costs because a matrix-vector multiplication

3 A matrix $A = D - L - U$ is consistently ordered if $\beta D^{-1}L + \beta^{-1}D^{-1}U$, $\beta \in \mathbb{R}$, $\beta \neq 0$ has eigenvalues not depending on β.

4 A consistently ordered matrix A has *property* $\mathscr{A}$ if it can be permuted into the form $\begin{pmatrix} D_1 & -U \\ -L & D_2 \end{pmatrix}$ where D_1, D_2 are diagonal matrices.

is necessary to compute the new residual anyway. In addition, many Krylov methods are parameter-free. Therefore, they are frequently used as solvers for large and sparse linear systems. In the following, we will go more into detail for two Krylov methods which are representative in their class and used for the numerical results presented in this thesis: the *conjugate gradient* (CG) method for SPD systems and the *biconjugate gradient stabilized* (BiCGSTAB) method for the general case.

The Gradient and the Conjugate Gradient Method

Developed by Hestenes and Stiefel [76], the CG method is an orthogonal projection method based on the *Ritz-Galerkin* approach where $\mathcal{L}^{(k)} = \mathcal{K}^{(k)}$. It is applicable to SPD systems and gives the solution in n steps assuming the computation is done in exact arithmetic. Actually, in practice it even finds a sufficiently good approximation in less than n steps. Our description is based on [3, 35, 100].

For SPD matrices holds using (2.10)

$$x = \arg\min_y \Phi_1(y) \quad \Leftrightarrow \quad \nabla\Phi_1(x) = 0 \quad \Leftrightarrow \quad Ax = b$$

as $\Phi_1(x)$ is convex and its second derivative yields A itself. Hence, in order to find the minimizer we have to specify a certain direction d and perform steps of size α down the surface Φ_1, i.e., perform 1D line searches $x^{(k+1)} = x^{(k)} + \alpha_k d^{(k)}$; starting from an initial guess $x^{(0)}$. The intuitive approach is to choose

$$d^{(k)} = -\nabla\Phi_1(x^{(k)}) = r^{(k)} = b - Ax^{(k)}$$

with the resulting stepsize

$$\min_{\alpha_k} \Phi_1(x^{(k)} + \alpha_k d^{(k)}) \quad \Leftrightarrow \quad \frac{\mathrm{d}}{\mathrm{d}\alpha_k}\Phi_1(x^{(k)} + \alpha_k d^{(k)}) \stackrel{!}{=} 0 \quad \Leftrightarrow \quad \alpha_k = \frac{\|r^{(k)}\|_2^2}{\langle r^{(k)}, Ar^{(k)}\rangle}. \tag{2.11}$$

This locally minimizing approach, known as *steepest descent* or the *gradient method* [35, 121], may suffer from poor convergence due to possibly linear dependent search directions. In general, it does not converge in at most n steps. Note that slow convergence especially occurs for ill-conditioned A. Furthermore, minimizing Φ_1 means to minimize the energy norm of the error vector of the linear SPD system which leads to a solution satisfying a stable state for many areas of application.

To achieve linear independence and a global optimum for every $x^{(k)}$, we are successively seeking directions $d^{(k)}$ which solve both the k-dimensional global minimization and the 1-dimensional local minimization, i.e.,

$$\min_{x \,\in\, \mathrm{span}\{d^{(0)},\ldots,d^{(k)}\}} \Phi_1(x) \quad = \quad \min_{\alpha_k} \Phi_1(x^{(k)} + \alpha_k d^{(k)}). \tag{2.12}$$

By choosing the search directions $d^{(k)}$ to be A-conjugate to the subspace

$$\mathrm{span}\{d^{(0)},\ldots,d^{(k-1)}\}, \text{ i.e.,} \qquad \langle d^{(j)}, Ad^{(k)}\rangle = 0 \quad \text{for} \quad j = 0,\ldots,k-1,$$

the new solution $x^{(k+1)} = x^{(k)} + \alpha_k d^{(k)}$ will satisfy (2.12). Based on $d^{(0)} = r^{(0)}$, we obtain A-orthogonal directions by applying Gram-Schmidt orthogonalization to $r^{(k+1)}$ with

$$d^{(k+1)} = r^{(k+1)} + \beta_k d^{(k)} \qquad \text{and} \qquad \beta_k = \frac{\|r^{(k+1)}\|_2^2}{\|r^{(k)}\|_2^2}.$$

Once the new direction is computed, we can solve the local minimization problem according to (2.11) with $\alpha_k = \frac{\|r^{(k)}\|_2^2}{\langle d^{(k)}, Ad^{(k)}\rangle}$. Hence, the only difference between the gradient and the CG method is the choice of the search directions. Algorithm 1 summarizes the main steps. As each $x^{(k)}$ is optimal, CG converges faster than the gradient method. See Figure 2.4 (a), where the A-orthogonal search directions lead to convergence after two steps in contrast to the gradient method. However, the convergence of CG depends on the spectral properties of the underlying (preconditioned) matrix.

Efficient implementations of the CG method require one matrix-vector product, three SAXPY[5] operations, and two dot products per iteration yielding the overall costs of

$$\#\text{flops}_{\text{CG}} = k(n^2 + 5n) \in \mathcal{O}(n^2)$$

in the general case. For sparse matrices with $\text{nnz}(A) \in \mathcal{O}(n)$, CG is optimal with $\#\text{flops}_{\text{CG}} \in \mathcal{O}(n)$ due to linear costs for the matrix-vector products.

The Biconjugate Gradient Stabilized Method

For regular A in the general case, we can apply the BiCGSTAB method published by van der Vorst [139] in 1992. It is based on the *biconjugate gradient* (BiCG) method and its transpose-free variant denoted as *conjugate gradient squared* (CGS) method. The solutions in BiCGSTAB satisfy the *Petrov-Galerkin* condition $\mathcal{L}^{(k)} = \mathcal{K}_{\text{T}}^{(k)}$, i.e., the residual $r^{(k)}$ is orthogonal to the transposed Krylov subspace defined as

$$\mathcal{K}_{\text{T}}^{(k)} := \mathcal{K}_{\text{T}}^{(k-1)} \otimes \{A^{(k-1)^T}\hat{r}^{(0)}\} \qquad \text{with} \qquad \mathcal{K}_{\text{T}}^{(1)} := \{\hat{r}^{(0)}\} \qquad \text{for} \qquad k \geq 1$$

and $\langle r^{(0)}, \hat{r}^{(0)}\rangle \neq 0$ for any arbitrary $\hat{r}^{(0)} \in \mathbb{R}^n$. The biconjugation used in BiCG implies two residual and two search direction updating sequences on A and A^T, corresponding to $\mathcal{K}^{(k)}$ and $\mathcal{K}_{\text{T}}^{(k)}$, namely

$$\begin{aligned} r^{(k+1)} &= r^{(k)} - \alpha_k A d^{(k)} && \text{and} && \hat{r}^{(k+1)} &= \hat{r}^{(k)} - \alpha_k A^T \hat{d}^{(k)}; \\ d^{(k+1)} &= r^{(k+1)} + \beta_k d^{(k)} && \text{and} && \hat{d}^{(k+1)} &= \hat{r}^{(k+1)} + \beta_k \hat{d}^{(k)}. \end{aligned}$$

They are used, for instance, to compute the stepsize $\alpha_k = \frac{\langle r^{(k)}, \hat{r}^{(k)}\rangle}{\langle Ad^{(k)}, \hat{d}^{(k)}\rangle}$. Considering those vectors as polynomials of degree k

$$r^{(k)} = \sum_{i=0}^{k} \xi_i A^{(i)} r^{(0)} \;\equiv\; \phi_k(A) r^{(0)}, \qquad \hat{r}^{(k)} = \sum_{i=0}^{k} \xi_i A^{(i)^T} \hat{r}^{(0)} \;\equiv\; \phi_k(A^T)\hat{r}^{(0)}$$

and accordingly $d^{(k)} = \psi_k(A) r^{(0)}$, $\hat{d}^{(k)} = \psi_k(A^T)\hat{r}^{(0)}$, it is possible to avoid the products with A^T as

$$\langle r^{(k)}, \hat{r}^{(k)}\rangle = \langle \phi_k(A) r^{(0)}, \phi_k(A^T)\hat{r}^{(0)}\rangle = \langle \phi_k^2(A) r^{(0)}, \hat{r}^{(0)}\rangle \quad \text{and}$$

5 SAXPY (scalar alpha times x plus y) is the computation of the expression $\alpha x + y$, for x,$y \in \mathbb{R}^n$, $\alpha \in \mathbb{R}$.

Algorithm 1: Conjugate gradient (CG) method.

Input : $A \in \mathbb{R}^{n\times n}$, $x^{(0)} \in \mathbb{R}^n$, $b \in \mathbb{R}^n$, $k_{\max} \geq 0$, $\varepsilon_{\mathrm{CG}} \geq 0$

Output: The solution of $Ax = b$.

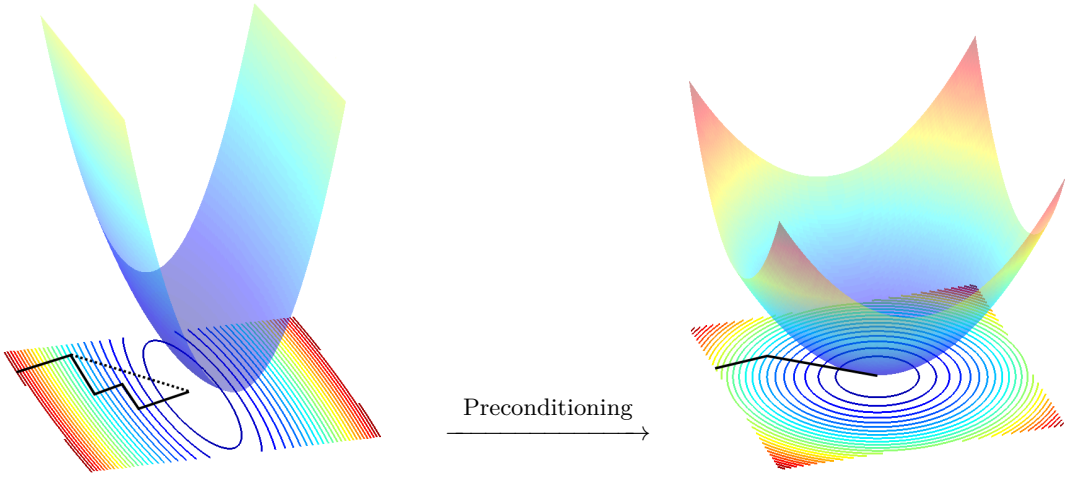
$r^{(0)} = b - Ax^{(0)}$
$d^{(0)} = r^{(0)}$
for $k = 0$ **to** $k_{\max}$ **do**
  $\alpha_k = \frac{\|r^{(k)}\|_2^2}{\langle d^{(k)}, Ad^{(k)}\rangle}$
  $x^{(k+1)} = x^{(k)} + \alpha_k d^{(k)}$
  $r^{(k+1)} = r^{(k)} - \alpha_k Ad^{(k)}$
  if $\|r^{(k+1)}\|_2 \leq \varepsilon_{\mathrm{CG}}$ **then**
    break
  end
  $\beta_k = \frac{\|r^{(k+1)}\|_2^2}{\|r^{(k)}\|_2^2}$
  $d^{(k+1)} = r^{(k+1)} + \beta_k d^{(k)}$
end

Algorithm 2: Biconjugate gradient stabilized (BiCGSTAB) method.

Input : $A \in \mathbb{R}^{n\times n}$, $x^{(0)} \in \mathbb{R}^n$, $b \in \mathbb{R}^n$, $k_{\max} \geq 0$, $\varepsilon_{\mathrm{BiCGSTAB}} \geq 0$

Output: The solution of $Ax = b$.

$\hat{r}^{(0)} = d^{(0)} = r^{(0)} = b - Ax^{(0)}$
for $k = 0$ **to** $k_{\max}$ **do**
  $\alpha_k = \frac{\langle r^{(k)}, \hat{r}^{(0)}\rangle}{\langle Ad^{(k)}, \hat{r}^{(0)}\rangle}$
  $s^{(k)} = r^{(k)} - \alpha_k Ad^{(k)}$
  $\gamma_{k+1} = \frac{\langle As^{(k)}, s^{(k)}\rangle}{\|As^{(k)}\|_2^2}$
  $x^{(k+1)} = x^{(k)} + \alpha_k d^{(k)} + \gamma_{k+1} s^{(k)}$
  $r^{(k+1)} = s^{(k)} - \gamma_{k+1} As^{(k)}$
  if $\|r^{(k+1)}\|_2 \leq \varepsilon_{\mathrm{BiCGSTAB}}$ **then**
    break
  end
  $\beta_k = \frac{\alpha_k \langle r^{(k+1)}, \hat{r}^{(0)}\rangle}{\gamma_{k+1}\langle r^{(k)}, \hat{r}^{(0)}\rangle}$
  $d^{(k+1)} = r^{(k+1)} + \beta_k (d^{(k)} - \gamma_{k+1} Ad^{(k)})$
end



(a) Ill-conditioned SPD problem.

(b) Well-conditioned SPD problem.

Figure 2.4.: Convergence behavior of the gradient method (solid line) and the conjugate gradient method (dotted line) both for an ill-conditioned and well-conditioned (preconditioned) SPD problem (2.10) with given contour levels.

$$\langle Ad^{(k)}, \hat{d}^{(k)} \rangle = \langle A\psi_k(A)r^{(0)}, \psi_k(A^T)\hat{r}^{(0)} \rangle = \langle A\psi_k^2(A)r^{(0)}, \hat{r}^{(0)} \rangle.$$

This formulation is the basic idea of CGS. Instead of squaring, BiCGSTAB uses different polynomials inducing an additional *degree of freedom* (DOF) γ_k to minimize the residual in each iteration. The residuals and search directions are thus defined via

$$r^{(k)} = \chi_k(A)\phi_k(A)r^{(0)} \quad \text{and} \quad d^{(k)} = \chi_k(A)\psi_k(A)r^{(0)} \tag{2.13}$$

using the recursively defined polynomial

$$\chi_k(t) := (1 - \gamma_k t)\chi_{k-1}(t) \qquad \text{with} \qquad \chi_0(t) := 1.$$

Applying the recursions of all polynomials $\chi_k(t)$, $\phi_k(t) = \phi_{k-1}(t) - \alpha_{k-1}t\psi_{k-1}(t)$, and $\psi_k(t) = \phi_k(t) + \beta_{k-1}\psi_{k-1}(t)$ to (2.13), we obtain the update sequences

$$r^{(k)} = s^{(k-1)} - \gamma_k A s^{(k-1)} \qquad \text{and} \qquad d^{(k)} = r^{(k)} + \beta_{k-1}(d^{(k-1)} - \gamma_k A d^{(k-1)}). \tag{2.14}$$

With these vectors the stepsize and search direction update parameters can be written without $\hat{r}^{(k)}$ and $\hat{d}^{(k)}$:

$$\alpha_k = \frac{\langle \phi_k(A)r^{(0)}, \phi_k(A^T)\hat{r}^{(0)} \rangle}{\langle A\psi_k(A)r^{(0)}, \psi_k(A^T)\hat{r}^{(0)} \rangle} = \frac{\langle r^{(k)}, \hat{r}^{(0)} \rangle}{\langle Ad^{(k)}, \hat{r}^{(0)} \rangle} \qquad \text{and}$$
$$\beta_k = \frac{\langle \phi_{k+1}(A)r^{(0)}, \phi_{k+1}(A^T)\hat{r}^{(0)} \rangle}{\langle \phi_k(A)r^{(0)}, \phi_k(A^T)\hat{r}^{(0)} \rangle} = \frac{\alpha_k \langle r^{(k+1)}, \hat{r}^{(0)} \rangle}{\gamma_{k+1}\langle r^{(k)}, \hat{r}^{(0)} \rangle}.$$

To overcome the drawback of an oscillating residual in BiCG and to produce a smooth descending progress, we have to determine the DOF γ_k such that it minimizes the residual from (2.14). This leads to

$$\min_{\gamma_k} \|(I - \gamma_k A)s^{(k-1)}\|_2^2 \quad \Leftrightarrow \quad \frac{\mathrm{d}}{\mathrm{d}\gamma_k}\left(\|(I - \gamma_k A)s^{(k-1)}\|_2^2\right) \stackrel{!}{=} 0 \quad \Leftrightarrow \quad \gamma_k = \frac{\langle As^{(k-1)}, s^{(k-1)} \rangle}{\langle As^{(k-1)}, As^{(k-1)} \rangle}$$

which is a minimum as the second derivative equals $2\|As^{(k-1)}\|_2^2 > 0$ for $s^{(k-1)} \neq 0$. Overall, Algorithm 2 summarizes the main steps of BiCGSTAB. For a more detailed description of the algorithm refer to [100, 109, 139]. A deliberate implementation will require two transpose-free matrix-vector products, four dot products and six SAXPYs yielding overall costs of

$$\#\text{flops}_{\text{BiCGSTAB}} = k(2n^2 + 10n) \in \mathcal{O}(n^2).$$

Further Krylov Subspace Methods

Since the publication of the CG method in 1952, a variety of other Krylov methods were introduced in the following decades until today. Basically, they differ both in the class of matrices they are able to solve and whether A is given explicitly. If so, matrix-vector products with A^T are possible. Only secondary they differ in the computational costs and storage requirements. An exception with respect to the latter is the *generalized minimum residual* (GMRES) method. See [13, 100, 109] for comparing remarks among the common methods.

For symmetric indefinite problems, Paige and Saunders [118] proposed the *symmetric LQ* (SYMMLQ) and the *minimum residual* (MINRES) method. An efficient solution of SPD problems can be gained by CG or the *conjugate residual* (CR) method. Note that in the SPD case SYMMLQ will generate the same solution iterates as CG but should not be preferred due to CG's efficiency. Because GMRES constructs a full orthogonal basis of the Krylov subspace and thus requires increasing amount of memory space, it should only be chosen if enough memory is available. The *quasi-minimal residual* (QMR) method and the BiCG method use a bi-orthogonal basis approach to find a solution satisfying the Petrov-Galerkin condition. As here, matrix-vector products with A^T are necessary, these methods are only applicable if A is not only given implicitly as an operator. In such cases the *transposed free QMR* (TFQMR) method, the *stabilized conjugate gradient* (CGS) method, or the BiCGSTAB method can be used because they avoid products with A^T.

Summarizing, the choice of one of these methods can be made according to the flowchart in Figure 2.5. See, for example, [100, 109] for a detailed description and special enhancements of the methods. For a historical survey on iterative methods for linear systems, refer to [124].

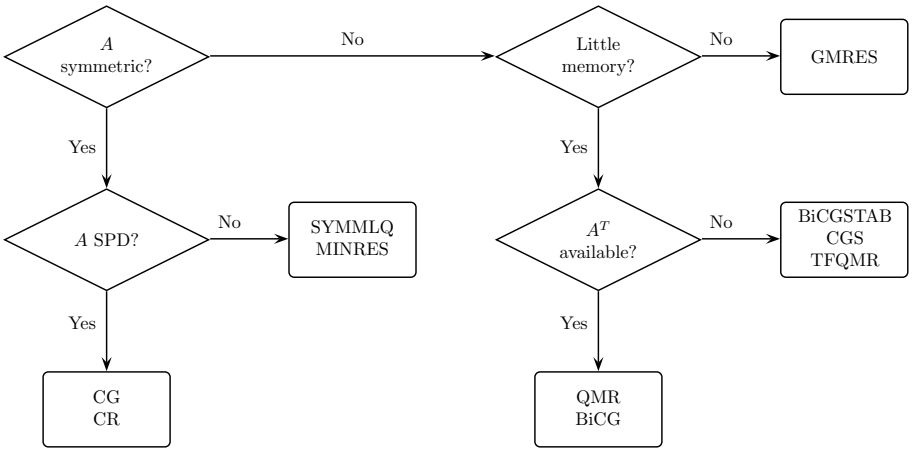


Figure 2.5.: Usage conditions of common Krylov subspace methods.

2.3. Sensitivity of Linear Systems and Preconditioning

The range of representable numbers is limited on today's computer architectures which leads to dependencies between the accuracy of floating point numbers and the underlying machine precision. Elementary operations on those numbers as well as every measurement of (physical) data is subject to round-off errors. Hence, conclusions regarding the dependency between input and output of numerical algorithms are of interest. In terms of linear systems (2.1) this means to provide a perturbation analysis between A, b, and x.

2.3.1. Perturbation Analysis

Following [53, 121, 127], we are interested in an estimate of the norm $\|\varepsilon\tilde{x}\|$ of the deviation $\varepsilon\tilde{x}$ in a linear system where the operator is perturbed by $\varepsilon\tilde{A} \in \mathbb{R}^{n\times n}$ and the right-hand side by $\varepsilon\tilde{b} \in \mathbb{R}^n$, i.e.,

$$(A + \varepsilon\tilde{A})(x + \varepsilon\tilde{x}) = b + \varepsilon\tilde{b} \tag{2.15}$$

with x being the exact solution of the unperturbed system $Ax = b$. To guarantee solvability, the condition $\|A^{-1}\|\|\varepsilon\tilde{A}\| < 1$ must hold for an induced matrix norm $\|\cdot\|$. Hence, $I + A^{-1}\varepsilon\tilde{A}$ is invertible. In general, an invertible matrix $I - B$ with $\|B\| < 1$ satisfies

$$(1 + \|B\|)^{-1} \leq \|(I - B)^{-1}\| \leq (1 - \|B\|)^{-1}. \tag{2.16}$$

From (2.16) it follows for $B = A^{-1}\varepsilon\tilde{A}$ that

$$\|(I + A^{-1}\varepsilon\tilde{A})^{-1}\| \leq \frac{1}{1 - \|A^{-1}\|\|\varepsilon\tilde{A}\|}. \tag{2.17}$$

Solving (2.15) for $\varepsilon\tilde{x} = (I + A^{-1}\varepsilon\tilde{A})^{-1}A^{-1}(\varepsilon\tilde{b} - \varepsilon\tilde{A}x)$, passing to norms, and using (2.17), we observe

$$\|\varepsilon\tilde{x}\| \leq \frac{\|A^{-1}\|}{1 - \|A^{-1}\|\|\varepsilon\tilde{A}\|}(\|\varepsilon\tilde{b}\| + \|\varepsilon\tilde{A}\|\|x\|).$$

Noticing that $\|b\| = \|Ax\| \leq \|A\|\|x\|$ and defining the so-called *condition number* of a matrix A relative to the norm $\|\cdot\|$ as the metric $\kappa(A) := \|A\|\|A^{-1}\|$, leads us to the following

Theorem 2.2 (Relative error estimation of perturbed linear systems) *For any given right-hand side $b \neq 0$ and the assumption that $(A+\varepsilon\tilde{A})$ is regular, every solution $x+\varepsilon\tilde{x}$ of the perturbed system (2.15) satisfies*

$$\frac{\|\varepsilon\tilde{x}\|}{\|x\|} \leq \kappa(A)\left(1 - \kappa(A)\frac{\|\varepsilon\tilde{A}\|}{\|A\|}\right)^{-1}\left(\frac{\|\varepsilon\tilde{b}\|}{\|b\|} + \frac{\|\varepsilon\tilde{A}\|}{\|A\|}\right).$$

Hence, the relative variation in the solution depends on the relative size of the perturbations which are mainly amplified by the condition number of the underlying system. Therefore, ill-conditioned systems with a large condition number are extremely sensible: small variations in the input may cause severe changes in the error and thus in the output. The performance of iterative methods strongly depends on the spectral properties of A, and in particular on $\kappa(A)$. See also the following Section 2.3.2. However, a small condition number may not be sufficient for fast convergence and accurate solutions.

The condition number κ_p depends on the selection of the p-norm $\|\cdot\|_p$. The conventional choice is the Euclidean norm defining the spectral condition number for regular and symmetric matrices via

$$\kappa_2(A) := \|A\|_2\|A^{-1}\|_2 = \frac{\max\limits_{\|x\|_2=1}\|Ax\|_2}{\min\limits_{\|x\|_2=1}\|Ax\|_2} = \frac{\sigma_{\max}}{\sigma_{\min}} \qquad \text{and} \qquad \kappa_2(A) := \frac{|\lambda_{\max}|}{|\lambda_{\min}|}, \tag{2.18}$$

respectively, where $\sigma_{\max}$, $\sigma_{\min}$ are the extremal singular values and $\lambda_{\max}$, $\lambda_{\min}$ the extremal eigenvalues of A. The formulae (2.18) allow for a geometrical interpretation of $\kappa_2(A)$ measuring the elongation of its associated hyperellipsoid $\{y : \ y = Ax, \|x\|_2 = 1\}$. For the SPD

case, see the elongated paraboloid illustrated in Figure 2.4. Note the following properties for the condition number in general:

- $\kappa(A) \geq \|AA^{-1}\| = \|I\| = 1$,
- $\kappa = \infty \Leftrightarrow A \neq 0$ is singular,
- $\kappa(\alpha A) = \kappa(A)$, $\forall \alpha \in \mathbb{R} \setminus \{0\}$,
- $\kappa(A) = 1$ if A is orthogonal.

2.3.2. Convergence Analysis of the (Preconditioned) Conjugate Gradient Method

To clarify the dependency of the convergence speed of iterative methods and the spectral properties of A we illustrate their relationship exemplary for the CG method (see Section 2.2.3) as its convergence analysis is evidence-based. See, for instance, [5, 46, 100, 109]. Assuming exact arithmetic, the CG method finds the solution after n iterations. An estimate concerning the rate of convergence gives

Theorem 2.3 (Approximation error of the CG method) *For an SPD $A \in \mathbb{R}^{n\times n}$ and $x \in \mathbb{R}^n$ being the unique solution of $Ax = b$, the approximation error $\|x - x^{(k)}\|_A$ of the k^{th} iterate $x^{(k)}$ of the conjugate gradient method can be estimated in the energy norm $\|y\|_A = \sqrt{\langle y, Ay\rangle}$ by*

$$\|x - x^{(k)}\|_A \leq 2\left(\frac{\sqrt{\kappa_2(A)} - 1}{\sqrt{\kappa_2(A)} + 1}\right)^k \|x - x^{(0)}\|_A.$$

Following Theorem 2.3, the convergence of CG depends on the spectral condition number: fast convergence can be achieved for small $\kappa_2(A)$. Hence, it is reasonable to transform ill-conditioned problems into equivalent but better conditioned ones. This technique is known as *preconditioning*: the original matrix A is replaced by, e.g., AM, such that $\kappa_2(AM) \ll \kappa_2(A)$ leads to faster convergence of iterative methods. However, a preconditioning matrix M of high quality is often problem dependent and may not be a reasonable choice in the general case. See also the comments in Section 2.3.3 on this.

Using a preconditioned system AM in which the SPD matrix M approximates A^{-1} in some sense slightly modifies the error bound in Theorem 2.3. Moreover, based on this estimation, we can derive an upper bound for the number of iterations k_{κ_2} which (P)CG will require to reduce the initial error by a factor ε. We obtain the approximation error in the norm $\|y\|_{AM} = \sqrt{\langle AMy, My\rangle}$ of the k^{th} iterate of the preconditioned CG method as

$$\|x-x^{(k)}\|_{AM} \leq 2\left(\frac{\sqrt{\kappa_2(AM)} - 1}{\sqrt{\kappa_2(AM)} + 1}\right)^k \|x-x^{(0)}\|_{AM} \quad \text{and} \quad k_{\kappa_2}(\varepsilon) \leq \left\lceil \frac{1}{2}\sqrt{\kappa_2(AM)}\log\left(\frac{2}{\varepsilon}\right)\right\rceil$$

for the corresponding estimation of iterations to reduce the initial error by ε. However, these bounds only explain PCG's global linear convergence and not the local effects which occur in many problems [5, 6]. Indeed, a well-known effect is the so called superlinear convergence phenomenon: the average speed of convergence increases as the iteration proceeds. This, for instance, can be observed for matrices resulting from standard five-point finite-difference discretizations of the 2D Poisson equation. Additionally, the quantity $\kappa_2(AM)$ does not provide a concrete construction of an appropriate CG preconditioner. In this context, Kaporin introduced the alternative approach using the so-called *K-condition number* [5, 101]. It is defined as the quantity

$$K(B) := \frac{\operatorname{trace}(B)}{n \det(B)^{\frac{1}{n}}} = \frac{\sum_{i=1}^{n} \lambda_i}{n \prod_{i=1}^{n} \lambda_i^{\frac{1}{n}}}, \tag{2.19}$$

where $B \in \mathbb{R}^{n \times n}$ and $\lambda_i > 0$, $i = 1, \ldots, n$ being the eigenvalues of B. In literature, (2.19) is sometimes also referred to as *Kaporin functional*. Recalling (A.4) for eigenvalues, the following properties hold:

$$K(B) \geq 1 \qquad \text{and} \qquad K(B) = 1 \qquad \text{if and only if} \qquad \lambda_1 = \ldots = \lambda_n. \tag{2.20}$$

Using (2.19), we receive the following upper bounds for PCG, namely

$$\frac{\|x - x^{(k)}\|_{AMA}}{\|x - x^{(0)}\|_{AMA}} \leq \left(K(AM)^{\frac{n}{k}} - 1\right)^{\frac{k}{2}} \quad \text{and} \quad k_K(\varepsilon) \leq \left\lceil n \log_2 K(AM) + \log_2 \left(\frac{1}{\varepsilon}\right) \right\rceil$$

for the superlinear convergence rate and for the iteration number, respectively. Taking the K-condition number into account facilitates several advantages:

- The K-condition number can give more accurate convergence bounds in some cases as it depends on the entire spectrum in contrast to κ_2 [5].
- The bound for the rate of convergence can be used to predict the superlinear convergence if k_K is sufficiently large satisfying $k_K > \log_2 K(AM)$ [7].
- The estimate for the iteration number suggests to minimize the quantity $K(AM)$ in order to obtain efficient preconditioners for the CG method. Hence, in contrast to κ_2, it constitutes a tool for the construction of preconditioners. The *factorized sparse approximate inverse* (FSAI) approach and its adaptive enhancement FSPAI are based on this fact; see Section 3.2.1.

For details on the K-condition number and the convergence of (P)CG, see [5, 6, 7, 101]. Finding an efficient preconditioner is a nontrivial task also depending on local effects occurring among certain classes of problems. Moreover, Axelsson and Kaporin even motivate for using different preconditioners among the different phases in PCG's convergence process [6]. In general, a sound convergence analysis of iterative methods is helpful to understand the requirements a preconditioner should fulfill.

2.3.3. Preconditioning and Parallel Preconditioning

Purpose and Classification

The technique of transforming a linear system into another system having the same solution but more favorable properties for iterative solution methods is known as preconditioning. The linear system at hand can be transformed with a preconditioner matrix M in the ways summarized in Table 2.2. Which form to use also depends on the iterative solver. While for the SPD case the eigenvalues remain the same for all preconditioning forms—resulting in equal convergence behavior for PCG—the situation may substantially differ among the used forms when, e.g., PGMRES is applied in the general case.

There is no limitation for the source of information to construct effective preconditioners. If available, a priori knowledge from the underlying physical problem can be used. However, in practice this is rarely the case. Therefore, preconditioners are usually derived from the

Table 2.2.: Classification and forms of preconditioning.

Preconditioning form	Preconditioner class	
	Forward based $M \approx A$	Inverse based $M \approx A^{-1}$
Left	$M^{-1}Ax = M^{-1}b$	$MAx = Mb$
Right	$AM^{-1}y = b,\ x = M^{-1}y$	$AMy = b,\ x = My$
Split	$M_1^{-1}AM_2^{-1}y = M_1^{-1}b,\ x = M_2^{-1}y$	$M_1AM_2y = M_1b,\ x = M_2y$

system matrix A. Following Section 2.3.2, the convergence of iterative methods depends on the spectral properties of A. Although it is impossible to determine properties yielding preconditioners of high quality in general, a preconditioner usually should meet the following requirements:

1. M is cheap to construct and to apply, i.e., the overhead using M pays off such that the iterative solver converges in less time than without using preconditioning. As especially the application on any given vector depends on the density of M, sparse preconditioners are desirable.

2. The preconditioned system modifies the spectrum of A in such a way that the transformed system is easier to solve iteratively. In most cases this can be achieved by M being a good approximation of A^{-1} in some sense, satisfying $\kappa_2(AM) \ll \kappa_2(A)$ and/or producing a clustering of eigenvalues. However, there are many examples where additional properties of M are desirable such as to produce gaps in the eigenspectrum, to yield a clustering of eigenvalues around several points, or to isolate the extremal eigenvalues.

3. For iterative solvers applicable to SPD A such as CG, M should be SPD or symmetric.

Note that condition 1 must not be satisfied if an iterative solver diverges. In such cases any costs are acceptable causing convergence and preconditioning can be seen as a technique yielding more robustness. Moreover, in order to find the solution of time-dependent or nonlinear problems, a long sequence of linear systems with the same coefficient matrix—but varying right-hand sides—has to be solved. Here, a preconditioner which is expensive to compute becomes feasible as the initial setup costs are amortized over several linear systems. In a parallel computing environment, matrix-vector products can be parallelized easily. Hence, when solving systems on such an architecture, the first requirement additionally implies that it should be possible to construct M in parallel easily. This is hardly satisfied for most of the common preconditioners. Sparse approximate inverses based on Frobenius norm minimization are an exception: the minimization with respect to this norm induces inherent parallelism (see Chapter 3).

Following [5, 100], to obtain a preconditioned variant of the CG method we modify Algorithm 1 by incorporating the preconditioner M in the inner products and by applying it to the residuals in

$$\text{line } 2: d^{(0)} = Mr^{(0)}, \qquad \text{line } 4: \alpha_k = \frac{\langle r^{(k)}, Mr^{(k)} \rangle}{\langle d^{(k)}, Ad^{(k)} \rangle},$$

$$\text{line } 11: d^{(k+1)} = Mr^{(k+1)} + \beta_k d^{(k)}, \text{ and} \qquad \text{line } 10: \beta_k = \frac{\langle r^{(k+1)}, Mr^{(k+1)} \rangle}{\langle r^{(k)}, Mr^{(k)} \rangle}.$$

Note that PCG in this form only requires the multiplication with an approximate inverse preconditioner $M \approx A^{-1}$ (see subsection below). Otherwise, the application of M^{-1} induces solving a linear system during each iteration which is computationally more expensive. See [16] for a survey on preconditioning for large linear systems, [13, 109] for algorithms on preconditioned iterative methods, and [3, 5, 35, 48, 109, 123] for more details on the various (parallel) preconditioners.

Splitting Based Preconditioners

In (2.9) we already mentioned the equivalence between the fixed point iteration $x^{(k+1)} = (I - M^{-1}A)x^{(k)} + M^{-1}b$, based on the matrix splitting $A = M - N$, and the left preconditioned system $M^{-1}Ax = M^{-1}b$. Hence, all the stationary methods from Table 2.1 can be understood as fixed point iterations for preconditioned systems while the specific forward preconditioners are listed in the column for M. The Jacobi preconditioner is denoted as $M = D$, the Gauss-Seidel preconditioner as $M = D - L$ and so forth.

Note that Jacobi preconditioners require very little storage, are easy to implement and inherently parallel. Among diagonal preconditioners Jacobi is (nearly) optimal in the sense of reducing the condition number. Gauss-Seidel or SOR preconditioning is almost never used because of technical reasons. Even if an optimal choice of ω can be computed, which usually is too expensive, the iterative method preconditioned with SOR simply reduces to the stationary (unpreconditioned) SOR method [3, 13].

Incomplete Factorization Based Preconditioners

The *incomplete LU* (ILU) preconditioner $M = \tilde{L}\tilde{U} \approx A$ belongs to the class of forward preconditioners. As already illustrated in Figure 2.3, the computation of an LU factorization of a sparse matrix A may cause fill-in yielding denser factors L and U compared to A. To overcome the drawback of a costly computation and large memory requirements, we can compute approximate factors satisfying

$$\min_{\mathcal{J}(\tilde{L}),\mathcal{J}(\tilde{U})\in\mathcal{J}} \|\tilde{L}\tilde{U} - A\|_2$$

by discarding part of the fill-in using some predefined pattern $\mathcal{J}$. Consequently, this approach is named incomplete LU preconditioning. There is plenty of techniques mainly differing in the rules that control the dropping of the fill-in in the incomplete factors [16, 35, 123]. We summarize the main variants commonly used in practice:

- **ILU(0)**: also known as no-fill ILU factorization imposes the property that $\mathcal{J}(\tilde{L} + \tilde{U}) = \mathcal{J}(A)$. Although no-fill incomplete factorizations are effective for certain problems such as low-order discretizations of certain PDEs leading to M-matrices[6] [16], they may yield poor convergence for more difficult problems.
- **ILU(l)**: allows controlled increase of $l \geq 0$ levels of allowed fill-in to yield more accurate factorizations. The dropping of an entry a_{ij} is based on its level of fill level_{ij}. Initially, $\text{level}_{ij} = 0$ for $a_{ij} \neq 0$, and ∞ otherwise. Components modified by the ILU

6 A nonsingular matrix $A \in \mathbb{R}^{n\times n}$ is called an M-matrix if $a_{ij} \leq 0$ for $i \neq j$, $i, j = 1, \ldots, n$, and $A^{-1} \geq 0$ [121].

process update their level via

$$\text{level}_{ij} = \min\{\text{level}_{ij}, \text{level}_{ik} + \text{level}_{kj} + 1\}.$$

An entry a_{ij} is dropped if $\text{level}_{ij} > l$. Hence, the higher the level of fill of an entry, the smaller its absolute value. For matrices whose diagonal is far from being dominant, ILU(l) may still yield poor results.

- **ILUT(p, τ)**: applies dropping based on the size of the entries in order to control the amount of fill-in. This dual-threshold strategy discards components according to $|a_{ij}| < \tau \|A_{:,i}\|_2$, where τ is an a priori chosen threshold parameter. Afterward, only the p largest remaining entries are kept. This technique produces efficient preconditioners among many problems.
- **MILU**: for discretized second-order scalar elliptic PDEs it may yield significant improvement forcing the preconditioner to preserve the row sum of A [16, 48], i.e., $\tilde{L}\tilde{U}e_S \stackrel{!}{=} Ae_S$ where $e_S^T := (1,1,\ldots,1)^T$. The preconditioner is denoted as *modified ILU* (MILU) preconditioner.

Note that similar strategies can be applied in the SPD case where incomplete Cholesky (IC) preconditioners are obtained, i.e., IC(0), modified IC (MIC) and so on.

Although ILU or IC approaches belong to the fastest preconditioning techniques, breakdowns may occur due to zero or even due to negative pivot elements for SPD problems. Stable behavior can only be assured for the class of H-matrices[7] [16, 22]. However, many matrices—also stemming from industrial applications—do not belong to this class. Therefore, various recent research efforts afforded robust incomplete factorization methods for the general SPD case such as the *robust incomplete factorization* (RIF) preconditioner by Benzi and Tůma [24]. RIF is computationally more expensive compared to IC but represents a reliable algorithm based on the *stabilized approximate inverse* (SAINV) preconditioner (see next subsection). For general matrices the situation is even worse: A can be far from diagonal dominance and even for a breakdown-free incomplete process of a well-conditioned system it is possible that $\tilde{L}$ and/or $\tilde{U}$ become ill-conditioned. Hence, the solution will deteriorate during the forward and backward substitution, respectively. In this context, preconditioners which do not require the solution of a linear system but approximate the inverse of A are a robust alternative immune from these problems.

Approximate Inverse Preconditioners

With the advent of vector computers in the 1970s, polynomial preconditioners become of interest as their application merely requires efficiently parallelizable matrix-vector multiplications. If A can be expressed via the splitting $A = K - N$, an intuitive approach is to use low order truncations of the *Neumann series* as approximate inverse preconditioner, i.e.,

$$M := \left(\sum_{i=0}^{k} (K^{-1}N)^i\right) K^{-1} \approx A^{-1} \qquad \text{with} \qquad k \geq 0. \tag{2.21}$$

7 A nonsingular matrix $A \in \mathbb{C}^{n\times n}$ is an H-matrix if the matrix $\bar{A} \in \mathbb{R}^{n\times n}$, resulting from the coefficients $\bar{a}_{ij} = |a_{ii}|$ for $i = j$ and $\bar{a}_{ij} = -|a_{ij}|$ for $i \neq j$, is an M-matrix [5].

Because the condition $\rho(K^{-1}N) < 1$ must be satisfied in order to obtain convergence, general splittings using $K \neq I$ are usually preferred. Another approach is based on finding, e.g., Chebyshev or least squares polynomial preconditioners $M := \phi(A)$ [35, 123], which satisfy

$$\min_M \|AM - I\| = \min_\phi \|A\phi(A) - I\| \qquad \text{with} \qquad \phi(A) \approx A^{-1}$$

in a specific norm. However, the lack of robustness, the limited effectiveness especially for unsymmetric problems, and unavailable but necessary a priori knowledge of the spectrum of A, prevents this approximate inverse approach to be used as a preconditioner in general.

Present-day used (sparse) approximate inverse preconditioners can be classified into two groups. Methods based on Frobenius norm minimization on the one hand and based on incomplete biconjugation on the other hand. The latter known as the *approximate inverse* (AINV) preconditioner by Benzi, Meyer, and Tůma [20, 21], is applicable to general sparse matrices which admit the factorization $A = LDU$. AINV is based on the computation of two A-biconjugate vector sets $\{Z_i\}_{i=1}^n$ and $\{W_i\}_{i=1}^n$, i.e., $W_i^T A Z_j = 0$ if and only if $i \neq j$. With nonsingular W and Z the inverse can be written as

$$A^{-1} = ZD^{-1}W^T = \sum_{i=1}^{n} \frac{Z_i W_i^T}{d_{ii}} \qquad \text{where} \qquad Z := U^{-1} \qquad \text{and} \qquad W^T := L^{-1}.$$

Briefly, AINV can be described as a two-sided generalized Gram-Schmidt orthogonalization process with respect to the bilinear form associated with A. Starting with unit basis vectors $Z_i = W_i = e_i$, the (right-looking) AINV consists of a nested loop computing

$$Z_j \leftarrow Z_j - \frac{A_{i,:}Z_j}{A_{i,:}Z_i} Z_i \qquad \text{and} \qquad W_j \leftarrow W_j - \frac{A_i^T W_j}{A_i^T W_i} W_i, \tag{2.22}$$

where $i = 1, 2, \ldots, n$ and $j = i + 1, \ldots, n$. After completion, the inverse can be computed via

$$ZD^{-1}W^T, \qquad \text{where} \qquad D^{-1} := \operatorname{diag}\left[(A_{1,:}Z_1)^{-1}, (A_{2,:}Z_2)^{-1}, \ldots, (A_{n,:}Z_n)^{-1}\right].$$

Although initially sparse, the nested loop produces fill-in in Z and W, respectively. By inducing a dropping rule similar to ILU methods, e.g., based on a drop tolerance, sparse factors $\tilde{Z}$ and $\tilde{W}$ are obtained to construct the factorized sparse approximate inverse preconditioner $M := \tilde{Z}D^{-1}\tilde{W}^T \approx A^{-1}$. If A is an H-matrix or round-off errors are absent, AINV represents a stable algorithm. In general though, these conditions are not satisfied and breakdowns or uncontrolled growth of the factors due to exceedingly small pivots—the denominators in (2.22)—may occur. For the SPD case where $A = LDL^T$, a stable but more costly version of AINV called *stabilized AINV* (SAINV) was introduced in [17]. It requires only one A-biconjugate vector set $\{Z_i\}_{i=1}^n$ such that $Z_i^T A Z_j = 0$ for $i \neq j$. Reducing the nested loop to

$$Z_j \leftarrow Z_j - \frac{\langle AZ_i, Z_j\rangle}{\langle AZ_i, Z_i\rangle} Z_i$$

yields a breakdown-free process due to the pivot elements $d_{ii} = \langle AZ_i, Z_i\rangle$. Refer to [17] for details on this. With the computed matrix $\tilde{Z} \approx L^{-T}$ the SAINV preconditioner is defined as $M := \tilde{Z}D^{-1}\tilde{Z}^T \approx A^{-1}$.

Approaches based on incomplete A-orthogonalization do not require an a priori sparsity pattern for M and usually outperform (factorized) methods based on Frobenius norm minimization [22] in sequential environments. Although, comparable to ILU methods in terms of robustness and rates of convergence, the inherent sequential part—the A-(bi)orthogonalization—can only be tackled by additional effort, e.g., using graph partitioning [19, 23] for large-scale problems. However, the known parallelization approaches for (S)AINV are only scalable in terms of parallel efficiency under certain conditions. In the two-level preconditioning approach, for instance, if the required Schur complements—the separator set stemming from a kind of dissection approach—remain small [16].

In this context, methods belonging to the second class which are based on Frobenius norm minimization provide a highly scalable preconditioner construction due to their inherent parallelism. They are robust, well defined for nonsingular A, and are able to solve difficult problems where ILU techniques fail [10]. The disadvantage of the so-called static methods, which require to choose an a priori sparsity structure for M, can be overcome by dynamic strategies. These, however, come along with a couple of input parameters. For a detailed survey and comparative study on sparse approximate inverses see [16, 22]. This thesis has a focus on parallel sparse approximate inverse preconditioners based on Frobenius norm and K-condition number minimization; we will elaborate on them in Chapter 3.

Further Preconditioning Techniques

Following [16, 124], the publication of Krylov subspace methods pushed modern preconditioning techniques more into the spotlight from the late 1960s on. The indispensable need and high problem dependency motivated plenty of specialized, problem tuned, or even combined preconditioning approaches over the last two decades.

For example, *Schwarz* type preconditioners [3, 123] which are based on a domain decomposition ansatz have been proven to be efficient parallel preconditioners for large classes of PDE related problems. Here, it is also known that A^{-1} is sparse in the wavelet basis having a so-called *finger* pattern. Therefore, Chan et al. [33] proposed to construct a sparse approximate inverse preconditioner $\tilde{M}$ for WAW^T, where W is an orthogonal wavelet transform matrix. This can yield improved results compared to ordinary SPAI preconditioners in the standard basis. To impose scalability on the fast and/or robust sequential preconditioners, multilevel approaches became of considerable importance, closing the gap between Krylov methods and the multigrid idea (see Section 2.4). *Multilevel ILU* (ILUM) [123] or multilevel sparse approximate inverses are typical examples for this preconditioning approach. Much effort is also put on parallelization strategies for a *parallel ILU* (PILU) preconditioner. Despite suffering from breakdown eventuality, it is possible to obtain—to some extend—scalable implementations based on ingenious algorithms. See the references in [16] on this.

Summing up, to obtain scalability using preconditioners which lack from inherent parallelism nontrivial approaches are applied, often resulting in problem dependent and sophisticated implementations. A robust, fast, parameter-free, and naturally parallel preconditioner which is efficient for a large class of problems is unavailable yet. Focusing on incomplete factorization and sparse approximate inverse preconditioners, a rough survey based on breakdown-freeness and inherent parallelism is given by the flowchart in Figure 2.6. Further attributes such as setup costs in a sequential environment, memory costs, or the quality (efficiency) among a broad class of problems are not considered.

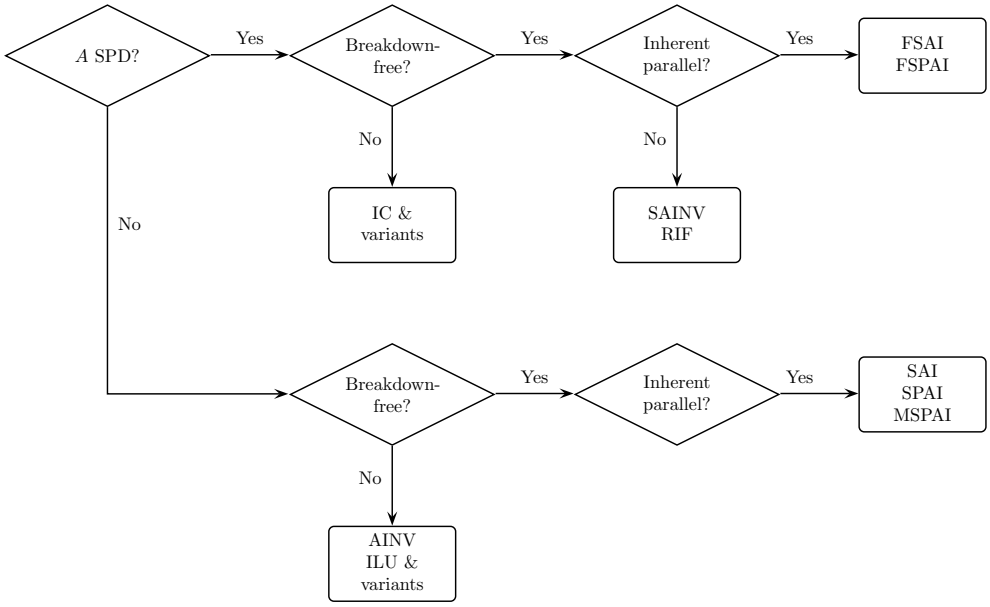


Figure 2.6.: Survey based on breakdown-freeness and inherent parallelism for incomplete factorization and sparse approximate inverse based preconditioners.

2.4. Multigrid

Multigrid (MG) is a general strategy for constructing hierarchical solvers by tackling the original physical problem, e.g., the PDE or the integral equation, directly. MG methods are known to be the fastest iterative methods for certain problems such as elliptic PDEs whose finite difference discretization leads to (ill-conditioned) SPD operators. Using a hierarchy of different structured meshes (discretizations) a different action is imposed on the various Fourier modes of the error corresponding to the computed solution of the underlying problem.

Basic Principle

The iterative process of stationary methods can be written as

$$x^{(k+1)} = Rx^{(k)} + M^{-1}b \qquad \text{with error} \qquad e^{(k+1)} = Re^{(k)}, \tag{2.23}$$

where M is chosen from Table 2.1 and $R := I - M^{-1}A$ is the iteration matrix. The error is a superposition of low and high frequency modes, which, for many problems, correspond to the eigenvectors of R. The different modes, also depending on the boundary conditions used in the given PDE, provide the basis for the Fourier expansion of the error. This observation leads to the basic idea behind MG strategies which consists of the following two principles [138]:

- **Smoothing.** Many stationary methods such as the damped Richardson, Jacobi, or Gauss-Seidel, have a strong smoothing property, i.e., oscillatory modes of the error are

eliminated effectively, but smooth modes are damped very slowly. Note that the latter is the reason for slow convergence of these so-called relaxation methods.

- **Coarse grid correction (CGC).** A smooth error term can be well approximated on a coarse grid. All computations on a coarse grid are less expensive as there are significantly fewer grid points.

A combination of these principles leads to the *two-grid cycle*: the natural basis of algorithms using several grids of different mesh size (V-cycle, W-cycle, FMG). Applying a few iterations of a stationary method on $A_h x^{(k)} = b_h$ produces the smooth approximation $\tilde{x}_h^{(k)}$. The corresponding smooth residual $r_h = b_h - A_h \tilde{x}_h^{(k)}$ is restricted to a coarser grid domain Ω_{2h} via $r_{2h} = E_h^{2h} r_h$, typically with half the number of points. After solving $A_{2h} e_{2h} = r_{2h}$ on Ω_{2h} and prolonging the correction $\tilde{e}_h = E_{2h}^h e_{2h}$ back on Ω_h, the new solution can be computed and smoothed again with a stationary method using the new initial guess $\tilde{x}_h^{(k)} + \tilde{e}_h$. See Briggs et al. [28] or Trottenberg et al. [138] for a detailed description of multigrid techniques and the choice of the various multigrid elements such as the restriction and prolongation operators E_h^{2h} and E_{2h}^h.

Briefly, the two-grid cycle consists of presmoothing, coarse grid correction, and postsmoothing. Consequently, an essential part of efficient MG methods is the presence of an effective smoother which in very few iterations will produce a smooth residual. The *local Fourier analysis* (LFA) [138] or *local mode analysis* (LMA) is a heuristic technique for the quantitative analysis and the design of smoothers and MG methods.

Smoothing Analysis with Local Fourier Analysis

By linearization and freezing of coefficients, LFA considers general linear discrete operators with constant coefficients defined on an infinite domain or with periodic boundary conditions. It can be used to analyze how smoothers such as relaxation methods act on the Fourier modes of the error. The following notation is strongly influenced by [35, 138].

Consider the general d-dimensional setting with fixed mesh width $h := (h_1, \ldots, h_d)$ and the infinite grid domain $\Omega_h := \{x = (x_1, \ldots, x_d) = \iota h = (\iota_1 h_1, \ldots, \iota_d h_d), \iota \in \mathbb{Z}^d\}$. The frequency of the grid function can be characterized by $\theta := (\theta_1, \ldots, \theta_d)$ with $\theta_k \in [-\pi, \pi]$ being the continuous wave number in the k^{th} dimension. Then, in the general case, the complex Fourier modes of the error are given by

$$\phi(\theta, x) = e^{\frac{i\theta x}{h}} = \prod_{k=1}^{d} e^{\frac{i\theta_k x_k}{h_k}}. \tag{2.24}$$

By coarsening of the underlying grid, typically with $2h$ in MG strategies, aliasing appears: all high frequency modes coincide with a low frequency on Ω_{2h} or vanish on Ω_{2h} [28, 138]. To obtain qualitative terms for the Fourier modes in case of standard coarsening, the first half of the Fourier basis is denoted to be the low frequency spectrum, while the second half is the high frequency one. In terms of $\phi(\theta, x)$ this means

$$\begin{aligned} \phi(\theta, x) \text{ is a low frequency} &\Leftrightarrow \theta \in \left[-\tfrac{\pi}{2}, \tfrac{\pi}{2}\right[^d \quad \text{and} \\ \phi(\theta, x) \text{ is a high frequency} &\Leftrightarrow \theta \in [-\pi, \pi[^d \setminus \left[-\tfrac{\pi}{2}, \tfrac{\pi}{2}\right[^d. \end{aligned} \tag{2.25}$$

As the oscillatory modes of the error tend to resemble the eigenvectors of the iteration matrix in general, LFA is used to analyze the smoothing property of high frequency modes. Note that if the modes coincide with the eigenvectors of R, LFA represents a rigorous analysis. After the k^{th} step of a relaxation scheme, the error at the point x of the domain can be expressed as $e_x^{(k)} = A(k)\phi(\theta,x)$, where $A(k)$ is the amplitude of the mode. In successive steps, the amplitudes are related by the recurrence $A(k+1) = R(\theta)A(k)$ where $R(\theta)$ is called the amplification factor for the mode θ. This enables to analyze the convergence of the modes separately. To obtain convergence for relaxation methods, $|R(\theta)| < 1$ must be satisfied for all θ. For MG methods only the oscillatory modes of the error must be damped, i.e., $|R(\theta)| < 1$ must only be satisfied on the high frequency part $\frac{\pi}{2} \leq |\theta| \leq \pi$. Hence, the worst case factor, by which the oscillatory modes are damped, is denoted as smoothing factor

$$\mu_{\text{smooth}} := \max_{\frac{\pi}{2} \leq |\theta| \leq \pi} |R(\theta)|.$$

Smoothing Analysis with Generating Functions

Matrices which are constant along their diagonals are denoted as *Toeplitz* matrices $T_n \in \mathbb{C}^{n\times n}$, whose entries are given by $T_{l,m} = t_{l-m}$. These matrices of the general form

$$T_n := \begin{pmatrix} t_0 & t_{-1} & \cdots & t_{2-n} & t_{1-n} \\ t_1 & t_0 & t_{-1} & & t_{2-n} \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ t_{n-2} & & t_1 & t_0 & t_{-1} \\ t_{n-1} & t_{n-2} & \cdots & t_1 & t_0 \end{pmatrix} \qquad \text{are related to} \qquad f(x) = \sum_{k=-\infty}^{\infty} t_k e^{ikx},$$

where $f(x)$ represents the corresponding symbol or generating function [56, 83]. Note that if $f(x)$ is continuous in $[-\pi,\pi]$ then its Fourier coefficients $t_k = \frac{1}{2\pi}\int_{-\pi}^{\pi} f(x)e^{-ikx}\mathrm{d}x$ are the coefficients of T_n; if $f(x)$ is real-valued then T_n is Hermitian. The connection also reflects in the range of values. The range of values of T_n—and consequently also all eigenvalues of T_n—are contained in the range of values of the symbol. For $T_n \in \mathbb{R}^{n\times n}$, the symbol is even with $f(x) \in [0,\pi]$ and reduces to a cosine series

$$f(x) = \frac{t_0}{2} + \sum_{k=1}^{\infty} t_k \cos(kx) \qquad \text{with coefficients} \qquad t_k = \frac{1}{\pi}\int_0^{\pi} f(x)\cos(kx)\mathrm{d}x.$$

The range of values of T_n still satisfies

$$\forall x \neq 0 \ : \qquad R_{T_n}(x) = \frac{x^T T_n x}{x^T x} \in \left[\operatorname*{ess\,inf}_{x\in[0,\pi]} f(x), \operatorname*{ess\,sup}_{x\in[0,\pi]} f(x)\right].$$

Hence, in the constant coefficient case the amplification factor $R = I - M^{-1}A$ can also be described as $1 - m^{-1}(x,y)a(x,y)$, where a and m are generating functions representing A and M. The technique of generating functions is similar to LFA. In 2D the functions are defined in $x,y \in [0,\pi]$ as the modes of the error are represented only by cosine terms. Therefore, the high frequency domain G is restricted to the first quadrant of the region in (2.25), i.e.,

$$G := [0,\pi]^2 \setminus [0,\tfrac{\pi}{2}]^2 \quad \text{with corners at} \quad (0,\tfrac{\pi}{2}),\ (\tfrac{\pi}{2},\tfrac{\pi}{2}),\ (\tfrac{\pi}{2},0),\ (\pi,0),\ (\pi,\pi),\ \text{and}\ (0,\pi). \quad (2.26)$$

Consequently, the smoothing factor is given by

$$\mu_{\text{smooth}} := \max_{x,y \in G} \left\{ |1 - m^{-1}(x,y)a(x,y)| \right\}. \tag{2.27}$$

For further details on smoothing analysis with generating functions see [83] and the references therein.

2.5. Discrete Inverse Problems

The fundamentals and the solution of (large) discrete inverse problems are described, e.g., in Hanke and Hansen's survey paper [62] or the books [69, 70] by Hansen. Given the internal structure of a model or system, a forward (or direct) problem represents the identification of the system's behavior for the input, i.e., evaluating the integral in (2.28) for known H and x. In contrast, the determination of the input from an observed output for a known model is called an inverse problem, i.e., the computation of x for given H and b. See also Figure 2.7.

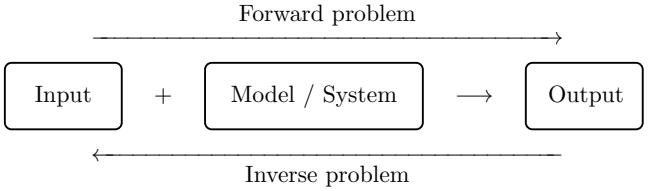


Figure 2.7.: Schematic illustration of forward and inverse problems.

Many inverse problems arising from, for example, image deblurring, signal processing, medical tomography, or inverse helioseismology applications can be formulated via a Fredholm integral equation of the first kind

$$\int_0^1 H(s,t)x(t)\mathrm{d}t = b(s) \qquad \text{with} \quad 0 \leq s \leq 1. \tag{2.28}$$

Both the square integrable linear kernel function H, which describes the underlying model, and the measured right-hand side b are known functions while x is the unknown input. An important property of (2.28) is its smoothing behavior: the mapping from x to b dampens the higher frequency components in x and the integration with H thus has a smoothing effect. Mathematically, this effect can be described using the Riemann-Lebesgue lemma[8] [70]. Hence, the inverse problem amplifies high frequencies, i.e., a small perturbation in b can correspond to an arbitrary large perturbation in x. Consequently, the inherent problem when solving (2.28) is that the solution may not depend continuously on the data.

In case that the inverse problem in continuous form (2.28) cannot be solved analytically, we have to discretize it and apply numerical solution methods. Using quadrature or expansion methods following [70], we obtain a linear discrete inverse problem

8 Consider the exemplary signal $x_p(t) := \sin(2\pi pt)$ being Riemann integrable on [0,1]. Then we obtain $\lim_{p\to\pm\infty} \int_0^1 H(s,t)x_p(t)\mathrm{d}t = 0$.

$$x \overset{\text{blur}}{\longrightarrow} Hx \overset{\text{noise}}{\longrightarrow} Hx + \eta = b \tag{2.29}$$

corresponding to (2.28). Here, $x \in \mathbb{R}^n$ is the original signal or image, $H \in \mathbb{R}^{n\times n}$ is the discretized kernel function, i.e., the finite dimensional operator—denoted as blur operator in image deblurring problems—$\eta \in \mathbb{R}^n$ is a vector representing the unknown perturbations such as noise or measurement errors, and $b \in \mathbb{R}^n$ is the observed signal and image, respectively.

According to Hadamard, an inverse problem is said to be *ill-posed* (in contrast to *well-posed*) if any of the following criteria is violated for an arbitrary input: the solution exists, is unique, and depends continuously on the data. If the first two conditions are not satisfied initially, we can at least compute solutions which are optimal in some sense by using a generalized inverse of H. However, the usually extremely ill-conditioned kernel H causes discrete inverse problems to be characterized as ill-posed. Therefore, to make the solution less sensible to errors, it must be stabilized, i.e., a regularization technique must be used.

Remark 2.2 *Throughout this thesis we assume that the errors in the right-hand side are Gaussian white noise. Hence, all components of the noise vector $\eta \in \mathbb{R}^n$ result from the same Gaussian distribution with zero mean and standard deviation ξ. Note that for the latter—the magnitude of the white noise—we use ξ instead of σ in order to avoid confusion with singular values. Denoting $\mathbb{E}(\cdot)$ as the expected value and ξ^2 as the variance, the following equations hold:*

$$\mathbb{E}(\eta) = 0, \qquad \mathbb{E}(\|\eta\|_2^2) = \xi^2 n, \qquad \textit{and} \qquad \operatorname{cov}(\eta) = \mathbb{E}(\eta\eta^T) = \xi^2 I.$$

See [70] for further forms of noise.

2.5.1. The Relationship Between the Singular Spectrum and the Meaningful Existence of a Solution

The singular spectrum of the finite-dimensional kernel H gives important insight into the behavior of discrete ill-posed problems resulting from (2.28). The *singular value decomposition* (SVD) [53] of a matrix $H \in \mathbb{R}^{m\times n}$, assuming $m \geq n$, is the decomposition

$$H = U \begin{pmatrix} \Sigma \\ 0 \end{pmatrix} V^T = \sum_{i=1}^{n} u_i \sigma_i v_i^T. \tag{2.30}$$

The matrices $U = (u_1, \ldots, u_m) \in \mathbb{R}^{m\times m}$ and $V = (v_1, \ldots, v_n) \in \mathbb{R}^{n\times n}$ consist of the left and right singular vectors, respectively, and are orthogonal, i.e., $U^TU = V^TV = I$. The matrix $\Sigma \in \mathbb{R}^{n\times n}$ contains the singular values of H in the form

$$\Sigma = \operatorname{diag}(\sigma_1, \ldots, \sigma_n) \qquad \text{with} \qquad \sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_n \geq 0.$$

For square integrable kernels in the continuous case, as they appear in Fredholm integral equations of the first kind, the corresponding decomposition into singular modes and singular values is known as the *singular value expansion* (SVE). The correlation between the singular modes and the Fourier modes $(2\pi)^{-\frac{1}{2}} e^{iks}$ [70, 74] can be transferred to the discrete setting:

- large singular values and singular vectors—small i within (2.30)—correspond to low frequencies and thus to the *signal subspace* while
- small singular values—large i—correspond to high frequencies and the *noise subspace*.

The direct and naive solution of (2.29) for the observed signal $b = b_{\text{exact}} + \eta$ is given by $x = H^{-1}b_{\text{exact}} + H^{-1}\eta$. Using the SVD of $H = U\Sigma V^T$, the solution can be expressed by

$$x = V\Sigma^{-1}U^T(b_{\text{exact}} + \eta) = \sum_{i=1}^{n} \frac{u_i^T b_{\text{exact}}}{\sigma_i} v_i + \sum_{i=1}^{n} \frac{u_i^T \eta}{\sigma_i} v_i = \sum_{i=1}^{n} \Phi_i \frac{u_i^T b}{\sigma_i} v_i \tag{2.31}$$

for the filter factors $\Phi_i = 1$. We observe in (2.31) that small singular values amplify the high frequency oscillations in the right-hand side. Hence, the smoothing effect of the forward problem also holds for square integrable kernels in discrete settings.

While in the continuous case the solution only exists when its 2-norm is bounded, i.e., the *Picard condition* [62, 70] is satisfied, the solution always exists for the finite-dimensional case. However, if the right-hand side components $|u_i^T b|$, on average, do not decay faster than the singular values, the norm of the solution can be extremely large making it useless in practice. The situation is even worse because of the perturbation in the right-hand side and the presence of round-off errors occurring due to finite precision arithmetic on computing architectures. The usual observation in the so-called discrete Picard plots is that the SVD coefficients $|u_i^T b|$ level off at some noise plateau. Hence, following [66, 70], a meaningful (regularized) solution for discrete ill-posed problems can only be obtained if the *discrete Picard condition* is satisfied: for all $\sigma_i > \tau$ the corresponding SVD coefficients $|u_i^T b|$ decay (on average) faster than the σ_i, where τ is the level at which the singular values level off due to round-off errors.

2.5.2. Direct Regularization Methods

Direct regularization methods compute the solution via direct methods. Based on a decomposition of H such as the QR factorization or the SVD, these methods can be seen as a spectral filter acting on the singular spectrum, diminishing the deterioration of the solution by noise. The filter factors Φ_i in (2.31) are to be chosen in such a way that the high frequency amplifying terms $u_i^T \eta \sigma_i^{-1}$ for small singular values are effectively filtered out, i.e., Φ_i tends to zero for decreasing σ_i to obtain a reconstruction mainly on the signal subspace. In fact, common direct regularization techniques differ in the choice of the filter factors.

Truncated SVD

An intuitive approach to improve the reconstruction and circumvent the contribution of noise to the solution would be to cut off the second summand within (2.31) and compute the solution merely on the signal subspace $\sum_{i=1}^{n}(u_i^T b_{\text{exact}})\sigma_i^{-1} v_i$. Unfortunately, as b_{exact} usually is not known in advance, an explicit splitting into the signal and noise subspace is impossible. Nevertheless, as large singular values correspond to the signal subspace, we can shrink the SVD expansion of (2.31) such that the solution will mostly consist of quantities $u_i^T b \sigma_i^{-1}$ corresponding to the signal part. We can suppress the noise contribution in (2.31) by using the filter factors

$$\Phi_i = \begin{cases} 1, & i = 1, \ldots, k \\ 0, & i = k+1, \ldots, n \end{cases} \qquad \text{corresponding to} \qquad x = \sum_{i=1}^{k} \frac{u_i^T b}{\sigma_i} v_i$$

which is equivalent to solving $\min_x \|x\|_2$ with subject to $\min_x \|H_k x - b\|_2$, where the rank deficient and better conditioned coefficient matrix H_k is the closest rank-k approximation to H. This direct regularization method is known as *truncated singular value decomposition* (TSVD) [65, 70]. There are more SVD based regularization methods such as the *modified TSVD*, the *damped SVD*, and the *selective SVD*; see [68, 70] and the references therein. However, all SVD based methods suffer from high computational costs due to the required decomposition which makes them difficult to handle for large-scale problems.

Tikhonov-Phillips Regularization

Another and one of the classical regularization methods is the *Tikhonov-Phillips regularization* (TPR) by Tikhonov and Phillips [120, 135]. It solves

$$\min_x \left\| \begin{pmatrix} H \\ \alpha I \end{pmatrix} x - \begin{pmatrix} b \\ 0 \end{pmatrix} \right\|_2^2 = \min_x \left\{ \|Hx - b\|_2^2 + \alpha^2 \|x\|_2^2 \right\} \Leftrightarrow (H^T H + \alpha^2 I)x = H^T b \quad (2.32)$$

instead of (2.29) for a fixed regularization parameter $\alpha \geq 0$. The three ingredients have a different action on the solution.

- The model fit $\|Hx - b\|_2^2$ measures how well x predicts the observed right-hand side.
- The penalty term $\|x\|_2^2$ controls the norm (regularity) of the solution to suppress high frequency components having large amplitudes.
- The weight α has to be chosen such that both minimization criteria yield the optimal value together: the computed solution x is as close as possible to the original problem and sufficiently regular. Note that in the limiting cases we obtain $x_{\alpha^2 \to \infty} = 0$ and $x_{\alpha^2 \to 0} = H^{-1} b$ representing the naive—with noise contaminated—reconstruction.

In fact, TPR can be seen as a spectral filtering method. Considering the decomposition $H = U\Sigma V^T$ in (2.32), we obtain

$$x = (V\Sigma^2 V^T + \alpha^2 V V^T)^{-1} V \Sigma U^T b = V(\Sigma^2 + \alpha^2 I)^{-1} \Sigma U^T b. \quad (2.33)$$

Using singular vectors and values, we can express (2.33) by (2.31) with the filter factors

$$\Phi_i = \frac{\sigma_i^2}{\sigma_i^2 + \alpha^2} \approx \begin{cases} 1, & \sigma_i \gg \alpha \\ \frac{\sigma_i^2}{\alpha^2}, & \sigma_i \ll \alpha \end{cases} \quad (2.34)$$

for the basis vectors v_i, corresponding to different frequency information. This filter function is structurally identical to the ideal Wiener filter [2], which optimally separates noise of spectral density α^2 from a signal of spectral density σ_i^2. Depending on the regularization weight, the low frequency SVD components contribute their full weight while the high frequency components get damped due to small Φ_i resulting from $\sigma_i \ll \alpha$. Note that, according to (2.34), it is reasonable to let α vary in the spectrum of H, i.e., $\alpha \in [\sigma_n, \sigma_1]$.

Regularization Including a Seminorm

Following [62, 70, 71], instead of using the 2-norm as a means to control the error in the regularized solution, another possibility is to use discrete seminorms of the form $\|Lx\|_2$ to

obtain regularity. With a so-called *regularization matrix* L, the problem (2.32) in standard form can be reformulated as TPR in general form

$$\min_x \left\| \begin{pmatrix} H \\ \alpha L \end{pmatrix} x - \begin{pmatrix} b \\ 0 \end{pmatrix} \right\|_2^2 = \min_x \left\{ \|Hx - b\|_2^2 + \alpha^2 \|Lx\|_2^2 \right\} \Leftrightarrow (H^T H + \alpha^2 L^T L)x = H^T b. \tag{2.35}$$

If the underlying signal x is smooth, L is usually chosen as a discrete approximation to the first or second derivative operator, i.e.,

$$L_1 = \begin{pmatrix} -1 & 1 & & \\ & \ddots & \ddots & \\ & & -1 & 1 \end{pmatrix} \in \mathbb{R}^{(n-1)\times n} \quad \text{or} \quad L_2 = \begin{pmatrix} 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2 & 1 \end{pmatrix} \in \mathbb{R}^{(n-2)\times n}, \tag{2.36}$$

not taking any boundary conditions into account. Applying L_k, $k = 1, 2$, will roughen up the oscillations in the reconstruction provoking large L_k-(semi)norms in the penalty term while on the smooth components it will have no effect. Consequently, rough oscillations caused by noisy components can be suppressed during the reconstruction and the regularized approximation will satisfy inherent smoothness properties. Therefore, for problems where the exact signal x is smooth, the solution of the general formulation (2.35) will be smoother and thus a more accurate reconstruction when using a differential operator L_k. In Section 6.5, we introduce new regularization matrices L_H depending on the spectrum of H.

If (2.35) is to be solved via spectral decomposition, a *generalized SVD* (GSVD) [53] of the matrix pair (H,L) has to be applied. Using filter factors similarly to the TSVD and thus truncating the GSVD to suppress the noise contribution is denoted as *truncated GSVD* (TGSVD) [68, 69, 70] in literature. Note that when using smoothing preconditioning with L in an iterative regularization method such as CGLS, a transformation to standard form has to be applied [70].

2.5.3. Iterative Regularization Methods

Iterative methods are especially suitable whenever it is possible to take advantage of the sparsity and the structure of H, as matrix factorizations used in direct (regularization) methods destroy this property. Moreover, there is no need for H being given explicitly because the elementary operations within the iterations are matrix-vector products. The latter makes iterative regularization methods simpler to parallelize and is quite a necessity for regularization methods handling the deblurring of images. Here the matrices are usually too large for being handled efficiently by direct methods both in time and space. Therefore, iterative regularization methods are an attractive alternative. Not least because both the TSVD and the TPR have problems to reconstruct discontinuous data [70].

Regularization by iteration has the inherent property of initially operating on the SVD components $u_i^T b \sigma_i^{-1} v_i$ corresponding to the largest singular values and the desired signal subspace, respectively. After approaching a regularized solution $x^{(k)}$ in the k^{th} iteration, the method converges to an undesired solution as it starts working on the noise subspace. In order to avoid this so-called *semiconvergent* behavior, an appropriate stopping criterion has to be used. However, for iterative regularization methods it is still an open question to provide an appropriate estimation for the optimal number of iterations. For certain Krylov

subspace methods such as the *conjugate gradient least squares* (CGLS) method [25, 76], the L-curve criterion (see Section 2.5.4) can be applied [70].

Conjugate Gradient Least Squares

CG applied to the unregularized normal equations $H^T Hx = H^T b$—without forming $H^T H$—has the desired regularization property: the low frequency components of the solution tend to converge faster than the high frequency components [68, 70]. CGLS [25, 76] is a stable way to implement the CG method on the normal equations for LS problems in the general case. Its solution at k^{th} iteration satisfies $x^{(k)} = \arg\min_x \|Hx - b\|_2$ and

$$x^{(k)} \in \mathcal{K}_{\text{NE}}^{(k)} := \mathcal{K}_{\text{NE}}^{(k-1)} \otimes \{(H^T H)^{(k-1)} H^T b\} \qquad \text{with} \qquad \mathcal{K}_{\text{NE}}^{(1)} := \{H^T b\} \qquad \text{for} \qquad k \geq 1.$$

$\mathcal{K}_{\text{NE}}^{(k)}$ is the k-dimensional Krylov subspace on the normal equations. Similar to direct methods, solutions from CGLS can be expressed via a filtered SVD expansion

$$\begin{aligned} x^{(k)} &= \sum_{i=1}^{k} c_i (H^T H)^{i-1} H^T b = \sum_{i=1}^{k} c_i (V \Sigma^2 V^T)^{i-1} V \Sigma U^T b = \sum_{i=1}^{k} c_i V \Sigma^{2(i-1)} \Sigma U^T b \\ &= V \underbrace{\left(\sum_{i=1}^{k} c_i \Sigma^{2i} \right)}_{\Phi^{(k)}} \Sigma^{-1} U^T b \quad \text{with} \quad \Phi^{(k)} := \operatorname{diag}\left(\sum_{i=1}^{k} c_i \sigma_1^{2i}, \sum_{i=1}^{k} c_i \sigma_2^{2i}, \ldots, \sum_{i=1}^{k} c_i \sigma_n^{2i} \right). \end{aligned}$$

Hence, the CGLS filter factors $\Phi^{(k)}$ are polynomials in the squared singular values. However, the proof for the coherence between the filter factors and the regularization property of CGLS is beyond the scope of this thesis. See Hanke and Hansen's work in [62, 69] and the references therein for further reading. Based on the formulation in [25], Algorithm 3 summarizes the

Algorithm 3: Preconditioned CGLS (PCGLS) using preconditioner M.

Input : $H \in \mathbb{R}^{n \times n}$, $b \in \mathbb{R}^n$, $k_{\max} \geq 0$, $M \in \mathbb{R}^{n \times n}$
Output: The solution $x^{(k)} = \arg\min_x \|Hx - b\|_2$.

$r^{(0)} = b - Hx^{(0)}$
$d^{(0)} = s^{(0)} = M(H^T r^{(0)})$
$\gamma_0 = \|s^{(0)}\|_2^2$
for $k = 0$ **to** $k_{\max}$ **do**
  $t^{(k+1)} = Md^{(k)}$
  $q^{(k+1)} = Ht^{(k+1)}$
  $\alpha_k = \frac{\gamma_k}{\|q^{(k+1)}\|_2^2}$
  $x^{(k+1)} = x^{(k)} + \alpha_k t^{(k+1)}$
  $r^{(k+1)} = r^{(k)} - \alpha_k q^{(k+1)}$
  $s^{(k+1)} = M(H^T r^{(k+1)})$
  $\gamma_{k+1} = \|s^{(k+1)}\|_2^2$
  $\beta_k = \frac{\gamma_{k+1}}{\gamma_k}$
  $d^{(k+1)} = s^{(k+1)} + \beta_k d^{(k)}$
end

main steps of PCGLS applying split preconditioning with preconditioner M. Note that $x^{(0)}$ is an initial approximation, e.g., $x^{(0)} = 0$. By using $M = I$, Algorithm 3 yields the unpreconditioned CGLS method.

As in (P)CGLS the solution norm and the residual norm are monotonic functions in k, i.e.,

$$\|x^{(k)}\|_2 \geq \|x^{(k-1)}\|_2 \qquad \text{and} \qquad \|r^{(k)}\|_2 \leq \|r^{(k-1)}\|_2$$

for every iteration k, respectively, it is possible to use the (discrete) L-curve criterion to estimate the number of iterations k_{opt} which yields an optimal reconstruction in (P)CGLS [70]; see Section 2.5.4.

Further Iterative Regularization Methods

An equivalent approach to CGLS is the *least squares QR* (LSQR) method which uses the Lanczos bidiagonalization [119]. However, both methods require products with H^T making them only applicable if H is not given implicitly as an operator. If H is not available in transposed version it is possible to use *MR-II* [60] in the symmetric indefinite case and *range restricted GMRES* (RRGMRES) [31] in the nonsymmetric case. Both methods are associated with MINRES and GMRES, respectively, but work on a shifted Krylov subspace with initial Krylov basis vector Hb instead of b. This has a smoothing effect on the high frequency components of the observed signal [70]. It is also the reason why MINRES and GMRES do not suppress the noise contribution sufficiently making them unsuitable for the reconstruction of ill-posed image deblurring problems [72]. In a summary, the choice of common iterative regularization methods can be made according to Figure 2.8.

Note that RRGMRES may fail to produce regularized solutions as it mixes the SVD components or if H has an unfavorable null space [70]. Therefore, the dashed arrow illustrates that RRGMRES cannot be used as a black-box method similar to, e.g., CGLS. Further methods worth mentioning are: the *algebraic reconstruction technique* (ART) with its fast initial convergence which made it quite popular for tomography problems in early days, the *ν-method* which is similar to CGLS but converges slower making it less sensitive to the estimated choice of the iteration number k, and so-called *hybrid methods* which combine direct and iterative regularization methods to stabilize the semiconvergence making overestimations still produce nearly optimal reconstructions. See, e.g., the *weighted-GCV* method by Chung et al. [39] which stabilizes the semiconvergence of LSQR, or the references therein.

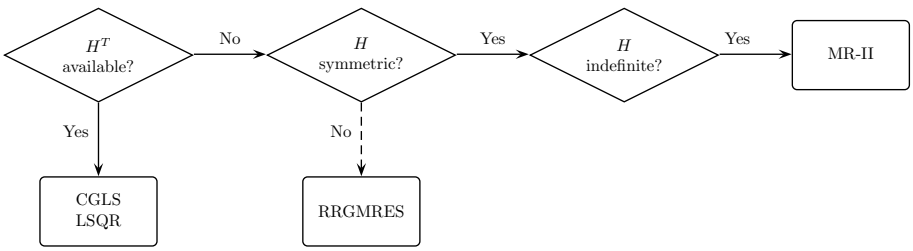


Figure 2.8.: Usage conditions of common iterative Krylov subspace methods with inherent regularization property.

Preconditioned Iterative Regularization Methods

Concerning discrete ill-posed problems, preconditioning usually should accelerate the convergence without destroying the quality of the reconstruction [61, 64, 115]. More precisely, applying a preconditioner to an iterative regularization method can have three positive effects:

- reduce the necessary number of iterations to reach the best reconstruction,
- result in a flat convergence curve such that it is easier to locate the iteration number k with the best solution,
- or result in a better reconstruction of the original signal.

In general, we have to expect that not all three conditions can be reached simultaneously. Therefore, we have to present different preconditioners depending on the application.

Using preconditioners can easily lead to a deterioration of the reconstruction by approximating the inverse also in the noise subspace, or by removing high frequency components in the original signal. Hence, an optimal preconditioner should treat the large singular values and act only on the signal part of the singular spectrum. It should have no effect on the smaller singular values not amplifying the noise. Compared to well-posed problems, this requirement makes it even more difficult to find suitable preconditioners for ill-posed problems. Following [64], a regularizing preconditioner M should have the following properties:

1. $M \approx |H|^{-1}$ on the signal subspace with $|H| = (H^T H)^{\frac{1}{2}}$, and
2. $M \approx I$ or $M \approx 0$ on the noise subspace.

For Toeplitz or circulant matrices, the eigendecomposition is known. Therefore, these conditions can be satisfied by manipulating the spectral values. Most of the preconditioners make use of properties of structured matrices. However, for general blur operators this is usually impossible. For general H, a preconditioner like ILU will lead to faster convergence but the reconstruction will deteriorate because the preconditioner also improves the solution relative to the unwanted noise subspace. Hence, it is even more demanding to develop preconditioners for general H, e.g., for spatially variant blur. Using preconditioners within iterative regularization methods to restore original data has been successfully proposed by Nagy et al. [63, 112, 113], mostly in connection with nearly structured problems. In Chapter 6, we present new (preconditioning) approaches to reconstruct blurred and with noise affected signals more accurately.

2.5.4. Estimating the Regularization Parameter

Finding the optimal regularization parameter—such as α for TPR, the truncation index k for the TSVD, or the number of iterations for iterative regularization methods—is a nontrivial task. There exist several methods which tend to work well under certain conditions but occasionally may fail to produce good reconstructions. Based on the presentation in [62, 68, 70], we briefly introduce common heuristic approaches.

The Discrepancy Principle

If the noise vector η is known—or at least some upper bound of $\|\eta\|_2$—the idea is to choose the regularization parameter such that the residual norm equals the discrepancy in the data. Using the expected value of the perturbation norm $\|\eta\|_2 = \sqrt{n}\xi$, including a safety factor $\nu_{\mathrm{dp}} > 1$ (e.g., $\nu_{\mathrm{dp}} = 1.5$), the optimal regularization parameter for TSVD and TPR is

$$k_{\mathrm{dp}} := \arg\min_k \|Hx_k - b\|_2 \geq \nu_{\mathrm{dp}}\sqrt{n}\xi \qquad \text{and}$$
$$\alpha_{\mathrm{dp}} := \alpha \quad \text{such that} \quad \|Hx_\alpha - b\|_2 = \nu_{\mathrm{dp}}\sqrt{n}\xi,$$

respectively. While this criterion is computationally cheaper than the following ones, it relies on the (accurate) knowledge of $\|\eta\|_2$, which, in general, is not available in practice. Even if $\|\eta\|_2$ is known exactly, the discrepancy principle tends to overestimate the optimum.

Generalized Cross-Validation

The *generalized cross-validation* (GCV) is a statistic-based estimation method which can be applied if an a priori estimate of the noise norm is unavailable. The basic idea is to compute a regularized solution of the modified ill-posed system resulting from omitting the i^{th} equation in the original problem. Afterward, this solution is used to predict the i^{th} component of b. A good choice of the regularization parameter should minimize the prediction errors for all components of the right-hand side. For TPR this can be achieved by

$$\min_\alpha \frac{\|H(H^TH + \alpha^2 I)^{-1}H^Tb - b\|_2^2}{\mathrm{trace}\,[H(H^TH + \alpha^2 I)^{-1}H^T - I]^2} = \min_\alpha \frac{\|Hx_\alpha - b\|_2^2}{\mathrm{trace}\,(HH^\dagger - I)^2}, \tag{2.37}$$

where $H^\dagger = (H^TH + \alpha^2 I)^{-1}H^T$ is the Tikhonov-Phillips *regularized inverse* which produces the regularized solution $x_\alpha := H^\dagger b$. Using the SVD of $H \in \mathbb{R}^{n\times n}$, the denominator can be expressed more generally in terms of filter factors:

$$\mathrm{trace}(HH^\dagger - I)^2 = \left(\sum_{i=1}^n \Phi_i - n\right)^2.$$

Hence, the regularization parameter for TPR and TSVD can be computed via

$$\alpha_{\mathrm{GCV}} := \arg\min_\alpha \frac{\|Hx_\alpha - b\|_2^2}{\left(\sum_{i=1}^n \frac{\sigma_i^2}{\sigma_i^2 + \alpha^2} - n\right)^2} \qquad \text{and} \qquad k_{\mathrm{GCV}} := \arg\min_k \frac{\|Hx_k - b\|_2^2}{(k-n)^2},$$

respectively. Note that for regularizing CG iterations the GCV is not straightforward to compute: the CG regularized inverse $H^\dagger$ is not unique, and therefore the GCV function (2.37) not unique either. See [69] for details on this. Although the GCV is often robust it may fail and produce severe undersmoothed results. Problems may especially occur when the GCV function has a very flat minimum or when the errors are highly correlated [75].

The (Discrete) L-Curve Criterion

Following [62, 67, 70, 75], the *L-curve* approach is based on the "L"-shaped curve defined by the point set

$$\{P_*\} = \{(\rho_*, \delta_*)\} := \{(\log\|Hx_* - b\|_2, \log\|Lx_*\|_2)\}\,.$$

The x_* are solutions from a test sequence $\{x_\alpha\}$ or $\{x_k\}$ resulting from TPR in general form, TSVD, or an iterative regularization method. Note that for regularization in standard form $\delta_* := \log\|x_*\|_2$. By plotting the (semi)norm of the penalty term versus the norm of the model fit in natural logarithmic scale, the features of the curve become more pronounced resulting in a steep part, a flat part, and a distinct corner separating these two regions. The optimal solution often corresponds to a point near this corner which represents its balance between being dominated by regularization errors and being dominated by errors in the right-hand side.

Regularization methods which depend on continuous regularization parameters such as TPR yield smooth L-curves on which the optimal regularization parameter can be located at the point of maximum curvature

$$\alpha_{\mathrm{L}} := \arg\max_{\alpha} \frac{\rho'_\alpha \delta''_\alpha - \rho''_\alpha \delta'_\alpha}{[(\rho'_\alpha)^2 + (\delta'_\alpha)^2]^{\frac{3}{2}}}.$$

However, any implementation of this approach requires a discretized treatment of the curve. The algorithm FindCorner published by Hansen and O'Leary in 1993 [75] and implemented in `l_corner` from [68] was historically one of the first implementations. It fits the point set via a 2D spline approach. It is also possible to apply the L-curve criterion to methods with a discrete regularization parameter. Here, the curve only consists of a finite point set and the optimal regularization parameter can be identified as

$$k_{\mathrm{L}} := k \qquad \text{where } k \text{ is the overall corner of the discrete L-curve.}$$

According to [73, 122], L-curves resulting from TSVD or CGLS regularization tend to cluster in a neighborhood around the corner and may have local steps or loss of convexity making `l_corner` providing a poor or even useless estimation. Hence, several approaches followed to provide robust estimations for this class of regularization methods. Besides the TriangleMethod [32] used to estimate the number of iterations in CG on the normal equations, Rodriguez and Theis proposed the CornerAlgorithm [122] for TSVD methods. Using vectors $v_k := P_{k+1} - P_k$ and angles $\theta_k := \measuredangle(v_k, v_{k+1})$ between them, the corner is estimated at the minimum of all z-coordinates $(\cdot)_z$ of the cross-product

$$(v_k \times v_{k+1})_z = \|v_k\|\,\|v_{k+1}\| \sin\theta_k = \det([v_k\ v_{k+1}]) =: z_k, \tag{2.38}$$

if $\min(z_k) < -\frac{1}{2}$ is satisfied. Based on the CornerAlgorithm, the AdaptivePruning algorithm by Hansen et al. [73] results in a robust, fast, and parameter-free method to estimate the optimal regularization parameter for TSVD and regularized CG iterations. By removing a varying number of points, several corners are located on the convex pruned L-curves relying on the angles between subsequent line segments, similar to (2.38). In a second stage, the overall corner from the candidate list is located as the point with shortest Euclidean distance to the intersection of the line segments. See [73] for a detailed description of the algorithm and MOOre Tools [96] for its implementation.

The (discrete) L-curve criterion remains a heuristic which acts on the assumption that the optimal solution is located at the corner of the curve. For some model problems we observed, similar to [73] for large-scale problems, that the optimal solution may correspond to a point on the horizontal or vertical branch in the L-curve and not to the point with maximum curvature, making the L-curve approaches produce weaker estimations. Further difficulties may arise for model problems with slowly decaying singular values where the corner of the curve becomes less distinct. Indeed, in many practical problems the L-curve may totally lose its characteristic shape. However, the L-curve criterion is at least as robust as the GCV.

It is important to point out that there is no robust criterion which yields good solutions in all cases. The estimation of the regularization parameter is a sensitive process depending on the underlying problem, the form of the noise, and the noise level. In Section 6.2 we will investigate two approaches suitable for (P)CGLS to estimate the optimal number of iterations. Note that there are further methods for the estimation of the (discrete) regularization parameter such as the *quasi-optimality criterion* or the *normalized cumulative periodogram* (NCP). See [62, 70, 74] and the references therein for these and further techniques.

2.5.5. Available Software for Discrete Ill-Posed Problems

For the analysis and solution of discrete ill-posed systems there are several well-known technical realizations. In this thesis, we focus on the first of the following MATLAB packages.

- REGULARIZATION TOOLS[9]: a package for the analysis and solution of discrete ill-posed problems developed and published by Hansen [68].
- RESTORETOOLS[10]: an object oriented package by Nagy et al. [114] with focus on image restoration.
- MOORE TOOLS[11]: a modular object oriented package with emphasis on iterative methods originally developed by Jacobsen and Jensen [96].

9 http://www2.imm.dtu.dk/~pch/Regutools/
10 http://www.mathcs.emory.edu/~nagy/RestoreTools/index.html
11 http://www2.imm.dtu.dk/~pch/MOOReTools/

Chapter 3

Sparse Approximate Inverses

The complexity and computational effort which comes along with the solution of high-dimensional problems makes the use of parallel and distributed architectures indispensable. Usually, at some point, large and sparse linear systems are involved and to be solved efficiently. Therefore, the search for preconditioners that are suitable for high performance computers is an important issue in order to obtain the solution in reasonable time. Besides the problems mentioned in Section 2.3.3, common preconditioning techniques such as ILU or IC are difficult to parallelize. See also Figure 2.6. On the other hand, preconditioners that are easier to parallelize, e.g., polynomial preconditioners, in turn are not effective for general large problems [58].

In this context, sparse approximate inverses based on Frobenius norm minimization represent a resort as the computation of the preconditioner is inherently parallel. It is important to point out that this approach relies on the assumption that for a given sparse matrix A, it is possible to compute a sparse M as a good approximation to A^{-1} [22]. Although the inverse of a sparse matrix A usually is nearly dense, the entries of A^{-1} are rapidly decaying away from the diagonal in many cases; so most of the entries are very small [45]. However, capturing the large (important) values of the inverse is a nontrivial task. Indeed, without a priori knowledge on A^{-1}, this fact caused the invention of plenty of different heuristic techniques, all focusing on the prescription or finding of a proper sparsity pattern for M.

We consider the general nonsymmetric case separately from the SPD one. Both parts are structured predominantly in the same manner. After introducing the inherent parallelism behind the underlying minimization approach, we focus on the computation for a fixed sparsity pattern $\mathcal{J}$. The difficulty of these so-called *static* approaches is to prescribe an a priori sparsity pattern. Hence, we shortly comment on common choices for $\mathcal{J}$. As it is hard to predict patterns which in general lead to a preconditioner of high quality, we introduce the SPAI approach by Grote and Huckle [58] and the factorized variant FSPAI by Huckle [81, 82]. These are adaptive, so-called *dynamic* methods which capture most profitable indices automatically. An index is considered profitable if it leads to a large reduction in the Frobenius norm and the K-condition number, respectively, and thus to a more accurate approximation.

For the general nonsymmetric case we also consider the modified SPAI (MSPAI) ansatz by Huckle and Kallischko [85] which allows M to have a certain action on imposed probing subspaces. In this context, we introduce a novel mask probing ansatz which allows for column specific probing conditions in contrast to the classical global probing approach. We will use MSPAI in order to derive smoothers for multigrid in Chapter 5 and regularizing preconditioners for iterative regularization methods within Chapter 6. The descriptions for the dynamic methods are accurate as we will present implementations for the algorithms in Chapter 7. Each part of the chapter fluently leads to new variants introduced at the end: a recursive multistep successive MSPAI and an exact version of FSPAI.

3.1. Frobenius Norm Minimization for the General Case

A well-known approach to construct an explicit sparse approximate inverse $M \approx A^{-1}$ is based on the idea of finding a matrix M which minimizes the Frobenius norm [15] of the residual matrix $AM - I$. For an a priori prescribed sparsity pattern $\mathcal{J}(M)$, chosen from the set of all possible patterns $\mathcal{J}$, this can be done in a static way by solving the LS problem

$$\min_{\mathcal{J}(M)\in\mathcal{J}} \|AM - I\|_F, \tag{3.1}$$

where $A \in \mathbb{R}^{n\times n}$ is the system matrix, $M \in \mathbb{R}^{n\times n}$ the right preconditioner to be computed, and I the n-dimensional identity matrix. The preconditioner will approximate the inverse with the idea to adopt the sparsity. Hence, when applying it in iterative methods, there is no need in solving a linear system as for incomplete LU approaches and the matrix-vector multiplications are cheap. Moreover, the key feature of minimizing the Frobenius norm is its 2-norm compatibility so that (3.1) can be decoupled into a sum of Euclidean norms

$$\min_{\mathcal{J}(M)\in\mathcal{J}} \|AM - I\|_F^2 = \sum_{k=1}^{n} \min_{\mathcal{J}(M)\in\mathcal{J}} \|(AM - I)e_k\|_2^2 = \sum_{k=1}^{n} \min_{\mathcal{J}(M_k)\in\mathcal{J}_k} \|AM_k - e_k\|_2^2. \tag{3.2}$$

The k^{th} column of M, I, and $\mathcal{J}$ is denoted by M_k, e_k, and $\mathcal{J}_k$ (3.3). Each summand in (3.2) represents an LS problem corresponding to one column M_k of M and can be solved independently from all others. This inherent parallelism of the *sparse approximate inverse* (SAI) approach is its main advantage over other preconditioners. It is possible to build parallel implementations which benefit from the features of modern supercomputers and thus provide a scalable parallel preconditioner which can be efficiently used for large-scale problems.

Remark 3.1 *Let M be the approximate inverse of A and $B := AM - I$. For every matrix norm $\|\cdot\|_C$ exists an equivalent matrix norm $\|\cdot\|_D$ such that*

$$\|AM - I\|_C = \|B\|_C = \|B^T\|_D = \|M^T A^T - I\|_D.$$

Computing a right preconditioner M for A is equivalent to the computation of a left preconditioner M^T for A^T because of the equivalence of norms in finite fields.

3.1.1. Computation for a Fixed Sparsity Pattern $\mathcal{J}$: SAI

An exact computation of the inverse of a sparse A usually leads to a dense preconditioner which should be avoided. Therefore, an a priori chosen sparsity structure $\mathcal{J}$ marks the allowable positions for the preconditioner values. Let $\mathcal{J}_k$ be the set of indices where M_k has nonzero entries, that is

$$\begin{aligned} \mathcal{J}_k &:= \big\{j \in \{1,\ldots,n\} : M_k(j) \neq 0\big\}, \\ q_k &:= |\mathcal{J}_k|\,. \end{aligned} \tag{3.3}$$

Hence, $\mathcal{J}_k$ is the extraction set containing the column indices of A which will reduce the system matrix to $A(.,\mathcal{J}_k)$ while evaluating the matrix-vector product $AM_k(\mathcal{J}_k)$. Due to the sparsity, the corresponding reduced system usually has a lot of zero rows which do not

influence the solution and thus can be ignored. The nonzero rows in $A(.,\mathcal{J}_k)$ can be expressed via the so-called *shadow* of $\mathcal{J}_k$ which denotes the index set

$$\begin{aligned} \mathcal{I}_k &:= \left\{ i \in \{1,\ldots,n\} : \sum_{j \in \mathcal{J}_k} |a_{ij}| \neq 0 \right\}, \\ p_k &:= |\mathcal{I}_k| \,. \end{aligned} \tag{3.4}$$

This provides the second reduction and leads to the submatrix $\hat{A}_k := A(\mathcal{I}_k,\mathcal{J}_k)$. Therewith, each summand in (3.2) can be reduced to a lower dimensional LS problem

$$\min_{\mathcal{J}(\hat{M}_k)=\mathcal{J}_k} \|\hat{A}_k \hat{M}_k - \hat{e}_k\|_2, \quad k = 1,\ldots,n, \tag{3.5}$$

with the definitions

$$\hat{A}_k := A(\mathcal{I}_k,\mathcal{J}_k) \in \mathbb{R}^{p_k \times q_k}, \qquad \hat{M}_k := M_k(\mathcal{J}_k) \in \mathbb{R}^{q_k}, \qquad \text{and} \quad \hat{e}_k := e_k(\mathcal{I}_k) \in \mathbb{R}^{p_k}. \tag{3.6}$$

The dimensions of the LS problems (3.5) are small if A and M are sparse. Hence, we can solve them by direct solution methods, e.g., by methods for normal equations or the modified Gram-Schmidt method. For nonsingular A, $\hat{A}_k$ has full column rank q_k and we can apply the (Householder) QR factorization $\hat{A}_k = \hat{Q}\hat{R}$, where $\hat{Q} \in \mathbb{R}^{p_k \times q_k}$ and $\hat{R} \in \mathbb{R}^{q_k \times q_k}$ according to (2.7). The nonzero entries of $\hat{M}_k$ are computed via

$$\hat{c} := \hat{Q}^T \hat{e}_k \qquad \text{and therewith} \qquad \hat{M}_k := \hat{R}^{-1}\hat{c}. \tag{3.7}$$

Note that a back mapping has to be performed to scatter the entries of $\hat{M}_k$ into the predefined positions in M_k.

The difficulty of SAI is to impose a proper sparsity structure on M. In general, the positions of the important values of the inverse are highly problem dependent and no a priori information is available. Hence, a central task in computing a sparse approximate inverse preconditioner based on Frobenius norm minimization is to identify promising entries. Over the years plenty of different (heuristic) techniques were established.

3.1.2. A Priori Choices of $\mathcal{J}$

An efficient sparsity pattern should act as a filter: avoid entries which contribute little to the quality of M. However, there is no guarantee that by capturing only the large entries the approximation will be a preconditioner of good quality [10]. For special structured matrices such as banded SPD matrices the analysis is known to predict efficient preconditioners. Here, the entries of A^{-1} are bounded in an exponentially decaying manner. Therefore, it is recommendable to prescribe a banded pattern for M if the mentioned decay is not too fast. See [22] and the references therein. However, for general matrices no analysis is available and the choice is usually based on empirical results.

Patterns $\mathcal{J}(A)$ and $\mathcal{J}(A^T)$

A general heuristic is to choose $\mathcal{J}(M) = \mathcal{J}(A)$ [16]. However, this choice may produce poor preconditioners when A^{-1} has large entries outside of $\mathcal{J}(A)$. Moreover, for general

unsymmetric A, where not all diagonal elements are nonzeros, it may happen that SAI produces preconditioners containing zero columns. This fact is not found in literature so far. Figure 3.1 illustrates an abstract example. For the matrix $A \in \mathbb{R}^{4\times 4}$ and $\mathcal{J}(M) = \mathcal{J}(A)$ the first column of M will result in a zero column $M_1 = 0$. This is because the nonzero in e_1 corresponds to a zero row in $A(.,\mathcal{J}_1)$, i.e., $e_1^T A M_1 = 0$. Note that even if the diagonal element is contained in $\mathcal{J}(M_1)$, M_1 will be a zero column.

$$\begin{pmatrix} 0 & \star & 0 & 0 \\ 0 & \star & \star & 0 \\ \star & 0 & \star & \star \\ \star & 0 & 0 & \star \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ \star \\ \star \end{pmatrix} = \begin{pmatrix} \star \\ 0 \\ 0 \\ 0 \end{pmatrix} \quad \Leftrightarrow \quad \begin{pmatrix} \star & 0 \\ \star & \star \\ 0 & \star \end{pmatrix} \begin{pmatrix} \star \\ \star \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \quad \Leftrightarrow \quad M_{1,3} = M_{1,4} = 0 \Rightarrow M \text{ is singular}$$

A $\mathcal{J}(M_1)$ e_1 $\hat{A}_1$ $\hat{M}_1$ $\hat{e}_1$

Figure 3.1.: Example for the construction of a singular SAI preconditioner due to the pattern choice $\mathcal{J}(M) = \mathcal{J}(A)$.

In practice, it is rarely the case that the diagonal of A contains zero components. Hence, the choice $\mathcal{J}(M) = \mathcal{J}(A)$ usually represents a feasible heuristic. However, a remedy and robust simple choice is at hand with

Theorem 3.1 (Nonzero column guarantee for SAI) *If A is regular, then the pattern $\mathcal{J}(A^T)$ will always satisfy the necessary condition $e_k^T A e_j \neq 0$ for any index $j \in \mathcal{J}_k$ and produce a SAI preconditioner with nonzero columns.*

Proof Let $\mathcal{J}_k$ be the k^{th} column of $\mathcal{J}(A^T)$. Then $\forall j \in \mathcal{J}_k$ holds

$$(A_k^T)_j = (A_j)_k = (Ae_j)_k \neq 0 \quad \Rightarrow \quad e_k^T A e_j \neq 0. \qquad \blacksquare$$

Note that similar to $\mathcal{J}(A)$, $\mathcal{J}(A^T)$ may also produce poor SAI preconditioners.

Pattern $\mathcal{J}$ as a Power of Sparsified A

For strongly anisotropic problems the components of the inverse usually decay slowly along one dimension reflecting the anisotropy. Here, the usage of $\mathcal{J}(A)$ most likely will behave isotropically and act very locally including small entries instead of large ones. In this case it can be beneficial to sparsify A a priori and use the resulting pattern as starting point for dynamic strategies like SPAI [132]; see Section 3.1.3. Thus, the important elements are captured while the number of nonzeros in M is kept small. Chow enhanced this idea by using powers of sparsified A as a pattern [37, 38]. This can be motivated by the Neumann expansion of the inverse according to (2.21) for $K = I$. While sparsification helps to detect the anisotropy, the powers of a matrix enlarge the local coupling, i.e., include components along the detected anisotropic direction. We obtain $\tilde{A}$ by sparsifying A based on a chosen threshold $\tau_{\text{thresh}} \geq 0$:

$$\tilde{a}_{ij} := \begin{cases} 1 & \text{if } i = j \text{ or } \left| \left(D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \right)_{ij} \right| > \tau_{\text{thresh}}, \\ 0 & \text{otherwise,} \end{cases} \qquad \text{where} \qquad d_{ii} := \begin{cases} |a_{ii}| & \text{if } |a_{ii}| > 0, \\ 1 & \text{otherwise.} \end{cases} \tag{3.8}$$

The Jacobi scaling with D is beneficial in order to simplify the choice of τ_{thresh}. After thresholding A, the SAI pattern is computed via $\mathcal{J}(M) = \mathcal{J}(\tilde{A}^k)$ for some small power k. However, the heuristic choice of k may produce dense results also depending on the structure of A. Therefore, the computed preconditioner is usually put to a postfiltration phase: components which satisfy $|m_{ij}| < \tau_{\text{filter}}$ are dropped to reduce the costs of applying M without intense corruption of the convergence rate.

Although this approach often requires less time to construct the preconditioner in comparison to dynamic strategies, it has to be used with caution:

- For general A, e.g., if the underlying PDE is isotropic, powers of A may produce poor but dense preconditioners.
- If the dropping in A or M is applied too aggressively, it is possible that M will be singular.
- Dropping may produce poor or useless results if significant information gets lost.

Note that for $\mathcal{J}(\tilde{A}^k)$ the matrix A^k is not constructed explicitly. Refer to [37] for details on the implementation in the Parallel Sparse Approximate Inverse Least-Squares (ParaSails) package by Chow [36].

Upper Bound Patterns

More sophisticated choices of patterns were considered by Huckle in [80]. Based on the Cayley-Hamilton theorem[12], it is possible to express the inverse of A and its corresponding pattern via

$$A^{-1} = -\frac{1}{c_0}\sum_{i=0}^{n-1} c_i A^i \qquad \text{and} \qquad \mathcal{J}(A^{-1}) \subseteq \mathcal{J}\left[(I + |A|)^{n-1}\right], \tag{3.9}$$

respectively. Similarly, we can expand the inverse of $A^T A$ and obtain

$$A^{-1} = -\frac{1}{d_0}\left(\sum_{i=0}^{n-1} d_i (A^T A)^i\right) A^T \quad \text{with the pattern} \quad \mathcal{J}(A^{-1}) \subseteq \mathcal{J}\left[(|A|^T|A|)^{n-1}|A|^T\right]. \tag{3.10}$$

As it is often beneficial to include components of A^T, we can replace $(I + |A|)$ by $(I + |A| + |A|^T)$ to obtain a third initial guess

$$\mathcal{J}(A^{-1}) \subseteq \mathcal{J}\left[(I + |A| + |A|^T)^{n-1}|A|^T\right]. \tag{3.11}$$

Any practical application of these patterns involves truncation in the polynomials after an index k. Fortunately, experiments showed that already for small $k \ll n - 1$, often for $k = 3$, the resulting approximation reaches a comparable—sometimes equal—quality to SPAI which adaptively chooses the most profitable pattern indices. Following [80, 99], we conclude several advantages:

12 Let A be an $n \times n$ matrix, and let $p(\lambda) = \det(\lambda I - A)$ be the characteristic polynomial of A. Then $p(A) = 0$.

- Although being heuristic, sparse patterns are at hand which likely will produce SAI preconditioners of good quality for many matrices with general nonsymmetric structure.
- Constructing M in a parallel distributed memory environment, upper bound patterns provide a more effective data communication between the processors: it is possible to distribute the columns of A a priori in such a way that no remote data exchange will be necessary for the computation of the local preconditioner columns. The notion is to perform heavily clustered global communication at the beginning in order to avoid expensive local communication afterward. However, in general we observed only small attainable speedup during the construction of M when applying this strategy [99, 128]. Note that on architectures with poor data throughput this may cause network congestion slowing down the overall setup stage.
- The SPAI approach computes the obtainable improvement of many indices during each update step. By using an upper bound pattern within SPAI, the number of indices—which have to be tested—can be significantly reduced. Thus, the setup becomes computationally cheaper. Experiments in [99] demonstrate this fact.

Note that if the upper bounds in (3.9)–(3.11) are used without absolute value and A contains negative entries, round-off errors may produce cancellation and thus sparser patterns. But still these patterns are contained in the upper bound patterns yielding comparable quality to SPAI.

3.1.3. Updating the Sparsity Pattern $\mathcal{J}$: SPAI

In 1997, Grote and Huckle published the *sparse approximate inverse* (SPAI) approach [58]. It is an additional feature in the Frobenius norm minimization that introduces different strategies for choosing new profitable indices in M_k in order to improve on an already computed approximation. In the following we assume—by solving (3.2) for a given index set $\mathcal{J}_k$—having already determined an optimal LS solution $M_k(\mathcal{J}_k)$ inducing the sparse vector M_k with residual r_k.

Pattern Updates

Dynamically, we want to augment new entries in M_k and solve (3.2) for an enlarged index set such that we obtain a reduction in the norm of the new residual without affecting the inherent parallelism. An improvement becomes possible if $\|r_k\|_2 = \|AM_k - e_k\|_2$ of a column is positive. Otherwise $r_k = 0$ and M_k is already the exact inverse of the k^{th} column of A. The aim is to reduce the magnitude of the nonzero entries of r_k which form the index set

$$\mathcal{L}_k := \big\{\ell \in \{1,\ldots,n\} : r_k(\ell) \neq 0\big\} \cup \{k\}.$$

For every $\ell \in \mathcal{L}_k$ we can specify an index set $\mathcal{N}_\ell := \{j : a_{\ell j} \neq 0\}$ which contains those nonzero elements of the ℓ^{th} row of A which are not in $\mathcal{J}_k$. The union of all these sets $\mathcal{N}_\ell$, $\ell \in \mathcal{L}_k$, results in a final index set

$$\tilde{\mathcal{J}}_k := \bigcup_{\ell \in \mathcal{L}_k} \mathcal{N}_\ell, \tag{3.12}$$

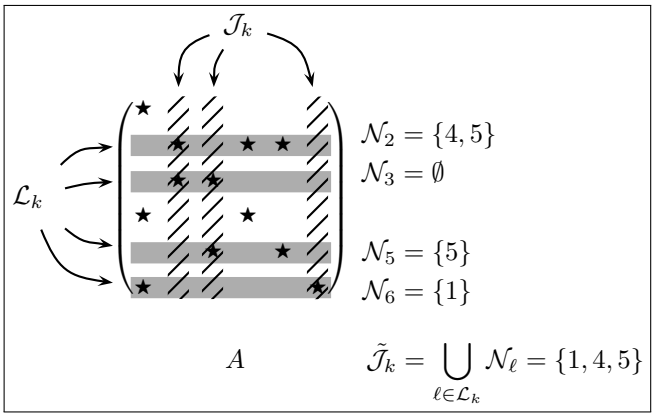


Figure 3.2.: Construction of index set $\tilde{\mathcal{J}}_k = \{1,4,5\}$ (3.12) referred to pattern $\mathcal{J}_k = \{2,3,6\}$ for an exemplary matrix A.

with $\tilde{\mathcal{J}}_k \cap \mathcal{J}_k = \emptyset$. See Figure 3.2 for an example. $\tilde{\mathcal{J}}_k$ represents the current set of DsOF which will lead to a reduction in r_k. As $|\tilde{\mathcal{J}}_k|$ can be large, the sparsity pattern should only be augmented by candidates yielding the most profitable reduction. To obtain a quality metric we consider the univariate minimization problem

$$\min_{m_{jk}} \|A(M_k + m_{jk}e_j) - e_k\|_2 = \min_{m_{jk}} \|r_k + m_{jk}Ae_j\|_2. \tag{3.13}$$

Using $Ae_j = A_j$, the solution of (3.13) is given by

$$\frac{\mathrm{d}}{\mathrm{d}m_{jk}}\|r_k + m_{jk}A_j\|_2^2 = 2r_k^T A_j + 2m_{jk}\|A_j\|_2^2 \overset{!}{=} 0 \Leftrightarrow m_{jk} = -\frac{r_k^T A_j}{\|A_j\|_2^2}. \tag{3.14}$$

As the second derivative $\frac{\mathrm{d}^2}{\mathrm{d}m_{jk}^2}\|r_k + m_{jk}A_j\|_2^2 = 2\|A_j\|_2^2$ is strictly positive, m_{jk} minimizes the new residual norm. Using (3.14) we can compute the 2-norm ρ_{jk} of the new residual which is gained by adding the index j to $\mathcal{J}_k$ via

$$\begin{aligned}
\rho_{jk}^2 := \|r_k + m_{jk}A_j\|_2^2 &= r_k^T r_k + 2m_{jk}r_k^T A_j + m_{jk}^2 A_j^T A_j \\
&= \|r_k\|_2^2 - 2\frac{(r_k^T A_j)^2}{\|A_j\|_2^2} + \frac{(r_k^T A_j)^2}{\|A_j\|_2^4}\|A_j\|_2^2 \\
&= \|r_k\|_2^2 - \frac{(r_k^T A_j)^2}{\|A_j\|_2^2}.
\end{aligned} \tag{3.15}$$

Hence, the factor which reduces the old residual norm is given by $\tau_{jk} := \|A_j\|_2^{-2}(r_k^T A_j)^2$. We update the current pattern with one or several indices $\mathcal{J}_{\text{new}} \subset \tilde{\mathcal{J}}_k$ which yield the largest reduction of $\|r_k\|_2$, i.e., which correspond to the smallest ρ_{jk}. Solving the LS problem for this augmented index set $\mathcal{J}_k = \mathcal{J}_k \cup \mathcal{J}_{\text{new}}$ (according to Section 3.1.1) will produce a more accurate approximation. This updating mechanism overcomes the main drawback of providing a priori sparsity patterns for SAI: based on an initial sparsity structure, e.g., $\mathcal{J}(M) = \mathcal{J}(I)$, the pattern is updated with profitable indices improving the quality. Without stopping criteria SPAI's updating mechanism will produce full patterns. See the following

Theorem 3.2 (Improvement guarantee) *There is at least one index j with $\tau_{jk} > 0$ if the current residual $r_k \neq 0$.*

Proof Assuming $r_k \neq 0$ and $\tau_{jk} = 0$ for all $j \in \{1,\ldots,n\}$, we obtain

$$\forall j : \tau_{jk} = \frac{(r_k^T A_j)^2}{\|A_j\|_2^2} = 0 \quad \Leftrightarrow \quad \forall j : A_j^T r_k = 0 \quad \Leftrightarrow \quad A^T r_k = 0 \quad \overset{A \text{ nonsingular}}{\Leftrightarrow} \quad r_k = 0$$

which is a contradiction of the assumption. ■

On the one hand, it usually suffices to perform only a couple of update steps to reach acceptable acceleration or convergence applied in an iterative solver. On the other hand, every update step involves both higher construction costs and a denser preconditioner. The search for a sensible trade-off between costs for constructing M and a sufficiently good approximation always remains a problem dependent task. Therefore, existing implementations (see Chapter 7) usually offer parameters which indirectly allow to control the fill-in and the accuracy of M. Following [79] and the numerical experiments using our codes, we can specify guidelines which proved to yield a good balance in general:

- Equivalently to the usage of iterative solvers, SPAI should stop improving on M_k if its residual norm satisfies a given tolerance, i.e., $\|r_k\|_2 \leq \varepsilon_{\text{SPAI}}$. Common choices are $\varepsilon_{\text{SPAI}} \in [0.1,0.5]$. Note that for difficult (severely ill-conditioned) problems it can be necessary to use a much smaller value for $\varepsilon_{\text{SPAI}}$.
- Applying a couple of pattern updates α (e.g., $\alpha \in \{1,\ldots,5\}$) will significantly improve the approximation.
- We recommend to add several indices per update step while bounding their number by β (e.g., $\beta \in \{3,\ldots,8\}$). This avoids cases where $\rho_{jk} \approx \|r_k\|_2$ due to $\tau_{jk} \to 0$ for an index j which turns out to be superfluous. This may happen as (3.13) only stands for a heuristic yielding local optimal enlargement of a previous $\mathcal{J}$. Adding several indices raises the probability of finding profitable index sets yielding large improvement.
- In order to preserve little fill-in, it is advantageous to bound the maximum number of possible entries in M_k by $p_{\max} \in \mathcal{O}\left(\frac{\text{nnz}(A)}{n}\right)$.
- One possible heuristic for augmenting the sparsity structure is to use the arithmetic mean

$$\rho_{\text{opt}} \leq \bar{\rho}_k := \frac{1}{|\tilde{\mathcal{J}}_k|} \sum_{j \in \tilde{\mathcal{J}}_k} \rho_{jk} \leq \|r_k\|_2 \tag{3.16}$$

  which is put into practice by the well-known SPAI implementation of Barnard et al. [10] or our (M)SPAI implementation [86, 87] (see Section 7.2). Indices will be accepted if their $\rho_{jk} \leq \bar{\rho}_k$. Thus, unprofitable indices with small improvement are omitted within each iteration [79].

We summarize the main steps of SPAI using pattern updates in Algorithm 4. Following [99], we receive a compact notation for the most commonly used parameters with the

Definition 3.1 (Update setting Υ) *By $\Upsilon_{\alpha,\beta}$ we denote the parameter for SPAI-like algorithms such that in each of the maximum α pattern update steps exactly β indices are added to the pattern of one column (unless all positions in that column are set). $\widetilde{\Upsilon}_{\alpha,\beta}$ indicates the mean value heuristic (3.16) by which only indices are added with corresponding values $\rho_{jk} \leq \bar{\rho}_k$.*

Algorithm 4: SPAI with univariate minimization using the mean value heuristic (3.16) and working on reduced intermediate solutions according to (3.17).

Input : $A \in \mathbb{R}^{n \times n}$, $\mathcal{J} \in \mathbb{B}^{n \times n}$, $\varepsilon_{\text{SPAI}} \geq 0$, $\alpha \geq 0$, $\beta \geq 0$
Output: Preconditioner $M \approx A^{-1}$, $M \in \mathbb{R}^{n \times n}$

for $k \leftarrow 1$ **to** n **do**
  $e_k \leftarrow I(.,k)$
  $\mathcal{J}_k \leftarrow \mathcal{J}(.,k)$
  for `step` $\leftarrow 0$ **to** α **do**
    $\mathcal{I}_k \leftarrow \left\{ i \in \{1,\ldots,n\} : \sum_{j \in \mathcal{J}_k} |a_{ij}| \neq 0 \right\}$
    $\hat{A}_k \leftarrow A(\mathcal{I}_k,\mathcal{J}_k)$
    $\hat{e}_k \leftarrow e_k(\mathcal{I}_k)$
    $\hat{Q},\hat{R} \leftarrow \text{qr}(\hat{A}_k)$
    $\hat{M}_k \leftarrow \hat{R}^{-1}\hat{Q}^T\hat{e}_k$
    $\hat{r}_k \leftarrow \hat{A}\hat{M}_k - \hat{e}_k$
    if $\|\hat{r}_k\|_2 < \varepsilon_{\texttt{SPAI}}$ **then**
      break
    end
    $\mathcal{L}_k \leftarrow \mathcal{I}_k \cup \{k\}$
    foreach $\ell \in \mathcal{L}_k$ **do**
      $\mathcal{N}_\ell \leftarrow \{j : a_{\ell j} \neq 0\}$
    end
    $\tilde{\mathcal{J}}_k \leftarrow \bigcup_{\ell \in \mathcal{L}_k} \mathcal{N}_\ell$
    $\bar{\rho}_k \leftarrow 0$
    foreach $j \in \tilde{\mathcal{J}}_k$ **do**
      $\rho_{jk} \leftarrow \left(\|\hat{r}_k\|_2^2 - \frac{\left[\hat{r}_k^T A_j(\mathcal{I}_k)\right]^2}{\|A_j(\mathcal{I}_k)\|_2^2} \right)^{\frac{1}{2}}$
      $\bar{\rho}_k \leftarrow \bar{\rho}_k + \rho_{jk}$
    end
    $\bar{\rho}_k \leftarrow \frac{\bar{\rho}_k}{\left|\tilde{\mathcal{J}}_k\right|}$
    for `idx` $\leftarrow 1$ **to** β **do**
      $j \leftarrow \arg\min_{j \in \tilde{\mathcal{J}}_k} \rho_{jk}$
      $\mathcal{J}_k \leftarrow \mathcal{J}_k \cup \{j : \rho_{jk} \leq \bar{\rho}_k\}$
      $\tilde{\mathcal{J}}_k \leftarrow \tilde{\mathcal{J}}_k \setminus \{j\}$
    end
  end
  $M_k(\mathcal{J}_k) \leftarrow \hat{M}_k$
end

Remark 3.2

1. *A different and more expensive way to determine a new profitable index j considers the more accurate problem*

$$\min_{\mathcal{J}(M_k)=\mathcal{J}_k\cup\{j\}} \|A(.,\mathcal{J}_k \cup \{j\})M_k(\mathcal{J}_k \cup \{j\}) - e_k\|_2$$

introduced by Gould and Scott [55]. For the augmented set $\mathcal{J}_k \cup \{j\}$ the optimal reduction of the residual is determined for the full minimization problem instead of for the univariate minimization (3.13).

2. *It is possible to update $\mathcal{J}_k$ on the reduced intermediate solutions $\|\hat{r}_k\|_2 = \|\hat{A}_k\hat{M}_k - \hat{e}_k\|_2$. Computing $\tilde{\mathcal{J}}_k$ via (3.12), the solution of the reduced univariate minimization problem*

$$\min_{m_{jk}} \|\hat{A}_k\hat{M}_k - \hat{e}_k + m_{jk}A(\mathcal{I},.)e_j\|_2 \;=\; \min_{m_{jk}} \|\hat{r}_k + m_{jk}A(\mathcal{I},j)\|_2 \tag{3.17}$$

is similar to (3.14) with new squared residual norms corresponding to (3.15).

3. *For complex valued problems $Ax = b$, where $A \in \mathbb{C}^{n\times n}$ and $x, b \in \mathbb{C}^n$, we obtain similar formulae (3.14) and (3.15):*

$$\begin{aligned}
\frac{\mathrm{d}}{\mathrm{d}m_{jk}}\|r_k + m_{jk}A_j\|_2^2 &= \frac{\mathrm{d}}{\mathrm{d}m_{jk}}\left(r_k^H r_k + r_k^H m_{jk}A_j + A_j^H m_{jk} r_k + A_j^H m_{jk}^2 A_j\right) \\
&= \frac{\mathrm{d}}{\mathrm{d}m_{jk}}\left(\|r_k\|_2^2 + 2m_{jk}\mathfrak{Re}\left[r^H A_j\right] + m_{jk}^2\|A_j\|_2^2\right) \stackrel{!}{=} 0 \\
\Leftrightarrow\quad m_{jk} &= -\frac{\mathfrak{Re}\left[r_k^H A_j\right]}{\|A_j\|_2^2}
\end{aligned}$$

and for the new squared residual norm

$$\begin{aligned}
\rho_{jk}^2 = \|r_k + m_{jk}A_j\|_2^2 &= \|r_k\|_2^2 + 2m_{jk}\mathfrak{Re}\left[r^H A_j\right] + m_{jk}^2\|A_j\|_2^2 \\
&= \|r_k\|_2^2 - 2\frac{\mathfrak{Re}\left[r_k^H A_j\right]}{\|A_j\|_2^2}\mathfrak{Re}\left[r_k^H A_j\right] + \frac{\left(\mathfrak{Re}\left[r_k^H A_j\right]\right)^2}{\|A_j\|_2^4}\|A_j\|_2^2 \\
&= \|r_k\|_2^2 - \frac{\left(\mathfrak{Re}\left[r_k^H A_j\right]\right)^2}{\|A_j\|_2^2}.
\end{aligned}$$

Note that our (M)SPAI implementation (see Chapter 7) works on the reduced problems (3.17) to gain higher performance and offers the possibility to setup complex preconditioners according to 3.

QR Updates

Augmenting the sparsity pattern during each update step results in expanding LS problems. Hence, more computational effort is induced due to expensive QR factorizations which are typically used to solve (3.5) in SPAI implementations. Instead of performing a new factorization, it is possible to update the one available from the previous computation step [79, 58]. Numerical experiments with our (M)SPAI implementation show that updating a

known factorization is an essential approach to save computational costs for the construction of M [99, 128].

Fully augmenting $\mathcal{J}_k$ with $\tilde{\mathcal{J}}_k$, we denote the set of new nonzero rows of $A(.,\mathcal{J}_k \cup \tilde{\mathcal{J}}_k)$ by

$$\tilde{\mathcal{I}}_k := \left\{ i \in \{1,\dots,n\} \setminus \mathcal{I}_k : \sum_{j \in \tilde{\mathcal{J}}_k} |a_{ij}| \neq 0 \right\},$$

whereas $\tilde{p}_k := \left|\tilde{\mathcal{I}}_k\right|$ and $\tilde{q}_k := \left|\tilde{\mathcal{J}}_k\right|$. This leads to an enlarged submatrix

$$\bar{A}_k := A(\mathcal{I}_k \cup \tilde{\mathcal{I}}_k, \mathcal{J}_k \cup \tilde{\mathcal{J}}_k) \in \mathbb{R}^{(p_k+\tilde{p}_k)\times(q_k+\tilde{q}_k)}$$

and enlarged LS problem (3.5). Adding new rows generates zero entries in $\bar{A}_k$. These can be collected in a lower left zero block by a row P_r and column P_c permutation of $\bar{A}_k$ such that

$$P_r \bar{A}_k P_c =: \tilde{A}_k = \begin{pmatrix} \hat{A}_k & A(\mathcal{I}_k,\tilde{\mathcal{J}}_k) \\ 0 & A(\tilde{\mathcal{I}}_k,\tilde{\mathcal{J}}_k) \end{pmatrix}.$$

Refer to Figure 3.3 for an illustration. Solving (3.5) we can use the available QR factorization

$$\hat{A}_k = QR = (\hat{Q},\tilde{Q}) \begin{pmatrix} \hat{R} \\ 0 \end{pmatrix} \qquad \text{with} \qquad Q \in \mathbb{R}^{p_k \times p_k},\ R \in \mathbb{R}^{p_k \times q_k},\ \text{and } \hat{R} \in \mathbb{R}^{q_k \times q_k}$$

to obtain

$$\tilde{A}_k = \begin{pmatrix} Q & \\ & I_{\tilde{p}_k} \end{pmatrix} \begin{pmatrix} R & Q^T A(\mathcal{I}_k,\tilde{\mathcal{J}}_k) \\ 0 & A(\tilde{\mathcal{I}}_k,\tilde{\mathcal{J}}_k) \end{pmatrix} = \begin{pmatrix} Q & \\ & I_{\tilde{p}_k} \end{pmatrix} \left(\begin{array}{c|c} \hat{R} & Q_1^T A(\mathcal{I}_k,\tilde{\mathcal{J}}_k) \\ \hline 0 & Q_2^T A(\mathcal{I}_k,\tilde{\mathcal{J}}_k) \\ 0 & A(\tilde{\mathcal{I}}_k,\tilde{\mathcal{J}}_k) \end{array}\right)$$

$$= \begin{pmatrix} Q & \\ & I_{\tilde{p}_k} \end{pmatrix} \left(\begin{array}{c|c} \overbrace{\begin{matrix} \ast & \ast & \ast \\ & \ast & \ast \\ & & \ast \end{matrix}}^{\hat{R}} & \begin{matrix} \tilde{\square} & \tilde{\square} & \tilde{\square} \\ \tilde{\square} & \tilde{\square} & \tilde{\square} \\ \tilde{\square} & \tilde{\square} & \tilde{\square} \end{matrix} \} B_1 \\ \hline \multirow{2}{*}{0} & \begin{matrix} \tilde{\square} & \tilde{\square} & \tilde{\square} \\ \tilde{\square} & \tilde{\square} & \tilde{\square} \end{matrix} \\ & \left.\begin{matrix} \tilde{\square} & \tilde{\square} & \tilde{\square} \\ \tilde{\square} & \tilde{\square} & \tilde{\square} \\ \tilde{\square} & \tilde{\square} & \tilde{\square} \end{matrix}\right\} B_2 \end{array}\right) = \begin{pmatrix} Q & \\ & I_{\tilde{p}_k} \end{pmatrix} \left(\begin{array}{c|c} \hat{R} & B_1 \\ \hline 0 & B_2 \end{array}\right).$$

The matrices Q_1, Q_2 are defined as

$$Q_1 := Q_{:,1:q_k} \in \mathbb{R}^{p_k \times q_k} \qquad \text{and} \qquad Q_2 := Q_{:,q_k+1:p_k} \in \mathbb{R}^{p_k \times (p_k-q_k)}.$$

The symbol $\ast$ represents the entries of $\hat{R}$ and $\tilde{\square}$ the entries of $Q^T A(\mathcal{I}_k,\tilde{\mathcal{J}}_k)$. Consequently, we can write $\tilde{A}_k$ as a matrix containing the old QR factorization and the two new blocks $B_1 \in \mathbb{R}^{q_k \times \tilde{q}_k}$ and $B_2 \in \mathbb{R}^{(p_k-q_k+\tilde{p}_k)\times \tilde{q}_k}$. In a summary, the new QR factorization of $\tilde{A}_k$ can be reduced to a QR factorization of the lower right block $B_2 = Q_B R_B$. Therefore, we obtain

$$\tilde{A}_k = \begin{pmatrix} Q & \\ & I_{\tilde{p}_k} \end{pmatrix} \begin{pmatrix} I_{q_k} & \\ & Q_B \end{pmatrix} \begin{pmatrix} \hat{R} & B_1 \\ 0 & R_B \end{pmatrix}. \tag{3.18}$$

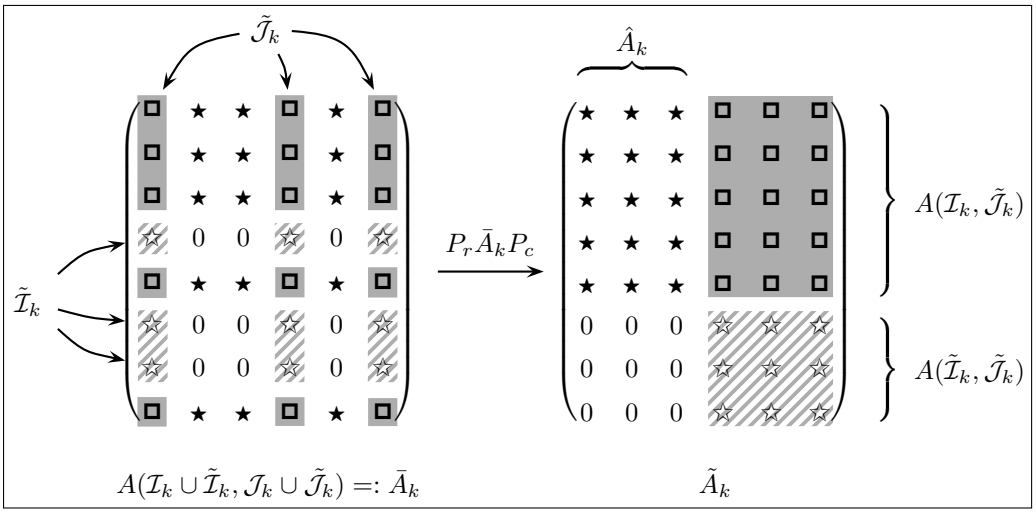


Figure 3.3.: Enlargement of $\hat{A}_k$ (★ values) with index sets $\tilde{\mathcal{I}}_k$ and $\tilde{\mathcal{J}}_k$ producing $\bar{A}_k$. Via row and column permutations P_r and P_c, the resulting matrix $\tilde{A}_k$ contains a zero block. The symbol ◘ stands for the entries in the old rows and new columns in $A(\mathcal{I}_k,\tilde{\mathcal{J}}_k)$, and ☆ denotes the entries in the new columns and new rows $A(\tilde{\mathcal{I}}_k, \tilde{\mathcal{J}}_k)$.

With $\tilde{e}_k := P_r \bar{e}_k = P_r e_k(\mathcal{I}_k \cup \tilde{\mathcal{I}}_k)$ and $\tilde{M}_k := P_c^T \bar{M}_k = P_c^T M_k(\mathcal{J}_k \cup \tilde{\mathcal{J}}_k)$, we have to solve the augmented LS problem

$$\min_{\mathcal{J}(\tilde{M}_k)=\mathcal{J}_k \cup \tilde{\mathcal{J}}_k} \|\tilde{A}_k \tilde{M}_k - \tilde{e}_k\|_2.$$

A final column permutation $M_k(\mathcal{J}_k \cup \tilde{\mathcal{J}}_k) = P_c \tilde{M}_k$ recovers the solution components in correct order. In this manner the computational costs are reduced to one application of Q^T and the QR factorization of the smaller block B_2, instead of computing the full QR factorization of the augmented system $\bar{A}_k$. Following Kallischko [99], Algorithm 5 summarizes the QR update procedure.

Algorithm 5: Updating an available QR factorization within SPAI.

Input : A, Q, R, $\mathcal{I}_k$, $\mathcal{J}_k$, $\tilde{\mathcal{I}}_k$, $\tilde{\mathcal{J}}_k$
Output: $M_k(\mathcal{J}_k \cup \tilde{\mathcal{J}}_k)$

1 $\bar{A}_k \leftarrow A(\mathcal{I}_k \cup \tilde{\mathcal{I}}_k, \mathcal{J}_k \cup \tilde{\mathcal{J}}_k)$
2 $\tilde{A}_k \leftarrow P_r \bar{A}_k P_c$, where $P_r, P_c \in \mathbb{B}^{(p_k+\tilde{p}_k)\times(q_k+\tilde{q}_k)}$
3 $\breve{A} \leftarrow Q^T A(\mathcal{I}_k, \tilde{\mathcal{J}}_k)$
4 $B_1 \leftarrow \breve{A}_{1:q_k,:}$
5 $B_2 \leftarrow \begin{pmatrix} \breve{A}_{(q_k+1):p_k,:} \\ A(\tilde{\mathcal{I}}_k, \tilde{\mathcal{J}}_k) \end{pmatrix}$
6 $Q_B, R_B \leftarrow \mathrm{qr}(B_2)$ (QR factorization of B_2)
7 $\tilde{M}_k \leftarrow \underset{\mathcal{J}(\tilde{M}_k)=\mathcal{J}_k \cup \tilde{\mathcal{J}}_k}{\arg\min} \|\tilde{A}_k \tilde{M}_k - \tilde{e}_k\|_2$
8 $M_k(\mathcal{J}_k \cup \tilde{\mathcal{J}}_k) \leftarrow P_c \tilde{M}_k$

Norm and Spectrum Analysis

As already introduced in Section 2.3.3, the convergence of iterative methods crucially depends on various properties of the (preconditioned) system. The success of a preconditioner mainly depends on its ability to redistribute the spectrum of the preconditioned matrix. We present some norm and spectrum properties of SPAI which show its theoretical effectiveness. Assuming that $\|r_k\|_2 = \|AM_k - e_k\|_2 < \varepsilon_{\text{SPAI}}$ implies

Theorem 3.3 (Norm properties of SPAI) *Let* $p := \max_{1\le k\le n}\{nnz \text{ in } r_k\} \ll n$ *because* A *and* M *are sparse. Then*

$$\begin{aligned} \|AM - I\|_F &\le \sqrt{n}\varepsilon_{SPAI}, & \|M - A^{-1}\|_F &\le \|A^{-1}\|_2\sqrt{n}\varepsilon_{SPAI} \\ \|AM - I\|_2 &\le \sqrt{n}\varepsilon_{SPAI}, & \|M - A^{-1}\|_2 &\le \|A^{-1}\|_2\sqrt{n}\varepsilon_{SPAI} \\ \|AM - I\|_1 &\le \sqrt{p}\varepsilon_{SPAI}, \quad \textit{and} & \|M - A^{-1}\|_1 &\le \|A^{-1}\|_1\sqrt{p}\varepsilon_{SPAI}. \end{aligned} \tag{3.19}$$

For the proof see [35, 58]. Taking Theorem 3.3 into account, SPAI satisfies spectral properties summarized in

Theorem 3.4 (Spectral properties of SPAI)

1. *With* $p := \max_{1\le k\le n}\{nnz \text{ in } r_k\}$ *the eigenvalues of* AM *are clustered at* 1 *and lie inside a circle of radius* $\sqrt{p}\varepsilon_{SPAI}$. *If* $\sqrt{p}\varepsilon_{SPAI} < 1$, *then the extremal eigenvalues* λ_{max} *and* λ_{min} *of* AM *satisfy*
$$\left|\frac{\lambda_{max}}{\lambda_{min}}\right| \le \frac{1+\sqrt{p}\varepsilon_{SPAI}}{1-\sqrt{p}\varepsilon_{SPAI}}.$$
2. *The singular values of* AM *are clustered at 1 and lie inside the interval* $[1-\delta, 1+\delta]$, *with* $\delta = \sqrt{n}\varepsilon_{SPAI}(2+\sqrt{n}\varepsilon_{SPAI})$. *If* $\delta < 1$, *then the condition number of* AM *satisfies*
$$\kappa_2(AM) \le \sqrt{\frac{1+\delta}{1-\delta}}.$$
3. *The preconditioned matrix* AM *has a controllable departure from normality*[13].

The proofs for *1.* and *2.* of Theorem 3.4 can be found in [35, 58]. For *3.* we follow [58] and perform the

Proof Let $Q(AM)Q^T = \Lambda_{AM} + N$ be a Schur decomposition[14] of AM, where Λ_{AM} denotes the matrix $\text{diag}(\lambda_1, \ldots, \lambda_n)$ and $(\lambda_k)_{k=1,\ldots,n}$ are the eigenvalues of AM. Then we observe

$$Q^T(AM - I)Q = \Lambda_{AM} - I + N \tag{3.20}$$

13 Let $Q^{-1}AQ = Q^TAQ = \Lambda_A + N$ be the Schur decomposition[14] for a matrix $A \in \mathbb{R}^{n\times n}$. Then the quantity
$$\|N\|_F = \sqrt{\|A\|_F^2 - \|\Lambda_A\|_F^2}$$
indicates how non-normal A is, i.e., gives A's departure from normality [35].

14 Also known as Schur's lemma [5]: for any matrix $A \in \mathbb{R}^{n\times n}$ there exists an orthogonal matrix Q, such that $Q^{-1}AQ = Q^TAQ = \Lambda_A + N$, where N is a strictly upper triangular matrix and Λ_A is a diagonal matrix containing the eigenvalues of A, i.e., $\Lambda_A := \text{diag}(\lambda_1, \ldots, \lambda_n)$.

and by using (3.19) that

$$\sum_{k=1}^{n} \left|\lambda_k - 1\right|^2 = \|\mathrm{diag}(AM - I)\|_2^2 \leq \|AM - I\|_F^2 \leq n\varepsilon_{\text{SPAI}}^2. \tag{3.21}$$

By passing (3.20) to norms we obtain

$$\|\Lambda_{AM} - I + N\|_F^2 \leq \|\Lambda_{AM} - I\|_F^2 + \|N\|_F^2 = \sum_{k=1}^{n} \left|\lambda_k - 1\right|^2 + \|N\|_F^2 \leq n\varepsilon_{\text{SPAI}}^2 + \|N\|_F^2.$$

Hence, the preconditioned system AM has a controllable departure from normality. Furthermore, (3.21) shows that SPAI also reduces A's departure from normality. ■

A large unbounded departure from normality causes strong oscillations of the eigenvalues if A is affected with a small error [35, 53]. This spectral instability may also deteriorate the numerical quality of certain iterative methods as they become backward unstable. Applying SPAI produces a matrix AM with a bounded small departure as $\|AM - I\|_F$ is minimized. Hence, the non-normality has only local influence on the spectrum and erroneous data in the input will not have a noticeable effect on the convergence of iterative solvers.

By means of Theorem 3.3 and the work in [140], it is possible to proof that under certain conditions SPAI will produce a nonsingular preconditioner. We gather the results in

Theorem 3.5 (Nonsingularity of SPAI)

1. *If* $\sqrt{p}\varepsilon_{SPAI} < 1$ *then* M *is nonsingular.*
2. *Let* $R := AM - I$ *be the residual matrix with* $R \in \mathbb{R}^{n\times n}$ *and* $\mathcal{J}(M) \in \mathcal{J} \supset \mathcal{J}(I)$*. If* $\sum_{k=1}^{n} |r_{kk}| < 1$*, then* $\|R\|_F = \|AM - I\|_F < 1$ *and* M *is nonsingular.*

The corresponding proofs can be found in [58] for *1.* and in [99, 140] for *2.* The latter also holds for complex valued systems when taking the properties of $\mathbb{C}$ into account. Hence, by computing the diagonal elements of $AM - I$, it is possible check for the nonsingularity of the SPAI matrix M.

In a summary, $\min_{\mathcal{J}(M)\in\mathcal{J}} \|AM - I\|_F$ generates a preconditioner which yields a preconditioned system of favorable properties accelerating the convergence of iterative methods: clustered eigenvalues and singular values with a reduction of the spectral condition number while maintaining a controllable departure from normality.

3.1.4. Modified SPAI: MSPAI

Holland, Shaw, and Wathen [77] have generalized the SPAI ansatz (3.1) allowing a sparse target matrix B on the right-hand side leading to

$$\min_{\mathcal{J}(M)\in\mathcal{J}} \|AM - B\|_F. \tag{3.22}$$

This approach is useful in connection with some kind of two-level preconditioning: first compute a standard sparse preconditioner B for A, i.e., target the operator A, and then improve this preconditioner by an additional Frobenius norm minimization with target B such that $B^{-1}AM \approx I$. This is useful for instance in advection-diffusion equations [77]

where the diffusion term involves a Laplacian system. Here, it is beneficial to target the Laplacian B_L and use multigrid for the action of B_L^{-1}. From an algorithmic point of view the minimization with sparse target matrix B, instead of I, introduces no additional difficulties. Simply $\mathcal{J}(M)$ should be chosen more carefully with respect to A and B.

The approach of approximating an arbitrary (target) matrix using only matrix-vector products is known as *probing*. Originally, this classical algebraic approximation technique [5, 8, 34] was applied to precondition Schur complements S. Especially by using 2D nonoverlapping domain decomposition methods in isotropic diffusion applications, it is possible to construct a banded matrix which approximates the large values of S accurately without explicitly forming S. Based on an a priori chosen matrix $E := (e_1, \ldots, e_l) \in \mathbb{B}^{n \times l}$ of probing vectors $e_i \in \mathbb{B}^n$, it is possible, e.g., to construct a banded preconditioner M for S by reading off the entries after equating $ME \stackrel{!}{=} SE$. See [34] for details on the original probing approach. Note that for matrices with general decay patterns it is unclear how to capture the large entries. Without explicit knowledge of the underlying problem A, probing allows to construct preconditioners which mimic the action of A on certain subspaces. The disadvantage is a proper choice of the probing vectors: the system for M must remain to be solved easily and it can be difficult to ensure properties such as symmetry or positive definiteness.

Generalized Form of Frobenius Norm Minimization

In [85, 99], Huckle and Kallischko proposed to combine the SPAI minimization (3.1) with targeting (3.22) and classical probing. This leads to the *modified* SPAI (MSPAI) ansatz

$$\begin{aligned} \min_{\mathcal{J}(M) \in \mathcal{J}} \|CM - B\|_F &= \min_{\mathcal{J}(M) \in \mathcal{J}} \left\| \begin{pmatrix} C_0 \\ \rho e^T \end{pmatrix} M - \begin{pmatrix} B_0 \\ \rho f^T \end{pmatrix} \right\|_F \\ &= \min_{\mathcal{J}(M) \in \mathcal{J}} \left(\|C_0 M - B_0\|_F + \rho^2 \|e^T M - f^T\|_F \right), \end{aligned} \tag{3.23}$$

with rectangular $C, B \in \mathbb{R}^{m \times n}$ and sparse matrices $C_0, B_0 \in \mathbb{R}^{n \times n}$. This generalized form is not restricted to special probing subspaces as it allows any choice of e and f. The resulting preconditioner M satisfies both $C_0 M \approx B_0$ and $e^T M \approx f^T$. We refer to the first n rows of (3.23), i.e., $C_0 M - B_0$, as *full approximation part* and to the additional rows as *probing part*. The weight $\rho \geq 0$ enables us to control how much emphasis is put on the *probing constraints*. The matrices $e, f \in \mathbb{R}^{n \times l}$, $l := m - n$, represent the l-dimensional subspace on which the preconditioner should be optimal. Note that we use lowercase letters for these special matrices to emphasize the relationship to classical probing vectors. Moreover, in many cases e and f represent only a 1D subspace, i.e., a vector. Choosing $\rho = 0$, $C_0 = A$, and $B_0 = I$ in (3.23) leads to the standard SPAI formulation (3.1).

The static and dynamic computation of M—denoted as MSAI and MSPAI, respectively—can be performed analogously to SAI and SPAI as presented in Section 3.1.1 and Section 3.1.3. We solve the occurring LS problems

$$\min_{\mathcal{J}(M) \in \mathcal{J}} \|CM - B\|_F^2 = \sum_{k=1}^{n} \min_{\mathcal{J}(M_k) \in \mathcal{J}_k} \|CM_k - B_k\|_2^2$$

independently of each other. Corresponding to (3.5), the initial sparse pattern $\mathcal{J}$ induces reduced LS problems

$$\min_{\mathcal{J}(\hat{M}_k)=\mathcal{J}_k} \|\hat{C}_k\hat{M}_k - \hat{B}_k\|_2, \qquad k = 1,\ldots,n,$$

which can be solved via QR factorization $\hat{C}_k = \hat{Q}\hat{R}$. We obtain an MSAI solution via

$$\hat{d} := \hat{Q}^T\hat{B}_k \qquad \text{and therewith} \qquad \hat{M}_k := \hat{R}^{-1}\hat{d}. \tag{3.24}$$

Based on an initial approximation, it is possible to update $\mathcal{J}$ in order to improve the quality of M. For each $j \in \tilde{\mathcal{J}}$ the new residuals can be obtained by solving the univariate minimization problem

$$\min_{m_{jk}} \|C(M_k + m_{jk}e_j) - B_k\|_2 = \min_{m_{jk}} \|r_k + m_{jk}Ce_j\|_2,$$

similarly to (3.14) and (3.15). Hence, MSPAI inherits the main advantages of SPAI: the computation remains inherently parallel and a profitable sparsity pattern is captured automatically by applying pattern updates. See also Figure 2.6. The field of applications using MSPAI is versatile: we can improve preconditioners resulting from ILU, IC, FSAI, FSPAI, or AINV (see [22] for an overview) by adding probing information. We also overcome the main drawbacks of MILU and classical probing such as the restriction to certain vectors like $(1,1,\ldots,1)^T$ as probing subspace and the rather difficult scalable implementation on parallel computing environments. Note that when $e = f$, the MSPAI approach (3.23) can be considered as

$$\min_{\mathcal{J}(M)\in\mathcal{J}} \|W(C_0M - B_0)\|_F \qquad \text{with} \qquad W := \begin{pmatrix} I \\ \rho e^T \end{pmatrix},$$

at which the norm emphasizes the special subspace by using the weight matrix W. Likewise, this can be seen as a regularization technique for a general probing method without restriction to the choice of the subspace in e. For large values of ρ much emphasis is put on the probing subspace while for small values the full approximation part is taken more into account. Note the relationship between ρ and the regularization parameter α inside the Tikhonov-Phillips regularization (2.32).

The generalized Frobenius norm minimization (3.23) allows to construct various preconditioners satisfying different probing conditions.

- **Sparse approximate inverse probing:** By adding probing constraints to SPAI, we can compute an unfactorized approximation of A^{-1}. Using $C_0 = A$, $B_0 = I$, and $f = e$, we obtain

$$\min_{\mathcal{J}(M)\in\mathcal{J}} \left\| \begin{pmatrix} A \\ \rho e^T A \end{pmatrix} M - \begin{pmatrix} I \\ \rho e^T \end{pmatrix} \right\|_F^2 = \min_{\mathcal{J}(M)\in\mathcal{J}} \|W(AM - I)\|_F^2, \tag{3.25}$$

  where M satisfies the conditions $M \approx A^{-1}$ and $e^TAM \approx e^T$. For a given sparse factorized approximation of $A^{-1} \approx UL$ or $UAL \approx I$, resulting from, e.g., FSPAI (see Section 3.2.3) or AINV, we can improve these preconditioners by keeping one of the factors fixed while recomputing the other one. We substitute $B_0 = I$, $C_0 = UA$, and $f = e$ within (3.23) and solve for $\tilde{L}$ restricting its pattern to lower triangular form

$$\min_{\mathcal{J}(\tilde{L})} \left\| \begin{pmatrix} UA \\ \rho e^T UA \end{pmatrix} \tilde{L} - \begin{pmatrix} I \\ \rho e^T \end{pmatrix} \right\|_F^2 = \min_{\mathcal{J}(\tilde{L})} \|W(UA\tilde{L} - I)\|_F^2. \tag{3.26}$$

The improved factor $\tilde{L}$ of L will satisfy both $UA\tilde{L} \approx I$ and $e^T UA\tilde{L} \approx e^T$. We obtain the improved upper triangular factor $\tilde{U}$ of U by solving the transposed ansatz $\min_{\mathcal{J}(\tilde{U})} \|W(\tilde{L}^T A^T \tilde{U}^T - I)\|_F^2$ limiting $\mathcal{J}(\tilde{U})$ to upper triangular form.

- **Explicit probing:** Setting $C_0 = I$ and $B_0 = A$, we are able to compute explicit sparse approximations to A via

$$\min_{\mathcal{J}(M)\in\mathcal{J}} \left\| \begin{pmatrix} I \\ \rho e^T \end{pmatrix} M - \begin{pmatrix} A \\ \rho e^T A \end{pmatrix} \right\|_F^2 = \min_{\mathcal{J}(M)\in\mathcal{J}} \|W(M-A)\|_F^2 . \tag{3.27}$$

In this case the derived approximation on a (nearly) dense A can have a considerably fewer nnz than A, but choosing $f^T = e^T A$, the preconditioner M will have a similar action on e^T as A. Hence, M will satisfy the conditions $M \approx A$ and $e^T M \approx e^T A$. Again, we can improve preconditioners which are computed in factorized form, e.g., an ILU factorization $A \approx LU$. Setting $C_0 = L$, $B_0 = A$, and $f^T = e^T A$, we observe

$$\min_{\mathcal{J}(\tilde{U})} \left\| \begin{pmatrix} L \\ \rho e^T L \end{pmatrix} \tilde{U} - \begin{pmatrix} A \\ \rho e^T A \end{pmatrix} \right\|_F^2 = \min_{\mathcal{J}(\tilde{U})} \left\|W(L\tilde{U} - A)\right\|_F^2 , \tag{3.28}$$

where $L\tilde{U} \approx A$ and $e^T L\tilde{U} \approx A$ are satisfied. Similarly to sparse approximate inverse probing, we can apply the same method on the transposed problem $\min_{\mathcal{J}(\tilde{L})} \|W(\tilde{U}^T \tilde{L}^T - A^T)\|_F^2$ in order to obtain an improved factor $\tilde{L}$.

- **Schur complement probing:** The classical probing approach was imposed for preconditioning of Schur complements [34] which typically arise in Stokes problems or by domain decomposition of PDEs, e.g., describing some pressure or flow behavior. Using the MSPAI approach, it becomes possible to compute a sparse approximation of the Schur complement $S_D = D - CA^{-1}B$ of the matrix

$$G = \begin{pmatrix} A & B \\ C & D \end{pmatrix} \quad \text{with its inverse} \quad G^{-1} = \begin{pmatrix} S_A^{-1} & -A^{-1}BS_D^{-1} \\ -D^{-1}CS_A^{-1} & S_D^{-1} \end{pmatrix}.$$

Observing that the lower right block of G^{-1} is the inverse of the Schur complement S_D, we can modify the general approach to compute an approximation $M_D \approx S_D^{-1}$ by solving

$$\min_{\mathcal{J}\binom{M_B}{M_D}\in\mathcal{J}} \left\| \begin{pmatrix} A & B \\ C & D \\ 0 & \rho e^T S_D \end{pmatrix} \begin{pmatrix} M_B \\ M_D \end{pmatrix} - \begin{pmatrix} 0 \\ I \\ \rho e^T \end{pmatrix} \right\|_F^2 .$$

In case that $e^T S_D$ is known exactly, the preconditioner M_D will also satisfy the condition $e^T S_D M_D \approx e^T$.

See [85, 99] for more details on the various probing variants and numerical experiments which demonstrate MSPAI's effectiveness in preconditioning of various PDE matrices and Schur complements.

Remark 3.3 *In many applications the linear system A is not available explicitly but only as a sparse approximation $\tilde{A}$, e.g., in domain decomposition methods. However, we assume to be able to compute the probing part exactly. For (3.25) and (3.28) we receive the slightly modified LS problems*

$$\min_{\mathcal{J}(M)\in\mathcal{J}} \left\| \begin{pmatrix} \tilde{A} \\ \rho e^T A \end{pmatrix} M - \begin{pmatrix} I \\ \rho e^T \end{pmatrix} \right\|_F^2 \qquad \textit{and} \qquad \min_{\mathcal{J}(\tilde{U})} \left\| \begin{pmatrix} L \\ \rho e^T L \end{pmatrix} \tilde{U} - \begin{pmatrix} \tilde{A} \\ \rho e^T A \end{pmatrix} \right\|_F^2,$$

respectively. For the former we obtain a sparse approximate inverse M for $\tilde{A} \approx A$ whereas for the latter we are able to improve a known decomposition. In both cases the preconditioner will act as A on the given probing subspace. The same holds for (3.26) and (3.27).

Mask Probing

It is also possible to include individual probing conditions for each column M_k. As a new approach we use *probing masks* defined by sparse row vectors $S_r \in \mathbb{R}^n$, $r = 1,\ldots,ln$, $l \geq 0$. The l masks which correspond to M_k contain the same pattern as M_k, i.e.,

$$\mathcal{J}(S_r) = \mathcal{J}(S_{r+n}) = \ldots = \mathcal{J}(S_{r+ln}) = \mathcal{J}(M_k).$$

We solve

$$\min_{\mathcal{J}(M)\in\mathcal{J}} \left(\|CM - B\|_F^2 + \rho^2 \|SM - T\|_F^2 \right)$$

with

$$S := (S_1, S_2, \ldots, S_n, S_{n+1}, \ldots, S_{2n}, S_{2n+1}, \ldots, S_{ln})^T \in \mathbb{R}^{(ln)\times n}$$

and

$$T := (T_1, T_2, \ldots, T_l)^T \in \mathbb{R}^{(ln)\times n}, \qquad \text{where} \qquad T_i := \operatorname{diag}([f_i]_1, [f_i]_2, \ldots, [f_i]_n) \in \mathbb{R}^{n\times n}$$

for $i = 1,\ldots,l$. Hence, the l masks for each column of M are stored in a sparse rectangular matrix S while the right-hand sides are stacked diagonal matrices in T. In the following we assume $l = 1$, i.e., $T = \operatorname{diag}(f_1,\ldots,f_n) \in \mathbb{R}^{n\times n}$ and $S \in \mathbb{R}^{n\times n}$ contains one mask for each column of M. The individual probing conditions result from $\operatorname{diag}(SM) \approx f \in \mathbb{R}^n$. Note that for $l = 0$ we obtain the standard MSPAI formulation (3.23).

Compared to MSPAI using *global probing* vectors e, f, this *local probing* approach gives considerably more freedom to choose probing conditions individually for each column of the preconditioner. Taking the sparsity into account, we add the condition

$$\min_{\mathcal{J}(\hat{M}_k)=\mathcal{J}_k} \left| \hat{S}_k \hat{M}_k - f_k \right|$$

to the Frobenius norm minimization of the k^{th} column. Corresponding to $\hat{M}_k$, $\hat{S}_k$ denotes the reduced form of S_k. Consider, for instance, a tridiagonal pattern in M, i.e., $\hat{M}_k = (m_{k-1,k}, m_{kk}, m_{k+1,k})^T$. Using the local probing mask $\hat{S}_k = (1,0,-1)$, it is possible to enforce a quasi symmetry in the column vector M_k such that $m_{k-1,k} \approx m_{k+1,k}$. In order to obtain a column sum of zero, we can also use the mask $\hat{S}_k = (-1,1,-1)$.

Definition 3.2 (Isotropic mask) *We call a mask $\hat{S}_k$ to be isotropic if for all columns $k \in \{1,\ldots,n\}$ the minimization $\min_{\mathcal{J}(\hat{M}_k)=\mathcal{J}_k} \left| \hat{S}_k \hat{M}_k - f_k \right|$ can be described by one probing vector $e \in \mathbb{R}^n$, with $e(\mathcal{J}_k) = \beta \hat{S}_k^T$, $\beta \in \mathbb{R}$, in the form $\min_{\mathcal{J}(M)\in\mathcal{J}} \left\| e^T M - f^T \right\|_2$.*

For example, the masks $\hat{S}_k = (-1,1,-1)$ and $\hat{S}_k = (1,1,1)$ are isotropic as they are related to $e_N = (1,-1,1,-1,\ldots)^T$ and $e_S = (1,1,\ldots,1)^T$, respectively.

Choosing the Probing Subspace

The choice of the probing vectors and probing masks is crucial for the quality of the resulting preconditioner and depends on the given model problem. We summarize various different heuristics suggested in [34, 85, 99, 130]:

- Motivated by MILU or the classical probing approach, the probing subspace e can be defined as a normalized set of l vectors

$$e := \sqrt{\frac{l}{n}}(e_1, \ldots, e_l) \qquad \text{where} \qquad (e_i)_j = 1 \quad \text{for} \quad j = i, l+i, 2l+i, \ldots \ .$$

- A set of vectors which form an orthogonal basis, e.g., e_i according to (3.29).
- l eigenvector (singular vector) approximations relative to the extremal eigenvalues (singular values) if no a priori knowledge is available. Although rough approximations are sufficient, additional costs must be made for this choice.
- Graph coloring approaches to obtain *structured probing* subspaces are an enhancement of the classical probing approach. These are especially efficient for preconditioning saddle-point problems if an a priori estimate of the probing pattern is at hand.

In Chapter 5 and 6 we introduce two applications which require preconditioners having a different action on the underlying domain's low and high frequency subspace, respectively. This necessity is appropriate for the probing approach in connection with MS(P)AI. We can model these characteristic subspaces in the following way:

- **Low frequency probing subspaces** can be modelled by the smooth vector $e_S := (1,1,\ldots,1)^T$. In order to allow larger subspaces, we can also add numerical eigenvector estimates to l large eigenvalues. Or, motivated by the close relationship between the sine transform and many PDE or image reconstruction problems, we can use, e.g., (Kronecker products of) l 1D vectors to form $e := (e_1, \ldots, e_l)$, where

$$e_i := \sqrt{\frac{2}{n+1}} \sin\left(\frac{\pi j i}{n+1}\right)_{j=1,\ldots,n} \qquad \text{for} \qquad i = 1, \ldots, l. \tag{3.29}$$

  These also represent smooth components and therefore the important part of the signal subspace.

- **High frequency probing subspaces** corresponding to the noise subspace can be modelled, e.g., via $e_N := (1,-1,1,-1,\ldots)^T =: e_{N1}$ or $e_{N2} := (1,0,-1,0,1,\ldots)^T$ and $e_{N3} := (0,1,0,-1,0,1,\ldots)^T$, representing typical vectors related to fast oscillations. To allow larger subspaces, we can also consider eigenvector estimates to near singular eigenvalues, or a subspace $e := (e_1, \ldots, e_l)$ containing oscillatory modes

$$e_i := \sqrt{\frac{2}{n+1}} \sin\left(\frac{\pi(2j+1)(n+i)}{2(n+1)}\right)_{j=1,\ldots,n} \qquad \text{for} \qquad i = 1, \ldots, l \ .$$

  In case of using probing masks, $\hat{S}_k = (-1,1,-1)$ and $\hat{S}_k = (1,0,-1)$ are related to e_{N1} and e_{N2}, e_{N3}, respectively.

Usually, the weight is chosen to be $\rho \in]0{,}10^2]$ depending on how large the action of M should be on the subspace. In many cases a saturated action is achieved already after $\rho \gtrsim 5$.

3.1.5. Multistep Successive Preconditioner: MSP

Following Section 3.1.2, it is impossible to predict a sparsity pattern for M which will lead to a preconditioner of good quality, in general. Using pattern updates likely reveals a sparse inverse of high quality. However, this iterative computation may be too expensive in a sequential environment compared to other preconditioning techniques like ILU or AINV. Motivated by Chow's approach using powers of sparsified A, Wang and Zhang [141] introduced a multistep strategy in which a SAI preconditioner $M^{(i+1)}$ is computed for the successive system $A^{(i+1)} = A^{(i)}M^{(i)}$ by using $M^{(i)}$ computed from a previous step. The motivation comes from factorized sparse approximate inverses: by constructing an approximate inverse from several matrices it is likely that this series is capable to hold more information than a single matrix itself. Furthermore, computing a few sparse matrices should be faster than computing a single matrix of comparable density. As SAI pattern Wang and Zhang propose to use $\mathcal{J}(M^{(i)}) = \mathcal{J}(A^{(i-1)})$ which may lead very fast to dense preconditioners and expensive large LS problems. In order to keep the computation efficient, they suggest to use a low number of steps, e.g., two ($\alpha = 1$) or three ($\alpha = 2$). Moreover, sparsification becomes essential. The authors propose to use a two-level dropping strategy at each step. While dropping of small components in $M^{(i)}$ after each computation is often used for sparse approximate inverse preconditioners, the additional approach of dropping components in $A^{(i)}$ at the beginning of each step must be treated with caution: there is no evidence which states that small entries in A will not produce important (large) entries in A^{-1}.

Algorithm 6 represents a slightly modified version of the *multistep successive preconditioner* (MSP) [141]. We omit the sparsification to create a reasonable basis for comparison with our following new approach, which is subject to a similar idea of improving current approximations via successive iterations. In both algorithms a common postdropping can be easily included. In its original version, the iteration of MSP is only stopped after performing a fixed number of heuristically chosen steps α. For some problems it might be hard to predict

Algorithm 6: Multistep successive preconditioner (MSP) [141] without sparsification.

Input : $A \in \mathbb{R}^{n\times n}$, $\alpha \geq 0$, $\varepsilon_{\text{MSP}} \geq 0$
Output: Preconditioner $M \approx A^{-1}$, $M \in \mathbb{R}^{n\times n}$

$A^{(0)} \leftarrow A$
$M \leftarrow I_n$
for $i \leftarrow 0$ **to** α **do**
  $M^{(i)} \leftarrow \underset{\mathcal{J}(M^{(i)})=\mathcal{J}(A^{(i)})}{\arg\min} \left\|A^{(i)}M^{(i)} - I_n\right\|_F$
  $M \leftarrow M^{(i)}M$
  if $\forall k \in \{1,\ldots,n\} : \|AM_k - e_k\|_2 \leq \varepsilon_{\texttt{MSP}}$ **then**
    break
  end
  $A^{(i+1)} \leftarrow A^{(i)}M^{(i)}$
end

α such that M will have a sufficiently good quality. To provide insight into the quality of the resulting preconditioner, we additionally impose a stopping criterion based on a tolerance metric ε_{MSP} motivated by the common usage in SPAI-like algorithms. Note that in [142] a *multilevel* MSP (MMSP) strategy is proposed which employs MSP on local submatrices. For the tested examples, MMSP outperforms the stand-alone MSP approach in terms of setup time. However, its convergence depends on the number of processors used due to local matrix reorderings.

3.1.6. Multistep MSPAI: MMSPAI

A drawback of the approach implemented in Algorithm 6 is that as long as there is at least one column violating the stopping criterion, a complete new successive step must be invoked (assuming α is not reached yet). This can be avoided by the following new approach. For the sake of clarity we use ε instead of $\varepsilon_{\text{SPAI}}$ as the SPAI-like tolerance metric. Let us first consider the standard SAI case starting with an available SAI preconditioner $M^{(0)}$ for $A^{(0)} = A$ computed from

$$\min_{\mathcal{J}(M^{(0)})\in\mathcal{J}} \|A^{(0)}M^{(0)} - I\|_F.$$

We apply a permutation P such that those γ_1 columns which already satisfy

$$\|A^{(0)}M_k^{(0)} - e_k\|_2 = \|r_k^{(0)}\|_2 \le \varepsilon, \tag{3.30}$$

are permuted to the front. We denote this submatrix as $M_{\le\varepsilon}^{(0)} \in \mathbb{R}^{n\times\gamma_1}$ while those columns which violate (3.30) are collected in $M_{>\varepsilon}^{(0)} \in \mathbb{R}^{n\times\delta_1}$. We obtain the system

$$P^T(A^{(0)}M^{(0)} - I)P \;=\; P^TA^{(0)}\left(M_{\le\varepsilon}^{(0)}\middle|M_{>\varepsilon}^{(0)}\right) - I. \tag{3.31}$$

Using $\tilde{A}^{(0)} := P^TA^{(0)}$, we can compute a SAI $M^{(1)} \in \mathbb{R}^{n\times\delta_1}$ for the remaining $\delta_1 = n - \gamma_1$ columns $M_{>\varepsilon}^{(0)}$ in (3.31) via

$$\begin{aligned}
&\min_{\mathcal{J}(M^{(1)})\in\mathcal{J}} \left\| \tilde{A}^{(0)}\left(M_{\le\varepsilon}^{(0)}\middle|M_{>\varepsilon}^{(0)}\right)\left(\begin{array}{c|c} I_{\gamma_1} & \\ 0 & M^{(1)} \end{array}\right) - I_n \right\|_F \\
= &\min_{\mathcal{J}(M^{(1)})\in\mathcal{J}} \left\| \left(\tilde{A}^{(0)}M_{\le\varepsilon}^{(0)}\middle|\underbrace{\tilde{A}^{(0)}\left(M_{\le\varepsilon}^{(0)}\middle|M_{>\varepsilon}^{(0)}\right)}_{=:A^{(1)}}M^{(1)}\right) - \left(\begin{array}{c|c} I_{\gamma_1} & 0 \\ \hline 0 & I_{\delta_1} \end{array}\right) \right\|_F \\
= &\min_{\mathcal{J}(M^{(1)})\in\mathcal{J}} \left\| A^{(1)}M^{(1)} - \begin{pmatrix} 0 \\ I_{\delta_1} \end{pmatrix} \right\|_F .
\end{aligned} \tag{3.32}$$

If the approximation is accurate enough, the resulting preconditioner arises from back permutation

$$M = \left[\left(M_{\le\varepsilon}^{(0)}\middle|M_{>\varepsilon}^{(0)}\right)\left(\begin{array}{c|c} I_{\gamma_1} & \\ 0 & M^{(1)} \end{array}\right)\right]P^T.$$

Otherwise, we can recursively apply this approach on (3.32) to obtain $M^{(2)}$ and so on. Thus we achieve refinement only where it is explicitly requested. With respect to the quality metric ε we obtain a preconditioner of higher quality and the setup costs are smaller than in Algorithm 6 due to less LS problems. Moreover, the setup remains inherently parallel: assuming a blockwise scattering of A's columns, each processor can locally apply the per-

mutations on its submatrix to compute a sequence of the processor-specific preconditioner part. Note that, similar to MSP, we need to prescribe a sparsity pattern for the various $M^{(i)}$; see Numerical Experiment 3.3.

This multistep approach can be transferred to the generalized form of Frobenius norm minimization (3.23). However, we must incorporate a slight modification in order to guarantee a correct probing subspace approximation. Based on an available MSAI $M^{(i)} \in \mathbb{R}^{n\times\delta_i}$, we illustrate a successive step of the *multistep modified SPAI* (MMSPAI) approach to compute $M^{(i+1)} \in \mathbb{R}^{n\times\delta_{i+1}}$. In the following we use the notations

$$F := \rho e^T \in \mathbb{R}^{l\times n}, \qquad G := \rho f^T \in \mathbb{R}^{l\times n}, \qquad \text{and} \qquad \tilde{M}^{(i)} := \left(M^{(i)}_{\leq\varepsilon} | M^{(i)}_{>\varepsilon}\right) \in \mathbb{R}^{n\times\delta_i}.$$

The scalar γ_{i+1} gives the number of columns of $\tilde{M}^{(i)}$ which satisfy $\|r^{(i)}_k\|_2 \leq \varepsilon$, i.e., $M^{(i)}_{\leq\varepsilon} \in \mathbb{R}^{n\times\gamma_{i+1}}$ while $M^{(i)}_{>\varepsilon} \in \mathbb{R}^{n\times\delta_{i+1}}$ denotes the number of columns which require improvement. The overall number of processed columns of M satisfying ε is defined as

$$\gamma := \sum_{s=0}^{i} \gamma_s, \qquad \text{where} \qquad \gamma_0 := 0 \qquad \text{and therefore} \qquad \delta_i = n - \gamma.$$

Corresponding to (3.31), we apply a permutation $P^{(i)} \in \mathbb{R}^{\delta_i\times\delta_i}$ that constructs $\tilde{M}^{(i)}$ via

$$\left(\begin{array}{cc|c} I_\gamma & 0 & 0 \\ 0 & P^{(i)T} & 0 \\ \hline 0 & 0 & I_l \end{array}\right) \left[\begin{pmatrix} C^{(i)} \\ \hline F^{(i)} \end{pmatrix} M^{(i)} - \begin{pmatrix} B^{(i)} \\ \hline G^{(i)} \end{pmatrix}\right] P^{(i)} = \begin{pmatrix} \tilde{C}^{(i)} \\ \hline F^{(i)} \end{pmatrix} \tilde{M}^{(i)} - \begin{pmatrix} B^{(i)} \\ \hline \tilde{G}^{(i)} \end{pmatrix},$$

in which $C^{(i)} \in \mathbb{R}^{n\times n}$, $F^{(i)} \in \mathbb{R}^{l\times n}$, $B^{(i)} \in \mathbb{R}^{n\times\delta_i}$, and $G^{(i)} \in \mathbb{R}^{l\times\delta_i}$. Note that in the first step $I_\gamma \in \mathbb{R}^{0\times 0}$ because $\gamma = 0$ and that $P^{(i)}$ has the same dimensions as $C^{(i)}$. The successive preconditioner $M^{(i+1)}$ can be computed via

$$\begin{aligned} &\min_{\mathcal{J}(M^{(i+1)})\in\mathcal{J}} \Bigg\| \underbrace{\begin{pmatrix} \tilde{C}^{(i)} \\ \hline F^{(i)} \end{pmatrix} \left(\begin{array}{c|c} I_\gamma & \\ 0 & \tilde{M}^{(i)} \end{array}\right)}_{=:\ (C^{(i+1)^T}, F^{(i+1)^T})^T} \left(\begin{array}{c|c} I_{\gamma+\gamma_{i+1}} & \\ 0 & M^{(i+1)} \end{array}\right) - \left(\begin{array}{c|c} B^{(i)}_1 & B^{(i)}_2 \\ \tilde{G}^{(i)}_1 & \tilde{G}^{(i)}_2 \end{array}\right) \Bigg\|_F = \\ &\min_{\mathcal{J}(M^{(i+1)})\in\mathcal{J}} \left\| \begin{pmatrix} C^{(i+1)} \\ \hline F^{(i+1)} \end{pmatrix} M^{(i+1)} - \begin{pmatrix} B^{(i+1)} \\ \hline G^{(i+1)} \end{pmatrix} \right\|_F, \end{aligned}$$

where

$$\begin{aligned} C^{(i+1)}, F^{(i+1)} &\in \mathbb{R}^{n\times n}, & M^{(i+1)} &\in \mathbb{R}^{n\times\delta_{i+1}}, \\ B^{(i)}_1 = B^{(i)}_{:,1:\delta_i-\delta_{i+1}} &\in \mathbb{R}^{n\times(\delta_i-\delta_{i+1})}, & \tilde{G}^{(i)}_1 &\in \mathbb{R}^{l\times(\delta_i-\delta_{i+1})}, \\ B^{(i+1)} := B^{(i)}_2 = B^{(i)}_{:,\delta_i-\delta_{i+1}+1:\delta_i} &\in \mathbb{R}^{n\times\delta_{i+1}}, \text{ and} & G^{(i+1)} := \tilde{G}^{(i)}_2 &\in \mathbb{R}^{l\times\delta_{i+1}}. \end{aligned}$$

We illustrate an iterative version of the MMSPAI approach in Algorithm 7. It is possible to omit the back permutation of M in line 23 and compute a preconditioner for the permuted system. For this purpose both the right-hand side b and the matrix A of the system to be solved must be permuted accordingly to the various $M^{(i)}$.

Remark 3.4 *Instead of using α successive steps, it is also possible to apply a small number of pattern updates, i.e., SPAI on the rear part $M^{(i)}_{\geq\varepsilon}$ after one (or a few) permutations.*

Algorithm 7: Multistep modified sparse approximate inverse (MMSPAI) preconditioner.

Input : $C \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times n}$, $e \in \mathbb{R}^{n\times l}$, $f \in \mathbb{R}^{n\times l}$, $\mathcal{J} \in \mathbb{B}^{n\times n}$, $\alpha \geq 0$, $\varepsilon_{\text{MMSPAI}} \geq 0$, $\rho \geq 0$

Output: Preconditioner $M \approx A^{-1}$, $M \in \mathbb{R}^{n\times n}$

$C^{(0)} \leftarrow C$

$F^{(0)} \leftarrow \rho e^T C^{(0)}$

$B^{(0)} \leftarrow B$

$G^{(0)} \leftarrow \rho f^T B^{(0)}$

$M \leftarrow I_n$

$\gamma \leftarrow \gamma_0 \leftarrow 0$

$\delta_0 \leftarrow n$

for $i \leftarrow 0$ **to** α **do**

$$M^{(i)} \leftarrow \underset{\mathcal{J}(M^{(i)})\in\mathcal{J}}{\arg\min} \left\| \begin{pmatrix} C^{(i)} \\ F^{(i)} \end{pmatrix} M^{(i)} - \begin{pmatrix} B^{(i)} \\ G^{(i)} \end{pmatrix} \right\|_F$$

$$\gamma_{i+1} \leftarrow \left| \left\{ k \in \{1,\ldots,\delta_i\} : \left\| \begin{pmatrix} C^{(i)} \\ F^{(i)} \end{pmatrix} M_k^{(i)} - \begin{pmatrix} B_k^{(i)} \\ G_k^{(i)} \end{pmatrix} \right\|_2 \leq \varepsilon_{\texttt{MMSPAI}} \right\} \right|$$

$\delta_{i+1} \leftarrow \delta_i - \gamma_{i+1}$

if $\delta_{i+1} = 0$ **then**

break

end

Construct $P^{(i)}$

$\tilde{M} \leftarrow M^{(i)} P^{(i)}$

$$M \leftarrow M \left(\begin{array}{c|c} I_\gamma & \\ 0 & \tilde{M} \end{array} \right)$$

$\tilde{G} \leftarrow G^{(i)} P^{(i)}$

$B^{(i+1)} \leftarrow B_{:,(\delta_i-\delta_{i+1})+1:\delta_i}$

$G^{(i+1)} \leftarrow \tilde{G}_{:,(\delta_i-\delta_{i+1})+1:\delta_i}$

$$\begin{pmatrix} C^{(i+1)} \\ F^{(i+1)} \end{pmatrix} \leftarrow \left(\begin{array}{cc|c} I_\gamma & 0 & 0 \\ 0 & P^{(i)T} & 0 \\ \hline 0 & 0 & I_l \end{array} \right) \begin{pmatrix} C^{(i)} \\ F^{(i)} \end{pmatrix} \left(\begin{array}{c|c} I_\gamma & \\ 0 & \tilde{M} \end{array} \right)$$

$\gamma \leftarrow \gamma + \gamma_{i+1}$

end

$$M \leftarrow M \left(\begin{array}{c|c} I_{\sum_{s=0}^{i}\gamma_s} & 0 \\ \hline 0 & P^{(i)T} \end{array} \right) \left(\begin{array}{c|c} I_{\sum_{s=0}^{i-1}\gamma_s} & 0 \\ \hline 0 & P^{(i-1)T} \end{array} \right) \cdots \left(\begin{array}{c|c} I_{\gamma_1} & 0 \\ \hline 0 & P^{(1)T} \end{array} \right) P^{(0)T}$$

Numerical Experiments

In order to obtain insight into the effect using an MMSPAI and an MSP preconditioner, we focus on three numerical experiments. Both Algorithms 6 and 7 are implemented in MATLAB [108] in a version which provides information about the spectrum and norm properties of the resulting preconditioned system. Furthermore, it detects the number of flops which are brought up during the solution of the LS problems. For this purpose, we use the LIGHTSPEED MATLAB TOOLBOX [110]. An additional second version reflects a code which is optimized on the basis of the MATLAB profiler and, in general, spends around 95% of its time for the

solution of the LS problems. It is geared to measure the setup time for the $M^{(i)}$ which is implemented via the difference of time stamps (date vectors). In order to reduce falsification, we minimize the number of running processes on the operating system and perform multiple runs of the same scenario. We consider a multistep SPAI variant of MMSPAI as we do not take any probing and target information into account, i.e., we use the identifiers A instead of C and I instead of B in MMSPAI. We setup preconditioners for matrices from MATRIX MARKET [116] and from the UF SPARSE MATRIX COLLECTION [44]. All computations are performed on the test environment specified in Appendix D.4.

Numerical Experiment 3.1 We illustrate the effect of each successive step in both approaches on the matrix FPGA_DCOP_03 in Table 3.1 and BCSSTK12 in Table 3.2, without using dropping in the preconditioner. We fix the number of successive steps to $\alpha = 2$. BiCGSTAB does not converge in 5000 iterations for the chosen right-hand side $b = (1,1,\ldots,1)^T$ and tolerance 1E−09 and 1E−08, respectively. To illustrate the effect of permutations, we give the resulting sparsity structure of M (not $M^{(i)}$) after each step. Note that the density of the various $M^{(i)}$ is much smaller than presented in the tables for M via $\text{nnz}(M)$. Although BCSSTK12 is an SPD matrix, we apply SAI and BiCGSTAB for more theoretical evidence.

When $M^{(i)}$ is not sparsified after each step, the MSP results in dense preconditioners while the MMSPAI produces only fill-in on those columns which do not satisfy the $\varepsilon_{\text{MMSPAI}}$ criterion. This affects the number of flops required to solve the LS problems and the time to setup the various $M^{(i)}$. We obtain the typical overshooting behavior of MSP: while many columns of $M^{(i)}$ already satisfy ε_{MSP}, an additional expensive step is performed. On the one hand, the larger number of DsOF is the reason for MSP's better spectral properties compared to MMSPAI. On the other hand, the higher density of M induces higher solver costs due to more costly matrix-vector multiplications. ●

Numerical Experiment 3.2 As the MSP may get dense very fast without using any dropping and thus become unhandy, we provide results for different matrices when applying a post-dropping on $M^{(i)}$ within each iteration in MMSPAI and MSP. We fix α to the minimum number of successive steps in MMSPAI to obtain convergence in BiCGSTAB for $b = (1,1,\ldots,1)^T$ and $\varepsilon_{\text{BiCGSTAB}} = 1\text{E}{-}09$. For each problem we compute two MSP preconditioners: one which has similar norm $\|AM - I\|_F$ and one which has similar density $\text{nnz}(M)$, compared to MMSPAI.

Following Table 3.3, despite using dropping in $M^{(i)}$, MMSPAI outperforms MSP in terms of setup and solver time for the given matrices. This is because MSP produces fill-in also on columns which yield only weak improvement in the next step. Consequently, MSP preconditioners which have similar density, compared to MMSPAI, are of poor quality yielding no convergence when used in BiCGSTAB. ●

Numerical Experiment 3.3 We are interested in heuristic patterns for the $M^{(i)}$ in Algorithm 7 which yield efficient preconditioners in general. Not taking sparsification into account, we use the following pattern strategies and preconditioning techniques, respectively:

1. MMSPAI$\big(A^{(i)}\big)$: approach according to Algorithm 7 using the pattern $\mathcal{J}(M^{(i)}) = \mathcal{J}(A^{(i)})$.
2. MMSPAI$\big((A^{(i)})_{\text{0-block}}\big)$: imposing a right upper 0-block for the pattern $\mathcal{J}(M^{(i)}) = \mathcal{J}(A^{(i)})$ which leads to a slightly modified minimization in line 9 of Algorithm 7:

$$\min_{\mathcal{J}(M^{(i)})\in\mathcal{J}} \left\| A^{(i)} \left(\begin{array}{c|c} I_\gamma & 0 \\ \hline 0 & M^{(i)} \end{array}\right) - \left(\begin{array}{c|c} I_\gamma & 0 \\ \hline 0 & I_{\delta_i} \end{array}\right) \right\|_F = \min_{\mathcal{J}(M^{(i)})\in\mathcal{J}} \left\| A^{(i)} \begin{pmatrix} 0 \\ M^{(i)} \end{pmatrix} - \begin{pmatrix} 0 \\ I_{\delta_i} \end{pmatrix} \right\|_F. \tag{3.33}$$

3. MMSPAI$\left(A^{i+1}\right)$: using powers of (unsparsified) A, motivated by the approach of Chow described in Section 3.1.2, i.e., $\mathcal{J}(M^{(i)}) = \mathcal{J}(A^{i+1})$.
4. MMSPAI$\left(A^{(i)T}A^{(i)}\right)$: using the more dense pattern $\mathcal{J}(M^{(i)}) = \mathcal{J}(A^{(i)T}A^{(i)})$.
5. MMSPAI$\left((A^{(i)T}A^{(i)})_{\text{0-block}}\right)$: using a right upper 0-block for the pattern $\mathcal{J}(M^{(i)}) = \mathcal{J}(A^{(i)T}A^{(i)})$ with modified minimization according to (3.33).
6. MMSPAI with SPAI$_{\text{rear part}}$: a hybrid approach in which pattern updates are performed on the rear part $M^{(i)}_{>\varepsilon}$ within each step. Hence, instead of using a priori patterns, profitable indices are captured automatically by SPAI according to Algorithm 4. Start pattern of SPAI is $\mathcal{J}\left(A^{(i)}\right)$.
7. SAI$\left(A^{i+1}\right)$: A multistep SAI based on a fixed number of steps α. It uses successively increasing powers of (unsparsified) patterns of A, i.e., $\mathcal{J}(M^{(i)}) = \mathcal{J}(A^{i+1})$.

The chosen setting for α and $\varepsilon_{\text{MMSPAI}}$ roughly reflects the minimum choice to obtain convergence in BiCGSTAB, corresponding to the pattern $\mathcal{J}(M^{(i)}) = \mathcal{J}(A^{(i)})$ in Algorithm 7. Note that we use $\mathcal{J}(M^{(0)}) = \mathcal{J}(A)$ as initial sparsity pattern for all strategies.

The numerical results in Table 3.4 demonstrate that MMSPAI is more efficient when a dense pattern (e.g., $A^{(i)T}A^{(i)}$) is invoked on those columns which do not satisfy $\varepsilon_{\text{MMSPAI}}$ after the first classification. Using larger values of $\varepsilon_{\text{MMSPAI}}$ yields a "skinny" $M^{(1)}$ where much improvement can be gained by a dense structure while the costs remain tolerable. Note that for some problems the unpermuted approach using successive powers on all columns leads to best results, e.g., for problem EX21. The 0-block variants are cheaper but useless because significant information of the inverse is being disregarded for the approximation. Here, an increase of α leads only to weak or almost no improvement along the successive steps. Applying SPAI to the rear part of $M^{(i)}$ produces additional costs: this hybrid approach is outperformed by the other variants. ●

Interpretation of the results: When constructing M via a sequence of preconditioners it is obvious to improve only those columns of $M^{(i)}$ which do not satisfy a certain quality metric. Thus, fill-in can be used specifically, leading to large improvement while preserving sparsity in a global sense. MMSPAI takes advantage of this approach resulting in reduced setup costs compared to an MSP based preconditioner. However, typical for SPAI-like techniques, the efficiency of MMSPAI is subject to the choice of $\mathcal{J}(M^{(i)})$. As usual, it is some kind of cost-benefit relation between having a sparse pattern, which leads to fast runtimes but weak spectral properties for M, in contrast to computationally expensive dense patterns leading to preconditioners of good quality. Our numerical experiments show that for MMSPAI it is reasonable to impose a dense pattern like $\mathcal{J}(A^{(i)T}A^{(i)})$ on the rear part $M^{(i)}_{>\varepsilon}$ after the first classification. Note that both the MSP and the MMSPAI are inherently parallel.

Table 3.1.: Comparison of the multistep MSPAI (Algorithm 7) and the MSP (Algorithm 6) for the matrix FPGA_DCOP_03 from [44].

	Matrix, Setting		
	A	FPGA_DCOP_03 [44]	
	n	1220	
	$\|A\|_F$	33.52	
	$\kappa_2(A)$	1.84E+13	
	ε_{MSP}, $\varepsilon_{\text{MMSPAI}}$	0.2	
	ρ	0.0	
	$\mathcal{J}(M^{(i)})$	$\mathcal{J}(A^{(i)})$	
α		**Multistep MSPAI**	**MSP**
0	time_{SAI}	0.27	0.16
	$\#\text{flops}_{\text{LS-solve}}$	5.17E+06	5.17E+06
	$\kappa_2(AM)$	7.37E+11	7.37E+11
	$\|AM-I\|_F$	17.92	17.92
	$\|AM-I\|_1$	1.58	1.58
	$\|AM-I\|_2$	1.66	1.66
	$\text{nnz}(M)$	4808	4808
	δ_1	1136	–
1	time_{SAI}	1.36	1.39
	$\#\text{flops}_{\text{LS-solve}}$	2.33E+08	2.35E+08
	$\kappa_2(AM)$	4.64E+15	8.86E+14
	$\|AM-I\|_F$	9.44	9.44
	$\|AM-I\|_1$	2.20	2.20
	$\|AM-I\|_2$	1.11	1.11
	$\text{nnz}(M)$	45018	45311
	δ_2	592	–
2	time_{SAI}	5.45	9.84
	$\#\text{flops}_{\text{LS-solve}}$	1.64E+09	3.87E+09
	$\kappa_2(AM)$	2.38E+01	1.35E+01
	$\|AM-I\|_F$	3.70	3.55
	$\|AM-I\|_1$	1.55	1.55
	$\|AM-I\|_2$	1.03	1.14
	$\text{nnz}(M)$	96406	148149
	δ_3	96	–
total time_{SAI}		7.08	11.39
total $\#\text{flops}_{\text{LS-solve}}$		1.88E+09	4.11E+09
	(P)BiCGSTAB		
	$\varepsilon_{\text{BiCGSTAB}}$	1E−09	
	rhs b	$(1,1,\ldots,1)^T$	
	no M	no convergence in 5000 iterations	
		Using $M^{(2)}$	
	time (iter.)	2.14 (330)	3.52 (159)

Table 3.2.: Comparison of the multistep MSPAI (Algorithm 7) and the MSP (Algorithm 6) for the matrix BCSSTK12 from [116].

	Matrix, Setting		
	A	BCSSTK12 [116]	
	n	1473	
	$\|A\|_F$	4.67E+09	
	$\kappa_2(A)$	2.21E+08	
	ε_{MSP}, $\varepsilon_{\text{MMSPAI}}$	0.6	
	ρ	0.0	
	$\mathcal{J}(M^{(i)})$	$\mathcal{J}(A^{(i)})$	
α		**Multistep MSPAI**	**MSP**
0	time_{SAI}	1.52	1.42
	$\#\text{flops}_{\text{LS-solve}}$	1.48E+08	1.48E+08
	$\kappa_2(AM)$	1.22E+05	1.22E+05
	$\|AM-I\|_F$	18.89	18.89
	$\|AM-I\|_1$	3.43	3.43
	$\|AM-I\|_2$	1.12	1.12
	$\text{nnz}(M)$	34,239	34,239
	δ_1	334	–
1	time_{SAI}	2.78	11.93
	$\#\text{flops}_{\text{LS-solve}}$	7.32E+08	3.10E+09
	$\kappa_2(AM)$	3.88E+04	2.55E+04
	$\|AM-I\|_F$	15.39	12.64
	$\|AM-I\|_1$	4.27	4.27
	$\|AM-I\|_2$	1.16	1.13
	$\text{nnz}(M)$	66,585	172,987
	δ_2	128	–
2	time_{SAI}	5.59	150.03
	$\#\text{flops}_{\text{LS-solve}}$	4.05E+09	7.57E+10
	$\kappa_2(AM)$	3.76E+04	5.64E+03
	$\|AM-I\|_F$	14.58	8.76
	$\|AM-I\|_1$	4.53	4.45
	$\|AM-I\|_2$	1.103	1.04
	$\text{nnz}(M)$	97,249	625,879
	δ_3	0	–
total time_{SAI}		9.89	163.39
total $\#\text{flops}_{\text{LS-solve}}$		4.93E+09	7.89E+10
	(P)BiCGSTAB		
	$\varepsilon_{\text{BiCGSTAB}}$	1E−08	
	rhs b	$(1,1,\ldots,1)^T$	
	no M	no convergence in 5000 iterations	
		Using $M^{(2)}$	
	time (iter.)	6.58 (981)	10.67 (289)

Table 3.3.: Comparison of MMSPAI and MSP for Numerical Experiment 3.2. For all problems BiCGSTAB does not converge in 5000 iterations for $\varepsilon_{\text{BiCGSTAB}} = 1\text{E}{-}09$ and $b = (1,1,\ldots,1)^T$. No convergence is denoted by †. Timing results have unit seconds.

Algorithm	Setting	$\kappa_2(AM)$	$\|AM-I\|_F$	nnz(M)	$\text{flops}_{\text{LS-solve}}$	t_{setup}	$t_{\text{PBiCGSTAB}}$ (iter.)	t_{total}
	MCCA [116], $n = 180$, $\kappa_2(A) = 4.02\text{E+}17$, $\|A\|_F = 2.32\text{E+}19$							
MMSPAI	$\alpha = 3$, $\varepsilon_{\text{MMSPAI}} = 0.2$, $\varepsilon_{\text{drop}} = 0.001$	1.02E+09	3.25	6,924	5.38E+07	0.44	0.22 (172)	0.66
MSP	$\alpha = 2$, $\varepsilon_{\text{MSP}} = 0.2$, $\varepsilon_{\text{drop}} = 0.0014$	2.76E+09	3.88	13,331	6.85E+07	0.77	0.63 (450)	1.39
MSP	$\alpha = 2$, $\varepsilon_{\text{MSP}} = 0.2$, $\varepsilon_{\text{drop}} = 0.05$	8.67E+12	4.25	9,736	2.79E+07	0.42	†	†
	QH882 [116], $n = 882$, $\kappa_2(A) = 2.46\text{E+}16$, $\|A\|_F = 2.26\text{E+}13$							
MMSPAI	$\alpha = 3$, $\varepsilon_{\text{MMSPAI}} = 0.3$, $\varepsilon_{\text{drop}} = 0.01$	1.21E+06	4.53	52,795	1.89E+09	2.14	1.50 (432)	3.64
MSP	$\alpha = 3$, $\varepsilon_{\text{MSP}} = 0.3$, $\varepsilon_{\text{drop}} = 0.001$	4.26E+06	3.49	276,317	7.31E+09	13.38	5.86 (450)	19.23
MSP	$\alpha = 2$, $\varepsilon_{\text{MSP}} = 0.3$, $\varepsilon_{\text{drop}} = 0.002$	4.71E+06	5.85	52,937	1.70E+08	1.16	†	†
	PORES_2 [116], $n = 1224$, $\kappa_2(A) = 1.09\text{E+}08$, $\|A\|_F = 1.50\text{E+}08$							
MMSPAI	$\alpha = 3$, $\varepsilon_{\text{MMSPAI}} = 0.4$, $\varepsilon_{\text{drop}} = 0.01$	8.76E+06	8.17	132,644	2.24E+09	7.41	4.02 (545)	11.43
MSP	$\alpha = 2$, $\varepsilon_{\text{MSP}} = 0.4$, $\varepsilon_{\text{drop}} = 0.01$	7.47E+06	9.13	150,121	8.10E+08	5.89	16.11 (1713)	22.00
MSP	$\alpha = 2$, $\varepsilon_{\text{MSP}} = 0.4$, $\varepsilon_{\text{drop}} = 0.02$	6.78E+07	9.40	136,222	5.80E+08	4.92	†	†
	FIDAP027 [116], $n = 974$, $\kappa_2(A) = 1.03\text{E+}06$, $\|A\|_F = 3.08\text{E+}01$							
MMSPAI	$\alpha = 2$, $\varepsilon_{\text{MMSPAI}} = 0.8$, $\varepsilon_{\text{drop}} = 0.1$	9.69E+04	8.48	141,788	1.96E+10	37.02	4.63 (527)	41.64
MSP	$\alpha = 2$, $\varepsilon_{\text{MSP}} = 0.8$, $\varepsilon_{\text{drop}} = 0.2$	7.59E+06	7.76	623,775	3.65E+10	127.13	12.44 (451)	139.56
MSP	$\alpha = 1$, $\varepsilon_{\text{MSP}} = 0.3$, $\varepsilon_{\text{drop}} = 0.8$	1.72E+16	13.71	155,730	1.99E+09	9.78	†	†

Table 3.4.: Comparison of different pattern strategies $\mathcal{J}(M^{(i)})$ in MMSPAI. For all problems BiCGSTAB does not converge in 5000 iterations using $\varepsilon_{\text{BiCGSTAB}} = 1\text{E}{-}09$ and $b = (1,1,\ldots,1)^T$. No convergence is denoted by †. Timing results have unit seconds.

	Algorithm	Setting	$\kappa_2(AM)$	$\|AM-I\|_F$	nnz(M)	δ_{i+1}	t_{setup}	$t_{\text{PBiCGSTAB}}$ (iter.)	t_{total}
		FIDAP_002 [116], $n = 441$, $\kappa_2(A) = 1.03\text{E+}10$, $\|A\|_F = 9.76\text{E+}08$							
MMSPAI	$A^{(i)}$	$\alpha = 2$, $\varepsilon_{\text{MMSPAI}} = 0.35$	6.82E+03	5.62	94,435	0	26.87	0.50 (67)	27.37
MMSPAI	$(A^{(i)})_{\text{0-bl.}}$	$\alpha = 2$, $\varepsilon_{\text{MMSPAI}} = 0.35$	6.08E+08	7.13	115,114	161	27.77	†	†
MMSPAI	A^{i+1}	$\alpha = 2$, $\varepsilon_{\text{MMSPAI}} = 0.35$	3.76E+09	7.43	119,649	179	36.69	†	†
MMSPAI	$A^{(i)T}A^{(i)}$	$\alpha = 2$, $\varepsilon_{\text{MMSPAI}} = 0.35$	1.53E+02	4.15	120,893	0	35.23	0.24 (24)	35.47
MMSPAI	$A^{(i)T}A^{(i)}$	$\alpha = 2$, $\varepsilon_{\text{MMSPAI}} = 0.55$	2.67E+04	7.32	46,807	0	8.47	1.66 (156)	10.13
MMSPAI	$(A^{(i)T}A^{(i)})_{\text{0-bl.}}$	$\alpha = 2$, $\varepsilon_{\text{MMSPAI}} = 0.35$	2.40E+09	6.97	123,287	140	41.56	†	†
MMSPAI	$\text{SPAI}_{\text{rear part}}$	$\bar{\Upsilon}_{1,5}$, $\alpha = 2$, $\varepsilon_{\text{MMSPAI}} = 0.35$	6.07E+02	5.36	94,234	0	34.61	0.36 (37)	34.97
$\text{SAI}(A^{i+1})$		$\alpha = 1$	3.36E+07	3.09	82,953	–	16.17	0.23 (27)	16.41
		EX21 [44], $n = 656$, $\kappa_2(A) = 5.76\text{E+}08$, $\|A\|_F = 2.21$							
MMSPAI	$A^{(i)}$	$\alpha = 3$, $\varepsilon_{\text{MMSPAI}} = 0.7$	2.14E+05	6.64	115,362	0	57.89	1.36 (253)	59.25
MMSPAI	$(A^{(i)})_{\text{0-bl.}}$	$\alpha = 3$, $\varepsilon_{\text{MMSPAI}} = 0.7$	2.36E+10	13.69	104,216	186	27.44	†	†
MMSPAI	A^{i+1}	$\alpha = 3$, $\varepsilon_{\text{MMSPAI}} = 0.7$	3.78E+11	13.65	116,864	182	51.95	†	†
MMSPAI	$A^{(i)T}A^{(i)}$	$\alpha = 3$, $\varepsilon_{\text{MMSPAI}} = 0.7$	9.02E+03	5.63	106,009	0	40.36	0.44 (76)	40.80
MMSPAI	$A^{(i)T}A^{(i)}$	$\alpha = 3$, $\varepsilon_{\text{MMSPAI}} = 0.9$	1.73E+06	6.55	101,452	0	35.66	8.91 (1259)	44.56
MMSPAI	$(A^{(i)T}A^{(i)})_{\text{0-bl.}}$	$\alpha = 3$, $\varepsilon_{\text{MMSPAI}} = 0.7$	3.01E+10	13.69	104,508	186	80.16	†	†
MMSPAI	$\text{SPAI}_{\text{rear part}}$	$\bar{\Upsilon}_{1,15}$, $\alpha = 2$, $\varepsilon_{\text{MMSPAI}} = 0.7$	5.00E+06	7.59	109,635	9	66.49	12.89 (1506)	79.38
$\text{SAI}(A^{i+1})$		$\alpha = 1$	1.23E+04	8.38	78,258	–	6.83	0.47 (98)	7.30
		CAVITY04 [116], $n = 317$, $\kappa_2(A) = 3.54\text{E+}06$, $\|A\|_F = 3.51\text{E+}02$							
MMSPAI	$A^{(i)}$	$\alpha = 2$, $\varepsilon_{\text{MMSPAI}} = 0.5$	4.64E+03	5.53	22,817	32	3.86	0.08 (51)	3.94
MMSPAI	$(A^{(i)})_{\text{0-bl.}}$	$\alpha = 2$, $\varepsilon_{\text{MMSPAI}} = 0.5$	3.14E+08	9.41	17,422	74	1.20	†	†
MMSPAI	A^{i+1}	$\alpha = 2$, $\varepsilon_{\text{MMSPAI}} = 0.5$	4.57E+08	9.42	14,422	74	0.88	†	†
MMSPAI	$A^{(i)T}A^{(i)}$	$\alpha = 2$, $\varepsilon_{\text{MMSPAI}} = 0.5$	4.44E+00	4.24	23,015	0	2.88	0.02 (12)	2.89
MMSPAI	$A^{(i)T}A^{(i)}$	$\alpha = 2$, $\varepsilon_{\text{MMSPAI}} = 0.9$	1.10E+02	5.12	22,388	0	2.58	0.05 (28)	2.63
MMSPAI	$(A^{(i)T}A^{(i)})_{\text{0-bl.}}$	$\alpha = 2$, $\varepsilon_{\text{MMSPAI}} = 0.5$	5.07E+08	9.41	18,608	74	2.64	†	†
MMSPAI	$\text{SPAI}_{\text{rear part}}$	$\bar{\Upsilon}_{1,5}$, $\alpha = 2$, $\varepsilon_{\text{MMSPAI}} = 0.5$	1.43E+02	5.00	22,817	18	8.73	0.05 (27)	8.78
$\text{SAI}(A^{i+1})$		$\alpha = 2$	1.58E+03	4.76	39,725	–	3.80	0.20 (88)	4.00

3.2. K-condition Number Minimization for the Symmetric Positive Definite Case

SPD systems typically occur in energy preserving systems where stable states are characterized by minimum energy [3]. Efficient iterative solvers like CG are constrained to such systems and therewith require preconditioners which are SPD, as well. Although SPAI is able to compute preconditioners for general A, symmetry or positive definiteness normally are not preserved for M if A is SPD. It is possible to use certain techniques which allow for a subsequent symmetrization of M such as

$$M + M^T \qquad \text{and} \qquad M + M^T - M^T A M,$$

or further approaches presented in [85, 99]. However, it cannot be assured that the resulting preconditioners or preconditioned systems are positive definite.

For SPD matrices Kolotilina and Yeremin introduced the *factorized sparse approximate inverse* (FSAI) approach [103, 104, 105, 106] which computes an approximation L to the inverse of the unknown Cholesky factor L_A of $A = L_A^T L_A$, i.e., $L \approx L_A^{-1}$, via

$$\min_{\mathcal{J}(L)\in\mathcal{J}} \|L_A L - I\|_F \qquad \text{such that} \qquad M = LL^T \approx L_A^{-1} L_A^{-T} = A^{-1}.$$

There is no need to know L_A because the resulting LS problem leads to a linear system in a reduced submatrix of A. We can decouple the Frobenius norm into a sum of Euclidean norms

$$\min_{\mathcal{J}(L)\in\mathcal{J}} \|L_A L - I\|_F^2 = \sum_{k=1}^{n} \min_{\mathcal{J}(L)\in\mathcal{J}} \|(L_A L - I)e_k\|_2^2 = \sum_{k=1}^{n} \min_{\mathcal{J}(L_k)\in\mathcal{J}_k} \|L_A L_k - e_k\|_2^2. \tag{3.34}$$

The minimum of each column of L in (3.34) is obtained for the normal equation $AL_k = L_A^T e_k$. However, the a priori imposed lower triangular sparsity structure $\mathcal{J}(L_k) \in \mathcal{J}_k$ leads to the reduced lower dimensional problem

$$\hat{A}_k \hat{L}_k = l_{kk} \hat{e}_k, \tag{3.35}$$

where, similar to (3.6) for SAI,

$$\hat{A}_k := A(\mathcal{J}_k, \mathcal{J}_k) \in \mathbb{R}^{q_k \times q_k}, \qquad \hat{L}_k := L_k(\mathcal{J}_k) \in \mathbb{R}^{q_k}, \qquad \text{and} \qquad \hat{e}_k := e_k(\mathcal{J}_k) \in \mathbb{R}^{q_k}.$$

As l_{kk} is unknown, the system (3.35) is solved typically for $l_{kk} = 1$ and a proper *diagonal scaling* $\operatorname{diag}(L^T A L) = I$ is applied afterward. In contrast to, for example, IC or SAINV, the computation of an FSAI is inherently parallel, always stable and the resulting preconditioner is well defined [16].

Remark 3.5 *The FSAI approach can also be modified to the unsymmetric and symmetric indefinite case; see [144] and [107], respectively. Concerning the former, approximate inverse factors $\tilde{L} \approx L$ and $\tilde{U} \approx U$ of an unsymmetric LU decomposable matrix $A = LU$ are computed. For the latter, the k^{th} component of the solution of (3.35) is not positive—in contrast to the SPD case—and thus the scaling is performed with the absolute value $l_{kk}^2 = |\hat{l}_{kk}|$. However, selecting $\mathcal{J}(L)$ must be done with caution, e.g., for $\mathcal{J}(L) = \mathcal{J}(\operatorname{low}(A))$ nonsingularity cannot be guaranteed.*

It is possible to proof that an FSAI preconditioner constructed via (3.34) is optimal in the sense that it minimizes the K-condition number (2.19) of the preconditioned matrix

$$K(L^TAL) = \frac{\text{trace}(L^TAL)}{n\det(L^TAL)^{\frac{1}{n}}} \tag{3.36}$$

over the set of matrices with a prescribed pattern $\mathcal{J}$ [105]. Hence, it is possible to derive factorized sparse approximate inverse preconditioners on the basis of (3.36). In the following we focus on a K-condition number based computation for L according to Huckle [81, 82].

3.2.1. Computation for a Fixed Sparsity Pattern $\mathcal{J}$: FSAI

We define the set of allowed entries in L_k by

$$\mathcal{J}_k := \{j \in \{k,\dots,n\} : L_k(j) \neq 0\} \qquad \text{with} \qquad \tilde{\mathcal{J}}_k := \mathcal{J}_k \setminus \{k\}.$$

To abbreviate our notation, we use

Definition 3.3 (FSAI Schur complement) *Consider the index set $\mathcal{I}_k := \{k,\dots,n\}\setminus\tilde{\mathcal{J}}_k$ induced by $\mathcal{J}_k$. We denote the Schur complement of $A(\tilde{\mathcal{J}}_k,\tilde{\mathcal{J}}_k)$ in $A_{k:n,k:n}$ by*

$$S(\mathcal{I}_k,\mathcal{I}_k) := A(\mathcal{I}_k,\mathcal{I}_k) - A(\tilde{\mathcal{J}}_k,\mathcal{I}_k)^TA(\tilde{\mathcal{J}}_k,\tilde{\mathcal{J}}_k)^{-1}A(\tilde{\mathcal{J}}_k,\mathcal{I}_k)$$

with the entries $s_{ij} = a_{ij} - A_i(\tilde{\mathcal{J}}_k)^TA(\tilde{\mathcal{J}}_k,\tilde{\mathcal{J}}_k)^{-1}A_j(\tilde{\mathcal{J}}_k)$. S is SPD with $s_{kk} > 0$.

In order to obtain the minimum of (3.36), we consider the splitting

$$L_k(\mathcal{J}_k) = l_{kk}\tilde{e}_k + \tilde{L}_k, \qquad \text{where} \qquad \tilde{e}_k := e_k(\mathcal{J}_k) \qquad \text{and} \qquad \tilde{L}_k := L_k(\mathcal{J}_k) - l_{kk}\tilde{e}_k.$$

Therewith, we expand $K(L^TAL)$ to

$$\begin{aligned}
\frac{\text{trace}(L^TAL)}{n\det(L^TAL)^{\frac{1}{n}}} &= \frac{\sum_{k=1}^n\left[L_k^TAL_k\right]}{n\det(A)^{\frac{1}{n}}\prod_{k=1}^n l_{kk}^{\frac{2}{n}}} = \frac{\sum_{k=1}^n\left[(l_{kk}\tilde{e}_k^T + \tilde{L}_k^T)A(\mathcal{J}_k,\mathcal{J}_k)(l_{kk}\tilde{e}_k + \tilde{L}_k)\right]}{n\det(A)^{\frac{1}{n}}\prod_{k=1}^n l_{kk}^{\frac{2}{n}}} \\
&= \frac{\sum_{k=1}^n\left[l_{kk}^2A_{kk} + l_{kk}A_k(\mathcal{J}_k)^T\tilde{L}_k + l_{kk}\tilde{L}_k^TA_k(\mathcal{J}_k) + \tilde{L}_k^TA(\mathcal{J}_k,\mathcal{J}_k)\tilde{L}_k\right]}{n\det(A)^{\frac{1}{n}}\prod_{k=1}^n l_{kk}^{\frac{2}{n}}} \\
&= \frac{\sum_{k=1}^n\left[l_{kk}^2a_{kk} + 2l_{kk}L_k(\tilde{\mathcal{J}}_k)^TA_k(\tilde{\mathcal{J}}_k) + L_k(\tilde{\mathcal{J}}_k)^TA(\tilde{\mathcal{J}}_k,\tilde{\mathcal{J}}_k)L_k(\tilde{\mathcal{J}}_k)\right]}{n\det(A)^{\frac{1}{n}}\prod_{k=1}^n l_{kk}^{\frac{2}{n}}}.
\end{aligned} \tag{3.37}$$

The differentiation of the expanded form relative to the unknowns $L_k(\tilde{\mathcal{J}}_k)$ and l_{kk} yields the optimal FSAI solution for column k. We insert the derivative

$$\begin{aligned}
\frac{\partial K(L^TAL)}{\partial L_k(\tilde{\mathcal{J}}_k)} &= n^{-1}\det(A)^{-\frac{1}{n}}\prod_{k=1}^n l_{kk}^{-\frac{2}{n}}\left[2l_{kk}A_k(\tilde{\mathcal{J}}_k) + 2A(\tilde{\mathcal{J}}_k,\tilde{\mathcal{J}}_k)L_k(\tilde{\mathcal{J}}_k)\right] \overset{!}{=} 0 \\
&\Leftrightarrow \quad L_k(\tilde{\mathcal{J}}_k) = -l_{kk}A(\tilde{\mathcal{J}}_k,\tilde{\mathcal{J}}_k)^{-1}A_k(\tilde{\mathcal{J}}_k)
\end{aligned}$$

into (3.37) and obtain

$$
\begin{aligned}
& n^{-1}\det(A)^{-\frac{1}{n}}\prod_{k=1}^{n} l_{kk}^{-\frac{2}{n}} \sum_{k=1}^{n} \Big[l_{kk}^2 a_{kk} + 2 l_{kk}\Big(- l_{kk} A(\tilde{\mathcal{J}}_k,\tilde{\mathcal{J}}_k)^{-1} A_k(\tilde{\mathcal{J}}_k)\Big)^T A_k(\tilde{\mathcal{J}}_k) \;+ \\
& \qquad + \;\Big(- l_{kk} A(\tilde{\mathcal{J}}_k,\tilde{\mathcal{J}}_k)^{-1} A_k(\tilde{\mathcal{J}}_k)\Big)^T A(\tilde{\mathcal{J}}_k,\tilde{\mathcal{J}}_k)\Big(- l_{kk} A(\tilde{\mathcal{J}}_k,\tilde{\mathcal{J}}_k)^{-1} A_k(\tilde{\mathcal{J}}_k)\Big)\Big] \\
= \; & \frac{\sum_{k=1}^{n}\big[l_{kk}^2 \big(a_{kk} - A_k(\tilde{\mathcal{J}}_k)^T A(\tilde{\mathcal{J}}_k,\tilde{\mathcal{J}}_k)^{-1} A_k(\tilde{\mathcal{J}}_k)\big)\big]}{n\det(A)^{\frac{1}{n}}\prod_{k=1}^{n} l_{kk}^{\frac{2}{n}}} \;=\; \frac{\sum_{k=1}^{n} l_{kk}^2 s_{kk}}{n\det(A)^{\frac{1}{n}}\prod_{k=1}^{n} l_{kk}^{\frac{2}{n}}}.
\end{aligned} \tag{3.38}
$$

The derivative of (3.38) with respect to l_{kk} and renamed indices yields

$$
\begin{aligned}
& \frac{\mathrm{d}}{\mathrm{d}l_{kk}}\left(\frac{\sum_{j=1}^{n} l_{jj}^2 s_{jj}}{n\det(A)^{\frac{1}{n}}\prod_{j=1}^{n} l_{jj}^{\frac{2}{n}}}\right) \;\overset{!}{=}\; 0 \\
\Leftrightarrow \quad & \frac{\mathrm{d}}{\mathrm{d}l_{kk}}\left(\frac{\sum_{j=1}^{n} l_{jj}^2 s_{jj}}{l_{kk}^{\frac{2}{n}}}\right) \;=\; \frac{\mathrm{d}}{\mathrm{d}l_{kk}}\left(l_{kk}^{2-\frac{2}{n}} s_{kk} + l_{kk}^{-\frac{2}{n}} \sum_{\substack{j=1\\ j\neq k}}^{n} l_{jj}^2 s_{jj}\right) \\
& = \left(2-\frac{2}{n}\right) l_{kk}^{1-\frac{2}{n}} s_{kk} - \frac{2}{n} l_{kk}^{-\frac{2}{n}-1} \sum_{\substack{j=1\\ j\neq k}}^{n} l_{jj}^2 s_{jj} \;\overset{!}{=}\; 0 \\
\Leftrightarrow \quad & \left(2-\frac{2}{n}\right) l_{kk}^2 s_{kk} - \frac{2}{n}\sum_{\substack{j=1\\ j\neq k}}^{n} l_{jj}^2 s_{jj} \;=\; 2 l_{kk}^2 s_{kk} - \frac{2}{n}\sum_{j=1}^{n} l_{jj}^2 s_{jj} \;\overset{!}{=}\; 0 \\
\Leftrightarrow \quad & l_{kk}^2 s_{kk} = \frac{1}{n}\sum_{j=1}^{n} l_{jj}^2 s_{jj}.
\end{aligned} \tag{3.39}
$$

The right-hand side of (3.39) is the same for all columns of L. Hence,

$$
\exists c > 0 \;:\; \forall k \;:\; l_{kk}^2 s_{kk} = c. \tag{3.40}
$$

The choice of c has no influence on the (spectral) properties of the preconditioned system, i.e., it has no influence on the quality of L. Without loss of generality, we choose $c = 1$ which—according to Theorem 3.6—corresponds to a normalization of L_k. Consequently, we can compute each column L_k independently from all other columns:

$$
y_k \;:=\; A(\tilde{\mathcal{J}}_k,\tilde{\mathcal{J}}_k)^{-1} A_k(\tilde{\mathcal{J}}_k), \tag{3.41}
$$

$$
l_{kk} \;=\; \frac{\sqrt{c}}{\sqrt{s_{kk}}} \;\overset{c=1}{=}\; \frac{1}{\sqrt{a_{kk} - A_k(\tilde{\mathcal{J}}_k)^T y_k}}, \tag{3.42}
$$

$$
L_k(\tilde{\mathcal{J}}_k) \;=\; -l_{kk} y_k. \tag{3.43}
$$

The approximate inverse of every column can be obtained by solving the small SPD system (3.41) which represents the computationally dominating part. Typically, the Cholesky factorization is used for this purpose. Minimizing the K-condition number solves the linear system and inherently leads to the normalization $\mathrm{diag}(L^T A L) \approx I$, while the Frobenius

norm minimization approach (3.34) requires the side condition $l_{kk} = 1$ with proper scaling afterward.

3.2.2. A Priori Choices of $\mathcal{J}$

The SPD case is similar to the general unsymmetric case according to Section 3.1.2: choosing a sparsity pattern $\mathcal{J}(L)$ to capture most profitable components remains a heuristic. Note that for symmetric A the condition $e_k^T A e_j \neq 0$ for $j \in \mathcal{J}_k$ is always satisfied according to Theorem 3.1. Hence, by choosing $\mathcal{J}(L) = \mathcal{J}(\text{low}(A))$, it is impossible to produce FSAI preconditioners containing zero columns because $\mathcal{J}(\text{low}(A)) = \mathcal{J}(\text{low}(A^T))$.

Pattern $\mathcal{J}$ as a Power of Sparsified A

Similar to the general case, it is possible to use Chow's approach [38] and sparsify A according to (3.8). Obtaining $\tilde{A}$ we can impose the pattern $\mathcal{J}(L) = \mathcal{J}(\text{low}(\tilde{A}^k))$ using some small power k. Depending on the structure of $\tilde{A}$, dropping the entries $|l_{ij}| < \tau_{\text{filter}}$ can be useful in order to get a more sparse FSAI preconditioner.

Upper Bound Patterns

Following Huckle [80], we can also specify an upper bound pattern $\mathcal{J}(L) = \mathcal{J}(L^{(\alpha)})$ for an FSAI L which contains the pattern of an FSPAI $L^{(\alpha)}$ constructed by a sweep of α steps; see next section. Using an arbitrary initial solution $L^{(0)}$ we observe

$$\mathcal{J}(L^{(\alpha)}) \subset \mathcal{J}(\underbrace{\text{low}(A \ldots \text{low}(A\,\text{low}(AL^{(0)})))}_{\alpha \text{ times}}). \tag{3.44}$$

For $\mathcal{J}(L^{(0)}) = \mathcal{J}(I)$, the bound (3.44) simplifies to $\mathcal{J}(L^{(\alpha)}) \subset \mathcal{J}(\text{low}(A \ldots \text{low}(A\,\text{diag}(A))))$.

3.2.3. Updating the Sparsity Pattern $\mathcal{J}$: FSPAI

Similar to SPAI, the *factorized sparse approximate inverse* (FSPAI) approach by Huckle [81, 82] overcomes the problem of providing a priori knowledge for $\mathcal{J}(L)$. The pattern of a current approximation is augmented with new promising entries in order to improve the preconditioner while preserving FSAI's inherent parallelism.

Pattern Updates

Let L_k be the computed FSAI solution of the k^{th} column of L for a given pattern $\mathcal{J}_k$. In the following we use the

Definition 3.4 (FSAI K-condition number) *By K_{old} we denote the K-condition number (3.36) for a given FSAI solution L while K_{new} denotes the K-condition number for an improved approximation L_{new}, resulting from an augmented index set $\mathcal{J}_k$, e.g., $\mathcal{J}_k \cup \{j\}$.*

Similar to SPAI, we are interested in a quality metric when adding the index j to $\mathcal{J}_k$ in the k^{th} column of L. As L is lower triangular we solve the univariate minimization

$$\min_{l_{jk}} \frac{\operatorname{trace}(L_{\text{new}}^T A L_{\text{new}})}{n \det(L_{\text{new}}^T A L_{\text{new}})^{\frac{1}{n}}} = \min_{l_{jk}} \frac{\operatorname{trace}\left[\left(L + e_j l_{jk} e_k^T\right)^T A \left(L + e_j l_{jk} e_k^T\right) \right]}{n \det(A)^{\frac{1}{n}} \prod_{k=1}^{n} l_{kk}^{\frac{2}{n}}} \tag{3.45}$$

for all $j > k$ in which $L_{\text{new}} := L + e_j l_{jk} e_k^T$. The denominator of (3.45) is positive and not a function in l_{jk} because $j \in \{k+1, \ldots, n\}$. Hence, the minimum of (3.45) is obtained at

$$\begin{aligned}
& \frac{\mathrm{d}}{\mathrm{d}l_{jk}} \left(\operatorname{trace}\left[\left(L + e_j l_{jk} e_k^T\right)^T A \left(L + e_j l_{jk} e_k^T\right) \right] \right) \\
= \; & \frac{\mathrm{d}}{\mathrm{d}l_{jk}} \Big(\operatorname{trace}\left(L^T A L\right) + 2 l_{jk} \operatorname{trace}\left(L^T A e_j e_k^T\right) + l_{jk}^2 a_{jj} \operatorname{trace}\left(e_k e_k^T\right) \Big) \qquad (3.46) \\
= \; & \frac{\mathrm{d}}{\mathrm{d}l_{jk}} \Big(\operatorname{trace}(L^T A L) + 2 l_{jk} A_j^T L_k + l_{jk}^2 a_{jj} \Big) \overset{!}{=} 0 \\
\Leftrightarrow \; & l_{jk} = -\frac{A_j^T L_k}{a_{jj}}, \qquad (3.47)
\end{aligned}$$

which is a minimum because the second derivative equals $2a_{jj} > 0$, due to SPD A. Inserting (3.47) into (3.46) and using the property $\operatorname{trace}(L^T A L) = n$ according to Corollary 3.1, the new K-condition number is bounded by

$$\begin{aligned}
1 \overset{(2.20)}{\leq} K_{\text{new}} &\leq \frac{\operatorname{trace}(L^T A L) + 2 l_{jk} A_j^T L_k + l_{jk}^2 a_{jj}}{n \det(L^T A L)^{\frac{1}{n}}} \\
&= \frac{n + 2\left(-\dfrac{A_j^T L_k}{a_{jj}}\right) A_j^T L_k + \left(-\dfrac{A_j^T L_k}{a_{jj}}\right)^2 a_{jj}}{n \det(L^T A L)^{\frac{1}{n}}} \\
&= \frac{n - \dfrac{(A_j^T L_k)^2}{a_{jj}}}{n \det(L^T A L)^{\frac{1}{n}}} = \det(L^T A L)^{-\frac{1}{n}} \left[1 - \frac{(A_j^T L_k)^2}{a_{jj} n} \right] \\
&= \left(1 - \frac{\tau_{jk}}{n}\right) K_{\text{old}}, \qquad \text{where} \qquad \tau_{jk} := \frac{(A_j^T L_k)^2}{a_{jj}} \qquad (3.48)
\end{aligned}$$

represents the reduction or quality metric for index j in column k. Due to the sparsity of A, only some indices in the shadow of $AL_k(\mathcal{J}_k)$ have to be considered. The set of all new indices which will lead to a reduction is defined as

$$\hat{\mathcal{J}}_k := \{ j : \; j > k \; \wedge \; A_j(\mathcal{J}_k)^T L_k(\mathcal{J}_k) \neq 0 \} \setminus \mathcal{J}_k \tag{3.49}$$

and can be efficiently detected by algorithms working on the graph with respect to the pattern of A. We update the current pattern with one or several indices $\mathcal{J}_{\text{new}} \subset \hat{\mathcal{J}}_k$ which yield the largest reduction of K_{new}, i.e., which correspond to the largest values τ_{jk}. Computing a new FSAI for this augmented index set $\mathcal{J}_k = \mathcal{J}_k \cup \mathcal{J}_{\text{new}}$ will result in a more accurate factorized sparse approximate inverse.

Similar to implementations of SPAI, it is possible to indirectly control the fill-in and the accuracy of the approximation of the FSPAI preconditioner by several parameters. The guidelines suggested in Section 3.1.3 for SPAI apply for FSPAI, as well. We can use $\Upsilon_{\alpha,\beta}$, $\bar{\Upsilon}_{\alpha,\beta}$, $p_{\max}$, and $\varepsilon_{\text{FSPAI}}$ in the same way. By default, our FSPAI implementation (see Section 7.3) makes use of the arithmetic mean value heuristic

$$\tau_{\text{opt}} \geq \bar{\tau}_k := \frac{1}{|\hat{\mathcal{J}}_k|} \sum_{j \in \hat{\mathcal{J}}_k} \tau_{jk} \tag{3.50}$$

as acceptance criterion, adding only those indices j with $\tau_{jk} > \bar{\tau}_k$. We terminate updating the pattern of an FSPAI if all τ_{jk} become smaller than the given tolerance $\varepsilon_{\text{FSPAI}}$. Algorithm 8 summarizes the main steps of FSPAI using pattern updates.

Remark 3.6

1. *Corresponding to the multivariate minimization approach for SPAI by Gould and Scott [55], Huckle proposed a related approach for FSPAI in [82]. Keeping the diagonal entry l_{kk} of a current FSPAI solution L_{old} fixed, we compute the K-condition number for the enlarged index set $\tilde{\mathcal{J}}_k \cup \{j\}$ for each $j > k$. The resulting linear system*

$$\begin{pmatrix} L_k(\tilde{\mathcal{J}}_k) \\ l_{jk} \end{pmatrix} = -l_{kk} \begin{pmatrix} A(\tilde{\mathcal{J}}_k,\tilde{\mathcal{J}}_k) & A_j(\tilde{\mathcal{J}}_k) \\ A_j(\tilde{\mathcal{J}}_k)^T & a_{jj} \end{pmatrix}^{-1} \begin{pmatrix} A_k(\tilde{\mathcal{J}}_k) \\ a_{jk} \end{pmatrix}$$

has the solution

$$\begin{aligned} l_{jk} &= -\frac{A_j(\tilde{\mathcal{J}}_k)^T L_{old} + l_{kk} a_{jk}}{s_{jj}} \qquad \text{and} \\ L_k(\tilde{\mathcal{J}}_k) &= L_{old} - l_{jk} A(\tilde{\mathcal{J}}_k,\tilde{\mathcal{J}}_k)^{-1} A_j(\tilde{\mathcal{J}}_k). \end{aligned}$$

Using the Cholesky factorization of $A(\tilde{\mathcal{J}}_k,\tilde{\mathcal{J}}_k)$ we can compute the K-condition number for every j and augment the current pattern with the most profitable index or indices.

2. *For sparse problems $Ax = b$, where $A \in \mathbb{C}^{n\times n}$ is Hermitian positive definite (HPD) and $x, b \in \mathbb{C}^n$, y_k is a complex vector which is to be obtained by solving the reduced HPD system $A(\tilde{\mathcal{J}}_k,\tilde{\mathcal{J}}_k)y_k = A_k(\tilde{\mathcal{J}}_k)$ corresponding to (3.41). Identifying the diagonal elements l_{kk} requires both the computation of the dot product $A_k(\tilde{\mathcal{J}}_k)^H y_k$, involving a complex conjugation for $A_k(\tilde{\mathcal{J}}_k)$, and the root of a complex number $w := \sqrt{z}$, $w, z \in \mathbb{C}$, occurring in line 7 or 10 in Algorithm 8. Based on the absolute value and the phase*

$$r := \sqrt{\mathfrak{Re}[z]^2 + \mathfrak{Im}[z]^2} \qquad \text{and} \qquad \psi := \arccos\left(\frac{\mathfrak{Re}[z]}{r}\right),$$

respectively, our FSPAI implementation (see Section 7.3) computes the first root of unity w of z via

$$\mathfrak{Re}[w] = \sqrt{r}\cos\left(\frac{\psi}{2}\right) \qquad \text{and} \qquad \mathfrak{Im}[w] = \sqrt{r}\sin\left(\frac{\psi}{2}\right).$$

Algorithm 8: FSPAI with univariate K-condition number minimization using the mean value heuristic (3.50).

Input : $A \in \mathbb{R}^{n\times n}$, $\mathcal{J} \in \mathbb{B}^{n\times n}$, $\varepsilon_{\text{FSPAI}} \geq 0$, $\alpha \geq 0$, $\beta \geq 0$
Output: Preconditioner $L \approx L_A^{-1}$, $L \in \mathbb{R}^{n\times n}$

for $k \leftarrow 1$ **to** n **do**
  $e_k \leftarrow I(.,k)$
  $\mathcal{J}_k \leftarrow \mathcal{J}(.,k)$
  for `step` $\leftarrow 0$ **to** α **do**
    $\tilde{\mathcal{J}}_k \leftarrow \mathcal{J}_k \setminus \{k\}$
    if $\tilde{\mathcal{J}}_k = \emptyset$ **then**
      $l_{kk} \leftarrow a_{kk}^{-\frac{1}{2}}$
    else
      $y_k \leftarrow A(\tilde{\mathcal{J}}_k,\tilde{\mathcal{J}}_k)^{-1} A_k(\tilde{\mathcal{J}}_k)$
      $l_{kk} \leftarrow \left(a_{kk} - A_k(\tilde{\mathcal{J}}_k)^T y_k\right)^{-\frac{1}{2}}$
      $L_k(\tilde{\mathcal{J}}_k) \leftarrow -l_{kk} y_k$
    end
    if `step` $= \alpha$ **then**
      break
    end
    $\hat{\mathcal{J}}_k \leftarrow \{j : \ j > k \ \wedge \ A_j(\mathcal{J}_k)^T L_k(\mathcal{J}_k) \neq 0\} \setminus \mathcal{J}_k$
    $\bar{\tau}_k \leftarrow 0$
    foreach $j \in \hat{\mathcal{J}}_k$ **do**
      $\tau_{jk} \leftarrow \dfrac{[A_j(\mathcal{J}_k)^T L_k(\mathcal{J}_k)]^2}{a_{jj}}$
      $\bar{\tau}_k \leftarrow \bar{\tau}_k + \tau_{jk}$
    end
    if $\max_{j\in\hat{\mathcal{J}}_k} \tau_{jk} \leq \varepsilon_{\texttt{FSPAI}}$ **then**
      break
    end
    $\bar{\tau}_k \leftarrow \dfrac{\bar{\tau}_k}{\left|\hat{\mathcal{J}}_k\right|}$
    for `idx` $\leftarrow 1$ **to** β **do**
      $j \leftarrow \arg\max_{j\in\hat{\mathcal{J}}_k} \tau_{jk}$
      $\mathcal{J}_k \leftarrow \mathcal{J}_k \cup \{j : \tau_{jk} \geq \bar{\tau}_k\}$
      $\hat{\mathcal{J}}_k \leftarrow \hat{\mathcal{J}}_k \setminus \{j\}$
    end
  end
end

Cholesky Updates

Similar to SPAI using QR Updates, we can update a given Cholesky factorization in order to reduce the time to setup an FSPAI for an augmented index set $\tilde{\mathcal{J}}_k \cup \mathcal{J}_{\text{new}}$. Here, $\mathcal{J}_{\text{new}}$ denotes the set of new candidates with $\tilde{q}_k := |\mathcal{J}_{\text{new}}|$. Hence, the unknown Cholesky factorization of the enlarged system $A(\tilde{\mathcal{J}}_k \cup \mathcal{J}_{\text{new}}, \tilde{\mathcal{J}}_k \cup \mathcal{J}_{\text{new}}) =: \tilde{L}^T\tilde{L}$ is based on an already known factorization

$$A(\tilde{\mathcal{J}}_k,\tilde{\mathcal{J}}_k) = P^T L^T L P, \tag{3.51}$$

making a fast update possible. P is a permutation matrix from the previous step while $\tilde{P}$ indicates a matrix which permutes the new indices to the top positions, i.e., for an arbitrary vector x we get

$$\left(\frac{x(\mathcal{J}_{\text{new}})}{x(\tilde{\mathcal{J}}_k)}\right) = \tilde{P}x(\tilde{\mathcal{J}}_k \cup \mathcal{J}_{\text{new}}).$$

Note that for the first update $P = I$. Using the known Cholesky factor L, the augmented system can be written as

$$\begin{aligned}
& A(\tilde{\mathcal{J}}_k \cup \mathcal{J}_{\text{new}}, \tilde{\mathcal{J}}_k \cup \mathcal{J}_{\text{new}}) = \tilde{P}^T \begin{pmatrix} A(\mathcal{J}_{\text{new}},\mathcal{J}_{\text{new}}) & A(\mathcal{J}_{\text{new}},\tilde{\mathcal{J}}_k) \\ A(\tilde{\mathcal{J}}_k,\mathcal{J}_{\text{new}}) & A(\tilde{\mathcal{J}}_k,\tilde{\mathcal{J}}_k) \end{pmatrix} \tilde{P} \\
= \; & \tilde{P}^T \begin{pmatrix} A(\mathcal{J}_{\text{new}},\mathcal{J}_{\text{new}}) & A(\mathcal{J}_{\text{new}},\tilde{\mathcal{J}}_k) \\ A(\tilde{\mathcal{J}}_k,\mathcal{J}_{\text{new}}) & P^T L^T L P \end{pmatrix} \tilde{P} \\
= \; & \tilde{P}^T \begin{pmatrix} I_{\tilde{q}_k} & 0 \\ 0 & P^T \end{pmatrix} \begin{pmatrix} A(\mathcal{J}_{\text{new}},\mathcal{J}_{\text{new}}) & A(\mathcal{J}_{\text{new}},\tilde{\mathcal{J}}_k)P^T \\ PA(\tilde{\mathcal{J}}_k,\mathcal{J}_{\text{new}}) & L^T L \end{pmatrix} \begin{pmatrix} I_{\tilde{q}_k} & 0 \\ 0 & P \end{pmatrix} \tilde{P} \\
= \; & \left[\begin{pmatrix} I_{\tilde{q}_k} & 0 \\ 0 & P \end{pmatrix} \tilde{P}\right]^T \begin{pmatrix} A(\mathcal{J}_{\text{new}},\mathcal{J}_{\text{new}}) & A(\mathcal{J}_{\text{new}},\tilde{\mathcal{J}}_k)P^T \\ PA(\tilde{\mathcal{J}}_k,\mathcal{J}_{\text{new}}) & L^T L \end{pmatrix} \left[\begin{pmatrix} I_{\tilde{q}_k} & 0 \\ 0 & P \end{pmatrix} \tilde{P}\right].
\end{aligned} \tag{3.52}$$

The inner part of (3.52) will satisfy the equation

$$\begin{pmatrix} A(\mathcal{J}_{\text{new}},\mathcal{J}_{\text{new}}) & A(\mathcal{J}_{\text{new}},\tilde{\mathcal{J}}_k)P^T \\ PA(\tilde{\mathcal{J}}_k,\mathcal{J}_{\text{new}}) & L^T L \end{pmatrix} \overset{!}{=} \tilde{L}^T\tilde{L} = \begin{pmatrix} \tilde{L}_1^T\tilde{L}_1 + \tilde{L}_2^T\tilde{L}_2 & \tilde{L}_2^T\tilde{L}_3 \\ \tilde{L}_3^T\tilde{L}_2 & \tilde{L}_3^T\tilde{L}_3 \end{pmatrix}$$

if and only if

$$\begin{aligned}
L^T L \overset{!}{=} \tilde{L}_3^T\tilde{L}_3 \quad &\Leftrightarrow \quad \tilde{L}_3 = L \\
PA(\tilde{\mathcal{J}}_k,\mathcal{J}_{\text{new}}) \overset{!}{=} \tilde{L}_3^T\tilde{L}_2 \quad &\Leftrightarrow \quad \tilde{L}_2 = L^{-T}PA(\tilde{\mathcal{J}}_k,\mathcal{J}_{\text{new}}) \\
A(\mathcal{J}_{\text{new}},\mathcal{J}_{\text{new}}) \overset{!}{=} \tilde{L}_1^T\tilde{L}_1 + \tilde{L}_2^T\tilde{L}_2 \quad &\Leftrightarrow \quad \tilde{L}_1 = \text{chol}\left[A(\mathcal{J}_{\text{new}},\mathcal{J}_{\text{new}}) - \tilde{L}_2^T\tilde{L}_2\right].
\end{aligned}$$

Thus, the Cholesky factorization of the augmented new system can be reduced to the old factorization $L^T L$ and an additional lower dimensional Cholesky factorization of a Schur complement block $A(\tilde{\mathcal{J}}_k,\tilde{\mathcal{J}}_k)$ in $A(\tilde{\mathcal{J}}_k \cup \mathcal{J}_{\text{new}}, \tilde{\mathcal{J}}_k \cup \mathcal{J}_{\text{new}})$ to obtain

$$\begin{aligned}
\tilde{L}_1 \quad &= \quad \text{chol}\left[A(\mathcal{J}_{\text{new}},\mathcal{J}_{\text{new}}) - A(\tilde{\mathcal{J}}_k,\mathcal{J}_{\text{new}})^T P^T L^{-1} L^{-T} PA(\tilde{\mathcal{J}}_k,\mathcal{J}_{\text{new}})\right] \\
&\overset{(3.51)}{=} \quad \text{chol}\left[S(\mathcal{J}_{\text{new}},\mathcal{J}_{\text{new}})\right] \in \mathbb{R}^{\tilde{q}_k \times \tilde{q}_k}.
\end{aligned} \tag{3.53}$$

Note that if $\mathcal{J}_{\text{new}}$ contains only one index j, (3.53) reduces to $\tilde{L}_1 = \sqrt{s_{jj}}$.

3.2.4. Theoretical Properties of FS(P)AI

Following our joint work with Bräckle in [30], we introduce properties of FS(P)AI. Consider L to be an FSAI solution for a given pattern $\mathcal{J}(L)$ and $\mathcal{J}_k = \tilde{\mathcal{J}}_k \cup \{k\}$.

Lemma 3.1 *Based on $\mathcal{J}_k$, the FSAI solution L_k satisfies*

$$A(\mathcal{J}_k,\mathcal{J}_k)L_k(\mathcal{J}_k) = \begin{pmatrix} s_{kk}l_{kk} \\ 0 \end{pmatrix} \in \mathbb{R}^{|\mathcal{J}_k|}.$$

Proof

$$\begin{aligned} A(\mathcal{J}_k,\mathcal{J}_k)L_k(\mathcal{J}_k) &= \begin{pmatrix} a_{kk} & A_k(\tilde{\mathcal{J}}_k)^T \\ A_k(\tilde{\mathcal{J}}_k) & A(\tilde{\mathcal{J}}_k,\tilde{\mathcal{J}}_k) \end{pmatrix} \begin{pmatrix} l_{kk} \\ L_k(\tilde{\mathcal{J}}_k) \end{pmatrix} \\ &\overset{(3.43)}{=} \begin{pmatrix} a_{kk} & A_k(\tilde{\mathcal{J}}_k)^T \\ A_k(\tilde{\mathcal{J}}_k) & A(\tilde{\mathcal{J}}_k,\tilde{\mathcal{J}}_k) \end{pmatrix} \begin{pmatrix} l_{kk} \\ -l_{kk}A(\tilde{\mathcal{J}}_k,\tilde{\mathcal{J}}_k)^{-1}A_k(\tilde{\mathcal{J}}_k) \end{pmatrix} \\ &= \begin{pmatrix} l_{kk}\left(a_{kk} - A_k(\tilde{\mathcal{J}}_k)^T A(\tilde{\mathcal{J}}_k,\tilde{\mathcal{J}}_k)^{-1}A_k(\tilde{\mathcal{J}}_k)\right) \\ l_{kk}A_k(\tilde{\mathcal{J}}_k) - l_{kk}A_k(\tilde{\mathcal{J}}_k) \end{pmatrix} = \begin{pmatrix} s_{kk}l_{kk} \\ 0 \end{pmatrix}. \end{aligned}$$

■

Theorem 3.6 *The column L_k of an FSAI solution with $\mathcal{J}(L_k) = \mathcal{J}_k$ is normalized with respect to the weighted Euclidean (energy) norm, i.e.,*

$$\|L_k\|_A = \sqrt{L_k^T A L_k} = 1.$$

Proof

$$L_k^T A L_k = L_k(\mathcal{J}_k)^T A(\mathcal{J}_k,\mathcal{J}_k)L_k(\mathcal{J}_k) \overset{\text{Lemma 3.1}}{=} L_k(\mathcal{J}_k)^T \begin{pmatrix} s_{kk}l_{kk} \\ 0 \end{pmatrix} = l_{kk}s_{kk}l_{kk} \overset{(3.42)}{=} 1.$$

Hence, we obtain $\|L_k\|_A = \sqrt{L_k^T A L_k} = 1$. ■

Corollary 3.1 *For every FSAI solution L holds $\frac{1}{n}\operatorname{trace}(L^T AL) = 1$.*

Proof

$$\frac{1}{n}\operatorname{trace}(L^T AL) = \frac{1}{n}\sum_{k=1}^{n} L_k^T A L_k \overset{\text{Theorem 3.6}}{=} \frac{1}{n}\sum_{k=1}^{n} 1 = 1.$$

■

Theorem 3.7 *If all $\tau_{jk} = 0$, the corresponding column L_k is the exact inverse of the k^{th} column of the Cholesky factor of A, i.e., if*

$$\forall k : \tau_{jk} = 0 \quad \textit{for all} \quad j \in \{k+1,\ldots,n\} \setminus \tilde{\mathcal{J}}_k, \qquad \textit{then} \qquad L^T AL = I.$$

Proof Let $\tilde{\mathcal{I}}_k = \{k{+}1,\ldots,n\}$. Following Theorem 3.6, $(L^T AL)_{kk} = 1$. For the non-diagonal components $L_i^T A L_k$ we assume $i > k$ without loss of generality. Therefore, $\mathcal{J}_i \subset \tilde{\mathcal{I}}_k$. Based on the assumption, we observe

$$\forall j \in \tilde{\mathcal{I}}_k \setminus \tilde{\mathcal{J}}_k : \tau_{jk} = 0 \quad \Leftrightarrow \quad A_j^T L_k = 0 \quad \overset{\text{Lemma 3.1}}{\Rightarrow} \quad A(\tilde{\mathcal{I}}_k,.)L_k = 0.$$

With $\mathcal{J}_i \subset \tilde{\mathcal{I}}_k$, the equality $L_i^T A L_k = L_i(\tilde{\mathcal{I}}_k)^T A(\tilde{\mathcal{I}}_k,.)L_k = 0$ is valid, verifying that all non-diagonal elements are zero. ■

Corollary 3.2 *For a full sparsity pattern $\mathcal{J}$ the FSAI solution L is the exact inverse of the Cholesky factor of A, i.e., $L^T AL = I$.*

Proof With $\{k+1,\ldots,n\} \setminus \tilde{\mathcal{J}}_k = \emptyset$ this follows immediately from Theorem 3.7. ■

Corollary 3.3 *For $\varepsilon_{FSPAI} = 0$, $\alpha \geq n$, and $\beta \geq 1$, FSPAI will compute the exact inverse $L = L_A^{-1}$.*

Proof Without any stopping criterion, FSPAI will augment $\mathcal{J}_k$ with indices j unless all $\tau_{jk} = 0$. Now we apply Theorem 3.7. ■

3.2.5. Exact FSPAI: EFSPAI

Even for Jacobi scaled SPD A it is possible that the K-condition number of the preconditioned system $L^T AL$ is unbounded. This results from the fact that although for a column k of the computed FSPAI L the termination condition $\tau_{jk} < \varepsilon_{\text{FSPAI}}$ is satisfied $\forall j > k$, its corresponding K-condition value can still be arbitrarily large. We illustrate this by the following

Example 3.1 Consider the two SPD matrices

$$A_1 = L_{A_1}^T L_{A_1} \in \mathbb{R}^{3\times 3} \quad \text{with} \quad L_{A_1} := \begin{pmatrix} \varepsilon_{\text{FSPAI}} & 0 & 0 \\ \sqrt{\varepsilon_{\text{FSPAI}}} & \sqrt{\varepsilon_{\text{FSPAI}}} & 0 \\ \sqrt{1-\varepsilon_{\text{FSPAI}}^2-\varepsilon_{\text{FSPAI}}} & \sqrt{1-\varepsilon_{\text{FSPAI}}} & 1 \end{pmatrix} \quad \text{and}$$

$$A_2 = L_{A_2}^T L_{A_2} \in \mathbb{R}^{3\times 3} \quad \text{with} \quad L_{A_2} := \begin{pmatrix} \varepsilon_{\text{FSPAI}} & 0 & 0 \\ \sqrt{1-2\varepsilon_{\text{FSPAI}}^2} & \sqrt{\varepsilon_{\text{FSPAI}}} & 0 \\ \varepsilon_{\text{FSPAI}} & \sqrt{1-\varepsilon_{\text{FSPAI}}} & 1 \end{pmatrix}.$$

Imposing the initial sparsity pattern $\mathcal{J}_1 = \{1,3\}$, $\mathcal{J}_2 = \{2,3\}$, and $\mathcal{J}_3 = \{3\}$, we compute the FSAI L_1 and L_2 corresponding to A_1 and A_2, respectively. For the possible candidate $j = 2$ in the first column, we observe

$$\tau_{2,1} = \frac{\varepsilon_{\text{FSPAI}}^2}{\varepsilon_{\text{FSPAI}}^2 + \varepsilon_{\text{FSPAI}}} < \varepsilon_{\text{FSPAI}} \quad \text{for } L_1 \text{ and} \quad \tau_{2,1} = \frac{\varepsilon_{\text{FSPAI}}(1-2\varepsilon_{\text{FSPAI}}^2)}{1-\varepsilon_{\text{FSPAI}}^2} < \varepsilon_{\text{FSPAI}} \quad \text{for } L_2.$$

In the limiting case, for both preconditioners holds $\lim_{\varepsilon_{\text{FSPAI}}\to 0} \tau_{2,1} = 0$, although the K-condition number is unbounded by

$$\lim_{\varepsilon_{\text{FSPAI}}\to 0} K(L_1^T A_1 L_1) = \lim_{\varepsilon_{\text{FSPAI}}\to 0} \left(1 + \frac{1}{\varepsilon_{\text{FSPAI}}}\right)^{\frac{1}{3}} = \infty \qquad \text{and}$$

$$\lim_{\varepsilon_{\text{FSPAI}}\to 0} K(L_2^T A_2 L_2) = \lim_{\varepsilon_{\text{FSPAI}}\to 0} \left(\frac{1}{\varepsilon_{\text{FSPAI}}^2} - 1\right)^{\frac{1}{3}} = \infty.$$

Hence, it may happen that FSPAI will stop with an arbitrarily poor, useless, or even spoiling preconditioner even though the current K-condition number is very large and/or it would be possible to further improve the approximation by augmenting an index. Note that this is no contradiction to the bound in (3.48), i.e., K_{old} must have been arbitrarily large as well

in the previous step. Consequently, we are interested in the exact reduction factor of the K-condition number when augmenting $\tilde{\mathcal{J}}_k$ with a new index j, i.e., a metric which avoids unbounded K-condition numbers. Our derivation is based on the joint work with Bräckle in [30]. For a comparison between K_{old} and K_{new} we only have to consider the diagonal elements l_{kk}: the numerator of (3.36) remains constant according to Corollary 3.1 and the denominator only depends on l_{kk}.

For simplification we consider the case when augmenting $\tilde{\mathcal{J}}_k$ with only one single index j. An augmentation with several indices is closely related to the *exact block version of FSPAI* (EBFSPAI) provided in Appendix C. Consider the definitions

$$\tilde{\mathcal{J}}_k^{+\mathrm{j}} := \tilde{\mathcal{J}}_k \cup \{j\} \qquad \text{and} \qquad \mathcal{I}_k^{-\mathrm{j}} := \mathcal{I}_k \setminus \{j\} \qquad \text{with} \qquad \mathcal{I}_k = \{k,\dots,n\} \setminus \tilde{\mathcal{J}}_k,$$

and the abbreviations S_{old} which denotes the Schur complement of $A(\tilde{\mathcal{J}}_k,\tilde{\mathcal{J}}_k)$ in $A_{k:n,k:n}$ while S_{new} is the Schur complement of $A(\tilde{\mathcal{J}}_k^{+\mathrm{j}},\tilde{\mathcal{J}}_k^{+\mathrm{j}})$ in $A_{k:n,k:n}$. Hence, by augmenting index j, the new Schur complement reads as

$$S_{\text{new}} = A(\mathcal{I}_k^{-\mathrm{j}},\mathcal{I}_k^{-\mathrm{j}}) - A(\tilde{\mathcal{J}}_k^{+\mathrm{j}},\mathcal{I}_k^{-\mathrm{j}})^T A(\tilde{\mathcal{J}}_k^{+\mathrm{j}},\tilde{\mathcal{J}}_k^{+\mathrm{j}})^{-1} A(\tilde{\mathcal{J}}_k^{+\mathrm{j}},\mathcal{I}_k^{-\mathrm{j}}).$$

Using Cholesky updates according to (3.52), we can compute S_{new} by incorporating the factorization L^TL for the original set $\tilde{\mathcal{J}}_k$. This leads to

$$\begin{aligned} S_{\text{new}} &= A(\mathcal{I}_k^{-\mathrm{j}},\mathcal{I}_k^{-\mathrm{j}}) - A(\tilde{\mathcal{J}}_k^{+\mathrm{j}},\mathcal{I}_k^{-\mathrm{j}})^T \left(\tilde{P}^T\tilde{L}^T\tilde{L}\tilde{P}\right)^{-1} A(\tilde{\mathcal{J}}_k^{+\mathrm{j}},\mathcal{I}_k^{-\mathrm{j}}) \\ &= A(\mathcal{I}_k^{-\mathrm{j}},\mathcal{I}_k^{-\mathrm{j}}) - Y^TY, \end{aligned} \tag{3.54}$$

where $Y = \tilde{L}^{-T}\tilde{P}A(\tilde{\mathcal{J}}_k^{+\mathrm{j}},\mathcal{I}_k^{-\mathrm{j}})$. With the known Cholesky factor L in $\tilde{L}$, we solve $\tilde{L}^TY = \tilde{P}A(\tilde{\mathcal{J}}_k^{+\mathrm{j}},\mathcal{I}_k^{-\mathrm{j}})$ to obtain Y:

$$\begin{pmatrix} \sqrt{(s_{\text{old}})_{jj}} & A_j(\tilde{\mathcal{J}}_k)^TL^{-1} \\ 0 & L^T \end{pmatrix} \begin{pmatrix} Y_1 \\ Y_2 \end{pmatrix} = \begin{pmatrix} A(j,\mathcal{I}_k^{-\mathrm{j}}) \\ A(\tilde{\mathcal{J}}_k,\mathcal{I}_k^{-\mathrm{j}}) \end{pmatrix}$$

$$\Leftrightarrow \quad Y_2 = L^{-T}A(\tilde{\mathcal{J}}_k,\mathcal{I}_k^{-\mathrm{j}}) \qquad \text{and} \tag{3.55}$$

$$Y_1 = \frac{A(j,\mathcal{I}_k^{-\mathrm{j}}) - A_j(\tilde{\mathcal{J}}_k)^TL^{-1}L^TA(\tilde{\mathcal{J}}_k,\mathcal{I}_k^{-\mathrm{j}})}{\sqrt{(s_{\text{old}})_{jj}}} = \frac{S_{\text{old}}(j,\mathcal{I}_k^{-\mathrm{j}})}{\sqrt{(s_{\text{old}})_{jj}}}. \tag{3.56}$$

Therewith, we can compute the new Schur complement using the old Schur complement as

$$\begin{aligned} S_{\text{new}} &\overset{(3.54)}{=} A(\mathcal{I}_k^{-\mathrm{j}},\mathcal{I}_k^{-\mathrm{j}}) - Y_1^TY_1 + Y_2^TY_2 \\ &\overset{(3.55)\,(3.56)}{=} A(\mathcal{I}_k^{-\mathrm{j}},\mathcal{I}_k^{-\mathrm{j}}) - A(\tilde{\mathcal{J}}_k,\mathcal{I}_k^{-\mathrm{j}})^T \underbrace{L^{-1}L^{-T}}_{A(\tilde{\mathcal{J}}_k,\tilde{\mathcal{J}}_k)^{-1}} A(\tilde{\mathcal{J}}_k,\mathcal{I}_k^{-\mathrm{j}}) - \frac{S_{\text{old}}(j,\mathcal{I}_k^{-\mathrm{j}})^TS_{\text{old}}(j,\mathcal{I}_k^{-\mathrm{j}})}{(s_{\text{old}})_{jj}} \\ &\overset{\text{Def. 3.3}}{=} S_{\text{old}}(\mathcal{I}_k^{-\mathrm{j}},\mathcal{I}_k^{-\mathrm{j}}) - S_{\text{old}}(j,\mathcal{I}_k^{-\mathrm{j}})^T(s_{\text{old}})_{jj}^{-1}S_{\text{old}}(j,\mathcal{I}_k^{-\mathrm{j}}). \end{aligned} \tag{3.57}$$

The relationship between K_{old} and K_{new}—corresponding to $\tilde{\mathcal{J}}_k$ and $\tilde{\mathcal{J}}_k^{+\mathrm{j}}$, respectively—can be described by

$$K_{\text{new}} \overset{\text{Corollary 3.1}}{=} \frac{1}{\det(L_{\text{new}}^TAL_{\text{new}})^{\frac{1}{n}}} \overset{(3.42)}{=} \frac{(s_{\text{new}})_{kk}^{\frac{1}{n}}}{\det(A)^{\frac{1}{n}}\prod_{\substack{j=1\\j\neq k}}^{n}(l_{\text{old}})_{jj}^{\frac{2}{n}}}, \tag{3.58}$$

because $(l_{\text{old}})_{jj} = (l_{\text{new}})_{jj}$ for $j \neq k$. Following (3.57), the diagonal components of the new Schur complement satisfy

$$\begin{aligned}(s_{\text{new}})_{kk} &= (s_{\text{old}})_{kk} - (s_{\text{old}})_{jk}^T (s_{\text{old}})_{jj}^{-1} (s_{\text{old}})_{jk} = (s_{\text{old}})_{kk} \left(1 - \frac{(s_{\text{old}})_{jk}^2}{(s_{\text{old}})_{jj}(s_{\text{old}})_{kk}}\right) \\ &= (s_{\text{old}})_{kk}(1-\mu_{jk}).\end{aligned} \tag{3.59}$$

Hence, for (3.58) we obtain the relationship

$$K_{\text{new}} = \frac{(s_{\text{old}})_{kk}^{\frac{1}{n}}(1-\mu_{jk})^{\frac{1}{n}}}{\det(A)^{\frac{1}{n}} \prod_{\substack{j=1 \\ j\neq k}}^{n} (l_{\text{old}})_{jj}^{\frac{2}{n}}} = (1-\mu_{jk})^{\frac{1}{n}} K_{\text{old}} \qquad \text{with} \qquad \mu_{jk} := \frac{(s_{\text{old}})_{jk}^2}{(s_{\text{old}})_{jj}(s_{\text{old}})_{kk}}$$

being the exact reduction factor when adding an index j to the sparsity pattern $\mathcal{J}_k$. It corresponds to the heuristic factor τ_{jk} in (3.48).

Comparing τ_{jk} with μ_{jk}

Interested in a brief comparison between the two metrics we state the

Lemma 3.2 *The heuristic reduction factor τ_{jk} in (3.48) can be written as*

$$\tau_{jk} = \frac{(s_{old})_{jk}^2}{a_{jj}(s_{old})_{kk}}.$$

Proof

$$\begin{aligned}\tau_{jk} &= \frac{\left[A_j(\mathcal{J}_k)^T L_k(\mathcal{J}_k)\right]^2}{a_{jj}} = \frac{1}{a_{jj}} \left[\begin{pmatrix} a_{kj} \\ A_j(\tilde{\mathcal{J}}_k) \end{pmatrix}^T \begin{pmatrix} l_{kk} \\ -l_{kk} A(\tilde{\mathcal{J}}_k, \tilde{\mathcal{J}}_k)^{-1} A_k(\tilde{\mathcal{J}}_k) \end{pmatrix} \right]^2 \\ &= \frac{l_{kk}^2}{a_{jj}} \left[a_{jk} - A_j(\tilde{\mathcal{J}}_k)^T A(\tilde{\mathcal{J}}_k, \tilde{\mathcal{J}}_k)^{-1} A_k(\tilde{\mathcal{J}}_k)\right]^2 = \frac{(s_{\text{old}})_{jk}^2}{a_{jj}(s_{\text{old}})_{kk}}.\end{aligned}$$

■

We obtain the relationship between both metrics in

Theorem 3.8 *The heuristic reduction factor τ_{jk} and the exact reduction factor μ_{jk} satisfy the relation*

$$0 \leq \tau_{jk} \leq \mu_{jk} < 1.$$

Proof The first inequality is trivial as A is positive definite. The second one is a consequence of

$$(s_{\text{old}})_{jj} \leq a_{jj} - A_j(\tilde{\mathcal{J}}_k)^T A(\tilde{\mathcal{J}}_k, \tilde{\mathcal{J}}_k)^{-1} A_j(\tilde{\mathcal{J}}_k) \overset{A(\tilde{\mathcal{J}}_k, \tilde{\mathcal{J}}_k)^{-1} \text{ is pos. def.}}{\leq} a_{jj}.$$

The last inequality results from (A.6) as S_{old} is SPD. ■

Corollary 3.4 *The heuristic reduction factor τ_{jk} and the exact reduction factor μ_{jk} induce the following relationship between the K-condition numbers:*

$$K_{new} = (1-\mu_{jk})^{\frac{1}{n}} K_{old} \;\leq\; (1-\tau_{jk})^{\frac{1}{n}} K_{old} \;\leq\; \left(1 - \frac{\tau_{jk}}{n}\right) K_{old}.$$

Proof The first inequality is trivial with Theorem 3.8. We obtain the second inequality applying Bernoulli's inequality (A.5):

$$\left(1 - \frac{\tau_{jk}}{n}\right)^n \geq 1 - \tau_{jk} \quad \Leftrightarrow \quad 1 - \frac{\tau_{jk}}{n} \geq (1 - \tau_{jk})^{\frac{1}{n}}. \qquad \blacksquare$$

3.2.6. Notes on Implementing EFSPAI

In this part we use the former Schur complement notation specified in Definition 3.3, i.e., with S and s we refer to S_{old} and s_{old}, respectively. Applying an inverse formulation of Lemma 3.2, we can write μ_{jk} in the form

$$\mu_{jk} = \frac{[A_j(\mathcal{J}_k)^T L_k(\mathcal{J}_k)]^2}{s_{jj}} = \frac{[A_j(\mathcal{J}_k)^T L_k(\mathcal{J}_k)]^2}{a_{jj} - A_j(\tilde{\mathcal{J}}_k)^T A(\tilde{\mathcal{J}}_k,\tilde{\mathcal{J}}_k)^{-1} A_j(\tilde{\mathcal{J}}_k)}. \tag{3.60}$$

Similar to FSPAI, we have to compute (3.60) only for those indices j which are contained in the set $\hat{\mathcal{J}}_k$ (3.49). However, replacing the τ_{jk} with (3.60) within line 19 of Algorithm 8 would require to solve a linear system

$$A(\tilde{\mathcal{J}}_k,\tilde{\mathcal{J}}_k) y_j = A_j(\tilde{\mathcal{J}}_k) \qquad \text{for each} \qquad j \in \hat{\mathcal{J}}_k. \tag{3.61}$$

Consequently, by incorporating μ_{jk} and (3.61) within the pattern updates—in order to obtain optimal improvement for L—comes along with higher costs compared to FSPAI's heuristic factor τ_{jk}. Therefore, a comparison between FSPAI and EFSPAI in terms of setup time is difficult, if not impossible. However, it is possible to reduce the setup costs for an EFSPAI avoiding (3.61) for the most part. Considering l to be the optimal index which was currently added to $\mathcal{J}_k$, we can update the Schur complement components of μ_{jk} such that the complexity of its evaluation during the next pattern update is reduced. We observe the following possibilities:

1. We can incorporate the Schur complement update according to (3.57) and (3.59), and update all s_{jj} via

$$s_{jj} = s_{jj} - \frac{s_{jl}^2}{s_{ll}}, \tag{3.62}$$

   taking symmetry into account. In a sequential environment we can initialize $s = \text{diag}(A)$ implying that for the update of (3.62) we only have to compute s_{jl} for all $j \in \hat{\mathcal{J}}_k$. However, the linear system $y_l = A(\tilde{\mathcal{J}}_k,\tilde{\mathcal{J}}_k)^{-1} A_l(\tilde{\mathcal{J}}_k)$ is only to be solved once in order to obtain all $s_{jl} = a_{jl} - A_j(\tilde{\mathcal{J}}_k)^T y_l$.

2. By computing the exact reduction factor via $\mu_{jk} = \frac{s_{jk}^2}{s_{jj} s_{kk}}$, it is possible both to update s_{jk} via

$$s_{jk} = s_{jk} - \frac{s_{jl} s_{lk}}{s_{ll}}, \tag{3.63}$$

   and s_{jj} according to (3.62). We summarize this version of EFSPAI in Algorithm 9. See also Figure 3.4 for an illustration of the update.

   The computation of the μ_{jk} based on (3.60)—or via updates of s_{jj} (3.62) and s_{jk} (3.63)—does not affect the inherent parallelism inherited by FSPAI. Still, each column of an EFSPAI can be constructed independently from all others. Merely the

pattern candidates are chosen according to μ_{jk} instead of τ_{jk}. Moreover, no additional communication between the processors becomes necessary. In order to understand this, let us compare μ_{jk} in (3.60) and τ_{jk} in proof of Lemma 3.2. We observe the single difference s_{jj} instead of a_{jj} in the denominator. However, both reduction factors depend on values contained in A_j where $j \in \hat{\mathcal{J}}_k$. Hence, a prefetching of the columns A_j in the current update step—to update the Schur complement components in Algorithm 9—instead of in the next update step—performed by FSPAI—implies no additional communication. The values in A_l required by s_{jl} and s_{ll}, respectively, are already available because μ_{lk}—corresponding to the augmented index l—has already been computed. In case that an s_{ij} component is not available, i.e., it is accessed for the first time, the corresponding matrix value a_{ij} is to be used instead. Therefore, we can either use an if-statement in the Schur update or initialize the s_{ij} components with the corresponding values a_{ij}. Although the latter consumes more memory, we use it in Algorithm 9 to simplify our notation. See S_{diag} and S_{col}. In any Schur update case, EFSPAI requires more memory compared to FSPAI in order to store and update its s_{ij} components.

3. To fully circumvent the solution of any linear system in (3.61), we can update all components of the Schur complement at once by

$$S(\hat{\mathcal{J}}_k^{+\mathrm{k}},\hat{\mathcal{J}}_k^{+\mathrm{k}}) = S(\hat{\mathcal{J}}_k^{+\mathrm{k}},\hat{\mathcal{J}}_k^{+\mathrm{k}}) - S_l(\hat{\mathcal{J}}_k^{+\mathrm{k}})^T s_{ll}^{-1} S_l(\hat{\mathcal{J}}_k^{+\mathrm{k}}) \tag{3.64}$$

where $\hat{\mathcal{J}}_k^{+\mathrm{k}} := \hat{\mathcal{J}}_k \cup \{k\}$. See Figure 3.4 for an exemplary illustration. This global Schur update preserves the parallelism of the second approach for the same reasons as mentioned above.

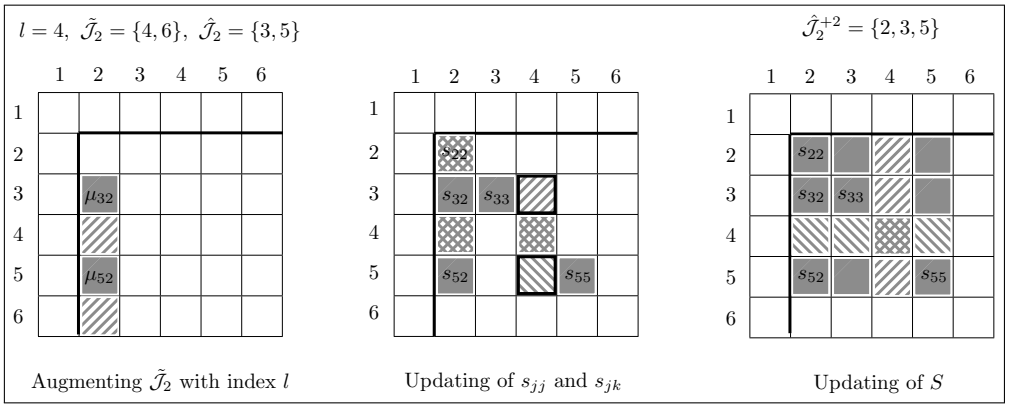


Figure 3.4.: Abstract example for updating s_{jj}, s_{jk}, and S (3.64) by augmenting the second column $\tilde{\mathcal{J}}_2 = \{6\}$ with index $l = 4$. Potential candidates are $\hat{\mathcal{J}}_2 = \{3{,}5\}$. The entries $s_{*,*}$ illustrate which components will be required for the computation of $\mu_{3,2}$ and $\mu_{5,2}$. The solid regions mark the entries of s and S, respectively, which require to be updated in order to be available for $\mu_{3,2}$ and $\mu_{5,2}$ in the next pattern update. The hatched (and crossed) regions illustrate the components which are additionally required to update the solid components. In the middle figure, the black bordered elements are s components which are necessary for the solid regions but have to be computed via solving a linear system according to (3.61) in line 26 of Algorithm 9.

Algorithm 9: Exact FSPAI with single index update per step.

Input : $A \in \mathbb{R}^{n\times n}$, $\mathcal{J} \in \mathbb{B}^{n\times n}$, $\varepsilon_{\text{EFSPAI}} \geq 0$, $\alpha \geq 0$
Output: Preconditioner $L \approx L_A^{-1}$, $L \in \mathbb{R}^{n\times n}$

for $k \leftarrow 1$ **to** n **do**
  $e_k \leftarrow I(.,k)$
  $\mathcal{J}_k \leftarrow \mathcal{J}(.,k)$
  $S_{\texttt{diag}} \leftarrow \text{diag}(A)$
  $S_{\texttt{col}} \leftarrow A_k$
  for $\texttt{step} \leftarrow 0$ **to** α **do**
    $\tilde{\mathcal{J}}_k \leftarrow \mathcal{J}_k \setminus \{k\}$
    if $\tilde{\mathcal{J}}_k = \emptyset$ **then**
      $l_{kk} \leftarrow a_{kk}^{-\frac{1}{2}}$
    else
      $y_k \leftarrow A(\tilde{\mathcal{J}}_k,\tilde{\mathcal{J}}_k)^{-1} A_k(\tilde{\mathcal{J}}_k)$
      $l_{kk} \leftarrow \left(a_{kk} - A_k(\tilde{\mathcal{J}}_k)^T y_k\right)^{-\frac{1}{2}}$
      $L_k(\tilde{\mathcal{J}}_k) \leftarrow -l_{kk} y_k$
    end
    if $\texttt{step} = \alpha$ **then**
      break
    end
    $\hat{\mathcal{J}}_k \leftarrow \{j : \ j > k \ \wedge \ A_j(\mathcal{J}_k)^T L_k(\mathcal{J}_k) \neq 0\} \setminus \mathcal{J}_k$
    foreach $j \in \hat{\mathcal{J}}_k$ **do**
      $\mu_{jk} \leftarrow \dfrac{(S_{\texttt{col}})_j^2}{(S_{\texttt{diag}})_j (S_{\texttt{diag}})_k}$
    end
    if $\max_{j\in\hat{\mathcal{J}}_k} \mu_{jk} \leq \varepsilon_{\texttt{EFSPAI}}$ **then**
      break
    end
    $l \leftarrow \arg\max_{j\in\hat{\mathcal{J}}_k} \mu_{jk}$
    $y_l \leftarrow A(\tilde{\mathcal{J}}_k,\tilde{\mathcal{J}}_k)^{-1} A_l(\tilde{\mathcal{J}}_k)$
    foreach $j \in \hat{\mathcal{J}}_k \cup \{k\}$ **do**
      $s_{jl} \leftarrow a_{jl} - A_j(\tilde{\mathcal{J}}_k)^T y_l$
      $(S_{\texttt{diag}})_j \leftarrow (S_{\texttt{diag}})_j - \dfrac{s_{jl}^2}{(S_{\texttt{diag}})_l}$
      $(S_{\texttt{col}})_j \leftarrow (S_{\texttt{col}})_j - \dfrac{s_{jl}(S_{\texttt{col}})_l}{(S_{\texttt{diag}})_l}$
    end
    $\mathcal{J}_k \leftarrow \mathcal{J}_k \cup \{l\}$
  end
end

Numerical Experiments

Our numerical results aim for a comparison between FSPAI and EFSPAI. We implement EFSPAI according to Algorithm 9 and adapt Algorithm 8 such that both MATLAB implementations coincide in data structures and algorithmic steps where possible. We use the test environment specified in Appendix D.4 and apply no dropping in L. In order to reduce falsification, we provide timings averaged from 5 runs of profiled codes on a system with minimized number of running processes, similar to our scenarios in Section 3.1.6. For the chosen SPD matrices from [44], CG does not converge in 5000 iterations for $b = (1,1,\ldots,1)^T$ and $\varepsilon_{\text{CG}} = 1\text{E}{-}09$. During all our experiments we add only one index per update step, i.e., we fix the setting to $\Upsilon_{\infty,1}$ and terminate the process when reaching $\varepsilon_{\text{(E)FSPAI}}$. Following Corollary 3.1, we additionally provide the variance ξ^2 of all eigenvalues of L^TAL around the expected value 1.

Numerical Experiment 3.4 We compare both algorithms with decreasing $\varepsilon_{\text{(E)FSPAI}}$ in Table 3.5. The additional costs of solving the linear system (3.61) in each step lead to an overall higher setup time for EFSPAI. Theorem 3.8 is also reflected in the results: the relation $\tau_{jk} \leq \mu_{jk}$ implies that EFSPAI will stop its update process regarding $\varepsilon_{\text{EFSPAI}}$ for some columns later than FSPAI. Hence, the resulting preconditioner has more nonzeros and is of higher spectral quality. As a consequence, PCG converges in less iterations. Note that for low $\varepsilon_{\text{(E)FSPAI}}$ the value of $\det(L^TAL)$ falls below the machine precision and is rounded to zero. Hence, we do not provide K-condition numbers in these cases. Regardless, the preconditioner L has full rank and can be applied in PCG. ●

Numerical Experiment 3.5 In Table 3.6 we impose a limit for the maximum number $p_{\max}$ of possible indices per column. Increasing $p_{\max}$ provokes more fill-in in EFSPAI. Similar to Numerical Experiment 3.4, the exact reduction factor leads to higher setup costs yielding a denser preconditioner of higher quality. ●

Interpretation of the results. Our results indicate that FSPAI's heuristic has a better cost-benefit ratio compared to EFSPAI. The additional costs required to construct an EFSPAI do not pay off in a total sense. For certain matrices such as those presented in Example 3.1, EFSPAI remains a robust preconditioner producing improved spectral properties for the preconditioned system. However, it is unlikely that such matrices arise in practice. Consequently, our results motivate for the usage of FSPAI as a parallel preconditioner for SPD systems.

Table 3.5.: Comparison of FSPAI and EFSPAI in Numerical Experiment 3.4 for matrices from [44]. The setting is chosen as $\Upsilon_{\infty,1}$ with varying $\varepsilon_{\text{(E)FSPAI}}$. No convergence is denoted by †. Timing results have unit seconds.

Algorithm	$\varepsilon_{\text{(E)FSPAI}}$	$\kappa_2(L^TAL)$	$K(L^TAL)$	ξ^2	$\|L_AL-I\|_F$	nnz(L)	t_{setup}	t_{PCG} (iter.)	t_{total}
EX33, $n = 1733$, $\kappa_2(A) = 7.02\text{E+}12$, $\|L_A - I\|_F = 1.39\text{E+}06$									
FSPAI	0.050	7.68E+09	–	0.400	3.00E+05	9,911	7.91	†	†
EFSPAI		9.36E+09	–	0.416	3.21E+05	9,401	10.58	†	†
FSPAI	0.010	3.60E+09	1.385	0.354	3.04E+05	13,477	11.65	0.89 (3813)	12.54
EFSPAI		3.01E+09	1.354	0.355	3.25E+05	13,679	17.27	0.52 (2200)	17.79
FSPAI	0.005	2.99E+09	1.359	0.336	3.04E+05	16,822	16.98	0.93 (3705)	17.91
EFSPAI		2.58E+09	1.330	0.338	3.25E+05	17,520	25.81	0.59 (2265)	26.40
FSPAI	0.001	5.61E+08	1.266	0.323	3.17E+05	29,388	42.95	0.75 (2400)	43.70
EFSPAI		3.37E+07	1.250	0.340	3.36E+05	31,836	68.99	0.56 (1709)	69.55
MSC01440, $n = 1440$, $\kappa_2(A) = 3.31\text{E+}06$, $\|L_A - I\|_F = 1.65\text{E+}04$									
FSPAI	0.40	1.25E+04	–	0.689	1.08E+02	1,477	0.91	0.18 (849)	1.09
EFSPAI		1.25E+04	–	0.689	1.08E+02	1,477	0.74	0.18 (849)	0.92
FSPAI	0.20	1.02E+04	–	0.522	1.08E+02	1,995	1.59	0.17 (772)	1.76
EFSPAI		1.05E+04	–	0.517	1.08E+02	2,015	1.91	0.17 (777)	2.08
FSPAI	0.10	3.38E+03	1.445	0.345	1.60E+02	3,577	3.88	0.11 (477)	3.99
EFSPAI		3.37E+03	1.435	0.332	1.60E+02	3,659	5.98	0.11 (470)	6.09
FSPAI	0.05	2.16E+03	1.356	0.265	2.72E+02	4,516	5.48	0.09 (370)	5.57
EFSPAI		2.16E+03	1.353	0.262	2.72E+02	4,568	8.52	0.09 (358)	8.61

Table 3.6.: Comparison of FSPAI and EFSPAI in Numerical Experiment 3.5 for matrices from [44]. The setting is chosen as $\Upsilon_{\infty,1}$ with varying $p_{\max}$. See the corresponding matrix header in the table for the value of $\varepsilon_{\text{(E)FSPAI}}$. Timing results have unit seconds.

Algorithm	$p_{\max}$	$\kappa_2(L^TAL)$	$K(L^TAL)$	ξ^2	$\|L_AL-I\|_F$	nnz(L)	t_{setup}	t_{PCG} (iter.)	t_{total}
BCSSTK11, $n=1473$, $\kappa_2(A)=2.21\text{E+}08$, $\|L_A-I\|_F=2.49\text{E+}05$, $\varepsilon_{\text{(E)FSPAI}}=0.1$									
FSPAI	2	3.93E+04	1.454	0.290	5.42E+02	2,188	0.72	0.15 (784)	0.87
EFSPAI		3.93E+04	1.454	0.290	5.42E+02	2,188	0.60	0.15 (784)	0.75
FSPAI	3	4.11E+04	1.440	0.281	5.44E+02	2,271	1.33	0.14 (775)	1.47
EFSPAI		3.74E+04	1.433	0.278	5.44E+02	2,317	1.49	0.14 (765)	1.63
FSPAI	4	3.88E+04	1.435	0.280	5.45E+02	2,307	1.41	0.15 (766)	1.56
EFSPAI		3.47E+04	1.428	0.276	5.46E+02	2,359	1.67	0.14 (754)	1.81
FSPAI	5	3.97E+04	1.434	0.280	5.45E+02	2,311	1.44	0.15 (768)	1.59
EFSPAI		3.53E+04	1.421	0.272	5.51E+02	2,395	1.74	0.14 (736)	1.88
MSC01050, $n=1050$, $\kappa_2(A)=4.54\text{E+}15$, $\|L_A-I\|_F=1.81\text{E+}04$, $\varepsilon_{\text{(E)FSPAI}}=0.01$									
FSPAI	5	4.03E+12	1.391	0.311	5.65E+01	3,882	3.33	0.24 (1446)	3.57
EFSPAI		2.85E+12	1.390	0.284	5.60E+01	3,914	5.08	0.27 (1538)	5.35
FSPAI	10	1.18E+10	1.195	0.146	1.81E+02	5,590	6.56	0.14 (813)	6.70
EFSPAI		1.07E+10	1.186	0.136	1.82E+02	5,938	11.33	0.11 (680)	11.44
FSPAI	15	5.34E+09	1.159	0.118	1.86E+02	6,114	8.00	0.09 (549)	8.09
EFSPAI		3.77E+09	1.138	0.097	1.90E+02	6,918	14.89	0.09 (477)	14.98
FSPAI	20	4.57E+09	1.154	0.113	1.87E+02	6,170	8.15	0.09 (511)	8.24
EFSPAI		2.31E+09	1.122	0.085	1.95E+02	7,250	16.45	0.06 (357)	16.51
FSPAI	25	4.57E+09	1.154	0.113	1.87E+02	6,170	8.15	0.09 (511)	8.24
EFSPAI		1.88E+09	1.115	0.079	2.04E+02	7,338	16.81	0.06 (327)	16.87

Chapter 4

Block Factorized Sparse Approximate Inverses based on K-condition Number Minimization

Block algorithms are typically used to improve the performance of dense matrix computations. Here, the application of BLAS 3 [26, 47] routines usually leads to an efficient cache utilization yielding computational speedup [16]. The situation differs for operations on sparse (approximate inverse) matrices. Indirect addressing which occurs during matrix-vector products dampens the gained speedup depending on the (block) structure of the matrix. However, also for sparse problems with inherent block structure, block algorithms are an attractive alternative to their pointwise counterparts. As a blockwise consideration may better fit the underlying physical problem, one can expect to gain robustness, more qualitative behavior, and a more efficient cache recovery. Still, the latter can come along with a faster computation using BLAS 3 routines, depending on the number and density of the blocks in the matrix. If such natural block structure resulting from, e.g., finite element discretizations of PDEs, is not available in advance, it is possible to impose it artificially by reordering algorithms like reverse Cuthill-McKee [40].

Almost every preconditioning approach has been generalized to its block variant. Among these we can find, e.g., the *Block SPAI* (BSPAI) [12], the *Block ILU* and *Block ILUM* [125, 145], or the *Block SAINV* (BSAINV) [18]. The effect of using block preconditioners can differ across the methods [16] and sometimes also across the structure and/or density of the system matrices. For instance, a BSPAI can be computed in less time compared to SPAI but its application usually yields slower convergence rates of the iterative solver [12]. However, the cost-benefit ratio of the setup and solver time normally is better for the block variant [84]. For matrices with natural block structure, BSAINV produces preconditioners of higher quality—in terms of convergence—compared to SAINV [18]. So far, there is no arbitrary scalable parallelization approach for BSAINV.

Recently, some research has been published concerning block factorized sparse approximate inverses based on Frobenius norm minimization. A combination of *block FSAI* (BFSAI) and ILU—denoted as BFSAI-ILU by Janna et al. [98]—shows to be superior to FSAI for smaller block sizes ($b \leq 8$) and coincides with FSAI in the case $b = 1$. Janna and Ferronato also proposed a hybrid approach consisting of an adaptive block FSAI and an incomplete Cholesky preconditioner denoted as ABF-IC [97]. The ABF automatically captures significant terms of higher powers of A without explicitly computing all its entries and shows to be able to outperform BFSAI in terms of total time. However, massively parallel experiments concerning the scalability beyond 32 processors remain of interest.

In this chapter we focus on a block version of the FSPAI preconditioner (BFSPAI) for SPD matrices having a block structure. After briefly describing the BSPAI for the general case, we show that the K-condition number minimized derivation of BFSPAI resembles the pointwise version of FSPAI. Hence, closely related to FSPAI, the resulting preconditioner preserves the advantages of inherent parallelism, stability, and adaptive capturing of promising blocks in the block pattern of the inverse. Note that this also means that it inherits the disadvantage of being computationally more expensive than other block preconditioners in a sequential environment. The chapter is based on our joint work with Bräckle [30].

As we deal with block sparsity structures we transfer our pointwise notation to the block case. We consider a block structure to be covered with full blocks while still being sparse, i.e., only few full blocks are contained in each block column/row; see Figure 4.1 (c) for an exemplary block sparsity pattern. For the rest of the chapter all indices are considered to be block indices and for simplicity we assume that n is an integer multiple of b. Hence, we denote the block columns $k \in \{1,\dots,n_b\}$ of M, L, A, and I by M_k, L_k, A_k, and I_k, with b being the block size and $n_b := \frac{n}{b}$ the number of block columns. The set of allowed block entries in the k^{th} block column is denoted as $\mathcal{J}_k$ while L_{jk} is the j^{th} block in the k^{th} block column L_k; the k^{th} block of I_k is the identity matrix I_b.

4.1. The General Case: Block SPAI

The block version of SPAI (BSPAI) [12] is similar to the pointwise SPAI. Again, it is possible to decouple the global problem into n_b minimization problems

$$\min_{\mathcal{J}(M)\in\mathcal{J}} \|AM - I\|_F^2 = \sum_{k=1}^{n_b} \min_{\mathcal{J}(M_k)\in\mathcal{J}_k} \|AM_k - I_k\|_F^2. \tag{4.1}$$

Based on the imposed sparse block pattern, the reduced LS problems in (4.1) can be solved via QR factorization, like in SPAI. The adaptive strategy using pattern updates can be transferred in order to capture profitable blocks in the approximate inverse. The set of potential new blocks is found according to (3.12) via

$$\tilde{\mathcal{J}}_k := \bigcup_{\ell\in\mathcal{L}_k} \mathcal{N}_\ell \qquad \text{with} \qquad \mathcal{L}_k := \{\ell \in \{1,\dots,n_b\} : R_{k\ell} \neq 0\} \cup \{k\}$$

and $\mathcal{N}_\ell := \{j : A_{\ell j} \neq 0\}$, where $R_k = AM_k - I_k$. We obtain the reduction factor of each block $j \in \tilde{\mathcal{J}}_k$ by solving the minimization problem

$$\min_{M_{jk}} \|A(M_k + I_j M_{jk}) - I_k\|_F \;=\; \min_{M_{jk}} \|R_k + AI_j M_{jk}\|_F. \tag{4.2}$$

With the expanded form

$$\|R_k + A_j M_{jk}\|_F^2 = \operatorname{trace}(R_k^T R_k) + 2\operatorname{trace}(M_{jk}^T A_j^T R_k) + \operatorname{trace}(M_{jk}^T A_j^T A_j M_{jk}) \tag{4.3}$$

we obtain the solution of (4.2) by setting the derivative of (4.3) with respect to M_{jk} equal zero:

$$\frac{\partial}{\partial M_{jk}} \left[\operatorname{trace}(R_k^T R_k) + 2\operatorname{trace}(M_{jk}^T A_j^T R_k) + \operatorname{trace}(M_{jk}^T A_j^T A_j M_{jk})\right]$$

$$\begin{aligned} &\overset{(A.1)}{(A.2)} && 2A_j^T R_k + A_j^T A_j M_{jk} + (A_j^T A_j)^T M_{jk} = 2A_j^T R_k + 2A_j^T A_j M_{jk} \overset{!}{=} 0 \\ &\Leftrightarrow && M_{jk} = -(A_j^T A_j)^{-1} A_j^T R_k. \end{aligned} \tag{4.4}$$

Due to the convexity of norm minimization the solution (4.4) is a minimum. In order to update the block pattern with profitable blocks, we have to compute the reduction factor τ_{jk} and the corresponding norm ρ_{jk} of the new residual similar to (3.15). Using (4.4) within (4.3), we get

$$\begin{aligned} \rho_{jk}^2 &= \|R_k + A_j M_{jk}\|_F^2 = \operatorname{trace}(R_k^T R_k) + 2\operatorname{trace}(M_{jk}^T A_j^T R_k) + \operatorname{trace}(M_{jk}^T A_j^T A_j M_{jk}) \\ &= \|R_k\|_F^2 + 2\operatorname{trace}\left[(-(A_j^T A_j)^{-1} A_j^T R_k)^T A_j^T R_k\right] + \\ &\qquad + \operatorname{trace}\left[(-(A_j^T A_j)^{-1} A_j^T R_k)^T A_j^T A_j (-(A_j^T A_j)^{-1} A_j^T R_k)\right] \\ &= \|R\|_F^2 - 2\operatorname{trace}\left(R_k^T A_j (A_j^T A_j)^{-1} A_j^T R_k\right) + \operatorname{trace}\left(R_k^T A_j (A_j^T A_j)^{-1} A_j^T R_k\right) \\ &= \|R_k\|_F^2 - \underbrace{\operatorname{trace}\left(R_k^T A_j (A_j^T A_j)^{-1} A_j^T R_k\right)}_{=:\ \tau_{jk}}. \end{aligned}$$

Summarizing, BSPAI can be implemented according to SPAI, e.g., by a blockwise version of Algorithm 4. In order to obtain a preconditioner of comparable quality with respect to SPAI, we have to adjust the tolerance to $\varepsilon_{\text{BSPAI}} = \sqrt{b}\varepsilon_{\text{SPAI}}$ and terminate the update process when $\|AM_k - I_k\|_F < \varepsilon_{\text{BSPAI}}$ [12].

Remark 4.1 *Recently, Huckle presented blocking strategies for sparse approximate inverses [84]. Preliminary results on block matrices indicate that the derived column block sparse approximate inverse (CBSAI) preconditioners are significantly cheaper to construct compared to SAI, remain inherently parallel, and may achieve comparable quality to ILU. This motivates for ongoing research for this type of preconditioners.*

4.2. Computation for a Fixed Block Sparsity Pattern $\mathcal{J}$: BFSAI

The rest of the chapter considers block factorized sparse approximate inverse preconditioners for SPD A. Our BFSAI derivation is based on the K-condition number minimization similar to the formulae for FSAI in Section 3.2.1. The block entries of the Schur complement S (3.3) read as

$$S_{ij} = A_{ij} - A_i(\tilde{\mathcal{J}}_k)^T A(\tilde{\mathcal{J}}_k, \tilde{\mathcal{J}}_k)^{-1} A_j(\tilde{\mathcal{J}}_k) \qquad \text{with} \qquad \tilde{\mathcal{J}}_k = \mathcal{J}_k \setminus \{k\}.$$

In order to obtain the minimum of $K(L^T AL)$ (3.36), we set its derivative equal to zero. Using the splitting $L_k(\mathcal{J}_k) := \tilde{I}_k L_{kk} + \tilde{L}_k$ where $\tilde{I}_k := I_k(\mathcal{J}_k)$ and $\tilde{L}_k := L_k(\mathcal{J}_k) - \tilde{I}_k L_{kk}$, we expand $K(L^T AL)$ to

$$\begin{aligned} \frac{\operatorname{trace}(L^T AL)}{n \det(L^T AL)^{\frac{1}{n}}} &= \frac{\sum_{k=1}^{n_b} \operatorname{trace}\left(L_k^T A L_k\right)}{n \det(A)^{\frac{1}{n}} \prod_{k=1}^{n_b} \det(L_{kk})^{\frac{2}{n}}} \\ &= \frac{1}{n \det(A)^{\frac{1}{n}} \prod_{k=1}^{n_b} \det(L_{kk})^{\frac{2}{n}}} \sum_{k=1}^{n_b} \Big[\operatorname{trace}\left(L_{kk}^T A_{kk} L_{kk}\right) + \end{aligned}$$

$$+ 2\,\mathrm{trace}\left(L_k(\tilde{\mathcal{J}}_k)^T A_k(\tilde{\mathcal{J}}_k) L_{kk}\right) + \mathrm{trace}\left(L_k(\tilde{\mathcal{J}}_k)^T A(\tilde{\mathcal{J}}_k,\tilde{\mathcal{J}}_k) L_k(\tilde{\mathcal{J}}_k)\right)\Big]. \tag{4.5}$$

The differentiation of the block expanded form relative to $L_k(\tilde{\mathcal{J}}_k)$ and L_{kk} yields the optimal BFSAI solution for the block column k. We insert the derivative

$$\begin{aligned}\frac{\partial K(L^T A L)}{\partial L_k(\tilde{\mathcal{J}}_k)} &\overset{(A.1)}{\overset{(A.2)}{=}} n^{-1}\det(A)^{-\frac{1}{n}}\prod_{k=1}^{n_b}\det(L_{kk})^{-\frac{2}{n}}\left[2A_k(\tilde{\mathcal{J}}_k)L_{kk} + 2A(\tilde{\mathcal{J}}_k,\tilde{\mathcal{J}}_k)L_k(\tilde{\mathcal{J}}_k)\right]\\ &\overset{!}{=} 0 \quad\Leftrightarrow\quad L_k(\tilde{\mathcal{J}}_k) = -A(\tilde{\mathcal{J}}_k,\tilde{\mathcal{J}}_k)^{-1}A_k(\tilde{\mathcal{J}}_k)L_{kk}\end{aligned}$$

into (4.5) and observe

$$\frac{\sum_{k=1}^{n_b}\mathrm{trace}\left[L_{kk}^T(A_{kk} - A_k(\tilde{\mathcal{J}}_k)^T A(\tilde{\mathcal{J}}_k,\tilde{\mathcal{J}}_k)^{-1}A_k(\tilde{\mathcal{J}}_k))L_{kk}\right]}{n\det(A)^{\frac{1}{n}}\prod_{k=1}^{n_b}\det(L_{kk})^{\frac{2}{n}}} = \frac{\sum_{k=1}^{n_b}\mathrm{trace}\left(L_{kk}^T S_{kk} L_{kk}\right)}{n\det(A)^{\frac{1}{n}}\prod_{k=1}^{n_b}\det(L_{kk})^{\frac{2}{n}}}. \tag{4.6}$$

The derivative of (4.6) with respect to L_{kk} and renamed indices reads as

$$\begin{aligned}&\frac{\partial}{\partial L_{kk}}\left(\frac{\sum_{j=1}^{n_b}\mathrm{trace}\left(L_{jj}^T S_{jj} L_{jj}\right)}{n\det(A)^{\frac{1}{n}}\prod_{j=1}^{n_b}\det(L_{jj})^{\frac{2}{n}}}\right)\overset{!}{=}0 \;\Leftrightarrow\; \frac{\partial}{\partial L_{kk}}\left(\frac{\sum_{j=1}^{n_b}\mathrm{trace}\left(L_{jj}^T S_{jj} L_{jj}\right)}{\det(L_{kk})^{\frac{2}{n}}}\right)\\ \overset{(A.2)}{\overset{(A.3)}{=}}\;& \frac{\det(L_{kk})^{\frac{2}{n}}(2S_{kk}L_{kk}) - \left[\sum_{j=1}^{n_b}\mathrm{trace}\left(L_{jj}^T S_{jj} L_{jj}\right)\right]\frac{2}{n}\det(L_{kk})^{\frac{2-n}{n}}\det(L_{kk})L_{kk}^{-T}}{\det(L_{kk})^{\frac{4}{n}}}\overset{!}{=}0\\ \Leftrightarrow\;& S_{kk}L_{kk} - \frac{1}{n}\sum_{j=1}^{n_b}\mathrm{trace}(L_{jj}^T S_{jj} L_{jj})L_{kk}^{-T}\overset{!}{=}0\\ \Leftrightarrow\;& L_{kk}^T S_{kk} L_{kk} = \frac{1}{n}\sum_{j=1}^{n_b}\mathrm{trace}(L_{jj}^T S_{jj} L_{jj}).\end{aligned} \tag{4.7}$$

The right-hand side of (4.7) is the same for all block columns of L. Hence,

$$\exists c > 0 \;:\; \forall k \;:\; L_{kk}^T S_{kk} L_{kk} = cI. \tag{4.8}$$

Without loss of generality, we choose $c = 1$ such that FSAI applied to the block pattern of BFSAI results in an equivalent solution. Note that similar to FSPAI, the choice of c has no influence on the properties of the preconditioned system and the quality of L. Finally, we can compute each block column of the BFSAI independently via

$$Y_k := A(\tilde{\mathcal{J}}_k,\tilde{\mathcal{J}}_k)^{-1}A_k(\tilde{\mathcal{J}}_k), \tag{4.9}$$

$$L_{kk} \overset{c=1}{=} \mathrm{chol}(S_{kk})^{-1} = \mathrm{chol}(A_{kk} - A_k(\tilde{\mathcal{J}}_k)^T Y_k)^{-1}, \quad \text{and} \tag{4.10}$$

$$L_k(\tilde{\mathcal{J}}_k) = -Y_k L_{kk}. \tag{4.11}$$

Note that a priori heuristic choices for $\mathcal{J}$—according to Section 3.2.2—can be transferred to BFSAI, as well.

4.3. Updating the Block Sparsity Pattern $\mathcal{J}$: BFSPAI

We are interested in incorporating block pattern updates to improve a current BFSAI solution. In order to obtain the reduction factor for a block entry L_{jk}, we keep all entries $L_k(\mathcal{J}_k)$ fixed while minimizing (3.45) with $L_{\text{new}} := L + I_j L_{jk} I_k^T$ and with respect to L_{jk} for all $j > k$. As the denominator of the K-condition number is independent of L_{jk}, the minimum is located at

$$\begin{aligned}
& \frac{\partial}{\partial L_{jk}} \Big(\operatorname{trace} \big[\left(L + I_j L_{jk} I_k^T\right)^T A \left(L + I_j L_{jk} I_k^T\right) \big] \Big) \\
= \quad & \frac{\partial}{\partial L_{jk}} \Big(\operatorname{trace}(L^T A L) + 2 \operatorname{trace}(L_k^T A_j L_{jk}) + \operatorname{trace}(L_{jk}^T A_{jj} L_{jk}) \Big) \stackrel{!}{=} 0 \\
\overset{\substack{(A.1)\\(A.2)}}{\Leftrightarrow} \quad & 2A_j^T L_k + 2A_{jj} L_{jk} \stackrel{!}{=} 0 \qquad (4.12) \\
\Leftrightarrow \quad & L_{jk} = -A_{jj}^{-1} A_j^T L_k. \qquad (4.13)
\end{aligned}$$

We can prove the positivity of the second derivative by transforming the first derivative into an equivalent matrix-vector form. I.e., for the stacked columns $\mathbb{R}^{b^2} \ni \vec{L}_{jk} := \left[(L_{jk})_1^T, (L_{jk})_2^T, \ldots, (L_{jk})_b^T\right]^T$ we obtain

$$\begin{aligned}
\frac{\partial}{\partial L_{jk}} (2A_j^T L_k + 2A_{jj} L_{jk}) \quad = \quad & \frac{\partial}{\partial L_{jk}} (2A_{jj} L_{jk}) \quad \equiv \quad \frac{\partial}{\partial \vec{L}_{jk}} 2 \operatorname{diag}\left(A_{jj}, A_{jj}, \ldots, A_{jj}\right) \vec{L}_{jk} \\
= \quad & 2 \operatorname{diag}\left(A_{jj}, A_{jj}, \ldots, A_{jj}\right), \qquad (4.14)
\end{aligned}$$

where (4.14) is SPD as A_{jj} is SPD. Hence, (4.13) is a minimum. Inserting L_{jk} into (4.12) the new K-condition number is bounded by

$$\begin{aligned}
1 \overset{(2.20)}{\leq} K_{\text{new}} \quad \leq \quad & \frac{n + 2 \operatorname{trace}\left[L_k^T A_j (-A_{jj}^{-1} A_j^T L_k)\right] + \operatorname{trace}\left[(-A_{jj}^{-1} A_j^T L_k)^T A_{jj} (-A_{jj}^{-1} A_j^T L_k)\right]}{n \det(L^T A L)^{\frac{1}{n}}} \\
= \quad & \frac{n - \operatorname{trace}(L_k^T A_j A_{jj}^{-1} A_j^T L_k)}{n \det(L^T A L)^{\frac{1}{n}}} \\
= \quad & \det(L^T A L)^{-\frac{1}{n}} \left[1 - \frac{1}{n} \operatorname{trace}(L_k^T A_j A_{jj}^{-1} A_j^T L_k)\right] \\
= \quad & \left(1 - \frac{\tau_{jk}}{n}\right) K_{\text{old}} \qquad \text{where} \qquad \tau_{jk} := \operatorname{trace}(L_k^T A_j A_{jj}^{-1} A_j^T L_k) \qquad (4.15)
\end{aligned}$$

denotes the reduction factor if augmenting $\mathcal{J}_k$ with the block index j. Again, only those block indices contained in $\hat{\mathcal{J}}_k$ (3.49) are potential candidates leading to a reduction in $K(L^T A L)$. The current pattern can be augmented with one or several block indices $\mathcal{J}_{\text{new}} \subset \hat{\mathcal{J}}_k$. Therewith, an algorithm for BFSPAI is easily obtained by transferring Algorithm 8 to the block case, incorporating the formulae (4.9)-(4.11) and (4.15). We can initialize BFSPAI with arbitrary lower triangular block sparsity patterns containing diagonal blocks. However, following Theorem B.1 in Appendix B, a block column L_k is not normalized. As L is applied twice (from left and from right) and $\tau_{jk} < b$ always holds according to Theorem C.1 in Appendix C, we use the modified stopping criterion

$$\varepsilon_{\text{BFSPAI}} = b \varepsilon_{\text{FSPAI}} \qquad (4.16)$$

in BFSPAI, similar to BSPAI. Note that it is possible to use a mean value heuristic corresponding to (3.50) in BFSPAI, as well.

Remark 4.2

1. *Adjusting the stopping criterion according to (4.16) may produce preconditioners with a slightly higher K-condition number than for the scalar case, i.e., a BFSPAI may be less effective than FSPAI. That is because while a profitable index j is augmented in FSPAI, the complete block containing j may violate (4.16) and be thus omitted in BFSPAI. On the other hand, (4.16) will lead to setup costs which are comparable to the scalar case.*
2. *It is possible to use block Cholesky updates in BFSPAI according to the derivation for FSPAI in Section 3.2.3. The only difference is that $\mathcal{J}_{new}$ contains one or several augmented block indices.*
3. *In a parallel environment it is possible to precompute and communicate the $A_{jj} = \tilde{L}^T \tilde{L}$ in advance to speedup the computation of the τ_{jk}. Consequently, they can be obtained by evaluating $\tau_{jk} = \|\tilde{L}^{-T} A_j^T L_k\|_F^2$.*
4. *In case of HPD systems, no difficulties occur in computing a complex BFSPAI. Merely complex analogons for (4.9) and (4.10) have to be considered.*
5. *Similar to FS(P)AI, we provide theoretical properties of BFS(P)AI in Appendix B.*
6. *Similar to EFSPAI, it is possible to derive the exact reduction factor in BFSPAI when augmenting $\mathcal{J}_k$ with a block. In Appendix C we derive the exact BFSPAI (EBFSPAI) whose formulae resemble the pointwise case derived in Section 3.2.5.*

4.4. Numerical Experiments

We are interested both in the setup time and the quality of BFSPAI in comparison to FSPAI. Again, we monitor timings of our MATLAB BFSPAI implementation via time stamps (date vectors). Additional information about L and properties of the preconditioned system indicate the quality of the preconditioners. By setting $b = 1$, we use a pointwise BFSPAI as an FSPAI analogy for the following reasons. First, we observe different runtime behavior among different environments of our scalar and blockwise implementation, respectively. Second, our BFSPAI implementation outperforms the scalar one on experimental Environment D.4. Consequently, by using only one implementation, overhead due to different data structures or different algorithmic realizations can be avoided. In all our experiments we add only one block (index) per update step and do not apply dropping in the preconditioners. For arbitrary matrices chosen from [44], we compute the overall block density via

$$\frac{\mathrm{nnz}(A)}{b^2 \cdot \mathrm{nnzb}(A)}, \tag{4.17}$$

where $\mathrm{nnzb}(A)$ denotes the number of nonzero blocks in A, i.e., blocks which at least contain one single scalar component. We restrict ourselves to equidistant blockings but use different block sizes b per matrix. Taking (4.17) into account, we only choose SPD matrices with block structure for our experiments whose blocks have a high density ($\gtrsim 80\%$). Note that finding of efficient blockings for an arbitrary matrix A—without processing all entries of

A—is not trivial [84]. See [59] for maximum weight matching algorithms in order to obtain a certain regular block structure.

Numerical Experiment 4.1 We compare BFSPAI using $b = 1$ to BFSPAI using larger block sizes for a fixed tolerance $\varepsilon_{\text{BFSPAI}} = b \cdot 0.1$ in Table 4.2. The time for constructing L is the average value from 10 setup runs. For many of the chosen matrices, arising from different structural problems, CG does not converge in 5000 iterations for $b = (1,1,\ldots,1)^T$ with $\varepsilon_{\text{CG}} = 1\text{E}{-}09$. For the matrix NASA2910 we illustrate a cut-out of the lower right part of the sparsity pattern of the scalar and block approximate inverse L in Figure 4.1 (b) and (c), respectively. The higher density obtained for larger block sizes is obvious.

Increasing the block size reduces the setup time significantly. Although the block preconditioner is more dense and its application in PCG more expensive, a block size of $b = 2$ or $b = 3$ already pays off for the setup, leading to smaller total costs. Observing the number of PCG iterations, BFSPAI produces preconditioners of approximately similar quality compared to the pointwise case (see matrix BCSSTK24). This approves for the modification of the stopping criterion in (4.16). Note that for $b > 1$ the value of $K(L^T AL)$ and $\|L_A L - I\|_F$ is sometimes larger than for $b = 1$; see Remark 4.2#1 for a possible reason. For NASA2910 and BCSSTK24 the value of $\det(L^T AL)$ falls below the machine precision and is rounded to zero. Hence, we do not provide K-condition numbers for these two matrices here. Regardless, the preconditioner L has full rank and can be used in PCG. ●

Numerical Experiment 4.2 Figure 4.2 illustrates which setup and total time becomes necessary to reach a certain number of iterations within PCG. Still applying the setting $\Upsilon_{\infty,1}$, we now vary $\varepsilon_{\text{FSPAI}} \in [1\text{E}{-}03,1.0]$. We omit values for $b = 1$ in the plots which are too large in order to focus on the important parts of the curves.

For a specific number of PCG iterations, the blockwise preconditioner can be constructed in significantly less time. As in MATLAB matrix-vector products usually outperform other operations, the solver costs do not carry heavy weight producing total costs comparable to the setup costs. Note the slightly increasing total costs for $b > 1$ which occur due to a worse cost-benefit ratio: a more dense preconditioner (still being sparse)—compared to the pointwise case—is applied in a large number of iterations. ●

Numerical Experiment 4.3 In Table 4.4 we analyze the effect of increasing block sizes in BFSPAI while in Table 4.3 we compare the pointwise and blockwise BFSPAI for similar K-condition numbers.

For a comparable K-condition number the blockwise BFSPAI preconditioner has a significant lower setup time while leading to a similar convergence rate. Although being more dense, a rather small block size is a good trade-off between saving setup time and producing high solver costs. This is illustrated for increasing block sizes in Table 4.4: we have to be aware of too much fill-in in L leading to higher solver costs for large block sizes. Here, the setup time may also start to increase; cf. $b = 66$ for matrix BCSSTK24. Following both tables, the K-condition number is a more robust indicator for the number of PCG iterations than the spectral condition number. See, for instance, NOS3 in Table 4.3 where the number of iterations is approximately the same although $\kappa_2(L^T AL)$ is different. ●

Numerical Experiment 4.4 In order to provide more insight, we additionally list runtime results for the experimental environment D.3 in Table 4.1. Here, the FSPAI is computed by a scalar MATLAB implementation and compared to our block implementation. Note the

difference in the setup timings. However, the final interpretation of the results obtained by our previous numerical experiments hold for this environment, as well. ●

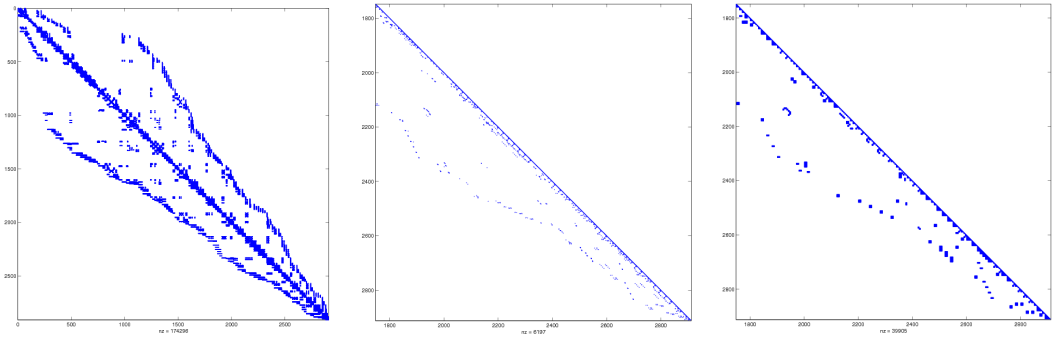
Table 4.1.: Comparison of FSPAI and BFSPAI for matrices from [44] on experimental Environment D.3. Setting is $\Upsilon_{\infty,1}$ with $\varepsilon_{\text{FSPAI}} = 0.1$ and $\varepsilon_{\text{BFSPAI}}$ adapted according to (4.16). CG does not converge in 5000 iterations for $\varepsilon_{\text{CG}} = 1\text{E}{-}09$ and $b = (1,1,\ldots,1)^T$. Timing results have unit seconds.

Matrix	Algorithm	t_{setup}	t_{PCG} (iter.)	t_{total}
BCSSTK11	FSPAI	4.41	0.31 (757)	4.72
	BFSPAI $b = 1$	5.29	0.31 (757)	5.60
	BFSPAI $b = 3$	0.97	0.32 (761)	1.29
BCSSTK14	FSPAI	9.15	0.13 (181)	9.28
	BFSPAI $b = 1$	9.96	0.13 (181)	10.09
	BFSPAI $b = 6$	0.63	0.15 (216)	0.78
BCSSTK24	FSPAI	39.61	1.83 (1282)	41.45
	BFSPAI $b = 1$	27.27	1.84 (1282)	29.11
	BFSPAI $b = 2$	6.64	2.10 (1279)	8.74
	BFSPAI $b = 13$	1.44	1.50 (709)	2.93

Interpretation of the results. Although, in general, results produced with MATLAB have to be treated with caution, our observations indicate the following behavior: we obtain similar behavior between the scalar and blockwise FSPAI which Barnard and Grote observed for SPAI and BSPAI [12]. The setup time for a BFSPAI is significantly lower and compensates for the slight loss of time in the solver due to its higher density. The advantages of inherent parallelism, the adaptive capturing of promising entries (blocks), and the robustness are preserved. Hence, we motivate for an implementation of BFSPAI in a high-level programming language in order to obtain more algorithmic and qualitative insight also into the behavior for larger model problems.

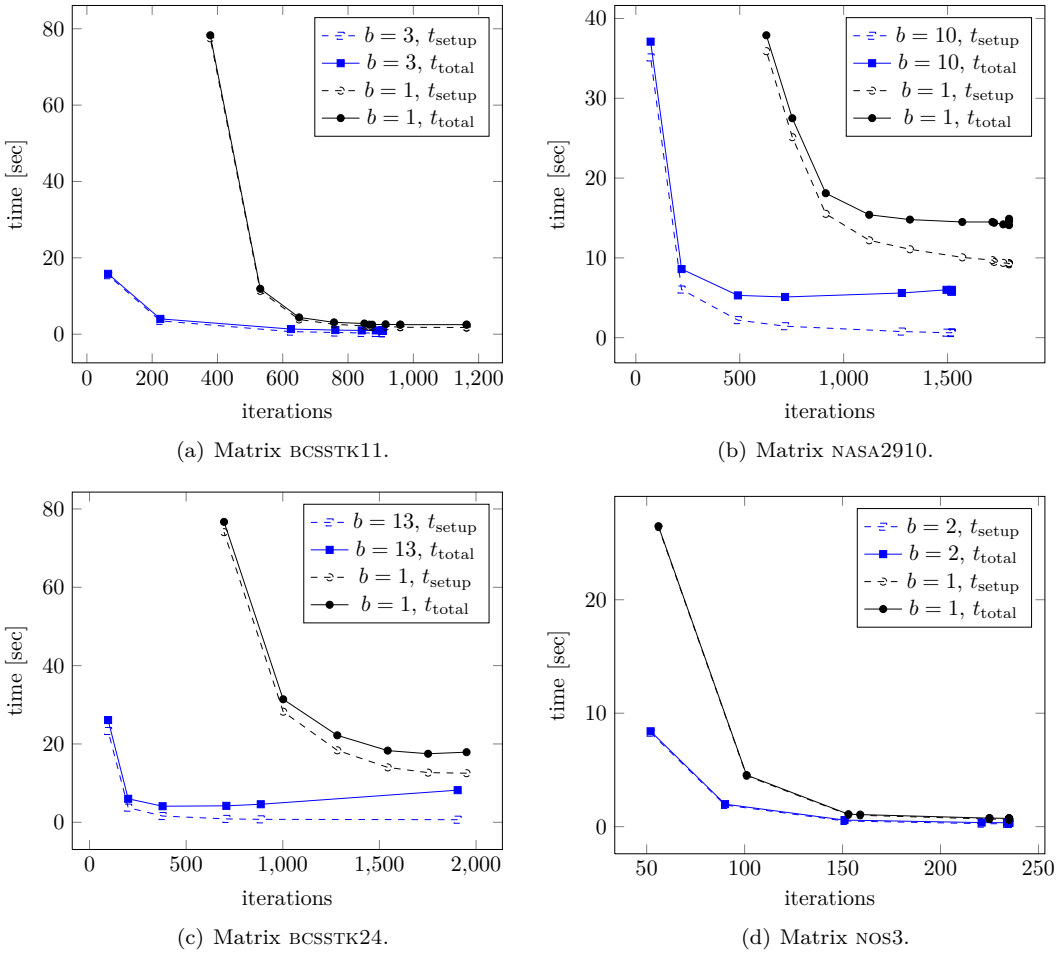
Table 4.2.: Comparison of BFSPAI and FSPAI in Numerical Experiment 4.1 for matrices from [44]. Setting is $\Upsilon_{\infty,1}$ with $\varepsilon_{\text{BFSPAI}} = b \cdot 0.1$. No convergence in 5000 CG iterations for $\varepsilon_{\text{CG}} = 1\text{E}{-}09$ and $b = (1,1,\ldots,1)^T$ is marked with †. Timing results have unit seconds. Setup times are average of 10 runs.

Block size	$\kappa_2(L^TAL)$	$\|L_AL - I\|_F$	nnz(L) (density %)	$K(L^TAL)$	t_{setup}	t_{PCG} (iter.)	t_{total}
	NASA2910, $n = 2910$, $\kappa_2(A) = 5.96\text{E+}06$, $\|L_A - I\|_F = 7.66\text{E+}04$, CG †						
$b = 1$	3.35E+04	29.46	6,197 (0.15 %)	–	27.02	2.97 (754)	29.99
$b = 5$	2.64E+04	29.21	24,015 (0.57 %)	–	2.04	5.63 (662)	7.66
$b = 10$	3.14E+04	27.81	39,905 (0.94 %)	–	1.48	10.59 (718)	12.08
	BCSSTK11, $n = 1473$, $\kappa_2(A) = 2.21\text{E+}08$, $\|L_A - I\|_F = 2.49\text{E+}05$, CG †						
$b = 1$	3.78E+04	21.47	2,317 (0.21 %)	1.43	2.71	0.75 (757)	3.46
$b = 3$	3.35E+04	21.50	4,115 (0.38 %)	1.43	0.44	0.94 (761)	1.38
	BCSSTK14, $n = 1806$, $\kappa_2(A) = 1.19\text{E+}10$, $\|L_A - I\|_F = 1.07\text{E+}06$, CG †						
$b = 1$	0.76E+03	16.70	2,875 (0.18 %)	1.18	5.76	0.30 (181)	6.06
$b = 6$	1.29E+03	19.09	6,772 (0.42 %)	1.26	0.29	0.47 (216)	0.76
	BCSSTK16, $n = 4884$, $\kappa_2(A) = 4.94\text{E+}09$, $\|L_A - I\|_F = 1.85\text{E+}06$, CG conv. in 523 iter.						
$b = 1$	570.23	24.63	5,342 (0.05 %)	1.14	25.40	1.02 (197)	26.41
$b = 3$	748.55	25.48	9,776 (0.08 %)	1.15	3.28	1.28 (218)	4.56
$b = 12$	619.26	24.09	27,098 (0.23 %)	1.13	1.13	1.99 (202)	3.12
	BCSSTK24, $n = 3562$, $\kappa_2(A) = 1.95\text{E+}11$, $\|L_A - I\|_F = 3.65\text{E+}07$, CG †						
$b = 1$	6.30E+04	30.97	6,569 (0.10 %)	–	19.24	4.84 (1283)	24.09
$b = 2$	6.58E+04	31.06	9,939 (0.16 %)	–	4.85	5.42 (1283)	10.27
$b = 13$	9.03E+03	26.28	35,396 (0.56 %)	–	0.87	8.13 (706)	9.00
	NOS3, $n = 960$, $\kappa_2(A) = 3.77\text{E+}04$, $\|L_A - I\|_F = 0.42\text{E+}03$, CG conv. in 271 iter.						
$b = 1$	1.04E+04	12.83	1,418 (0.31%)	1.20	1.00	0.08 (159)	1.08
$b = 2$	2.98E+04	15.12	1,600 (0.34%)	1.29	0.27	0.11 (221)	0.38



(a) Pattern of NASA2910. (b) BFSPAI L with $b = 1$. (c) BFSPAI L with $b = 10$.

Figure 4.1.: Sparsity pattern $\mathcal{J}$ of matrix NASA2910 [44] in (a). (b) and (c) show a cut-out of the lower right part of the pointwise and blockwise $L \approx L_A^{-1}$, respectively.



(a) Matrix BCSSTK11. (b) Matrix NASA2910.

(c) Matrix BCSSTK24. (d) Matrix NOS3.

Figure 4.2.: Comparison between FSPAI ($b = 1$) and BFSPAI for various matrices from [44] and setting $\Upsilon_{\infty,1}$ in Numerical Experiment 4.2. For decreasing $\varepsilon_{\text{BFSPAI}}$ the number of iterations in PCG to reach $\varepsilon_{\text{CG}} = 1\text{E}{-}09$ is plotted against the setup and total time.

Table 4.3.: Comparison of BFSPAI and FSPAI for similar K-cond. number and setting $\Upsilon_{\infty,1}$ in Numer. Exp. 4.3 for matrices from [44]. Rhs for CG is $b = (1,1,\ldots,1)^T$, $k_{\max} = 5000$, and $\varepsilon_{\text{CG}} = 1\text{E}{-}09$. Timing results have unit seconds. $\varepsilon_{\text{BFSPAI}}$ is adapted according to (4.16).

Block size	$K(L^TAL)$	$\varepsilon_{\text{FSPAI}}$	$\kappa_2(L^TAL)$	nnz(L) (density %)	t_{setup}	t_{PCG} (iter.)	t_{total}
	BCSSTK14, $n = 1806$, $\kappa_2(A) = 1.19\text{E}{+}10$, CG †						
$b = 1$	1.1474	0.06	581.55	3,502 (0.22 %)	6.88	0.19 (157)	7.06
$b = 6$	1.1469	0.05	591.89	18,964 (1.16 %)	0.66	0.38 (178)	1.03
	NOS3, $n = 960$, $\kappa_2(A) = 3.77\text{E}{+}04$, CG conv. in 271 iter.						
$b = 1$	1.1897	0.05	9.64E+03	1,480 (0.32 %)	1.05	0.06 (153)	1.11
$b = 5$	1.1832	0.066	1.16E+04	4,779 (1.04 %)	0.17	0.09 (152)	0.27
	BCSSTK16, $n = 4884$, $\kappa_2(A) = 4.94\text{E}{+}09$, CG conv. in 523 iter.						
$b = 1$	1.1379	0.1	570.23	5,342 (0.05 %)	24.98	0.94 (197)	25.92
$b = 4$	1.1366	0.0525	567.41	13,478 (0.11 %)	3.45	1.06 (197)	4.52

Table 4.4.: Increasing block size in Numerical Experiment 4.3 for matrices from [44] and setting $\Upsilon_{\infty,1}$. CG does not converge in 5000 iterations for $\varepsilon_{\text{CG}} = 1\text{E}{-}09$ and $b = (1,1,\ldots,1)^T$. Timing results have unit seconds. $\varepsilon_{\text{BFSPAI}}$ is adapted according to (4.16).

Block size	$\varepsilon_{\text{FSPAI}}$	$\kappa_2(L^TAL)$	$\|L_AL - I\|_F$	nnz(L) (density %)	$K(L^TAL)$	t_{setup}	t_{PCG} (iter.)	t_{total}
	NASA2910, $n = 2910$, $\kappa_2(A) = 5.96\text{E}{+}06$, $\|L_A - I\|_F = 7.66\text{E}{+}04$, CG †							
$b = 1$	0.01	7.22E+03	18.87	21,676 (0.51 %)	1.1409	137.66	1.17 (289)	138.83
$b = 2$		8.66E+03	19.48	43,571 (1.03 %)	1.1520	51.48	1.70 (322)	53.19
$b = 5$		3.97E+03	16.36	88,045 (2.08 %)	1.1027	8.58	1.81 (225)	10.39
$b = 15$		4.45E+03	14.61	203,560 (4.81 %)	1.0818	5.95	3.16 (210)	9.11
$b = 30$		3.06E+03	12.72	314,160 (7.42 %)	1.0615	6.70	3.83 (174)	10.53
	BCSSTK24, $n = 3562$, $\kappa_2(A) = 1.95\text{E}{+}11$, $\|L_A - I\|_F = 3.65\text{E}{+}07$, CG †							
$b = 1$	0.4	777.88	25.95	4,895 (0.08 %)	1.1558	21.69	1.06 (224)	22.75
$b = 2$		804.37	25.92	7,117 (0.11 %)	1.1559	7.91	1.09 (225)	9.00
$b = 4$		742.65	25.40	11,370 (0.18 %)	1.1489	2.94	1.11 (218)	4.05
$b = 11$		638.02	24.34	24,919 (0.39 %)	1.1356	1.25	1.28 (209)	2.53
$b = 33$		466.75	22.25	70,784 (1.12 %)	1.1113	1.31	1.61 (183)	2.92
$b = 66$		374.65	21.01	126,988 (2.00 %)	1.0982	2.08	2.05 (167)	4.13

Chapter 5

Smoothing with Modified Sparse Approximate Inverses

Following Section 2.4, we briefly recall the crucial observation leading to multigrid methods: applying a stationary method (e.g., Gauss-Seidel) gives a satisfactory reduction of the error on the subspace related to high frequency components. Consequently, the error will mainly contain smooth components and can be projected to a smaller linear system. This reduced system can be tackled recursively by the same approach based on smoothing steps and projection on a further reduced system. On the coarsest level the small linear system can be solved explicitly. Afterward, the coarse solutions have to be prolonged step by step back to the finer levels including postsmoothing steps.

In this chapter, we are mainly interested in the smoother being an essential part in MG methods. More precisely, we investigate the smoothing property of modified sparse approximate inverses which presents original research. Parts of it were published in [90]. To derive a convergent method, the smoother has to reduce the error in the high frequency subspace. For a given matrix A and a (sparse) approximate inverse smoother M, the iteration k is described by $x^{(k+1)} = x^{(k)} + M(b - Ax^{(k)})$ and thus the error is given by $I - MA$; see also (2.23). As the eigendecomposition is analytically well-known for discretizations of the constant coefficient Laplace operator [123], it is possible to fully discuss the convergence behavior of MG in this special case. In order to analyze the smoothing property the eigenvalues are separated into high and low frequency eigenvectors. The projection $P = U^T(I - MA)U$ on the high frequency subspace U gives the smoothing factor defined as the spectral radius of P. Following Section 2.4, we can use generating functions for a smoothing analysis in the constant coefficient case and describe the smoothing factor as

$$\mu_{\text{smooth}} := \max_{x,y \in G} \{|1 - m(x,y)a(x,y)|\},$$

similar to (2.27). Here, a and m are generating functions representing A and M while G denotes the high frequency domain corresponding to (2.26).

We focus on a static MSPAI (MSAI) as a smoother for the general case. SAI was already considered by Tang and Wan in [133] and SPAI by Bröker et al. in [29]. We derive smoothers for 1D and 2D discretizations of the Laplace operator. In the latter case we provide results for the classical 5-point stencil as well as the 9-point stencil. The two sections of the chapter are structured in a similar way: we derive the optimal smoother for the underlying problem and translate the minimization conditions into local mask probing conditions for MSAI. Afterward, we transfer them to global (classical) probing vectors and additionally analyze the effect of global probing conditions with action on A and with action on non-constant coefficient problems. While presenting the numerical results clustered on a single page at

the end of each section, we provide specific experiments with brief interpretation throughout the chapter.

5.1. 1D Model Problems

We consider the standard 1D discretized Laplace operator with constant coefficients which is of the form $A_1 x = b$, with $A_1 := \text{tridiag}\left(-\frac{1}{2},1,-\frac{1}{2}\right)$. The matrix A_1 is related to the generating function or symbol

$$a_1(x) = 1 - \frac{e^{ix} + e^{-ix}}{2} = 1 - \cos(x). \tag{5.1}$$

The high frequency part of A_1 is represented by (5.1) for $x \in \left[\frac{\pi}{2},\pi\right] =: G_1$. Hence the optimal smoothing parameter ω in the Jacobi smoother $1 - \omega a_1(x)$ is found by solving the problem

$$\min_{\omega} \max_{x \in G_1} |\underbrace{1 - \omega a_1(x)}_{=:\, s(x)}| = \min_{\omega} \max_{x \in G_1} |s(x)| \overset{\omega=\frac{2}{3}}{=} \frac{1}{3}. \tag{5.2}$$

The solution can be found by replacing (5.2) with the maximum over the two boundary values

$$\min_{\omega} \max \left\{ \left|s\left(\tfrac{\pi}{2}\right)\right|, |s(\pi)| \right\}.$$

5.1.1. Analytical Derivation of the Optimal Smoother

As approximate inverse smoother we choose a trigonometric polynomial of the same degree

$$m_1(x) = a + 2b\cos(x) \tag{5.3}$$

and a tridiagonal Toeplitz matrix $M = \text{tridiag}(b,a,b)$, respectively. The smoothing condition for (5.3) can be written as

$$\min_{a,b} \max_{x \in G_1} |1 - m_1(x)a_1(x)| = \min_{a,b} \max_{x \in G_1} \Big| \underbrace{1 - \big[a + 2bh(x)\big]\big[1 - h(x)\big]}_{=:\, s(x)} \Big|, \tag{5.4}$$

where $h(x) := \cos(x)$ with $h(x) \in [-1,0]$. Considering the boundary values of G_1 and $h(x)$, respectively, we obtain the first two minimization conditions in (5.6). An additional third condition can be derived for the local extreme point in $s(x)$. Inserting

$$\frac{\mathrm{d}}{\mathrm{d}h(x)} \Big[1 - \big[a + 2bh(x)\big]\big[1 - h(x)\big]\Big] \overset{!}{=} 0 \qquad \Leftrightarrow \qquad h(x) = \frac{1}{2} - \frac{a}{4b} \tag{5.5}$$

into $s(x)$, we end up with the overall minimization problem

$$\min_{a,b} \left\{ |a - 1|, |2(a - 2b) - 1|, \left| \frac{(a + 2b)^2}{8b} - 1 \right| \right\}. \tag{5.6}$$

By equating the first two conditions we deduce $b = \frac{a}{4}$ which we use to linearize and solve the equation $\frac{(a+2b)^2}{8b} - 1 \stackrel{!}{=} -[2(a-2b)-1]$. We finally obtain $(a,b) = \left(\frac{16}{17},\frac{4}{17}\right)$ which leads to

$$\mathsf{M3}_{\text{opt}} := \text{tridiag}\left(\frac{4}{17},\frac{16}{17},\frac{4}{17}\right). \tag{5.7}$$

The number in our notation for optimal smoothers symbolizes the used stencil discretization, i.e., $\mathsf{M3}_{\text{opt}}$ is the optimal smoother corresponding to a 3-point stencil. The related smoothing factor for this optimal preconditioner $m_1(x)$ is $\frac{1}{17} = 0.0588$, which is significantly smaller than the optimal smoothing factor of $\frac{1}{3} = 0.3\overline{33}$ related to $\omega = \frac{2}{3}$ for the Jacobi smoother.

5.1.2. Individual Probing Masks

The minimization conditions (5.6) on a and b can also be seen as conditions for the entries of the matrix directly. A column k of M is described by the three values

$$\hat{M}_k := (m_{k-1,k}, m_{k,k}, m_{k+1,k})^T, \tag{5.8}$$

where the main diagonal entry is related to a and the upper and lower entries to b. Therefore, we can translate the two linear minimization conditions into individual probing masks $\hat{S}_k$ on entries of M directly:

$$\mathsf{S3}_{\mathsf{L1}}: \min_{\mathcal{J}(\hat{M}_k)} \left|(0,1,0)\hat{M}_k - 1\right| \qquad \text{and} \qquad \mathsf{S3}_{\mathsf{L2}}: \min_{\mathcal{J}(\hat{M}_k)} \left|(-1,1,-1)\hat{M}_k - \frac{1}{2}\right|.$$

Notice that we refer to individual probing conditions as local probing conditions and to $\mathcal{J}(\hat{M}_k)$ as the pattern according to the three entries in (5.8). The number in our mask notation again corresponds to the underlying stencil discretization while the subscript denotes either local ($\mathsf{S}*_{\mathsf{L}*}$) or global ($\mathsf{S}*_{\mathsf{G}*}$) minimization conditions. There are several possibilities to eliminate and replace the quadratic condition by linear conditions that can be related to masks itself. In one possible method we assume that the linear conditions of (5.6) are satisfied exactly. Inserting $a = 1$ into the second condition yields $b = \frac{1}{4}$. We replace the denominator of the quadratic condition with this value and obtain the new linear condition

$$\min_{a,b} \left|\frac{(a+2b)^2}{8 \cdot \frac{1}{4}} - 1\right| \;=\; \min_{a,b} \left|(a+2b)^2 - 2\right| \;\Rightarrow\; \min_{a,b} \left|a + 2b - \sqrt{2}\right|$$

which is related to the probing condition with isotropic probing mask (see Definition 3.2)

$$\mathsf{S3}_{\mathsf{L3}}: \min_{\mathcal{J}(\hat{M}_k)} \left|(1,1,1)\hat{M}_k - \sqrt{2}\right|.$$

Here, $\hat{S}_k = (-1,1,-1)$ of $\mathsf{S3}_{\mathsf{L2}}$ is related to $e_N = (1,-1,1,-1,\ldots)^T$ and $\hat{S}_k = (1,1,1)$ of $\mathsf{S3}_{\mathsf{L3}}$ to $e_S = (1,1,\ldots,1)^T$.

In our following numerical experiments we are interested in a comparison between MSAI with probing conditions, the optimal smoother $\mathsf{M3}_{\text{opt}}$ (5.7), and the tridiagonal preconditioner given by SAI. The SAI matrix is derived by minimizing the Frobenius norm in which the reduced LS problem for a typical column of M is given by

$$\min_{\mathcal{J}(\hat{M}_k)} \left\| \begin{pmatrix} -0.5 & 0 & 0 \\ 1 & -0.5 & 0 \\ -0.5 & 1 & -0.5 \\ 0 & -0.5 & 1 \\ 0 & 0 & -0.5 \end{pmatrix} \hat{M}_k - \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} \right\|_2 . \tag{5.9}$$

The inner columns of the solution of (5.9) have Toeplitz structure described by the matrix tridiag $\left(\frac{2}{5}, \frac{6}{5}, \frac{2}{5}\right)$ related to the generating function $m(x) = \frac{6}{5} + \frac{4}{5}\cos(x)$. Preconditioning with the SAI matrix results in the smoothing factor 0.250. Indeed, this is a degradation compared to the smoothing factor 0.0588 for the optimal derived smoother $\mathsf{M3}_{\text{opt}}$ but still an improvement compared to Jacobi. Note that MS(P)AI minimizes the 2-norm of a combination of conditions while $\mathsf{M3}_{\text{opt}}$ is deduced from minimizing the 1-norm. Therefore, to derive efficient smoothing factors by MS(P)AI, it is important to find a convenient combination and weighting of probing conditions.

Numerical Experiment 5.1 We consider the matrix A_1 of dimension $n = 1\text{E}{+}03$. Following Table 5.1 and Figure 5.1 (a), an approximate inverse preconditioner—satisfying the individual probing conditions—yields highly reduced smoothing factors compared to SAI and Jacobi. Depending on the probing mask and weight ρ, reductions down to $\mu_{\text{smooth}} = 0.075$ are possible for $\mathsf{S3}_{\mathsf{L3}}$ weighted with $\rho = 100$. Moreover, the choice of the subspace weight is stable, i.e., increasing values lead to a saturation of μ_{smooth}. ●

5.1.3. Global Probing Vectors

Considering the conditions $\mathsf{S3}_{\mathsf{L2}}$ and $\mathsf{S3}_{\mathsf{L3}}$ with isotropic masks, it is possible to derive conditions with high and low frequency global probing vectors (see Section 3.1.4) for the generalized Frobenius norm minimization (3.23) of MSAI:

$$\min_{\mathcal{J}(M)\in\mathcal{J}} \left\| e_{N1}^T M - \frac{1}{2} e_{N1}^T \right\|_2 \quad \text{and} \quad \min_{\mathcal{J}(M)\in\mathcal{J}} \left\| e_S^T M - \sqrt{2} e_S^T \right\|_2 ,$$

respectively. In the following we use a different approach and derive global probing conditions with respect to the optimal derived preconditioner $\mathsf{M3}_{\text{opt}}$. This can also be applied to more general probing vectors, e.g., $e_{N2} = (1,0,-1,0,1,\ldots)^T$ or $e_{N3} = (0,1,0,-1,0,1,\ldots)^T$. For a given probing vector e, we observe that $e^T \mathsf{M3}_{\text{opt}} = \beta e^T$ with $\beta \in \mathbb{R}$. As we want to find the smoother M, based on MSAI with the same action on e^T, we define probing conditions $e^T M \approx \beta e^T$ and obtain

$$\mathsf{S3}_{\mathsf{G1}}: \min_{\mathcal{J}(M)\in\mathcal{J}} \left\| e_S^T M - \frac{24}{17} e_S^T \right\|_2 , \qquad \mathsf{S3}_{\mathsf{G2}}: \min_{\mathcal{J}(M)\in\mathcal{J}} \left\| e_{N1}^T M - \frac{8}{17} e_{N1}^T \right\|_2 ,$$

$$\mathsf{S3}_{\mathsf{G3}}: \min_{\mathcal{J}(M)\in\mathcal{J}} \left\| e_{N2}^T M - \frac{16}{17} e_{N2}^T \right\|_2 , \quad \text{and} \quad \mathsf{S3}_{\mathsf{G4}}: \min_{\mathcal{J}(M)\in\mathcal{J}} \left\| e_{N3}^T M - \frac{16}{17} e_{N3}^T \right\|_2 .$$

Numerical Experiment 5.2 Figure 5.1 (b) shows the impact of using these global probing conditions for matrix A_1. Inducing MSAI to satisfy $\mathsf{S3}_{\mathsf{G1}}$ leads to a strong reduction of the smoothing factor in comparison to SAI, close to the optimal value of $\mathsf{M3}_{\text{opt}}$. A combination of global conditions can lead to an efficient smoother as well. Again, the smoothing factor remains stable for increasing values of ρ. ●

5.1.4. Global Probing Vectors with Action on General A

In a similar way it is possible to derive global probing conditions with action on general A, i.e., $\min_{\mathcal{J}(M)\in\mathcal{J}}\|e^TAM-\beta e^T\|_2$. For probing vectors such as e_{N1}, e_{N2}, and e_{N3}, the approximation $e^TA\approx const\cdot e^T$ holds and therefore $e^TAM\approx const\cdot e^TM$ up to some boundary perturbations. Note that for an approximate inverse preconditioner which should nearly be exact on e the condition $e^TAM\approx e^T$ should be satisfied.

As e^TA_1 can be determined easily for the high frequency probing vectors, we compute the optimal factor β such that $e^TA_1M=\beta e^T=e^TA_1\mathsf{M3}_{\text{opt}}$ is satisfied. We thus denote the exact conditions

$$\min_{\mathcal{J}(M)\in\mathcal{J}}\left\|e^TA_1M-e^TA_1\mathsf{M3}_{\text{opt}}\right\|_2=\min_{\mathcal{J}(M)\in\mathcal{J}}\left\|e^TA_1M-\frac{16}{17}e^T\right\|_2 \tag{5.10}$$

for the probing vectors e_{N1}, e_{N2}, and e_{N3} as $\mathsf{S3}_{\mathsf{G5}}$, $\mathsf{S3}_{\mathsf{G6}}$, and $\mathsf{S3}_{\mathsf{G7}}$. Additionally, when not taking the boundary values into account, we can consider approximations of these conditions in which the probing vectors—except for multiples—merge into each other. For $\beta=1$ the approximate conditions are indicated by $\mathsf{S3}_{\tilde{\mathsf{G5}}}$, $\mathsf{S3}_{\tilde{\mathsf{G6}}}$, and $\mathsf{S3}_{\tilde{\mathsf{G7}}}$. We do not use e_S here because $e_S^TA_1$ results in the zero vector.

As a generalization we are interested in the impact of using global probing conditions when considering the 1D tridiagonal matrix B_1 with varying coefficients and k^{th} row defined by

$$(B_1)_{k,:}:=(0,\ldots,0,-b_{k-1},b_{k-1}+b_k,-b_k,0,\ldots,0)\qquad\text{with}\qquad b:=\left(-\frac{1}{2}-\frac{j}{2n}\right)_{j=0,\ldots,n}.$$

Similar to (5.10), we use the exact conditions $\min_{\mathcal{J}(M)\in\mathcal{J}}\|e^TB_1M-\frac{16}{17}e^T\|_2$ for the high frequency probing vectors denoted as $\mathsf{S3}_{\mathsf{G5}}$, $\mathsf{S3}_{\mathsf{G6}}$, and $\mathsf{S3}_{\mathsf{G7}}$ while $\mathsf{S3}_{\tilde{\mathsf{G5}}}$, $\mathsf{S3}_{\tilde{\mathsf{G6}}}$, and $\mathsf{S3}_{\tilde{\mathsf{G7}}}$ correspond to the approximate ones.

Numerical Experiment 5.3 Referring to Figures 5.1 (c) and (d) shows that using a subspace spanned by all conditions leads to the optimal smoothing factor $\mu_{\text{smooth}}=0.0588$. Besides being stable for increasing values of ρ, the approximate subspace leads to a reduction by a factor of 2, compared to SAI. Incorporating single probing conditions, e.g., $\mathsf{S3}_{\mathsf{G6}}$ in Figure 5.1 (c), yields only weak improvement. Here the probing conditions with action on A_1 nearly coincide with local probing masks or global conditions, up to boundary effects. Furthermore, this deterioration at the boundaries leads to a degradation of the smoothing factor. Compare $\mathsf{S3}_{\mathsf{G3}}$ in Figure 5.1 (b) and $\mathsf{S3}_{\mathsf{G6}}$ in (c). ●

5.2. 2D Model Problems

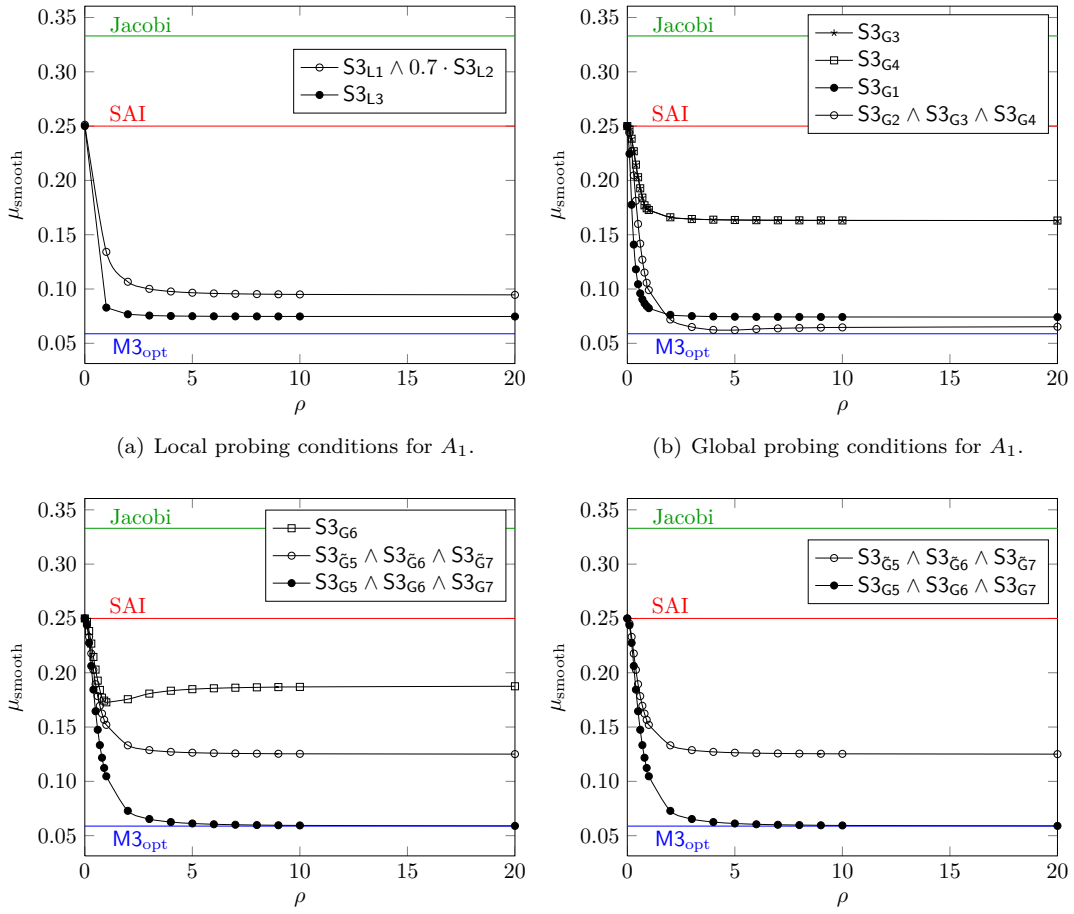
We consider the block tridiagonal matrix

$$A_2:=\frac{A_1\otimes I+I\otimes A_1}{2}\;=\;\text{blocktridiag}\left(0,-\frac{1}{4},0\,\middle|\,-\frac{1}{4},1,-\frac{1}{4}\,\middle|\,0,-\frac{1}{4},0\right),$$

with $A_2\in\mathbb{R}^{n^2\times n^2}$ related to the generating function

Table 5.1.: Smoothing factors by using $\mathsf{M3}_{\text{opt}}$ (5.7), SAI, and MSAI with local probing conditions for the constant coefficient system A_1 of dimension $n = 1\text{E}{+}03$.

$\mathsf{M3}_{\text{opt}}$	SAI	ρ	Local probing conditions $\rho\mathsf{S3}_{\mathsf{L1}} \wedge 0.7 \cdot \mathsf{S3}_{\mathsf{L2}}$	$\rho\mathsf{S3}_{\mathsf{L3}}$
0.0588	0.250	1.0	0.134	0.083
		2.0	0.107	0.077
		10.0	0.095	0.075
		100.0	0.095	0.075



(a) Local probing conditions for A_1.

(b) Global probing conditions for A_1.

(c) Global probing conditions with action on A_1.

(d) Global probing conditions with action on B_1.

Figure 5.1.: Smoothing factor μ_{smooth} against subspace weight ρ. MSAI is compared to Jacobi, SAI, and $\mathsf{M3}_{\text{opt}}$ (5.7) for various probing conditions. Both the constant coefficient sysem A_1 and the system with varying coefficients B_1 are of dimension $n = 1\text{E}{+}03$. Subfigure (a) illustrates results from Table 5.1. (c) and (d) show global probing conditions with action on the underlying system.

$$a_2(x,y) = 1 - \frac{\cos(x) + \cos(y)}{2}.$$

As before, the smoothing corresponds to the rectangle

$$G_2 := \left\{(x,y) : x \in \left[\tfrac{\pi}{2},\pi\right] \wedge y \in [0,\pi]\right\}.$$

Hence, the solution of $\min_\omega \max_{x,y\in G_2} \left|1 - \omega a_2(x,y)\right|$ yields the optimal Jacobi smoother with smoothing factor $\frac{3}{5} = 0.6$ for $\omega = \frac{4}{5}$.

5.2.1. Analytical Derivation of the Optimal Smoother

As approximate inverse smoother we use the trigonometric polynomial

$$m_2(x,y) = a + 2b\left[\cos(x) + \cos(y)\right].$$

We observe the smoothing condition

$$\min_{a,b} \max_{x,y\in G_2} |1 - m_2(x,y)a_2(x,y)| = \min_{a,b} \max_{x,y\in G_2} \Bigg| \underbrace{1 - [a + 2bh(x,y)]\left[1 - \frac{h(x,y)}{2}\right]}_{=:\ s(x,y)} \Bigg|,$$

where $h(x,y) := \cos(x) + \cos(y)$ with $h(x,y) \in [-2,1]$. The corners of G_2 result in the first two minimization conditions of (5.11). We can derive the third condition by inserting

$$\frac{\mathrm{d}}{\mathrm{d}h(x,y)} s(x,y) \stackrel{!}{=} 0 \qquad \Leftrightarrow \qquad h(x,y) = 1 - \frac{a}{4b}$$

into $s(x,y)$ and obtain the minimization problem

$$\min_{a,b} \left\{ |2a - 8b - 1|, \left|\frac{a}{2} + b - 1\right|, \left|\frac{a}{2} + b + \frac{a^2}{16b} - 1\right| \right\}. \tag{5.11}$$

Similar to the 1D case, we equate the linear conditions and obtain $a = 6b$ which we use to linearize and solve $\frac{a}{2} + b + \frac{a^2}{16b} - 1 \stackrel{!}{=} -[\frac{a}{2} + b - 1]$. Hence, the optimal smoothing preconditioner is given by

$$\mathsf{M5}_{\text{opt}} := \text{blocktridiag}\left(0,\frac{8}{41},0 \,\middle|\, \frac{8}{41},\frac{48}{41},\frac{8}{41} \,\middle|\, 0,\frac{8}{41},0\right) \tag{5.12}$$

with smoothing factor $\frac{9}{41} = 0.2195$.

5.2.2. Individual Probing Masks

Again, we can interpret the minimizations (5.11) on a and b as conditions for the entries of the smoother directly. A column k of M is now described by the five degrees of freedom

$$\hat{M}_k := (0,m_{k-n,k},0|m_{k-1,k},m_{k,k},m_{k+1,k}|0,m_{k+n,k},0)^T. \tag{5.13}$$

Thus, we gain the following local probing conditions for the 5-point stencil A_2:

$$\mathsf{S5_{L1}}: \min_{\mathcal{J}(\hat{M}_k)} \left| (0,-2,0\,|-2,2,-2|\,0,-2,0)\,\hat{M}_k - 1 \right| \qquad \text{and}$$
$$\mathsf{S5_{L2}}: \min_{\mathcal{J}(\hat{M}_k)} \left| \left(0,\frac{1}{4},0\,\middle|\,\frac{1}{4},\frac{1}{2},\frac{1}{4}\middle|\,0,\frac{1}{4},0\right)\hat{M}_k - 1 \right|.$$

Making use of the fact that the linear conditions in (5.11) have the same absolute value $a = 6b$ for the optimal a and b, we can replace the quadratic term in the third condition either by

$$\frac{a^2}{16\frac{a}{6}} = \frac{3a}{8} \qquad \text{or} \qquad \frac{(6b)^2}{16b} = \frac{9b}{4}.$$

Consequently, we end up with two additional linear conditions

$$\min_{a,b} \left\{ \left| \frac{7a}{8} + b - 1 \right|, \left| \frac{a}{2} + \frac{13b}{4} - 1 \right| \right\} \tag{5.14}$$

which can be translated into the local probing conditions

$$\mathsf{S5_{L3}}: \min_{\mathcal{J}(\hat{M}_k)} \left| \left(0,\frac{1}{4},0\,\middle|\,\frac{1}{4},\frac{7}{8},\frac{1}{4}\middle|\,0,\frac{1}{4},0\right)\hat{M}_k - 1 \right| \qquad \text{and}$$
$$\mathsf{S5_{L4}}: \min_{\mathcal{J}(\hat{M}_k)} \left| \left(0,\frac{13}{16},0\,\middle|\,\frac{13}{16},\frac{1}{2},\frac{13}{16}\middle|\,0,\frac{13}{16},0\right)\hat{M}_k - 1 \right|.$$

We can combine the conditions $\mathsf{S5_{L2}}$ and $\mathsf{S5_{L4}}$ to obtain an isotropic mask to which the low frequency probing vector e_S corresponds to. We add the diagonal and subdiagonal values of the weighted conditions $r \cdot \mathsf{S5_{L2}}$ and $s \cdot \mathsf{S5_{L4}}$ and set them to be equal. The isotropy condition leads to the equation $\frac{r+s}{2} = \frac{r}{4} + \frac{13s}{16}$ with the solution $r = \frac{5s}{4}$. We obtain

$$a = b = \frac{r+s}{2} = \frac{9s}{8} \qquad \text{and} \qquad r + s = \frac{5s}{4} + s = \frac{9s}{4}$$

for the right-hand side, which leads to the additional individual isotropic probing condition

$$\mathsf{S5_{L5}}: \min_{\mathcal{J}(\hat{M}_k)} \left| (0,1,0\,|1,1,1|\,0,1,0)\,\hat{M}_k - 2 \right|.$$

Numerical Experiment 5.4 Consider A_2 of size $n = 1024$. Following Figure 5.2 (a), an MSAI satisfying individual probing conditions reduces the smoothing factor in comparison to SAI with factor 0.339. The smoothing factor can be reduced further towards the optimal value of $\mathsf{M5}_{\text{opt}}$ when utilizing more DsOF by a combination of probing masks. ●

5.2.3. Global Probing Vectors and Global Probing Vectors with Action on A

Similar to the 1D model problems we observe that $(e \otimes f)^T \mathsf{M5}_{\text{opt}} = \beta (e \otimes f)^T$ for certain e,f, and β. Therefore, we define global probing conditions by using $(e \otimes f)^T M = \beta (e \otimes f)^T$. Regarding the 2D case, we use global vectors resulting from the Kronecker products

$$\bar{e}_S := e_S \otimes e_S, \quad \bar{e}_{N1} := e_{N1} \otimes e_{N1}, \quad \bar{e}_{N2} := e_{N2} \otimes e_{N2}, \quad \text{and} \quad \bar{e}_{N3} := e_{N3} \otimes e_{N3}.$$

We obtain the global probing conditions

$$\mathsf{S5}_{\mathsf{G1}}: \min_{\mathcal{J}(M)\in\mathcal{J}} \left\| \bar{e}_S^T M - \frac{80}{41}\bar{e}_S^T \right\|_2, \qquad \mathsf{S5}_{\mathsf{G2}}: \min_{\mathcal{J}(M)\in\mathcal{J}} \left\| \bar{e}_{N1}^T M - \frac{16}{41}\bar{e}_{N1}^T \right\|_2,$$
$$\mathsf{S5}_{\mathsf{G3}}: \min_{\mathcal{J}(M)\in\mathcal{J}} \left\| \bar{e}_{N2}^T M - \frac{48}{41}\bar{e}_{N2}^T \right\|_2, \quad \text{and} \quad \mathsf{S5}_{\mathsf{G4}}: \min_{\mathcal{J}(M)\in\mathcal{J}} \left\| \bar{e}_{N3}^T M - \frac{48}{41}\bar{e}_{N3}^T \right\|_2.$$

For the Kronecker vectors $\bar{e}_{N1}$, $\bar{e}_{N2}$, and $\bar{e}_{N3}$ it is possible to derive global probing conditions with action on A_2. We compute the optimal factor β such that

$$(e\otimes e)^T A_2 M = \beta (e\otimes e)^T = (e\otimes e)^T A_2 \mathsf{M5}_{\text{opt}}$$

is satisfied. Thus, we obtain for $\bar{e}_{N1}$ the exact condition

$$\mathsf{S5}_{\mathsf{G5}}: \min_{\mathcal{J}(M)\in\mathcal{J}} \left\| \bar{e}_{N1}^T A_2 M - \bar{e}_{N1}^T A_2 \mathsf{M5}_{\text{opt}} \right\|_2 = \min_{\mathcal{J}(M)\in\mathcal{J}} \left\| \bar{e}_{N1}^T A_2 M - \frac{32}{41}\bar{e}_{N1}^T \right\|_2$$

and similarly for $\bar{e}_{N2}$ and $\bar{e}_{N3}$ the conditions

$$\mathsf{S5}_{\mathsf{G6}}: \min_{\mathcal{J}(M)\in\mathcal{J}} \left\| \bar{e}_{N2}^T A_2 M - \frac{48}{41}\bar{e}_{N2}^T \right\|_2 \quad \text{and} \quad \mathsf{S5}_{\mathsf{G7}}: \min_{\mathcal{J}(M)\in\mathcal{J}} \left\| \bar{e}_{N3}^T A_2 M - \frac{48}{41}\bar{e}_{N3}^T \right\|_2.$$

Again, for $\beta = 1$, we indicate the approximate conditions by $\mathsf{S5}_{\tilde{\mathsf{G}}\mathsf{5}}$, $\mathsf{S5}_{\tilde{\mathsf{G}}\mathsf{6}}$, and $\mathsf{S5}_{\tilde{\mathsf{G}}\mathsf{7}}$.

Numerical Experiment 5.5 Following Figure 5.2 (b), an approximation on the subspace spanned by the global probing vectors reduces the smoothing factor compared to SAI and Jacobi, similar to MSAI with individual probing masks. The global condition $\mathsf{S5}_{\mathsf{G1}}$ leads to a stable smoothing factor of 0.252. Referring to Figure 5.2 (c), it is possible to reach the optimal smoothing factor when using the global probing subspace satisfying the conditions with action on A_2 exactly. A heuristic approximation, at which the probing vectors are reproducing themselves, leads to a slightly lower saturated smoothing factor after a tight lower amplitude at the beginning. Incorporating the single high frequency condition $\mathsf{S5}_{\mathsf{G5}}$, it is possible to reduce the smoothing factor solidly to 0.278. ●

5.2.4. 9-Point Stencil Derivations

As a second 2D example we consider the 9-point stencil of A_2, which is the block tridiagonal matrix

$$A_3 := \text{blocktridiag}\left(-\frac{1}{8},-\frac{1}{8},-\frac{1}{8}\,\middle|\,-\frac{1}{8},1,-\frac{1}{8}\,\middle|\,-\frac{1}{8},-\frac{1}{8},-\frac{1}{8}\right)$$

related to the generating function

$$a_3(x,y) = 1 - \frac{\cos(x)+\cos(y)+\cos(x-y)+\cos(x+y)}{4}.$$

Smoothing still corresponds to the rectangle G_2 and we use the pattern of $a_3(x,y)$ for our approximate inverse smoother

$$m_3(x,y) = a + 2b[\cos(x)+\cos(y)+\cos(x-y)+\cos(x+y)].$$

Hence, we consider the smoothing condition

$$\min_{a,b}\max_{x,y\in G_2}|1-m_3(x,y)a_3(x,y)| = \min_{a,b}\max_{x,y\in G_2}\Bigg|\underbrace{1-[a+2bh(x,y)]\left[1-\frac{h(x,y)}{4}\right]}_{=:\,s(x,y)}\Bigg|,$$

where

$$h(x,y) := \cos(x)+\cos(y)+\cos(x-y)+\cos(x+y) \qquad \text{and} \qquad h(x,y)\in[-1,2]\,.$$

With the boundary values $h(x,y)=-1$ and $h(x,y)=2$ we get the linear conditions in (5.15). The third quadratic condition can be derived by inserting

$$\frac{\mathrm{d}}{\mathrm{d}h(x,y)}s(x,y)\stackrel{!}{=}0 \qquad \Leftrightarrow \qquad h(x,y)=2-\frac{a}{4b}$$

into $s(x,y)$. We therefore obtain the overall minimization conditions at high frequency values

$$\min_{a,b}\left\{\left|\frac{3(a-4b)}{2}-1\right|,\left|\frac{3(a+2b)}{4}-1\right|,\left|\frac{a}{2}+2b+\frac{a^2}{32b}-1\right|\right\}. \tag{5.15}$$

Equating the linear conditions yields $a=10b$ which can be used to solve

$$\frac{a}{2}+2b+\frac{a^2}{32b}-1 \;\stackrel{!}{=}\; -\frac{3(a+2b)}{4}+1.$$

The optimal solution is given by $(a,b)=\left(\frac{160}{153},\frac{16}{153}\right)$ which leads to

$$\mathsf{M9}_{\mathrm{opt}} = \mathrm{blocktridiag}\left(\frac{16}{153},\frac{16}{153},\frac{16}{153}\,\middle|\,\frac{16}{153},\frac{160}{153},\frac{16}{153}\,\middle|\,\frac{16}{153},\frac{16}{153},\frac{16}{153}\right) \tag{5.16}$$

with smoothing factor $\frac{1}{17}=0.0588$. The direct translation into linear probing masks, in which a column k of M is described by the nine degrees of freedom

$$\hat{M}_k := (m_{k-n-1,k}, m_{k-n,k}, m_{k-n+1,k}|m_{k-1,k}, m_{k,k}, m_{k+1,k}|m_{k+n-1,k}, m_{k+n,k}, m_{k+n+1,k})^T,$$

reveals the individual probing conditions

$$\mathsf{S9}_{\mathsf{L1}}: \min_{\mathcal{J}(\hat{M}_k)}\left|\left(-\frac{3}{4},-\frac{3}{4},-\frac{3}{4}\,\middle|-\frac{3}{4},\frac{3}{2},-\frac{3}{4}\middle|-\frac{3}{4},-\frac{3}{4},-\frac{3}{4}\right)\hat{M}_k-1\right| \qquad \text{and}$$

$$\mathsf{S9}_{\mathsf{L2}}: \min_{\mathcal{J}(\hat{M}_k)}\left|\left(\frac{3}{16},\frac{3}{16},\frac{3}{16}\,\middle|\,\frac{3}{16},\frac{3}{4},\frac{3}{16}\middle|\,\frac{3}{16},\frac{3}{16},\frac{3}{16}\right)\hat{M}_k-1\right|.$$

The linear conditions are identical in the optimum with $a=10b$. To linearize the quadratic condition in (5.15) we can replace its quadratic term either by $\frac{10ba}{32b}=\frac{5a}{16}$ or $\frac{(10b)^2}{32b}=\frac{25b}{8}$. We thus gain the two additional linear conditions

$$\min_{a,b}\left\{\left|\frac{13a}{16}+2b-1\right|,\left|\frac{a}{2}+\frac{41b}{8}-1\right|\right\}$$

which can be translated into the local probing conditions

$$\mathsf{S9_{L3}}: \min_{\mathcal{J}(\hat{M}_k)} \left| \left(\frac{1}{4},\frac{1}{4},\frac{1}{4}\left|\frac{1}{4},\frac{13}{16},\frac{1}{4}\right|\frac{1}{4},\frac{1}{4},\frac{1}{4}\right)\hat{M}_k - 1 \right| \qquad \text{and}$$
$$\mathsf{S9_{L4}}: \min_{\mathcal{J}(\hat{M}_k)} \left| \left(\frac{41}{64},\frac{41}{64},\frac{41}{64}\left|\frac{41}{64},\frac{1}{2},\frac{41}{64}\right|\frac{41}{64},\frac{41}{64},\frac{41}{64}\right)\hat{M}_k - 1 \right|.$$

Similar to the 5-point stencil, we can derive an isotropic mask which is related to e_S. Equalizing the sum of $r \cdot \mathsf{S9_{L2}}$ and $s \cdot \mathsf{S9_{L4}}$ we receive

$$\frac{3r}{4} + \frac{s}{2} \stackrel{!}{=} \frac{3r}{16} + \frac{41s}{64} \qquad \text{which leads to} \qquad r = \frac{s}{4}.$$

The isotropic entries become $a = b = \frac{11s}{16}$ and $\frac{5s}{4}$ for the right-hand side. We end up with an additional isotropic probing condition

$$\mathsf{S9_{L5}}: \min_{\mathcal{J}(\hat{M}_k)} \left| (1,1,1|1,1,1|1,1,1)\,\hat{M}_k - \frac{20}{11} \right|.$$

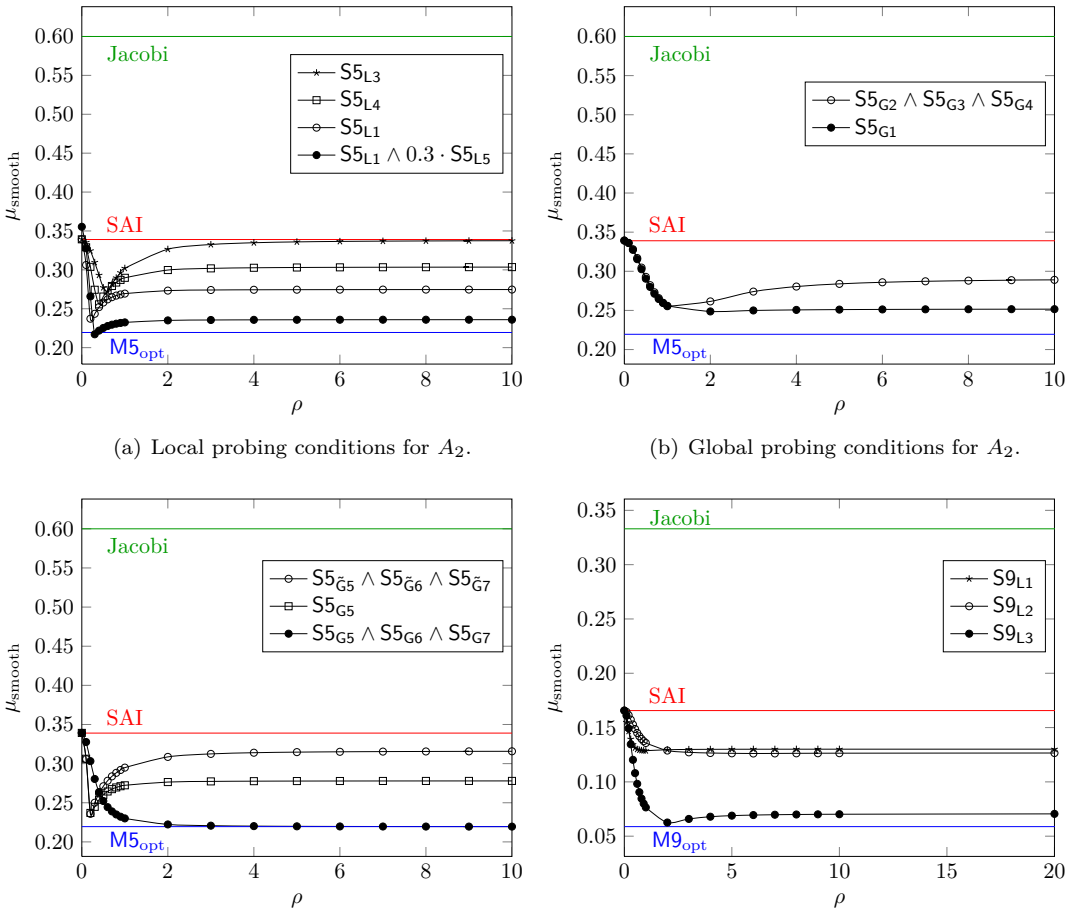
Numerical Experiment 5.6 Following Table 5.2 and Figure 5.2 (d), all local individual probing conditions are stable for increasing values of ρ and lead to a reduced smoothing factor compared to SAI. ●

Again, it is possible to derive global probing conditions, e.g., with action on A_3, by using Kronecker products of the 1D probing vectors. By observing $(e \otimes f)^T A\mathsf{M9_{opt}} = \beta(e \otimes f)^T$, similar to above, we can define conditions with $(e \otimes f)^T AM = \beta(e \otimes f)^T$ for some e, f, and β. Moreover, as nine DsOF are available, we are not restricted to Kronecker products between equal vectors. We have the possibility to use global probing conditions between combinations of them, e.g.,

$$(e_{N1} \otimes e_{N2})^T A_3 M \stackrel{!}{=} \left[(e_{N1} \otimes e_{N2})^T A_3 \mathsf{M9_{opt}}\right](e_{N1} \otimes e_{N2})^T.$$

5.3. Interpretation of the Results

In order to derive preconditioners with special behavior on certain subspaces we can translate analytic minimization problems in functions into MS(P)AI minimizations for vectors based on masks. In Chapter 6 we will apply the presented approach to regularization of discrete inverse ill-posed problems. For multigrid methods the results show that the smoothing property of approximate inverses like SAI can be improved significantly by using MSAI with appropriate probing masks or probing vectors. In special cases we can analytically determine the optimal approximate inverse smoother and its corresponding smoothing factor. SAI is far from being optimal in this class but including convenient individual or global probing conditions we can nearly reach the optimum with MSAI. Moreover, an increasing weighting of the subspace leads to stable behavior. Our tests on systems with varying coefficients and for the 2D case demonstrate that even the usage of global probing conditions with action on A, only satisfying the approximation $e^T AM \approx e^T$, reduce the smoothing factor in comparison to SAI and the damped Jacobi method. Note that the presented approach can be transferred analogously to the dynamic version (MSPAI) using pattern updates. Similar to the observations in [29] for SPAI, it is possible that MSPAI will also improve an MSAI smoother locally where needed.



(a) Local probing conditions for A_2.

(b) Global probing conditions for A_2.

(c) Global probing conditions with action on A_2.

(d) Local probing conditions for A_3.

Figure 5.2.: Smoothing factor μ_{smooth} against subspace weight ρ. MSAI is compared to Jacobi, SAI, $\mathsf{M5}_{\text{opt}}$ (5.12), and $\mathsf{M9}_{\text{opt}}$ (5.16) for various probing conditions. Both matrices A_2 and A_3 are of dimension $n = 1024$. Subfigure (d) illustrates the results of the first three conditions from Table 5.2.

Table 5.2.: Smoothing factors by using $\mathsf{M9}_{\text{opt}}$ (5.16), SAI, and MSAI with local probing conditions for the 2D 9-point stencil A_3 of dimension $n = 1024$.

$\mathsf{M9}_{\text{opt}}$	SAI	ρ	Local probing conditions				
			$\rho\mathsf{S9}_{\mathsf{L1}}$	$\rho\mathsf{S9}_{\mathsf{L2}}$	$\rho\mathsf{S9}_{\mathsf{L3}}$	$\rho\mathsf{S9}_{\mathsf{L4}}$	$\rho\mathsf{S9}_{\mathsf{L5}}$
0.0588	0.1657	0.2	0.147	0.161	0.150	0.116	0.095
		1.0	0.129	0.136	0.077	0.066	0.069
		2.0	0.130	0.129	0.063	0.063	0.069
		10.0	0.126	0.126	0.070	0.062	0.068
		100.0	0.130	0.127	0.071	0.062	0.068

Chapter 6

Improving the Solution of Discrete Ill-Posed Inverse Problems

This chapter deals with the improvement of the reconstruction of discrete ill-posed problems based on well-known regularization techniques. In the first part, we use MSAI's probing facility in order to construct regularizing preconditioners for iterative regularization methods. By acting as zero on the noise subspace they avoid the deterioration of the reconstruction by itself. Similar to Chapter 5, we analytically derive optimal preconditioners for a structured artificial 1D and 2D problem and translate the occurring minimization conditions into mask and global probing conditions afterward. For signals with strongly increasing values near the boundaries or interior discontinuities, we analyze the effect of a heuristically corrected probing subspace around special sections in the signal vector. Applying MSAI as regularizing preconditioner is original research published in [90].

As the quality of the reconstruction comes along with the choice of the regularization parameter its robust and accurate estimation is advantageous. We investigate a B-spline based approach for discrete L-curves and provide insight into the quality compared to a slightly adjusted discrepancy principle and state of the art L-curve algorithms. Motivated by special treatment of the MSAI components around signal instabilities, we present the effect of TPR, TSVD, and (P)CGLS when taking the whole signal data into account. For a certain class of problems, incorporating approximations of the signal data in form of a diagonal matrix improves the reconstruction, sometimes significantly. This effect can be enforced by applying the approach iteratively, i.e., construct a new diagonal matrix from an improved reconstruction and use it again for an additional improvement. In the end of this section, we will motivate for a partitioned reconstruction as a general improvement technique for ill-posed problems. Parts of the data based regularization approach can be found in [91].

In the last part, we pass on TPR in general form. We present seminorms depending on the underlying blur operator. The heuristic idea is to obtain a penalty term which is able to discriminate between the signal and the noise subspace, that is, which is able to better evaluate whether the reconstructed solution belongs more to the signal than to the noise. This operator dependent seminorm approach is published in [94]. The chapter ends by combining data based regularization and smoothing norms in regularization matrices for TPR which we recently analyzed in [93]. As several different topics are covered in this chapter, from which some are empirically-based heuristics, numerical experiments are provided throughout the various sections.

6.1. Regularization with Modified Sparse Approximate Inverses

An optimal preconditioner for iterative regularization methods should treat the large eigenvalues and have no effect on the smaller eigenvalues not amplifying the noise (see also Section 2.5.3). Since S(P)AI is an efficient smoother and satisfies the first condition, we propose MS(P)AI probing as regularizing preconditioner for general H to suppress a reconstruction on the noise space. For general matrices, where the eigendecomposition is unknown, it is usually impossible to derive preconditioners satisfying these conditions by manipulating the eigenvalues. We thus use the probing facility of MS(P)AI in order to derive a different approximation quality on the signal and noise subspace, respectively:

- For the signal space, we could use the vector e_S representing smooth components, and therefore the important part of the signal subspace. Here, we expect that the signal subspace is already taken into account by S(P)AI itself and thus we omit this probing facility.
- For the noise subspace, we use e_{N1}, e_{N2}, and e_{N3} as typical vectors related to fast oscillations. In case of using probing masks, $\hat{S}_k = (-1,1,-1)$ and $\hat{S}_k = (1,0,-1)$ are related to e_{N1} and e_{N2}, e_{N3}, respectively.

For higher dimensional problems, probing vectors typically result from a Kronecker product of 1D probing vectors. The conditions in MS(P)AI are given by $AM \approx I$ for the full approximation part—in order to derive a preconditioner of good quality and fast convergence on the signal subspace—and by $\rho e_N^T M \approx 0$ with $\rho > 0$ for the probing part—in order to avoid a deterioration of the reconstruction by M on the noise subspace. Similar to Chapter 5, we focus on a static MSPAI (MSAI) as regularizing preconditioner.

6.1.1. 1D Model Problems

We consider the matrix

$$H_1 := \text{tridiag}\left(\frac{1}{2},1,\frac{1}{2}\right) \qquad \text{related to the symbol} \qquad h_1(x) = 1 + \frac{e^{ix} + e^{-ix}}{2} = 1 + \cos(x).$$

As approximate inverse preconditioner we choose the trigonometric polynomial of symbol $m_1(x) = a + 2b\cos(x)$ related to a Toeplitz matrix $M = \text{tridiag}(b,a,b)$. The entries a and b should be determined such that the preconditioner M acts both nearly as the inverse on the signal subspace and as zero on the noise subspace. For $x \in \{0, \frac{\pi}{2}, \pi\}$, this leads to the minimization conditions

$$\begin{aligned} & \min_{a,b} \left\{ \left|m_1(0)h_1(0) - 1\right|, \left|m_1\left(\tfrac{\pi}{2}\right)h_1\left(\tfrac{\pi}{2}\right) - 1\right|, \rho\left|m_1(\pi)\right| \right\} \\ = \quad & \min_{a,b} \left\{ \left|2(a+2b) - 1\right|, \left|a - 1\right|, \rho\left|a - 2b\right| \right\}. \end{aligned} \tag{6.1}$$

By using the factor ρ, we introduce a weighting between the signal and the noise conditions. We obtain the optimal solution of (6.1) by solving the equality

$$2a + 4b - 1 = a - 1 = -\rho\,(a - 2b)\,.$$

For this purpose, we observe both

$$2a+4b-1 = a-1 \quad \text{leading to} \quad \min_{a,b}\{|4b+1|,6\rho|b|\} \tag{6.2}$$

$$\text{and} \quad 2a+4b-1 = 1-a \quad \text{leading to} \quad \min_{a,b}\left\{|a-1|,\rho\left|\frac{5a}{2}-1\right|\right\}. \tag{6.3}$$

Equalizing the conditions and inserting the solution into $|a-1|$ yields the optima

$$\left|-\frac{6\rho}{4-6\rho}\right|, \left|-\frac{6\rho}{4+6\rho}\right| \quad \text{for (6.2) and} \quad \left|-\frac{3\rho}{5\rho-2}\right|, \left|-\frac{3\rho}{5\rho+2}\right| \quad \text{for (6.3).}$$

In the limiting case for $\rho \to \infty$ they reach the values 1 and $\frac{3}{5}$, respectively. Hence, we consider a and b leading to the last. As $\frac{3\rho}{5\rho+2} \leq \frac{3\rho}{5\rho-2}$ holds for $\rho \geq 0$, we obtain the overall minimum for $(a,b) = \left(\frac{2+2\rho}{2+5\rho}, \frac{\rho-0.5}{2+5\rho}\right)$ resulting in the optimal regularizing preconditioner

$$\mathsf{M3}_\rho := \text{tridiag}\left(\frac{\rho-0.5}{2+5\rho}, \frac{2+2\rho}{2+5\rho}, \frac{\rho-0.5}{2+5\rho}\right). \tag{6.4}$$

For only preconditioning on the signal subspace ($\rho = 0$) we obtain $\mathsf{M3}_0 = \text{tridiag}\left(-\frac{1}{4}, 1, -\frac{1}{4}\right)$ while for only regularizing on the noise subspace

$$\lim_{\rho\to\infty} \mathsf{M3}_\rho = \text{tridiag}\left(\frac{1}{5}, \frac{2}{5}, \frac{1}{5}\right) = \frac{2}{5}H_1 = \frac{2}{5}H_1^T.$$

Therefore, the preconditioner for $\rho \to \infty$ is equivalent to the normal equations. With the parameter ρ we can choose between preconditioning on the signal subspace or suppression of noise.

Similar to Chapter 5, the conditions in (6.1) can be seen as conditions for the entries of the preconditioner directly, where a column k is described by the three entries in (5.8). Both the first and the second condition are covered by SAI and are not used in MSAI, as we can think of SAI approximation as acting mainly on the signal subspace. The translation of the last condition of (6.1) into an individual probing condition with probing mask $\hat{S}_k$ yields

$$\mathsf{R3_{L1}}: \min_{\mathcal{J}(\hat{M}_k)} \left|(-1,1,-1)\hat{M}_k\right|. \tag{6.5}$$

Our notation is related to Chapter 5: the subscript denotes either a local ($\mathsf{R*_{L*}}$) or global ($\mathsf{R*_{G*}}$) regularizing probing condition. It is possible to transfer the individual condition (6.5) to the global condition $e^T M \approx 0$ with probing vector e_{N1}, representing high frequency noisy components. Thus, we force M to act as approximately zero on the noise subspace. We refer to the high frequency global probing condition as

$$\mathsf{R3_{G1}}: \min_{\mathcal{J}(M)\in\mathcal{J}} \left\|e_{N1}^T M\right\|_2.$$

Numerical Experiment 6.1 In a first experiment, we are interested in the quality of $\mathsf{M3}_\rho$. We consider the ill-posed problem with operator H_1^8 and the original smooth signal

$$x_1 := \sin\left(\frac{4\pi j}{n}\right)_{j=1,\dots,n}$$

of size $n = 1\text{E}{+}03$ affected by Gaussian white noise with $\xi = 0.01\%$. By using powers of H, i.e., examining H_1^{2k} with $k > 0$, we make the problem more ill-conditioned and thus enforce the difference between the reconstructions of all methods. Here, $\kappa_2(H_1^8) = 1.365\text{E}{+}18$. As iterative regularization methods we use (P)CG and CGLS. Although, in general, an appropriate stopping criterion has to be applied, we assume to know the exact signal x and keep track of the relative reconstruction error (RRE) $\|x - \tilde{x}\|_2/\|x\|_2$ in order to observe the semiconvergent behavior.

Figure 6.1 shows that for increasing values of ρ the reconstruction remains stable and the region of optimal reconstruction quality is broader and smoother compared to unpreconditioned CG. Moreover, by putting more weight to the regularization property of $\mathsf{M3}_\rho$, it is possible to approximate the distribution of CGLS. We achieve almost similar reconstruction quality in fewer iterations (cf. Table 6.1). Hence, with this family of preconditioners we can steer the iteration to obtain faster convergence (small ρ) or slightly better reconstruction quality ($\rho > 1$). ●

Numerical Experiment 6.2 We incorporate the MSAI regularization property to reconstruct a perturbed signal. We ensure the MSAI M to be SPD via corresponding powers $(MM^T)^k$. As a more general problem we consider the reconstruction of x_1 blurred by the tridiagonal operator B_2 with varying coefficients. Its k^{th} row is given by

$$(B_2)_{k,:} := (0,\dots,0,w_{k-1},w_{k-1}+w_k,w_k,0,\dots,0) \qquad \text{for} \qquad w := \left(\frac{2j^2}{n^2}+1\right)_{j=0,\dots,n}$$

with $k \in \{1,\dots,n\}$. We choose $n = 1\text{E}{+}03$ and affect the signal with white noise of magnitude 1% and 0.1%, respectively.

Following Figure 6.2, CGLS yields both better reconstruction after slightly more iterations and smoother convergence compared to CG. In case of applying MSAI to CG, satisfying the global probing condition $\mathsf{R3}_{\mathsf{G1}}$, we are able to reconstruct x_1 with smaller error and in a smooth way. These positive effects can be enforced by applying MSAI to CGLS. The optimal reconstruction is achieved after 1 iteration while the convergence curve is much smoother and mostly below the curve of CG and CGLS, respectively. ●

6.1.2. Handling Discontinuities and Boundaries

The global reconstruction quality of signals with strongly increasing entries near the boundaries or nonanalytic interior points is usually significantly worse than the reconstruction of the corresponding discontinuous-free signal. A closer look at the error between the original and the reconstructed signal vector reveals that the signal is recovered usually very well for interior components but spoiled in the neighborhood of occurring discontinuities. Using MS(P)AI as regularizing preconditioner not only allows to incorporate filtering for noise reduction but can also be adjusted to the system matrix, e.g., to the blur operator, and to the data vector x. The deterioration of the reconstruction at discontinuities of x can be reduced by modifying the probing conditions relatively to the variation of the signal data. Our following numerical experiments illustrate that the development of regularizing preconditioners should also take into account the underlying signal structure, if possible.

Table 6.1.: RRE $\|x_1 - \tilde{x}_1\|_2/\|x_1\|_2$ for the 1D problem H_1^8 of size n = 1E+03 with signal x_1 affected by noise $\xi = 0.01\%$.

Regularization method		Minimal error	at iteration
CG		2.1870E−04	5
CGLS		1.8998E−04	62
PCG with $\mathsf{M3}_\rho$	$\rho = 0.4$	2.3699E−04	4
	$\rho = 1$	1.9746E−04	12
	$\rho = 5$	1.9236E−04	34
	$\rho = 100$	1.9226E−04	48

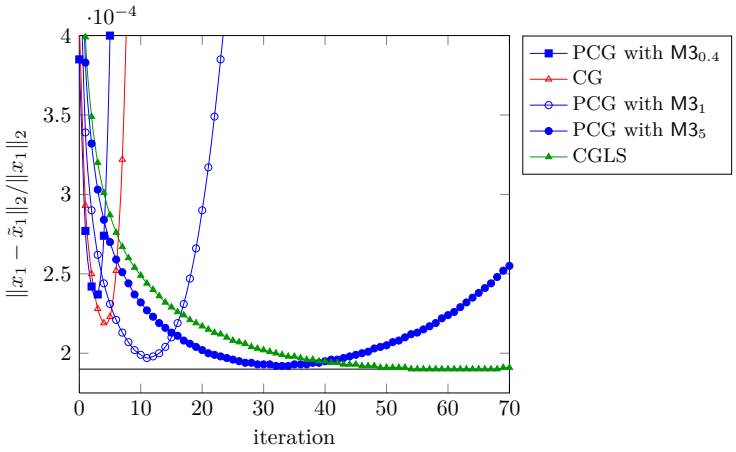


Figure 6.1.: RRE with respect to the iteration number for the 1D problem H_1^8 with signal x_1 of size n = 1E+03 affected by noise $\xi = 0.01\%$. CG is compared to PCG using the optimal preconditioner $\mathsf{M3}_\rho$ (6.4) for different weights ρ.

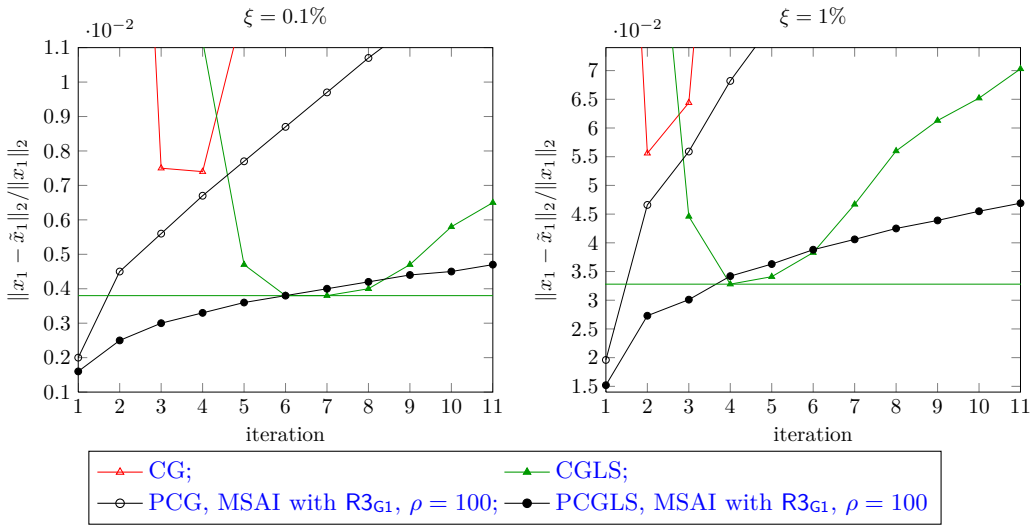


Figure 6.2.: RRE versus (P)CG(LS) iteration for the 1D operator B_2 and signal x_1 of size n = 1E+03 affected by noise $\xi = 0.1\%$ (left) and $\xi = 1\%$ (right), respectively.

Numerical Experiment 6.3 We observe the behavior for the blur operator H_1^4 with $\kappa_2(H_1^4) = 1.390\text{E+}18$ and a signal which has strongly increasing entries near the boundaries as well as an additional nonanalytic point in the middle:

$$x_2 := \left(\frac{1}{j} - \frac{1}{n+1-j} + \frac{1}{\frac{n}{2}+\frac{1}{4}-j} + \sin\left(\frac{4j\pi}{n}\right)\right)_{j=1,\dots,n}.$$

The problem is chosen to be of size $n = 1\text{E+}03$ and x_2 is perturbed by white noise of magnitude 0.1%. We incorporate some correction at the boundary by changing the components $(e_{N1})_1 = \tau$ and $(e_{N1})_n = \pm\tau$ in which τ is a weight factor of heuristic choice, in most cases $\tau \in [0,1]$ is sufficient. Additionally, we correct our probing subspace in the middle by $(e_{N1})_{\frac{n}{2}} = 0$ and $(e_{N1})_{\frac{n}{2}\pm1} = \frac{1}{2}(e_{N1})_{\frac{n}{2}\pm1}$, again with heuristic weight $\frac{1}{2}$.

Figure 6.3 and Table 6.2 show that PCG with MSAI reaches its optimum with slightly smaller value after more iterations but in a stable and smoother semiconvergent manner compared to CG. The reconstruction $\tilde{x}_2$ is by a factor of 2.86 times more accurate when using boundary corrections and 4.72 times when using both corrections within the probing subspace of MSAI. ●

Likewise, it is possible to use subspace corrections within mask probing. We change the individual probing condition $\mathsf{R3_{L1}}$ for the first and last column of the preconditioner in order to take into account the missing value -1 which lies outside the vector. This lost information can be incorporated by using the probing masks

$$\hat{S}_1 = \hat{S}_n = [1-\tau, -1+(-1+\tau)] = (1-\tau,\tau-2)$$

for the first and the last column. Note that interior discontinuities can be treated similarly by modifying the related masks. Furthermore, it is also possible for $\mathsf{M3}_\rho$ to build in similar corrections, e.g., by weighting the nondiagonal entries.

Numerical Experiment 6.4 We consider the problem `deriv2(n,2)` from [68]. It results from a discretization of a Fredholm integral equation of the first kind (2.28) whose dense operator H_d is the Green's function for the second derivative. I.e., we reconstruct the problem

$$H_\mathsf{d}(s,t) = \begin{cases} s(t-1), & s < t \\ t(s-1), & s \geq t \end{cases} \quad \text{with} \quad b_\mathsf{d}(s) = e^s + (1-e)s - 1 \quad \text{and} \quad x_\mathsf{d}(t) = e^t$$

being the right-hand side and the exact signal, respectively. Our problem has size $n = 2\text{E+}03$ and x_d is affected by $\xi = 0.001\%$ and $\xi = 0.01\%$. We use (P)CGLS and force the MSAI to act as approximately zero on the noise subspace by incorporating the high frequency probing conditions $\mathsf{R3_{L1}}$ and

$$\mathsf{R3_{L2}}: \min_{\mathcal{J}(\hat{M}_k)} \left|(1,0,-1)\hat{M}_k\right|$$

simultaneously. We weight the subspace with $\rho = 1\text{E+}03$ and symmetrize the preconditioner via $M + M^T$. To avoid deterioration at the boundaries we adjust M by resetting the values

$$M_{1,1} = M_{2,2}, \quad M_{2,1} = M_{3,2}, \quad M_{n,n} = M_{n-1,n-1}, \quad \text{and} \quad M_{n-1,n} = M_{n-2,n-1}. \tag{6.6}$$

Following Figure 6.4, applying regularizing conditions within an MSAI with corrected boundary values yields an improved reconstruction in fewer iterations. We observed similar behavior for other noise levels and for the problem `deriv(n,1)` as well. ●

Table 6.2.: Optimal RRE $\|x_2-\tilde{x}_2\|_2/\|x_2\|_2$ for the 1D problem H_1^4 with signal x_2 of size n = 1E+03 and white noise of magnitude 0.1%. $\rho = 100$.

Regularization method		Minimal error	at iteration
Tikhonov-Phillips ($\alpha = 0.3$)		0.1286	–
CG		0.1240	8
PCG with ρR3$_{\text{G1}}$	no correction	0.1210	130
	$\tau = 0.25$	0.0433	58
	$\tau = 0.25$, middle correction	0.0263	52

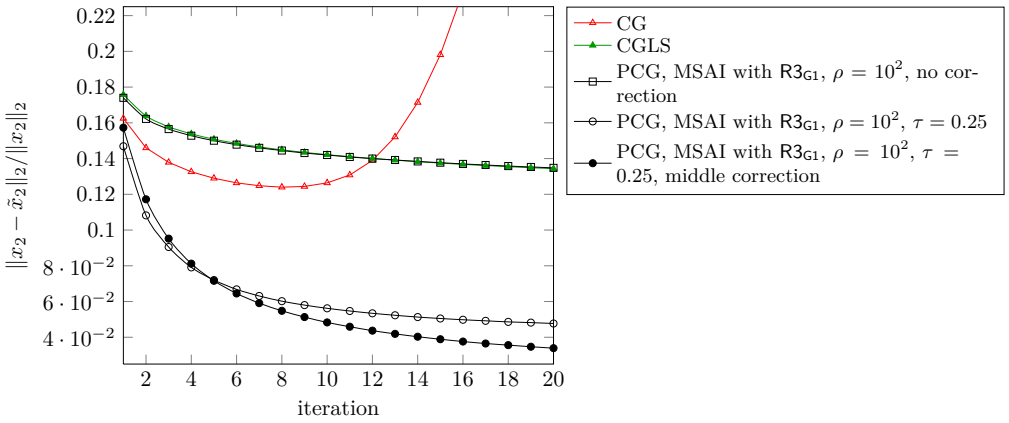


Figure 6.3.: RRE with respect to iteration number for H_1^4 of dimension n = 1E+03 and the signal x_2 affected by $\xi = 0.1\%$. CG is compared to CGLS and to PCG using MSAI with R3$_{\text{G1}}$ and $\rho = 100$.

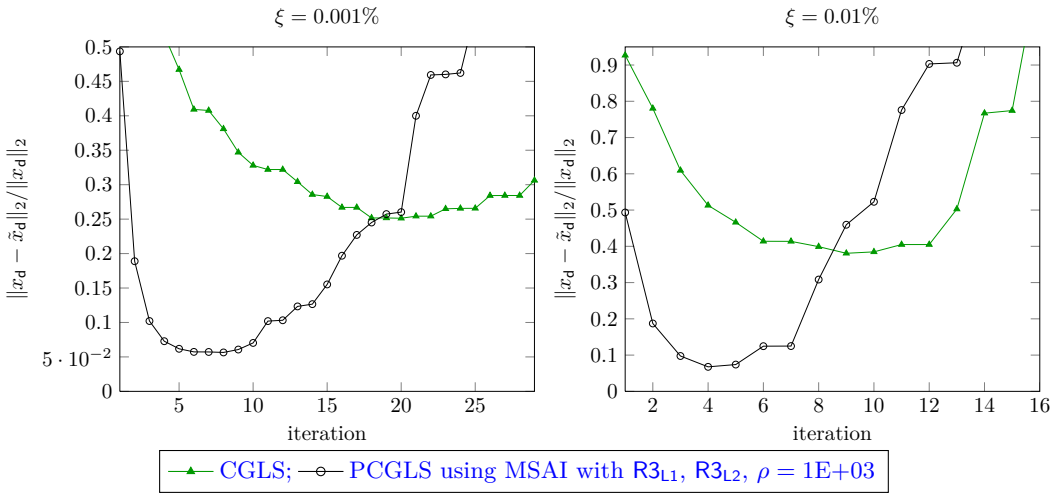


Figure 6.4.: RRE with respect to iteration number for the `deriv2(2000,2)` problem from [68]. x_{d} is affected with $\xi = 0.001\%$ (left) and $\xi = 0.01\%$ (right), respectively. MSAI is symmetrized via $M + M^T$.

Remark 6.1 *We also applied corrections according to (6.6) within the derived optimal preconditioner* $\mathsf{M3}_\rho$*. For signals containing the mentioned instabilities, we observed improved reconstructions similar to MSAIs with corrected subspaces.*

6.1.3. 2D Model Problems

In 2D we consider the matrix

$$H_2 := \frac{H_1 \otimes I + I \otimes H_1}{2} \;=\; \text{blocktridiag}\left(0,\frac{1}{4},0\left|\frac{1}{4},1,\frac{1}{4}\right|0,\frac{1}{4},0\right)$$

with its corresponding symbol $h_2(x,y) = 1 + \frac{\cos(x)}{2} + \frac{\cos(y)}{2}$ and preconditioner $m_2(x,y) = a + 2b[\cos(x) + \cos(y)]$. With $\cos(x) + \cos(y) =: h(x,y) \in [-1,2] =: G_3$ we have to minimize

$$\min_{a,b} \max_{x,y \in G_3} \left|1 - [a + 2bh(x,y)]\left[1 + \frac{h(x,y)}{2}\right]\right|$$

relative to the signal subspace and $\min_{a,b} \rho|a - 4b|$ for $h(x,y) = -2$ relative to the noise subspace. The minimization problem which is to be solved reads as

$$\min_{a,b} \left\{|1 - 2a - 8b|\,, \left|1 - \frac{a}{2} + b\right|, \rho\,|a - 4b|\right\}.$$

Similar to the 1D case, we observe both

$$1 - 2a - 8b \;=\; 1 - \frac{a}{2} + b \quad \text{leading to} \quad \min_{a,b}\{|1 + 4b|,10\rho|b|\} \tag{6.7}$$

$$\text{and} \quad 1 - 2a - 8b \;=\; \frac{a}{2} - b - 1 \quad \text{leading to} \quad \min_{a,b}\left\{\left|\frac{3}{5} + \frac{12b}{5}\right|, \rho\left|\frac{4}{5} - \frac{34b}{5}\right|\right\}. \tag{6.8}$$

Equalizing these conditions and inserting into $\left|1 - \frac{a}{2} + b\right|$ yields the optima

$$\left|\frac{4 - 5\rho}{2 - 5\rho}\right|, \left|\frac{4 + 5\rho}{2 + 5\rho}\right| \quad \text{for (6.7) and} \quad \left|\frac{15\rho}{17\rho + 6}\right|, \left|\frac{15\rho}{17\rho - 6}\right| \quad \text{for (6.8).}$$

In the limiting case $\rho \to \infty$ they reach the values 1 and $\frac{15}{17}$, respectively. Considering a and b leading to the two latter optima and observing $\frac{15\rho}{17\rho+6} \leq \frac{15\rho}{17\rho-6}$ for $\rho \geq 0$, we obtain the optimal regularizing block tridiagonal preconditioner

$$\mathsf{M5}_\rho := \text{blocktridiag}\left(0, 2\rho - \frac{3}{2}, 0\left|2\rho - \frac{3}{2}, 9 + 8\rho, 2\rho - \frac{3}{2}\right|0, 2\rho - \frac{3}{2}, 0\right). \tag{6.9}$$

The individual probing masks for a reduced inner column of the preconditioner described by (5.13) as well as the high frequency probing vector for the 2D case can be derived via Kronecker products of the 1D probing vectors. Thus, we obtain respectively the individual and global regularizing probing condition

$$\mathsf{R5_{L1}}: \min_{\mathcal{J}(\hat{M}_k)} \left|[(-1,1,-1) \otimes (-1,1,-1)]\hat{M}_k\right| \quad \text{and} \quad \mathsf{R5_{G1}}: \min_{\mathcal{J}(M) \in \mathcal{J}} \left\|(e_{N1} \otimes e_{N1})^T M\right\|_2.$$

Numerical Experiment 6.5 We are interested in the reconstruction quality using both $\mathsf{M5}_\rho$ and MSAI incorporating $\mathsf{R5}_{\mathsf{G1}}$. We choose the artificial 2D operator H_2^4 with $\kappa_2(H_2^4) =$ 1.232E+12 and the signal x_1 of size $n = 50^2$ affected by white noise of magnitude 0.1%.

Referring to Figure 6.5, we are able to approximate the convergence of CGLS with increasing values of ρ in $\mathsf{M5}_\rho$. Applying MSAI with $\mathsf{R5}_{\mathsf{G1}}$ it is almost possible to reach the optimal value but in less iterations. Higher values of ρ lead to smooth and broad convergence curves similar to $\mathsf{M5}_\rho$. CG has its optimal value after 9 iterations with error 2.219E−03 while preconditioning with MSAI using $\mathsf{R5}_{\mathsf{G1}}$ and $\rho = 1$ after 12 iterations with value 1.496E−03. CGLS reaches the error 1.282E−03 after 44 iterations. ●

Numerical Experiment 6.6 Finally, we consider the `blur` problem [68] which is deblurring images degraded by atmospheric turbulence blur. The operator H_{b} is an $n^2 \times n^2$ symmetric, doubly block Toeplitz matrix that models blurring of an $n \times n$ image by a Gaussian point spread function

$$G(x, y) := \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}. \tag{6.10}$$

The parameter σ controls the width of H_{b} and thus the amount of smoothing and ill-posedness. $H_{\mathsf{b}} = A_1 \otimes A_1$ is symmetric block banded and possibly positive definite depending on n and σ. We choose $n = 150$, bandwidth 4, and $\sigma = 1$, i.e., we invoke `blur(150,4,1)`, where H_{b} is of size $150^2 \times 150^2$. The original signal is denoted by x_{b}. In case of preconditioning H_{b}, MSAI is symmetrized via $M + M^T$ while for $H_{\mathsf{b}}^T H_{\mathsf{b}}$ the preconditioner $M^T M$ is applied. For MSAI we impose the blocktridiagonal pattern according to (5.13). In view of the structure of H_{b}, we build the high frequency subspace by Kronecker products of oscillatory probing vectors in the regularizing global conditions $\mathsf{R5}_{\mathsf{G1}}$,

$$\mathsf{R5}_{\mathsf{G2}} : \min_{\mathcal{J}(M)\in\mathcal{J}} \left\|(e_{N1} \otimes e_{N2})^T M\right\|_2, \qquad \text{and} \qquad \mathsf{R5}_{\mathsf{G3}} : \min_{\mathcal{J}(M)\in\mathcal{J}} \left\|(e_{N2} \otimes e_{N1})^T M\right\|_2,$$

all weighted with $\rho = 1$. Following Figure 6.6 and Table 6.3, we obtain a better reconstruction when applying MSAI in contrast to CG(LS). Note that we observed similar behavior for other bandwidths and values of σ as well. ●

Table 6.3.: Optimal RRE $\|x_{\mathsf{b}} - \tilde{x}_{\mathsf{b}}\|_2 / \|x_{\mathsf{b}}\|_2$ at given iteration for the `blur(150,4,1)` problem with white noise of magnitude 0.01%, 0.1%, 1%, and 10%. $\rho = 1$.

Regularization method	$\xi = 0.01\%$		$\xi = 0.1\%$	
	optimal RRE	at it.	optimal RRE	at it.
CG	25.637	3	41.828	1
CGLS	24.123	19	37.807	2
PCG, MSAI with $\mathsf{R5}_{\mathsf{G1,G2,G3}}$	23.620	6	39.007	1
PCGLS, MSAI with $\mathsf{R5}_{\mathsf{G1,G2,G3}}$	23.778	51	38.023	3
Regularization method	$\xi = 1\%$		$\xi = 10\%$	
	optimal RRE	at it.	optimal RRE	at it.
CG	591.0	1	13.820E+03	1
CGLS	82.5	1	1.253E+03	1
PCG, MSAI with $\mathsf{R5}_{\mathsf{G1,G2,G3}}$	189.8	1	4.104E+03	1
PCGLS, MSAI with $\mathsf{R5}_{\mathsf{G1,G2,G3}}$	69.7	1	0.962E+03	1

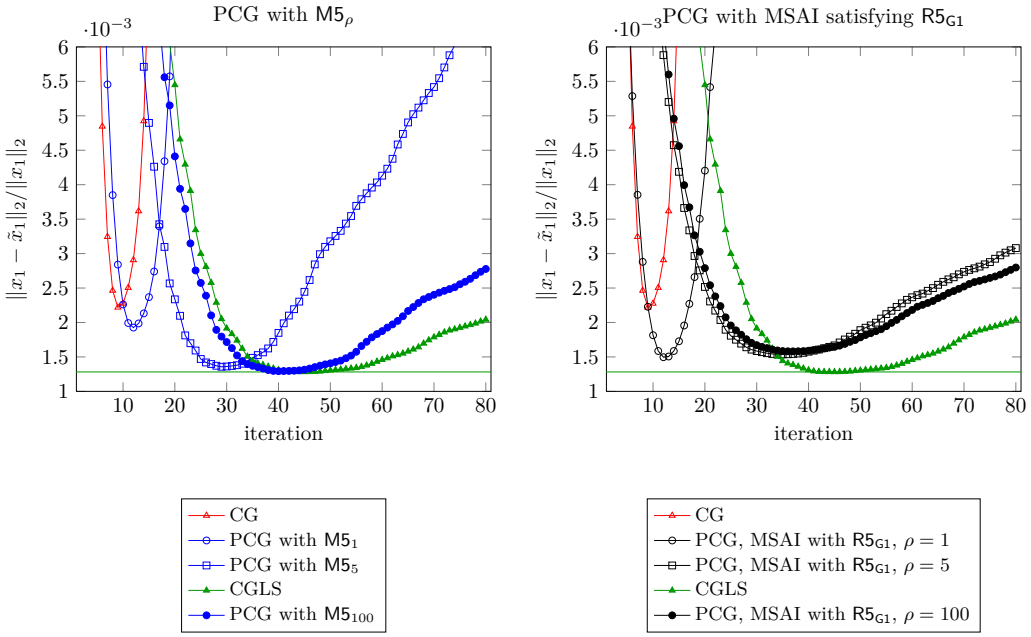


Figure 6.5.: RRE with respect to iteration number for the operator H_2^4 of size $n = 50^2$ and signal x_1 affected by $\xi = 0.1\%$.

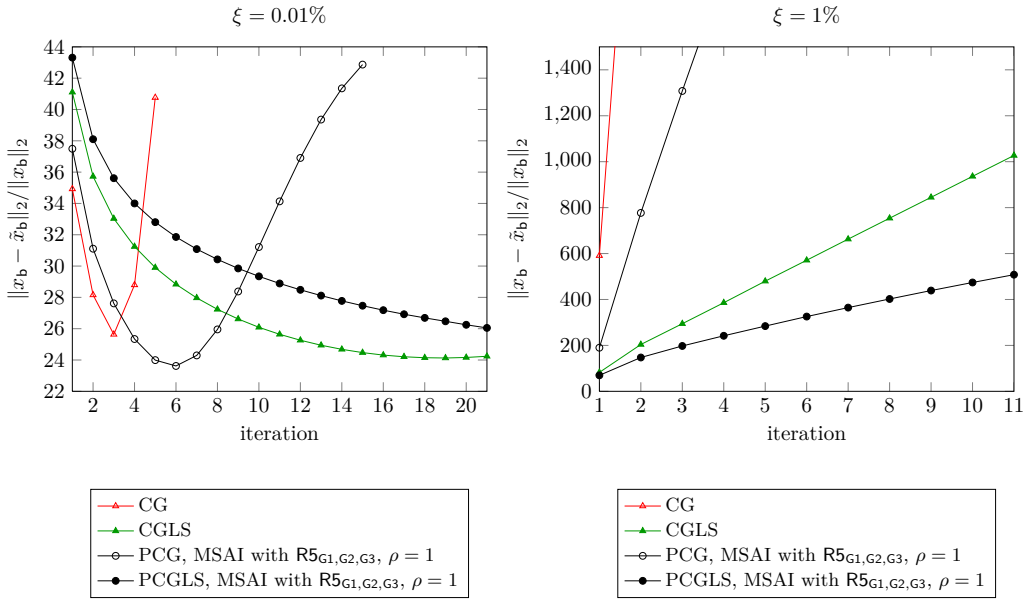


Figure 6.6.: RRE with respect to iteration number for the `blur(150,4,1)` problem from [68]. Noise is of magnitude 0.01% (left) and 1% (right), respectively. MSAI is used as $M + M^T$ within PCG and as $M^T M$ within PCGLS.

Interpretation of the results. For the recovery process of blurred signals, a different subspace approach than in Chapter 5 becomes necessary. For structured operators, we can analytically derive regularizing approximate inverse preconditioners based on generating functions, or by incorporating masks or probing vectors in MSAI. We can adjust these preconditioners to the system matrix and reduce the deterioration by noise or at signal instabilities by noise conditions and their modification relative to the variation of the signal. Applying them in PCG or PCGLS, we observe faster convergence or a better reconstruction compared to the unpreconditioned case.

6.2. Estimating the Regularization Parameter for CGLS

The quality of the reconstruction of discrete ill-posed problems heavily depends on the estimation of the optimal regularization parameter. Numerical examples where the solution is computed iteratively by a Krylov subspace method (e.g., CGLS) using an appropriate stopping criterion are less frequently taken into consideration. We analyze a discrete L-curve approach (see Section 2.5.4) based on smoothing and B-splines and provide a comparison to the AdaptivePruning technique [73] and to a slightly adjusted discrepancy principle [60]. Note that the two former approaches can be used to estimate the optimum of direct regularization methods as well, as long as discrete L-curves are available.

Our approach is slightly based upon the Algorithm FindCorner from [75] as we use some sort of spline method combined with a smoothing step. As a first step, we iterate to a sufficient large number of iterations within (P)CGLS, for example, $n = 200$, and compute the points $P_k = \left(\|r^{(k)}\|_2, \|\tilde{x}^{(k)}\|_2\right)$. Following [75], for many problems it is advantageous to use an intuitive logarithmic scaling of the P_k to emphasize the flat branches and the corner of the L-curve. However, when using (P)CGLS, we observed that this is no mandatory prerequisite to get well shaped L-curves. Here, our experiments show that the usage of a linear scaling may also yield robust behavior of the algorithms. Using P_k we build up a cubic spline interpolant in B-form. The B-form of our (univariate) piecewise polynomial function f is specified by its knot sequence $\|r^{(k)}\|_2$ and by its B-spline coefficient sequence. We use B-splines [121] to receive local support along the domain to overcome the drawback that local variations in the P_k have influence on the whole function degrading the shape of the curve. We refer to the approach as B-spline approach.

Due to numerical errors, local jumps in the curve lead to undesired or useless B-splines, falsifying the true corner of the curve as here the curvature locally deviates from the intrinsic one. Therefore, we provide a smoothing of the original data P_k. As we observed different data ranges along the branches of the L-curve for some model problems, we perform smoothing of different magnitude along the abscissa and the ordinate via

$$\tau_x := c \cdot 10^{-2} \left| \|r^{(1)}\|_2 - \|r^{(n)}\|_2 \right| \qquad \text{and} \qquad \tau_y := c \cdot 10^{-2} \left| \|\tilde{x}^{(1)}\|_2 - \|\tilde{x}^{(n)}\|_2 \right|.$$

The constant c is a heuristic value and can be determined by observing the shape of the L-curve. Most of our experiments revealed $c = 10^{-2}$, i.e., almost always we use 0.01% of smoothing to receive smoother L-curves. If the residual-norm distance between two neighboring points P_i and P_j is smaller than their solution-norm distance, then we use the smoothing criterion

$$\left| \|r^{(i)}\|_2 - \|r^{(j)}\|_2 \right| < \tau_x, \qquad \text{otherwise} \qquad \left| \|\tilde{x}^{(i)}\|_2 - \|\tilde{x}^{(j)}\|_2 \right| < \tau_y.$$

If the criterion is satisfied, we include the point between P_i and P_j into the new smoothed point set S, instead of P_i and P_j. See Figure 6.7 for an illustration. Note that the smoothing approach could be performed several times if the shape is still not smooth enough. For some model problems an additional post-smoothing step is helpful. Therefore, we remove close-by points from S which either in their residual-norm or solution-norm distance are closer to each other than 1E−08. Based on this new set of smoothed points S, we build the new B-spline interpolant with its corresponding piecewise polynomial function f_s. As the optimal number of iterations for (P)CGLS is located at the point with maximum curvature, we compute the curvature at every sample S_k via

$$\kappa(S_k) = \frac{f_s''(\|r^{(k)}\|_2)}{\left[1 + f_s'(\|r^{(k)}\|_2)^2\right]^{\frac{3}{2}}}.$$

Hence, the corner is located at the point with absolute maximum curvature

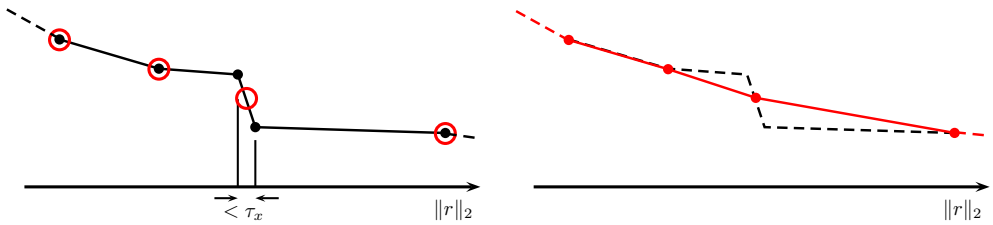
$$S_{\text{opt}} = \arg\max_k(|\kappa(S_k)|).$$

Due to the smoothing approach, it is necessary to perform a back mapping of S_{opt} to locate P_{opt} on the original discrete L-curve. For this purpose, we identify the points P_i and P_j which are closest to S_{opt} via their residual-norm distance. The true corner P_{opt} is then the point with bigger absolute curvature when evaluated on f_s.

If the perturbation norm $\|e\|_2$ (or ξ) is known within reasonable accuracy in advance, we are interested in using an adjusted discrepancy principle as a stopping heuristic in (P)CGLS and its qualitative comparison to estimators based on L-curves such as the AdaptivePruning algorithm or our B-spline approach. As in (P)CGLS the series $\{r^{(k)}\}$ is monotonically decreasing, we terminate the reconstruction at iteration k if

$$\|r^{(k-1)}\|_2 - \|r^{(k)}\|_2 < \xi \tag{6.11}$$

is satisfied. This criterion is much cheaper than, for instance, the L-curve criterion because it can be computed during each iteration. Moreover, there is no need for another invocation with the estimated k or to store every solution along the estimation process, as, here, after (P)CGLS returns, the solution is immediately available. To simplify our notation, we refer to the criterion in (6.11) as DiscrepancyPrinciple in the following.



(a) Local jump which satisfies the τ_x criterion. (b) Smoothed point set S.

Figure 6.7.: Smoothing step on cut-out of artificial discrete L-curve. The local jump would lead to undesired B-splines. Applying the τ_x criterion, the new point set S will lead to a smoother B-spline interpolant.

Numerical Experiment 6.7 We compare the estimation quality between our B-SPLINE approach, the ADAPTIVEPRUNING algorithm, and the DISCREPANCYPRINCIPLE. We use a similar test scenario and the notation as presented in [73] but use CGLS to compute the solution of our test problems. As illustrated in Table 6.4, our test problems are mainly chosen from [68], except No. 14 and No. 15 which are two ill-conditioned coefficient matrices from [108]. For the `lotkin` operator we use the smooth signal $(x)_j = \sin\left(\frac{j\pi}{n}\right)_{j=1,\dots,n}$ with a discontinuity in the middle $(x)_{\frac{n}{2}} = 4$. For the `prolate` operator we use the constant pulse sequence $(x)_{100\cdot i} = 200$, $i = 1,\dots,10$. To evaluate the quality of the reconstructions, we constitute $\tilde{x}_{k_{\text{opt}}}$ as the OPTIMAL solution where $k_{\text{opt}} := \arg\min_k \|x - \tilde{x}_k\|_2$. We fix our problem size to $n = 1024$ and for each test problem p we compute regularized solutions for 10 different magnitudes of noise:

$$\xi \in \{7 \cdot 10^{-d}, 4 \cdot 10^{-d}, 1 \cdot 10^{-d}, 7 \cdot 10^{-6}\} \qquad \text{with} \qquad d \in \{3,4,5\}. \tag{6.12}$$

Similar to [73], we measure the quality of the solutions using the metric

$$Q_{p,\xi} = \frac{\|x - \tilde{x}_k\|_2}{\|x - \tilde{x}_{k_{\text{opt}}}\|_2}.$$

The minimum value $Q_{p,\xi} = 1$ is optimal, and values $Q_{p,\xi} > 100$ are considered off the scale and are set to 100.

Following Figure 6.8, the B-SPLINE approach produces solutions which are sometimes off the scale. This we observed, similar to [73], for the `l_corner` approach as well. That is, because a parametric spline is a function sensitive to its knot distribution making maximum curvature based methods fit only the local behavior of the L-curve and thus sometimes produce poor estimations. For the problems No. 8, 9, and 10 this approach seems to work better while for No. 3, 7, and 13 it produces worse solutions compared to the ADAPTIVEPRUNING algorithm.

The most stable behavior among all problems results from the DISCREPANCYPRINCIPLE. However, we observed that both the B-SPLINE and the ADAPTIVEPRUNING algorithm may produce better estimations. The bottom plot of Figure 6.8 illustrates a zoomed in view, where all values $Q_{p,\xi} > 10^{\frac{1}{10}}$ are set to $10^{\frac{1}{10}}$. Besides the problems No. 9, 10, 14, the results are more accurate using the ADAPTIVEPRUNING algorithm. Nevertheless, if the perturbation norm or ξ are known in advance—and especially for problems where the optimum is reached after few iterations ($\lesssim 30$)—the criterion in (6.11) provides a cheap and robust estimation of k, yielding tolerable solutions while circumventing the drawbacks of an L-curve approach. ●

Table 6.4.: Test problems used for the comparison of the estimation approaches. All problems from [68] use the default solution. For `lotkin` we use the signal $(x)_j = \sin\left(\frac{j\pi}{n}\right)_{j=1,\dots,n}$, $(x)_{\frac{n}{2}} = 4$ and for `prolate` the constant pulse sequence $(x)_{100\cdot i} = 200$, $i = 1,\dots,10$.

No.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Name	`blur`	`deriv2-1`	`foxgood`	`gravity-1`	`spikes`	`shaw`	`wing`	`phillips`	`heat`	`tomo`	`ilaplace-1`	`gravity-3`	`baart`	`lotkin`	`prolate`
Type	Regularization Tools [68]													[108]	

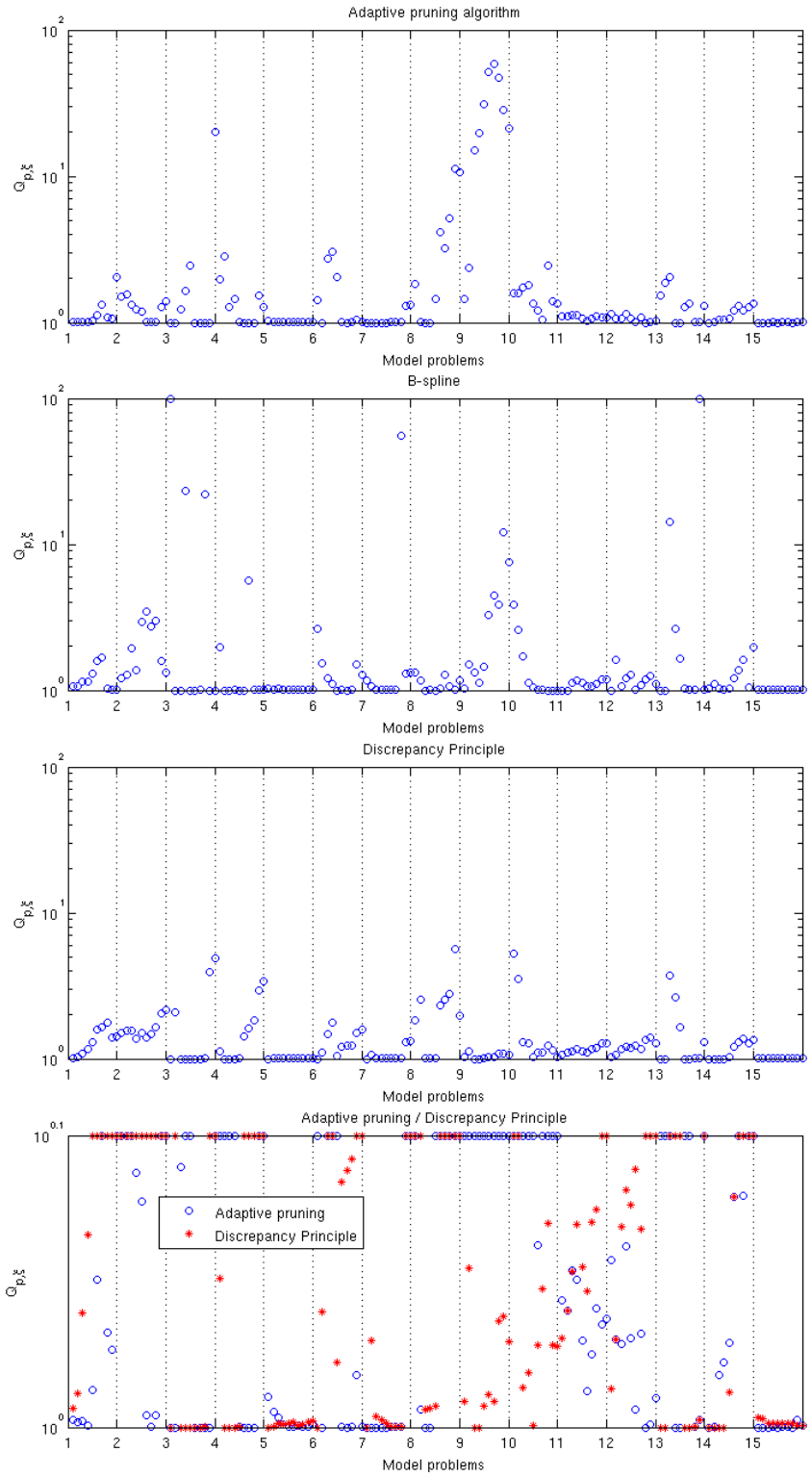


Figure 6.8.: Measure of the quality metric $Q_{p,\xi}$ for the estimation approaches used to reconstruct all model problems from Table 6.4 for ten realizations of ξ (6.12) and fixed problem size $n = 1024$. A measure of one is optimal. All values above 100 are set to 100.

Interpretation of the results. Similar to [73] for large-scale problems, we observed in some cases that the real optimum is not located at the corner of the L-curve, but more outside along its branches. This may bring up a deviation from the optimum for L-curve based algorithms. Similar to direct regularization techniques, the estimation of the number of iterations for CGLS is sensible and depends on the smoothness and the characteristic shape of the L-curve which is not always guaranteed to be satisfied. Further difficulties may appear for model problems with slowly decaying singular values where the corner of the curve becomes less distinct. Moreover, in many practical problems the L-curve may totally lose its characteristic shape. For model problems where the reconstruction leads to an unspoiled L-curve, our results indicate that the B-SPLINE approach is able to provide a good estimation of the true optimum. However, it involves smoothing parameters which might require adjustment for certain problem settings, in contrast to the ADAPTIVEPRUNING algorithm. For more results, see Section 6.3 where we also apply the approaches within TPR and image reconstruction.

6.3. Data Based Regularization for Discrete Deconvolution Problems

6.3.1. Motivation for Incorporating the Signal Data

Numerical Experiment 6.3 shows that the preconditioner should also take into account the behavior of the original or blurred data vector. Smoothing, e.g., with $M = \text{tridiag}(\frac{1}{2},1,\frac{1}{2})$ makes sense to remove noisy components only as long as the data is continuous. At discontinuities smoothing would cause additional errors. We may use a modified tridiagonal smoothing preconditioner with j^{th} row $(0,\ldots,0,r_{j-1},1,r_j,0,\ldots,0)$ to obtain $r_j \approx \frac{1}{2}$ near continuous components, but $r_j \approx 0$ near discontinuities. Hence, it may help to incorporate the behavior of the original signal x or some approximation from previous steps, e.g., the blurred data vector b, defining the preconditioner M_b with j^{th} row

$$(M_b)_{j,:} := (0,\ldots,0,r_{j-1},1,r_j,0,\ldots,0) \text{ for } r := \left(\frac{1}{2}\cdot\frac{1}{1+(\rho\,|b_j-b_{j+1}|)^k}\right)_{j=1,\ldots,n-1}. \quad (6.13)$$

The parameters ρ and k have to be chosen in such a way that discontinuities are revealed as good as possible. Note that if the original data x is known, we could define M_x according to (6.13) with $k = 1$ and incorporating x-components instead of b-components. Furthermore, we can consider an iterative process where a first approximation x_1 on x is used to define the tridiagonal preconditioner M_1 delivering an improved approximation x_2 which again gives a new preconditioner M_2, and so on.

Numerical Experiment 6.8 We are interested in the behavior applying M_b and M_x as smoothing preconditioners within CG and CGLS. We reduce the `blur` problem from [68] to the 1D case and denote it `blur1D`. Hence, we consider as blur operator the 1D analogon $H_{\texttt{b1D}}$ of (6.10) and reduce the 2D right-hand side to appropriate size, referring to it as $x_{\texttt{b1D}}$. We choose $n = 1\text{E}{+}03$, $\xi = 0.1\%$, bandwidth 4, and the variances $\sigma = 1$ and $\sigma = 10$, respectively. Following Figure 6.9, adjusting the preconditioner relative to discontinuities of x improves the reconstruction. Note that ρ and k are chosen heuristically. ●

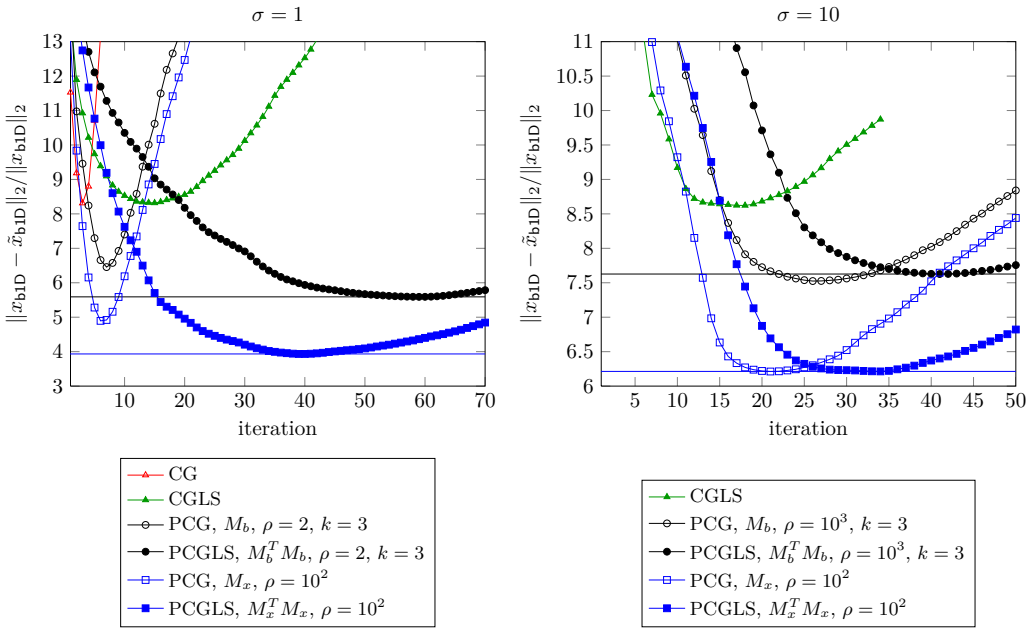


Figure 6.9.: RRE with respect to iterations for the `blur1D(1000,4,`σ`)` problem with $\xi = 0.1\%$. The blur operator $H_{\mathtt{b1D}}$ has $\sigma = 1$ (left) and $\sigma = 10$ (right), respectively.

Applying smooth preconditioners to eminently continuous data does not destroy the reconstruction process and may lead to slight quality improvement for certain problems. However, our experiments revealed that for signals which consist of nearly zero components and are only weakly blurred, i.e., if the signal structure is preserved, an incorporation of the signal data in form of a diagonal matrix leads to better reconstruction results. It is important to point out that the idea of estimating the signal structure from the data may not work for inverse problems where these two domains—the one containing the exact signal and the one containing the observed signal—are fundamentally different.

In the following, we give a motivation for the effectiveness of taking the signal values into account. Assume

$$D := \operatorname{diag}[(x+\theta)_1, \ldots, (x+\theta)_n] \tag{6.14}$$

is the diagonal preconditioner built from an approximation $x+\theta$ of x, where x is the original (exact) signal and θ is some deviation from it. Note that $x + \theta$ can, but must not be the observed right-hand side b. The computed solution $\tilde{x}$, i.e., the reconstruction, can be written as $\tilde{x} = x + D\delta$, with the deviation $D\delta$.

Theorem 6.1 *Assuming $(x+\theta)_{j=1,\ldots,n} \neq 0$, the component-wise relative error with respect to the approximation $x + \theta$ of the reconstruction $\tilde{x}$ of the unregularized equation $H^T H\tilde{x} = H^T b$ satisfies*

$$\left| \frac{(\tilde{x} - x)_j}{(x+\theta)_j} \right| = \left| (U\Sigma^{-1}V^T\eta)_j \right|,$$

where $U\Sigma V^T$ is the spectral decomposition of DH^T.

Proof Applying (6.14) to the unregularized equation, we observe

$$\begin{aligned} H^T H\tilde{x} = H^T b &\Leftrightarrow (DH^T HD)D^{-1}\tilde{x} = DH^T b &\Leftrightarrow \\ (DH^T HD)D^{-1}(x+D\delta) &= DH^T(Hx+\eta) &\Leftrightarrow \\ (DH^T HD)D^{-1}(x+\theta-\theta+D\delta) &= DH^T[HDD^{-1}(x+\theta-\theta)+\eta]. \end{aligned}$$

Using the identity $D^{-1}(x+\theta) =: \vec{1}$ and the SVD of $DH^T = U\Sigma V^T$, we obtain

$$\begin{aligned} (DH^T HD)(\vec{1}+\delta-D^{-1}\theta) = DH^T(HD\vec{1}+\eta-H\theta) &\Leftrightarrow \\ DH^T HD\delta = DH^T\eta &\Leftrightarrow \\ \Sigma^2 U^T\delta = \Sigma V^T\eta &\Leftrightarrow \\ \delta = U\Sigma^{-1}V^T\eta & \end{aligned} \tag{6.15}$$

Therefore, using the deviation (6.15), the computed solution $\tilde{x} = x + D(U\Sigma^{-1}V^T\eta)$. With the component-wise consideration

$$(\tilde{x}-x)_j = D_j(U\Sigma^{-1}V^T\eta)_j = (x+\theta)_j(U\Sigma^{-1}V^T\eta)_j,$$

we receive the relative error with respect to the approximation $x+\theta$ as

$$\left|\frac{(\tilde{x}-x)_j}{(x+\theta)_j}\right| = \left|(U\Sigma^{-1}V^T\eta)_j\right|. \qquad \blacksquare$$

Hence, for a solution $\tilde{x} \neq 0$, the relative error is in the order of the underlying data noise η if the elements of Σ^{-1} are not arbitrary large, i.e., $\Sigma^{-1} \in \mathcal{O}(1)$.

As the elements of Σ^{-1} usually can be arbitrary large, we get rid of the noise contribution by truncating the small singular values in the original spectral decomposition $\Sigma = \mathrm{diag}(\sigma_1,\ldots,\sigma_n)$ and denote

$$\Sigma_k := \mathrm{diag}(\sigma_1,\ldots,\sigma_k,0,\ldots,0) \in \mathbb{R}^{n\times n}.$$

Furthermore, we denote the pseudoinverse of Σ_k as

$$\Sigma_k^\dagger := \mathrm{diag}(\sigma_1^{-1},\ldots,\sigma_k^{-1},0,\ldots,0) \in \mathbb{R}^{n\times n}.$$

Theorem 6.2 *After truncating the small singular values, corresponding to noise, the regularized solution of $(DH^T HD)D^{-1}\tilde{x} = DH^T b$ is given by $D^{-1}\tilde{x} = U\Sigma_k^\dagger V^T b$ and bounded by the underlying data noise η via*

$$\|D^{-1}(\tilde{x}-x)\|_2 \leq \frac{1}{\sigma_k}\|\eta\|_2 + e_{reg} \qquad \textit{and} \qquad \|D^{-1}(\tilde{x}-x)\|_2^2 \leq \frac{1}{\sigma_k^2}\|\eta\|_2^2 + e_{reg}^2,$$

where e_{reg} denotes the regularization error.

Proof Replacing Σ^{-1} by $\Sigma_k^\dagger$ results in the solution

$$\begin{aligned} D^{-1}\tilde{x} &= U\Sigma_k^\dagger V^T b = U\Sigma_k^\dagger V^T(HDD^{-1}x+\eta) = U\Sigma_k^\dagger V^T(V\Sigma U^T D^{-1}x+\eta) \\ &= U\Sigma_k^\dagger\Sigma U^T D^{-1}x + U\Sigma_k^\dagger V^T\eta = U\left(\begin{array}{c|c} I_k & 0 \\ \hline 0 & 0 \end{array}\right)U^T D^{-1}x + U\Sigma_k^\dagger V^T\eta \\ &= D^{-1}x - U\left(\begin{array}{c|c} 0 & 0 \\ \hline 0 & I_{n-k} \end{array}\right)U^T D^{-1}x + U\Sigma_k^\dagger V^T\eta. \end{aligned}$$

Hence, with $U := (U_1, U_2)^T$, we obtain

$$D^{-1}(\tilde{x} - x) = U\Sigma_k^\dagger V^T \eta - U \left(\begin{array}{c|c} 0 & 0 \\ \hline 0 & I_{n-k} \end{array}\right) \left(\begin{array}{c} U_1^T \\ \hline U_2^T \end{array}\right) D^{-1}x \tag{6.16}$$

which, by passing to norms, can be written as

$$\|D^{-1}(\tilde{x} - x)\|_2^2 = \Big[\underbrace{\|U\Sigma_k^\dagger V^T \eta\|_2}_{\text{perturbation error}} \Big]^2 + \Big[\underbrace{\|U_2 U_2^T D^{-1}x\|_2}_{\text{regularization error} = e_{\text{reg}}} \Big]^2. \tag{6.17}$$

Finally, (6.17) can be bounded by

$$\|D^{-1}(\tilde{x} - x)\|_2^2 \le \frac{1}{\sigma_k^2}\|\eta\|_2^2 + e_{\text{reg}}^2. \qquad \blacksquare$$

An immediate consequence of Theorem 6.2 is

Corollary 6.1 *After truncating the small singular values, the component-wise relative error of the solution $\tilde{x}$, with respect to the approximation $x + \theta$, is bounded by*

$$\left|\frac{\tilde{x}_j - x_j}{(x+\theta)_j}\right| \le \frac{n}{\sigma_k} \max|\eta_j| + \left|(U_2 U_2^T D^{-1}x)_j\right|.$$

Proof Using (6.14) and passing (6.16) to components, we obtain

$$\left|\frac{\tilde{x}_j - x_j}{(x+\theta)_j}\right| \le \left|(U\Sigma_k^\dagger V^T \eta)_j\right| + \left|(U_2 U_2^T D^{-1}x)_j\right|$$

in which the component-wise perturbation error can be bounded by

$$\left|(U\Sigma_k^\dagger V^T \eta)_j\right| \le \|U\Sigma_k^\dagger V^T \eta\|_\infty \le \underbrace{\|U\|_\infty}_{\le \sqrt{n}} \underbrace{\|\Sigma_k^\dagger\|_\infty}_{\le \sigma_k^{-1}} \underbrace{\|V^T\|_\infty}_{\le \sqrt{n}} \|\eta\|_\infty \le \frac{n}{\sigma_k}\|\eta\|_\infty = \frac{n}{\sigma_k}\max|\eta_j|. \qquad \blacksquare$$

Representing a heuristic, we ignore the regularization error e_{reg} in Theorem 6.2 and observe the following cases where $\sigma_k(H)$ and $\sigma_k(DH^T)$ denote the k^{th} singular value of H and DH^T, respectively:

1. For $D = I$, we obtain the absolute errors

$$\begin{aligned} |\tilde{x}_j - x_j| &\lesssim \frac{n}{\sigma_k(H)}\max|\eta_j| \in \frac{n}{\sigma_k(H)}\mathcal{O}(\|\eta\|_\infty) \quad \text{and} \\ \|\tilde{x} - x\|_2 &\lesssim \frac{1}{\sigma_k(H)}\|\eta\|_2 \in \frac{1}{\sigma_k(H)}\mathcal{O}(\|\eta\|_2). \end{aligned}$$

2. For D according to (6.14), we obtain the relative error with respect to $x + \theta$ as

$$\begin{aligned} \left|\frac{\tilde{x}_j - x_j}{(x+\theta)_j}\right| &\lesssim \frac{n}{\sigma_k(DH^T)}\max|\eta_j| \in \frac{n}{\sigma_k(DH^T)}\mathcal{O}(\|\eta\|_\infty) \quad \text{and} \\ \left\|\frac{\tilde{x}_j - x_j}{(x+\theta)_j}\right\|_2 &\lesssim \frac{1}{\sigma_k(DH^T)}\|\eta\|_2 \in \frac{1}{\sigma_k(DH^T)}\mathcal{O}(\|\eta\|_2). \end{aligned}$$

3. For general D, we obtain an arbitrary weighting possibility.

Consequently, our heuristic taking the signal values into account weights components differently according to their size, i.e., different components of different size in x become of similar importance in contrast to standard regularization with $D = I$. General D allows general weighting to enforce reconstruction for certain sections of the signal vector.

Remark 6.2 *Being of theoretical interest, similarly to M_x mentioned above, constructing D out of exact signal data, i.e., D_x according to (6.18), and applying it as data based preconditioner (see Section 6.3.2), we observe strong improvement among many problems. See also Figure 6.11 (d) on this.*

6.3.2. Data Based Regularization Methods

Due to the given signal x or b we construct the data based preconditioner $D \in \mathbb{R}^{n\times n}$ as a diagonal matrix with entries

$$(D_x)_{ii} := |x_i| + \varepsilon_{\text{D}} \qquad \text{and} \qquad (D_b)_{ii} := |b_i| + \varepsilon_{\text{D}}, \tag{6.18}$$

respectively. The parameter ε_{D} should be chosen $\varepsilon_{\text{D}} \ll 1$ or $\varepsilon_{\text{D}} \in \mathcal{O}(\eta)$ if $b_i = 0$, to guarantee the nonsingularity of D_b. For $b_i \neq 0$ we can choose $\varepsilon_{\text{D}} = 0$. The same holds for D_x.

Following our numerical experiments, we receive best results using D or D^2 depending on the regularization method. Other powers of D lead to less accurate approximations while still improving the reconstruction. For the TSVD the decomposition $HD = U\Sigma V^T$ leads to the modified ill-posed problem

$$(DH^THD)D^{-1}x = DH^Tb \;\Leftrightarrow\; x = DV\Sigma^{-1}U^Tb. \tag{6.19}$$

For TPR we use the direct application of D to define an appropriate norm $\|x\|_D^2 = x^TD^TDx$. We modify (2.32) to

$$\begin{aligned} (DH^THD)D^{-1}x = DH^Tb \quad &\Leftrightarrow \\ (DH^THD + \alpha^2 I)D^{-1}x = DH^Tb \quad &\Leftrightarrow \\ (H^TH + \alpha^2 D^{-2})x = H^Tb \quad &\Leftrightarrow \quad \min_x \left\{ \|Hx - b\|_2 + \alpha^2 \|x\|_{D^{-1}} \right\}. \end{aligned} \tag{6.20}$$

Note the connection of the preconditioned system DH^THD in (6.19) to the generalized SVD of the matrix pair (H, D^{-1}) related to the seminorm $\|x\|_{D^{-1}}$. Although we observed improvement for (6.20), slightly better results can be obtained by incorporating the signal values only once in TPR. Therefore, we reconstruct the signal using $D^{\frac{1}{2}}$ in

$$\min_x \left\{ \|Hx - b\|_2 + \alpha^2 \|x\|_{D^{-\frac{1}{2}}} \right\} \Leftrightarrow (H^TH + \alpha^2 D^{-1})x = H^Tb.$$

Using PCGLS, we reconstruct the observed signal by using split preconditioning with $M = D^{\frac{1}{2}}$ within Algorithm 3. Note that we also examine the preconditioning effect for PMINRES [118] and PMR-II [60] in Numerical Experiment 6.14. In Section 6.6, we combine data based regularization with smoothing norms (2.36) in TPR of general form.

6.3.3. Improvement by Outer Iterations

It is possible to further improve on a first computed solution and start an iterative process of building diagonal preconditioners based on the current reconstruction. Following Algorithm 10, we either use b or a first reconstruction from an unpreconditioned regularization method as initial solution $\tilde{x}^{(0)}$. For a given number of steps we construct $D_{\tilde{x}^{(s)}}$ from a previously computed reconstruction $\tilde{x}^{(s-1)}$ and compute a new solution $\tilde{x}^{(s)}$. As this approach can be applied both to direct and iterative regularization methods, we refer to any method presented in Section 2.5 by using the identifier Regularization_method. Note that for every call of the Regularization_method, there must precede a method for the estimation of the optimal regularization parameter.

Algorithm 10: Data based regularization with outer iterations (ROI).

Input : $H \in \mathbb{R}^{n\times n}$, $b \in \mathbb{R}^n$, $\varepsilon_{\mathrm{D}} \geq 0$, steps ≥ 0, $\nu_{\mathrm{dp}} \geq 0$, $\xi \geq 0$
Output: $\tilde{x} \in \mathbb{R}^n$

$\tilde{x}^{(0)} \leftarrow b$ or $\tilde{x}^{(0)} \leftarrow$ Regularization_method(H, b, I)
for $s \leftarrow 1$ **to** steps **do**
  $(D_{\tilde{x}^{(s)}})_{ii} \leftarrow |\tilde{x}_i^{(s-1)}| + \varepsilon_{\mathrm{D}}$
  $\tilde{x}^{(s)} \leftarrow$ Regularization_method$(H, b, D_{\tilde{x}^{(s)}})$
  if $\|r^{(s)}\|_2 < \nu_{\mathrm{dp}}\sqrt{n}\xi$ **then**
    break
  end
end

In general, the solution improves in the first steps. As in later steps the error saturates and the reconstruction does not change evidently, it is reasonable to perform only a few number of iterations. In most cases, it is sufficient to choose steps $\in$ [1,5]. Unfortunately, the class of discrete ill-posed problems for which this approach yields improved reconstructions can not be classified exactly. The prerequisite "weakly distorting blur operators for signals containing nearly zero components" is making the method sensible or even spoiling the solution for problems which do not clearly satisfy the requested property. It turned out, that providing a heuristic stopping criterion for the outer iterations can be a resort to this problem. Following the discrepancy principle [70], we can stop the outer iterative process before reaching the maximum number of given steps if the residual norm $\|r^{(s)}\|_2$ drops below the expected value of the perturbation norm $\|e\|_2 = \sqrt{n}\xi$ using the "safety factor" $\nu_{\mathrm{dp}} = 1.5$ for $\nu_{\mathrm{dp}}\|e\|_2 = \nu_{\mathrm{dp}}\sqrt{n}\xi$. Note that for this the error norm or the deviation ξ (magnitude of white noise) have to be known in advance. Regardless, if the reconstruction should be as accurate as possible this can be a method of choice for a certain class of problems, e.g., Gaussian blurred pulse sequences or weakly distorted astronomical images.

6.3.4. Numerical Experiments

Besides some few model problems created on our own and from [108], we mainly focus on problems from Regularization Tools [68]. We affect our right-hand sides with Gaussian white noise of different magnitude and perform all computations on normalized values. Hence, we measure the quality of the reconstruction via the relative reconstruction error (RRE). If not mentioned otherwise, we use $\varepsilon_{\mathrm{D}} = 1\mathrm{E}{-}08$ for $D_{\tilde{x}}$ and $\tilde{x}^{(0)} = b$ as initial solution

for Algorithm 10 (ROI). All our following experiments are performed on Environment D.3 with implementations for MATLAB.

Direct Data Based Regularization Methods

For TPR we use two different ways to estimate the optimal regularization parameter α. We constitute the method as OPTIMAL by using the MATLAB function `fminbnd` to search for the local minimizer α in our Tikhonov-Phillips function handle. We refine the search by setting the termination tolerance to 1E−09 = `TolX`. See also Listing 6.1 on how we incorporate the function handle in `fminbnd`. Providing an optimal solution has a more hypothetical character. Computing the minimal error $\min_\alpha \|x - \tilde{x}_\alpha\|_2$ illustrates the maximum obtainable improvement for a perfect estimator. Moreover, one could think of a quasi-optimal estimator by visually choosing the subjective best or sharpest image from a set of reconstructions.

Additionally, we compute an estimation via `l_corner` from [68]. Similarly to its implementation, we fix the size of our parameter space to $p = 200$ (number of evaluations for α_k) and perform a scaling of the various α_k in the form

$$\alpha_k = \alpha_{k+1} \left(\frac{\sigma_{\max}}{\alpha_p} \right)^{\frac{1}{p-1}}, \qquad k = p-1, \ldots, 1.$$

Here, $\alpha_p = \max(\sigma_{\min}, 16\sigma_{\max}\varepsilon_{\text{mach}})$ at which $\varepsilon_{\text{mach}}$ denotes the machine precision and is defined as $\varepsilon_{\text{mach}} = 2.2204\text{E}{-16}$ in our case. For every α_k we compute the (preconditioned) Tikhonov-Phillips solution $\tilde{x}_{\alpha_k}$ and generate the points $\{P_k\} = (\|H\tilde{x}_{\alpha_k} - b\|_2, \|\tilde{x}_{\alpha_k}\|_2)$ of the L-curve. The search of the location of the corner is performed afterwards using `l_corner`.

In case of reconstructing the signal via TSVD, we estimate the regularization parameter with the ADAPTIVEPRUNING algorithm implemented in [96]. For the data based TSVD solution we use the SVD of HD, compute the solution $\tilde{x}_k$ with the estimated optimal truncation parameter k via `tsvd` [68], and apply the preconditioner to the solution by $\tilde{x}_k \leftarrow D_{\tilde{x}^{(s)}}\tilde{x}_k$. Similar to TPR, we provide an OPTIMAL solution by computing the minimal error $\min_k \|x - \tilde{x}_k\|_2$ with respect to the true solution.

Numerical Experiment 6.9 : 1D Gaussian blur of a constant pulse sequence. We consider the signal x_{p} of size $n = 800$ which is a pulse rate with constant positive discontinuities after every 100 samples:

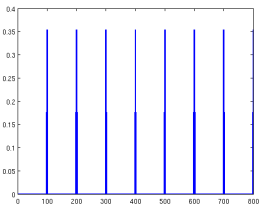
$$(x_{\mathsf{p}})_{100 \cdot i} = 5, \quad i = 1, \ldots, 8. \tag{6.21}$$

We blur x_{p} with the 1D Gaussian operator H_{b1D} with bandwidth 6, $\sigma = 4$, and add noise of magnitude $\xi = 0.05\%$. We reconstruct the signal both in its basic way without preconditioner and with $D_{\tilde{x}_{\mathsf{p}}^{(s)}}$ using ROI, where s denotes the number of performed outer iterations. We fix the maximum number of outer iterations to 5, i.e., steps = 5. Note that $\kappa_2(H_{\mathsf{b1D}}) = 1.91\text{E}{+04}$ and that the results for $D_{\tilde{x}_{\mathsf{p}}^{(1)}}$ reflect a single application of the data based preconditioner.

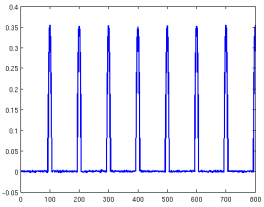
Following Table 6.5, we obtain a better reconstruction of x_{p} incorporating $D_{\tilde{x}_{\mathsf{p}}^{(s)}}$. Performing more outer iterations leads to further improvement. In contrast to TSVD, the discrepancy principle in ROI stops the iteration for TPR (`l_corner`) after the first step. While this criterion prevents further improvement for this example, it may prevent degradation of the solution for other problems. The ADAPTIVEPRUNING algorithm yields nearly optimal

Table 6.5.: RRE for the signal x_{p} blurred with `blur1D(800,6,4)` and affected by noise of magnitude $\xi = 0.05\%$ using TPR and TSVD in algorithm ROI.

`blur1D(800,6,4)`, $n = 800$, $band = 6$, $\sigma = 4$, $\xi = 0.05\%$, $\varepsilon_{\mathrm{D}} = 1\mathrm{E}{-}08$, steps $= 5$, $(x_{\mathsf{p}})_{100 \cdot i} = 5, \quad i = 1, \ldots, 8$



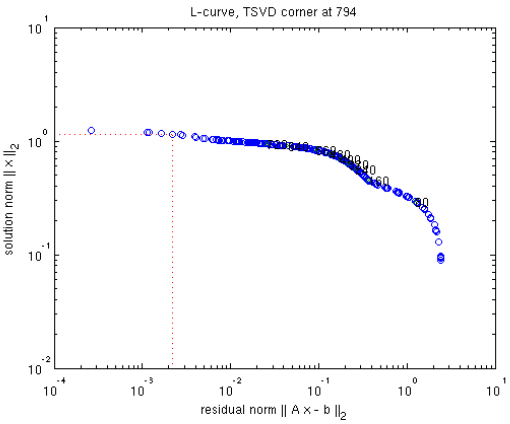
x_{p}



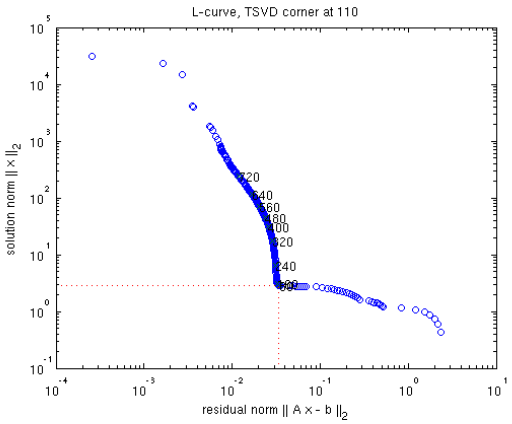
$H_{\mathsf{b1D}} x_{\mathsf{p}} + \eta$

$\|x_{\mathsf{p}} - \tilde{x}_{\mathsf{p}}\|_2 / \|x_{\mathsf{p}}\|_2$ (Regularization parameter)

	Tikhonov-Phillips		TSVD	
Preconditioner	Optimal	`l_corner`	Optimal	Adap.Prun.
I	0.2328 (0.0418)	0.8111 (1.1503)	0.2212 (766)	0.5830 (794)
$D_{\tilde{x}_{\mathsf{p}}^{(1)}}$	0.0384 (0.0238)	0.2357 (0.0005)	0.0300 (82)	0.0319 (110)
$D_{\tilde{x}_{\mathsf{p}}^{(2)}}$	0.0224 (0.0333)	– stopped	0.0015 (8)	0.0018 (11)
$D_{\tilde{x}_{\mathsf{p}}^{(3)}}$	0.0176 (0.0360)	–	0.0013 (8)	0.0013 (8)
$D_{\tilde{x}_{\mathsf{p}}^{(4)}}$	0.0153 (0.0397)	–	0.0013 (8)	0.0013 (8)
$D_{\tilde{x}_{\mathsf{p}}^{(5)}}$	0.0141 (0.0419)	–	0.0013 (8)	0.0013 (8)



(a) Without preconditioner.



(b) With diagonal preconditioner $D_{\tilde{x}_{\mathsf{p}}^{(1)}}$.

Figure 6.10.: Discrete L-curves for the problem `blur1D(800,6,4)` solved with TSVD without preconditioner and with $D_{\tilde{x}_{\mathsf{p}}^{(1)}}$ using the AdaptivePruning algorithm to estimate the truncation index.

estimations of the truncation index making the TSVD regularization to produce saturated values similar to the OPTIMAL case. Here, and for some other problems, we observe that after applying $D_{\tilde{x}}$, the solution norms $\|\tilde{x}\|_2$ are amplified which results in sharper L-curves (cf. Figure 6.10). In such cases estimators should be able to locate the corner more accurately. ●

Numerical Experiment 6.10 : 2D Gaussian blur problem. We consider the test problem `blur` [68], choose $H_{\mathsf{b}} \in \mathbb{R}^{32^2 \times 32^2}$, $band = 3$, and $\sigma = 2$, i.e., we invoke `blur(32,3,2)`. Note that H_{b} corresponds to the kernel in (6.10). The stacked right-hand side x_{b} is affected with $\xi = 0.01\%$. Using ROI, we bound the outer iterations by steps = 5.

Using $D_{\tilde{x}_{\mathsf{b}}^{(s)}}$ yields better reconstructions both for TPR and TSVD as illustrated in Table 6.6. The discrepancy principle in ROI takes effect after the first improvement for TPR but it does not stop the reconstruction after the third step for TSVD producing worse results along further iterations. This reflects the inaccurate behavior of the stopping criterion which we observed for some model problems. Due to poor estimations of α using `l_corner`, we get nearly similar results for I and $D_{\tilde{x}_{\mathsf{b}}^{(1)}}$. When using our B-SPLINE approach, e.g., we obtain more distinct errors which illustrates the sensitivity of the estimation process. Figure 6.11 gives the mesh plots of the OPTIMAL TPR solutions. Comparing subfigures (b) and (c) shows that data based regularization provides a solution with better reconstruction, especially near zero values. However, the improvement is not restricted to these zero regions: e.g., we observed improved reconstruction also around the discontinuities of the pulse sequence (6.21). Being of theoretical interest, subfigure (d) gives the solution using $D_{\tilde{x}_{\mathsf{b}}^{(1)}}$ for $\tilde{x}_{\mathsf{b}}^{(0)} = x_{\mathsf{b}}$. Taking the exact signal values into account in most cases leads to improved reconstructions, even for more general blur operators and signals. See also Table 6.9 and Figure 6.16 (d). ●

Numerical Experiment 6.11 : test problem `wing`. The test problem by G. M. Wing taken from [68] has a discontinuous solution. The constituent parts of the discretized Fredholm integral equation are

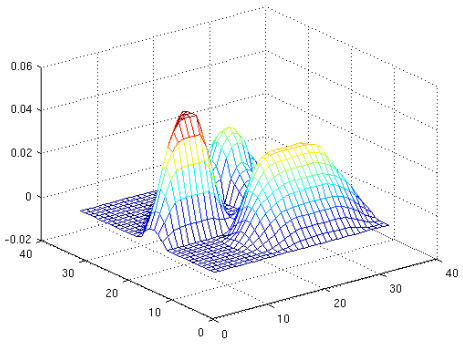
$$H_{\mathsf{w}}(s,t) = te^{-st^2}, \qquad b_{\mathsf{w}}(s) = \frac{e^{-\frac{s}{9}} - e^{-\frac{4s}{9}}}{2s}, \text{ and} \qquad x_{\mathsf{w}}(t) = \begin{cases} 1, & \text{for } \frac{1}{3} < t < \frac{2}{3} \\ 0, & \text{elsewhere} \end{cases},$$

being the kernel, the right-hand side, and the exact solution, respectively. With $n = 800$ we invoke `wing(800,`$\frac{1}{3}$`,`$\frac{2}{3}$`)`, affect the blurred right-hand side with white noise of magnitude 0.1%, and perform a maximum of 5 outer iterations. The operator is severely ill-conditioned with $\kappa_2(H_{\mathsf{w}}) = 1.65\text{E+}22$.

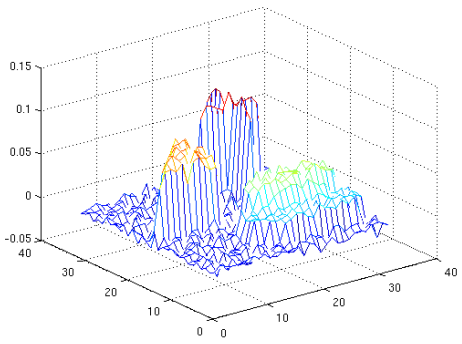
Following Table 6.7, a single application of $D_{\tilde{x}_{\mathsf{w}}^{(1)}}$ in the first step yields slightly worse solutions. For TSVD the stopping criterion tolerates further outer iterations and therefore, we are able to improve the reconstruction. Although, for $D_{\tilde{x}_{\mathsf{w}}^{(1)}}$ the estimated and OPTIMAL reconstruction is worse than for the unpreconditioned case, the solution becomes better for $D_{\tilde{x}_{\mathsf{w}}^{(3)}}$. However, the discrepancy principle does not prevent from spoiling the solution after 3 iterations. Note that this is an example where the signal structure is strongly distorted by the underlying kernel. ●

Table 6.6.: RRE for the `blur(32,3,2)` [68] problem with $\xi = 0.01\%$ for TPR and TSVD. The reconstruction is performed with Algorithm 10.

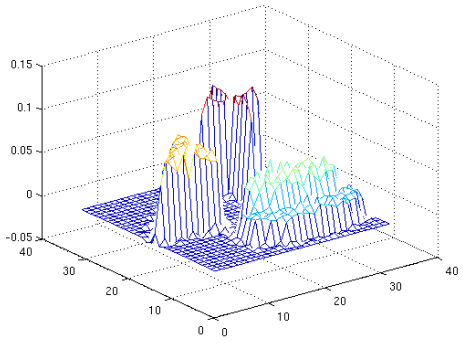
`blur(32,3,2)`, $n = 32^2$, $band = 3$, $\sigma = 2$, $\xi = 0.01\%$, $\varepsilon_{\mathrm{D}} = 1\mathrm{E}{-}08$, steps $= 5$					
	$\|x_{\mathtt{b}} - \tilde{x}_{\mathtt{b}}\|_2 / \|x_{\mathtt{b}}\|_2$ (Regularization parameter)				
	Tikhonov-Phillips			TSVD	
Precond.	OPTIMAL	`l_corner`	B-SPLINE	OPTIMAL	ADAP.PRUN.
I	0.1813 (0.0042)	0.6967 (3E−04)	0.3757 (0.0670)	0.1898 (766)	0.8619 (972)
$D_{\tilde{x}_{\mathtt{b}}^{(1)}}$	0.1148 (5E−04)	0.6479 (3E−06)	0.1345 (0.0012)	0.1384 (417)	0.1583 (506)
$D_{\tilde{x}_{\mathtt{b}}^{(2)}}$	– stopped	– stopped	– stopped	0.0997 (310)	0.1243 (363)
$D_{\tilde{x}_{\mathtt{b}}^{(3)}}$	–	–	–	0.0971 (287)	0.1247 (304)
$D_{\tilde{x}_{\mathtt{b}}^{(4)}}$	–	–	–	0.0997 (273)	0.1309 (274)
$D_{\tilde{x}_{\mathtt{b}}^{(5)}}$	–	–	–	0.0998 (273)	0.1789 (232)



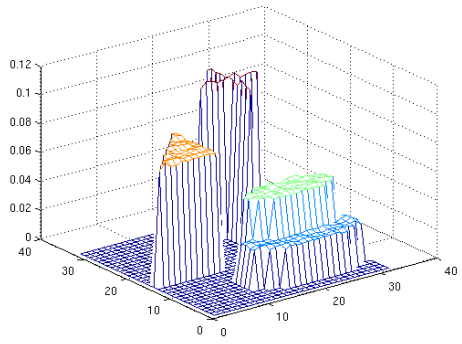
(a) Observed signal $b_{\mathtt{b}} = H_{\mathtt{b}} x_{\mathtt{b}} + \eta$.



(b) No preconditioning yields error 0.1813.



(c) Using $D_{\tilde{x}_{\mathtt{b}}^{(1)}}$, $\tilde{x}_{\mathtt{b}}^{(0)} = b_{\mathtt{b}}$ yields error 0.1148.

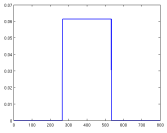


(d) Using $D_{\tilde{x}_{\mathtt{b}}^{(1)}}$, $\tilde{x}_{\mathtt{b}}^{(0)} = x_{\mathtt{b}}$ yields error 0.0150.

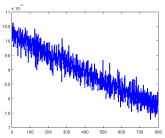
Figure 6.11.: Impact of using $\tilde{x}_{\mathtt{b}}^{(0)} = b_{\mathtt{b}}$ and $\tilde{x}_{\mathtt{b}}^{(0)} = x_{\mathtt{b}}$ as initial solutions in algorithm ROI during OPTIMAL TPR of the problem `blur(32,3,2)` with $\xi = 0.01\%$.

Table 6.7.: RRE using Tikhonov-Phillips regularization and TSVD for the `wing(800,`$\frac{1}{3}$`,`$\frac{2}{3}$`)` problem [68] with $\xi = 0.1\%$. The reconstruction is performed with Algorithm 10.

`wing(800,`$\frac{1}{3}$`,`$\frac{2}{3}$`)`, $n = 800$, $\xi = 0.1\%$, $\varepsilon_{\mathrm{D}} = 1\mathrm{E}{-}08$, steps $= 5$



x_{w}



$H_{\mathsf{w}}x_{\mathsf{w}} + \eta$

$\|x_{\mathsf{w}} - \tilde{x}_{\mathsf{w}}\|_2/\|x_{\mathsf{w}}\|_2$ (Regularization parameter)

	Tikhonov-Phillips		TSVD	
Preconditioner	OPTIMAL	`l_corner`	OPTIMAL	ADAP.PRUN.
I	0.6041 (0.0038)	0.6042 (0.0035)	0.6039 (2)	0.6039 (2)
$D_{\tilde{x}_{\mathsf{w}}^{(1)}}$	0.6092 (E−04)	0.6094 (0.0002)	0.6137 (3)	0.6240 (2)
$D_{\tilde{x}_{\mathsf{w}}^{(2)}}$	– stopped	– stopped	0.4122 (2)	0.4561 (2)
$D_{\tilde{x}_{\mathsf{w}}^{(3)}}$	–	–	0.3934 (2)	0.3813 (2)
$D_{\tilde{x}_{\mathsf{w}}^{(4)}}$	–	–	0.7248 (2)	0.6550 (2)
$D_{\tilde{x}_{\mathsf{w}}^{(5)}}$	–	–	1.4154 (2)	1.3208 (2)

Iterative Data Based Regularization Methods

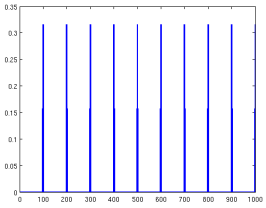
Numerical Experiment 6.12 : test problem `prolate`. We blur our pulse sequence x_{p} (6.21) with the `prolate` matrix from [108] which has symmetric Toeplitz form and is ill-conditioned with $\kappa_2(H_{\mathsf{p}}) = 5.29\mathrm{E}{+}16$. Our model problem has size $n = 1\mathrm{E}{+}03$ and is positive definite as we choose $w = 0.34$, i.e., we invoke `gallery('prolate', 1000,0.34)`. The observed signal also contains negative entries.

Referring to Table 6.8, we are able to further improve on a computed solution similar to the experiments using direct regularization methods. As we want to show the gained accuracy in later iterations, we switch off the discrepancy principle within ROI to avoid the break in the outer iterations which occurs after the first step. Using B-SPLINES leads to inferior estimations of k compared to the DISCREPANCYPRINCIPLE but yields better solutions in contrast to the ADAPTIVEPRUNING algorithm. This is because the shape of the L-curve is getting degraded and the optimum no longer corresponds to the point with maximum curvature. Here, the DISCREPANCYPRINCIPLE is a robust and more accurate estimator.

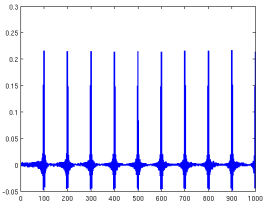
Interested in the convergence behavior when using $D_{\tilde{x}}$ as a preconditioner within (P)CGLS, we illustrate the semiconvergent curves for $D_{\tilde{x}^{(1)}}$ and $D_{\tilde{x}^{(5)}}$ for the problems `blur(32,3,2)` (see Numerical Experiment 6.10) and `gallery('prolate',1000,0.34)` with x_{p} in Figure 6.12. Applying data based preconditioning to these problems results in flat convergence curves, especially for the `prolate` operator where CGLS starts to operate on the unwanted noise subspace after a few iterations and thus heavily spoils the reconstruction. This flattened effect is enforced along the outer iterations yielding the desirable L-shaped convergence behavior for $D_{\tilde{x}^{(5)}}$ (cf. Figure 6.12 (a)). As there is no intersection between the convergence curves, the reconstruction is always more accurate in each iteration. The fact that the point

Table 6.8.: RRE for x_{p} (6.21) blurred with the operator `gallery('prolate',1000, 0.34)` [108] and affected with $\xi = 0.1\%$. Algorithm 10 is used without stopping criterion and with (P)CGLS.

`gallery('prolate',1000, 0.34)`, $n = 1000$, $\xi = 0.1\%$, $\varepsilon_{\mathrm{D}} = 1\mathrm{E}{-}08$, steps $= 5$, $(x_{\mathsf{p}})_{100 \cdot i} = 5, \quad i = 1, \ldots, 10$



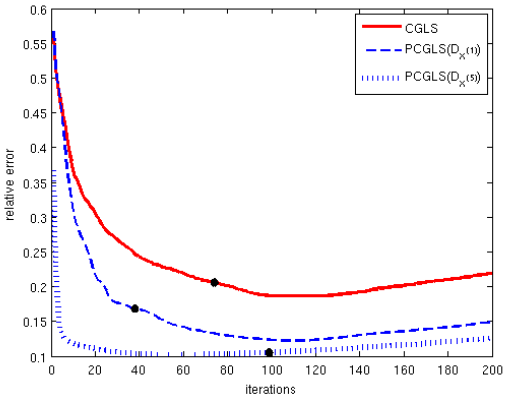
x_{p}



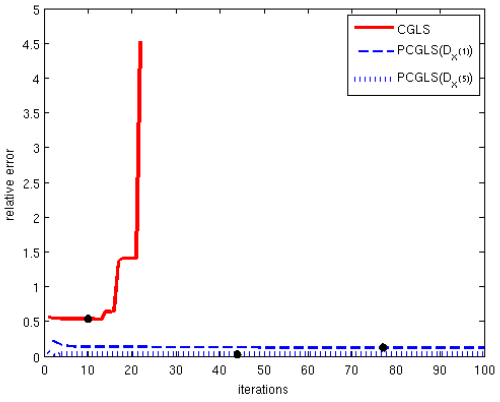
$H_{\mathsf{p}} x_{\mathsf{p}} + \eta$

(P)CGLS, $\|x_{\mathsf{p}} - \tilde{x}_{\mathsf{p}}\|_2 / \|x_{\mathsf{p}}\|_2$ (Regularization parameter)

Preconditioner	Optimal	Adapt.Prun.	B-spline	Discr.Princ.
I	0.5330 (13)	0.5380 (10)	0.5380 (10)	0.5404 (6)
$D_{\tilde{x}_{\mathsf{p}}^{(1)}}$	0.1213 (78)	0.1339 (166)	0.1213 (77)	0.1355 (8)
$D_{\tilde{x}_{\mathsf{p}}^{(2)}}$	0.0341 (5)	0.0491 (36)	0.0430 (16)	0.0392 (4)
$D_{\tilde{x}_{\mathsf{p}}^{(3)}}$	0.0115 (2)	0.0346 (91)	0.0307 (32)	0.0146 (3)
$D_{\tilde{x}_{\mathsf{p}}^{(4)}}$	0.0057 (2)	0.0470 (101)	0.0265 (17)	0.0069 (3)
$D_{\tilde{x}_{\mathsf{p}}^{(5)}}$	0.0041 (2)	0.0315 (40)	0.0256 (44)	0.0048 (3)



(a) Behavior for Numerical Experiment 6.10.



(b) Behavior for Numerical Experiment 6.12.

Figure 6.12.: Convergence behavior of CGLS and PCGLS with $D_{\tilde{x}^{(1)}}$ and $D_{\tilde{x}^{(5)}}$ for model problems `blur(32,3,2)` (a) and `gallery('prolate',1000,0.34)` (b). The RRE depends on the iteration k. The black dots are the B-spline estimations for the optimal number of iterations.

of maximum curvature on the L-curve does not exactly correspond to the optimal one is the reason why the estimation of k_{opt} does not always correspond to the exact optimum. ●

Numerical Experiment 6.13 : visual effects on 2D signals. We are interested in the visual effects occurring in the solutions when using (P)CGLS as regularization method in Algorithm 10 for the following test problems:

1. The 2D signal $x_{\mathtt{b}}$ resulting from the `blur` problem (see Numerical Experiment 6.10) and affected with $\xi = 0.08\%$. The 2D Gaussian kernel with $\sigma = 4$, bandwidth 8, and size $n = 100^2$ has $\kappa_2(H_{\mathtt{b}}) = 4.59\text{E+}13$. Note that $H_{\mathtt{b}}$ has a stronger blurring effect.
2. `tomo(n,f)` from [68] which is a tomography problem where the domain $[0,n] \times [0,n]$ is divided into n^2 cells of unit size, and a total of `round(`fn^2`)` rays in random directions penetrate the domain. The stacked right-hand side is the same as for the first problem, i.e., $x_{\mathtt{b}}$. We use `f` $= 1$, $\xi = 0.02\%$, and $n = 60$. $\kappa_2(H_{\mathtt{t}}) = 8.85\text{E+}17$.
3. A quadratic cut-out $[1{,}150] \times [1{,}150]$ of the image `Fig0920(a)(text_image)` from [54] blurred with the 2D Gaussian operator (6.10) using $band = 4$, $\sigma = 3$, and affected with $\xi = 0.1\%$.

For problem 1 and 3 we switch off the discrepancy principle in Algorithm 10 in order to illustrate the effect of 5 outer iterations. For problem 2 we limit their maximum number by 8.

Figure 6.13 and Figure 6.14 picture the solutions. Using data based preconditioning yields higher reconstruction quality for the given model settings. Visually, there is less noise perturbation in the background; compare Figure 6.13 (b) with (c) and (e) with (f). The shape of the objects is reconstructed more accurately. See, for instance, the white cross in Figure 6.13 (c) or the writing in Figure 6.14 (c). Note that we were able to find settings for these problems where data based preconditioning leads to worse results. Especially for settings affected with large noise an application of $D_{\tilde{x}}$ produces poorer reconstructions compared to I. For some of these settings the usage of outer iterations can be a remedy to circumvent this problem because the reconstruction is worse for $D_{\tilde{x}^{(1)}}$ but improves along further iterations. ●

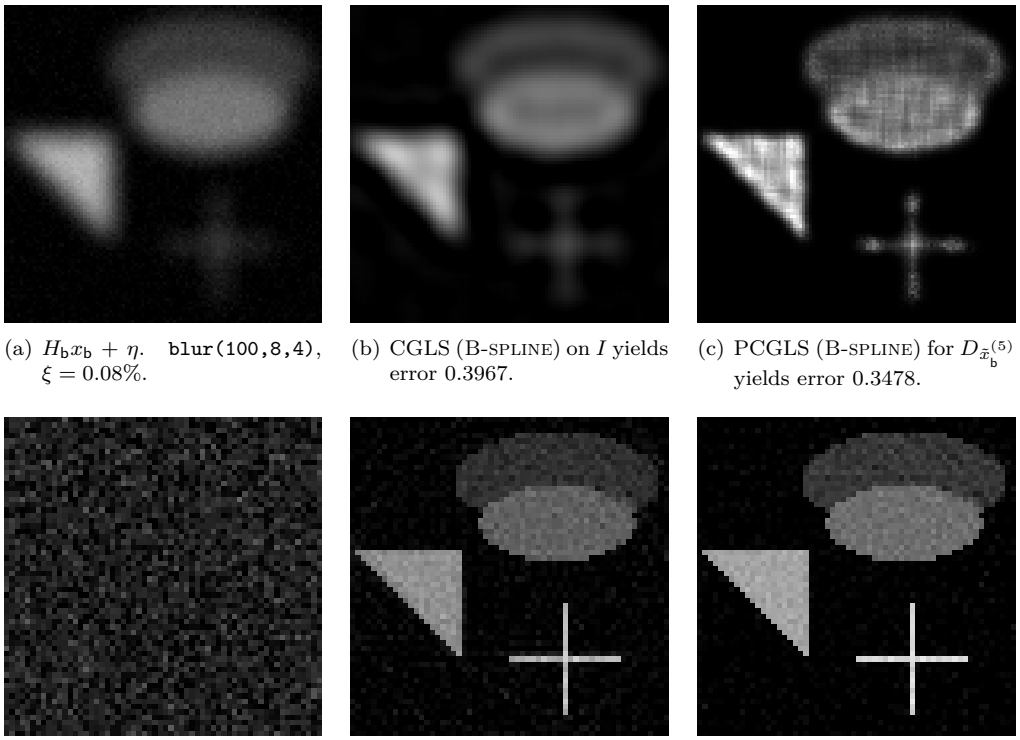
Numerical Experiment 6.14 : further iterative regularization methods. We are interested in the behavior using data based preconditioning within MINRES and MR-II, representing further regularizing Krylov subspace methods. We focus on the model problem `blur(32,4,8)` with $\kappa_2(H_{\mathtt{b}}) = 1.39\text{E+}13$ and on `gallery('gcdmat',1024)` with $\kappa_2(H_{\mathtt{g}}) = 8.67\text{E+}04$ from [108], which is the greatest common divisor matrix[15]. For the latter kernel we use the point symmetric signal $x_{\mathtt{s}}$ with a positive and negative sawtooth around $\frac{n}{2}$:

$$(x_{\mathtt{s}})_{i=1:\frac{n}{2}-100,\ \frac{n}{2}+100:n} = 2,\ (x_{\mathtt{s}})_{i=\frac{n}{2}-100:\frac{n}{2}-1} = i - \frac{n}{2} + 101,\ (x_{\mathtt{s}})_{i=\frac{n}{2}:\frac{n}{2}+100} = i - \frac{n}{2} - 100.$$

We perform the reconstruction using a maximum of 5 outer iterations in ROI and use the optimal number of iterations k_{opt} within all methods.

Following Table 6.9, we receive similar results among the selected Krylov methods and obtain improved solutions using $D_{\tilde{x}}$. For the `blur` problem the discrepancy principle in ROI stops the reconstruction process after the first iteration. The additional theoretical results for

15 For a finite ordered set $\mathcal{F} = \{x_1, x_2, \ldots, x_n\}$ of distinct positive integers, the greatest common divisor (GCD) matrix $F \in \mathbb{R}^{n \times n}$ on $\mathcal{F}$ is defined by the components $f_{ij} = \gcd(x_i, x_j)$, where gcd(.) returns the greatest common divisor between two integers.



(a) $H_{\mathrm{b}}x_{\mathrm{b}} + \eta$. `blur(100,8,4)`, $\xi = 0.08\%$.

(b) CGLS (B-SPLINE) on I yields error 0.3967.

(c) PCGLS (B-SPLINE) for $D_{\tilde{x}_{\mathrm{b}}^{(5)}}$ yields error 0.3478.

(d) $H_{\mathrm{t}}x_{\mathrm{b}} + \eta$. `tomo(60,1)`, $\xi = 0.02\%$.

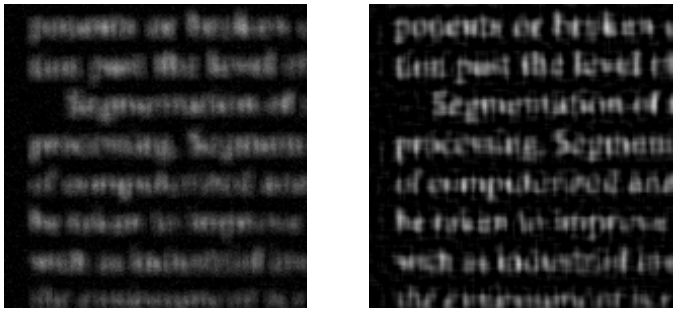
(e) CGLS (B-SPLINE) on I yields error 0.2027.

(f) PCGLS (DISCR.PRINC.) for $D_{\tilde{x}_{\mathrm{b}}^{(8)}}$ yields error 0.1153.

Figure 6.13.: Reconstruction of problem `blur(100,8,4)` with $\xi = 0.08\%$ in subfigures (a)–(c) and problem `tomo(60,1)` with $\xi = 0.02\%$ in subfigures (d)-(f). Algorithm 10 uses (P)CGLS with switched off discrepancy principle for the first model problem.

$D_{\tilde{x}^{(1)}}$ with initial solution $\tilde{x}^{(0)} = x$ show the maximum obtainable improvement after one outer iteration if $\xi = 0\%$. In this case we always observe a strong improvement for the model problems which implies that the less b is affected by noise, the bigger the improvement will be when data based regularization is used. ●

Interpretation of the results. In case of model problems where H preserves the shape of the original signal, i.e., performs a weak blurring, and the right-hand side contains nearly zero components, the usage of data based regularization will improve the reconstruction, especially when the perturbation in the right-hand side is of small magnitude and discontinuities are available. Here, incorporating the signal values in a diagonal matrix D is favorable concerning its cheap construction and inversion. Further improvement can be achieved by iteratively applying D to the solution iterates. As a heuristic stopping rule we used the discrepancy principle which, for some examples, showed pessimistic behavior, i.e., the outer iteration stopped although it would have been possible to further improve on the solution.



(a) $H_{\mathsf{b}} x_{\mathsf{text}} + \eta$. $\xi = 0.1\%$

(b) CGLS (Adapt.Pruning) for I yields error 0.7193.

(c) PCGLS (Adapt.Pruning) for $D_{\tilde{x}_{\mathsf{b}}^{(5)}}$ yields error 0.595.

(d) PCGLS (Discr.Princ.) for $D_{\tilde{x}_{\mathsf{b}}^{(5)}}$ yields error 0.567.

Figure 6.14.: RRE for a cut-out of the image `Fig0920(a)(text_image)` from [54] blurred with the 2D Gaussian operator (6.10) using $band = 4$, $\sigma = 3$, and $\xi = 0.1\%$. Reconstruction is perfomed with Algorithm 10 with switched off discrepancy principle.

Table 6.9.: RRE using the optimal number of iterations for the `blur(32,4,8)` [68] and the `gallery('gcdmat',1024)` [108] problem, respectively. steps = 5.

`blur(32,4,8)`, $\xi = 0.015\%$, $\varepsilon_{\mathrm{D}} = 1\mathrm{E}{-}08$, steps $= 5$			
	$\|x_{\mathsf{b}} - \tilde{x}_{\mathsf{b}}\|_2 / \|x_{\mathsf{b}}\|_2$ (Regularization parameter)		
Preconditioner	(P)CGLS	(P)MINRES	(P)MR-II
I	0.2648 (67)	0.2617 (82)	0.2660 (147)
$D_{\tilde{x}_{\mathsf{b}}^{(1)}}$, $\tilde{x}_{\mathsf{b}}^{(0)} = b_{\mathsf{b}}$	0.1718 (69)	0.1684 (536)	0.1745 (243)
$D_{\tilde{x}_{\mathsf{b}}^{(1)}}$, $\tilde{x}_{\mathsf{b}}^{(0)} = x_{\mathsf{b}}$	0.0206 (18)	0.0205 (148)	0.0199 (36)
`gallery('gcdmat',1024)`, $\xi = 0.5\%$, $\varepsilon_{\mathrm{D}} = 1\mathrm{E}{-}08$, steps $= 5$, x_{s}			
	$\|x_{\mathsf{s}} - \tilde{x}_{\mathsf{s}}\|_2 / \|x_{\mathsf{s}}\|_2$ (Regularization parameter)		
Preconditioner	(P)CGLS	(P)MINRES	(P)MR-II
I	0.4116 (25)	0.4072 (30)	0.4231 (44)
$D_{\tilde{x}_{\mathsf{s}}^{(1)}}$, $\tilde{x}_{\mathsf{s}}^{(0)} = b_{\mathsf{s}}$	0.3033 (25)	0.3001 (95)	0.3021 (56)
$D_{\tilde{x}_{\mathsf{s}}^{(3)}}$, $\tilde{x}_{\mathsf{s}}^{(0)} = b_{\mathsf{s}}$	0.2350 (21)	0.2340 (67)	0.2329 (42)
$D_{\tilde{x}_{\mathsf{s}}^{(5)}}$, $\tilde{x}_{\mathsf{s}}^{(0)} = b_{\mathsf{s}}$	0.2260 (20)	0.2278 (65)	0.2216 (31)
$D_{\tilde{x}_{\mathsf{s}}^{(1)}}$, $\tilde{x}_{\mathsf{s}}^{(0)} = x_{\mathsf{s}}$	0.1182 (11)	0.1169 (33)	0.1138 (24)

6.4. Partitioned Regularization

The optimal reconstruction for subparts of the signal is not equivalent to the optimal reconstruction of the whole signal. Indeed, our experiments allow to speculate that the noise spoils the solution for some subparts earlier or with different magnitude than for others during the reconstruction. This is (also) likely based on the fact that the subparts are differently contaminated by the noise. Therefore, another possibility to improve on the quality of the reconstruction is to perform the regularization process locally on parts of the signal with different regularization parameters. Our experiments revealed that this holds for blur operators and signals in general. Although the results motivate for a partitioned reconstruction, a practical residual-free method to estimate the regularization parameter for subparts of the signal is not available yet. Therefore, we point out that this section has a more theoretical character.

We split up the signal $\tilde{x}^{(s)}$ equidistantly into a given number of partitions. For problems with the mentioned prerequisites of data based regularization, we can improve on the reconstruction by embedding the partitioning approach within Algorithm 10. We denote c as the size of each partition of the signal and we assume that the number of partitions is a divisor of the problem size n. Following Algorithm 11, we estimate the optimal regularization parameter for each signal part using the current preconditioner $D_{\tilde{x}}^{(s)}$ and store it in a buffer, here denoted as `reg_params`. The new solution $\tilde{x}^{(s)}$ is computed via performing the reconstruction on each partition with a different regularization parameter.

Algorithm 11: Data based regularization with outer iterations and partitioning (ROIP).

Input : $H \in \mathbb{R}^{n\times n}$, $b \in \mathbb{R}^n$, $\varepsilon_{\mathrm{D}} \geq 0$, `steps` ≥ 0, $\xi \geq 0$, $\nu_{\mathrm{dp}} \geq 0$, `partitions` ≥ 1
Output: $\tilde{x} \in \mathbb{R}^n$

$\tilde{x}^{(0)} \leftarrow b$ or $\tilde{x}^{(0)} \leftarrow$ Regularization_method(H, b, I)
$c \leftarrow \frac{n}{\texttt{partitions}}$
for $s \leftarrow 1$ **to** `steps` **do**
  $(D_{\tilde{x}^{(s)}})_{ii} \leftarrow |\tilde{x}_i^{(s-1)}| + \varepsilon_{\mathrm{D}}$
  for $l \leftarrow 1$ **to** `partitions` **do**
    reg_params$[l]$ $\leftarrow$ estimate optimal regularization parameter for partition $\tilde{x}^{(s)}_{(l-1)c+1:lc}$ using $D_{\tilde{x}^{(s)}}$
  end
  $\tilde{x}^{(s)} \leftarrow$ Regularization_method$(H, b, D_{\tilde{x}^{(s)}}$, reg_params$)$
  if $\|r^{(s)}\|_2 < \nu_{\mathrm{dp}}\sqrt{n}\xi$ **then**
    break
  end
end

It is not necessary to execute several complete runs when using an iterative regularization method. Here, it is sufficient to iterate to the maximum number of all estimated optimal iterations. At the end of each iteration k, every partition of the reconstruction $\tilde{x}^{(s)}$ is updated with the corresponding partition of the current solution $x^{(k)}$. But only if the optimal number of iterations for this part is not reached yet, i.e., for each partition l of the signal we update the solution $\tilde{x}^{(s)}$ via

$$\textbf{if } \text{reg\_params}[l] \geq k \qquad \textbf{then } \tilde{x}^{(s)}_{(l-1)c+1:lc} = x^{(k)}_{(l-1)c+1:lc}.$$

For general blur operators, an improved reconstruction can be achieved without $D_{\tilde{x}}$ in the Regularization_method, and, consequently, not performing outer iterations. Usually, we observe saturation in the reconstruction quality when the number of partitions is increased. Note that the maximum number of partitions is bounded by the dimension of the underlying model problem.

The quality of a partitioned reconstruction even more depends on the quality of the estimation of the regularization parameter. Moreover, estimating a series of regularization parameters may become very costly. For the TPR or the TSVD regularization one could think of a parallel computation of the solution on p processors, where p is equal to the number of partitions used, for example. After broadcasting the model problem parameters, every processor is able to construct the preconditioner locally, estimate the local regularization parameter α_l and k_l, respectively, and compute its local solution part $\tilde{x}^{(s)}_{(l-1)c+1:lc}$ independently from all other processors. A master process gathers the solution parts and builds up the final solution. Using iterative methods this parallel approach is not necessary as along the way to the maximum number of estimated iterations all other solution parts are computed as well. Here, at least, the parameter estimation could be parallelized.

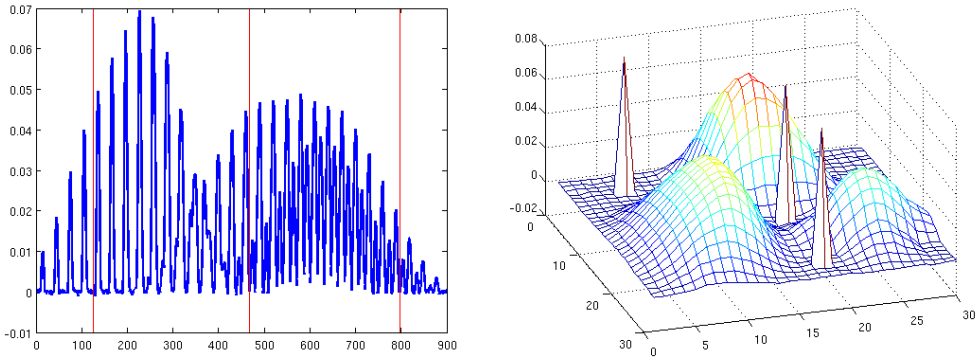
Numerical Experiment 6.15 : estimated regularization parameter. We are interested in the partitioned TPR regularization approach. By manually choosing 4 non-equidistant partitions such that their boundaries are embedded around zero components, we minimize the influence of the residual in a global sense. For each of the separated signal clusters we estimate α via L-curves and take special care to invoke corner estimations on well shaped curves. See Figure 6.15 for the manually imposed boundaries in the chosen 2D problem `blur(30,4,2)` with $\xi = 0.06\%$ and $\xi = 0.1\%$

Following Table 6.10, the reconstruction with 4 partitions leads to an improvement for noise level 0.06%. Changing the setting to noise level 0.1% leads to worse results due to (several) poor estimations of the regularization parameter via `l_corner`. Compare the OPTIMAL reconstruction which again yields improvement. Hence, only slight over or under estimations may accumulate to an overall larger reconstruction error, making a highly accurate estimation technique mandatory. Note that there might be a correlation between the size of the partitions and the shape of the L-curve, in general. ●

Numerical Experiment 6.16 : visual effects in image reconstruction. We focus on iterative (data based) regularization using (P)CGLS in ROIP to reconstruct a cut-out of the `hubble-original` image from [54] of size 100×100. We affect the image signal x_{h} with a 2D Gaussian blur and with white noise of magnitude 0.02%. The number of partitions is chosen to be 5% of the global size, i.e., 500.

Referring to Figure 6.16, a partitioned reconstruction improves the unpreconditioned CGLS reconstruction. Combining data based regularization with partitioning further improves the quality. Note that for large noise we observed a degradation of the solution using data based regularization while partitioning always yields improved results. ●

Numerical Experiment 6.17 : general blur operators. We are interested in the partitioned direct and iterative regularization approach without using outer iterations for general blur operators. We consider various examples from [68] and perform the reconstructions with TPR, TSVD, and CGLS using a different number of partitions. We restrict ourselves to the OPTIMAL case being able to compute the optimal regularization parameter using the exact



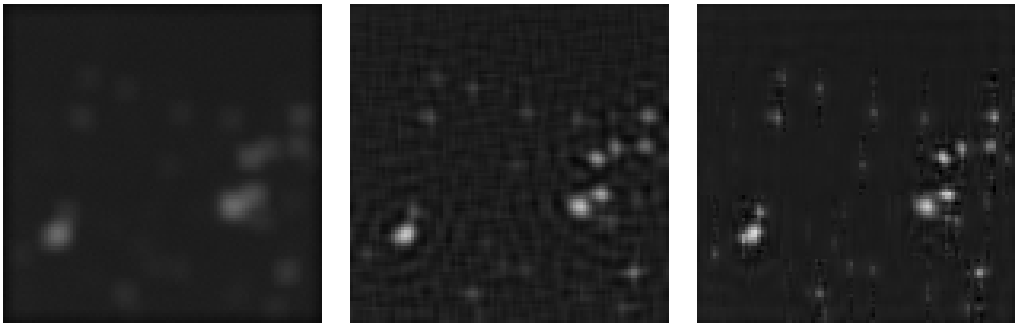
(a) Partition boundaries of $b_{\mathtt{b}}$ in 2D.

(b) Corresponding partition boundaries on mesh plot of de-stacked $b_{\mathtt{b}}$.

Figure 6.15.: Non-equidistant partitioning of the observed signal from test problem `blur(30,4,2)` [68] affected with $\xi = 0.06\%$. Instead of planes, we use peaks in (b) to mark the partition boundaries from a de-stacked column-wise point of view of the 2D signal.

Table 6.10.: RRE for problem `blur(30,4,2)` [68] using partitioned Tikhonov-Phillips regularization in Algorithm 11. No outer iterations and 4 partitions are used.

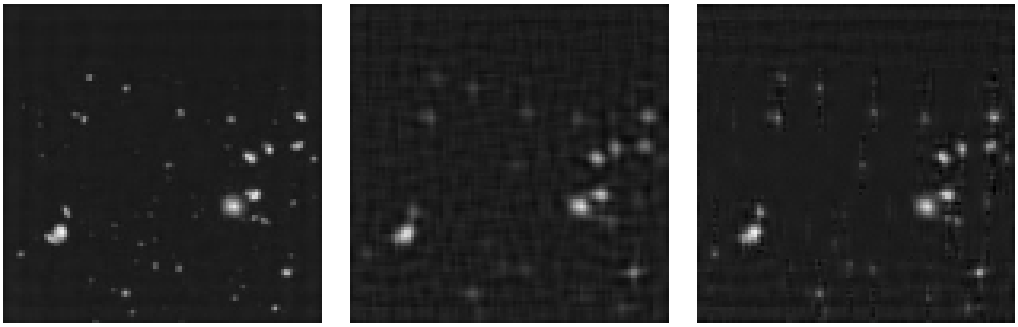
Tikhonov-Phillips	Partitions	α	$\|x_{\mathtt{b}} - \tilde{x}_{\mathtt{b}}\|_2 / \|x_{\mathtt{b}}\|_2$
`blur(30,4,2)`, $n = 30^2$, $band = 4$, $\sigma = 2$, $\xi = 0.06\%$			
OPTIMAL	$(\tilde{x}_{\mathtt{b}})_{1:900}$	0.0165	0.3562
	$(\tilde{x}_{\mathtt{b}})_{1:126}$	0.0288	0.3464
	$(\tilde{x}_{\mathtt{b}})_{127:468}$	0.0202	
	$(\tilde{x}_{\mathtt{b}})_{469:798}$	0.0124	
	$(\tilde{x}_{\mathtt{b}})_{799:900}$	0.0176	
`l_corner`	$(\tilde{x}_{\mathtt{b}})_{1:900}$	0.0085	0.4228
	$(\tilde{x}_{\mathtt{b}})_{1:126}$	0.0158	0.4088
	$(\tilde{x}_{\mathtt{b}})_{127:468}$	0.0085	
	$(\tilde{x}_{\mathtt{b}})_{469:798}$	0.0073	
	$(\tilde{x}_{\mathtt{b}})_{799:900}$	0.0639	
`blur(30,4,2)`, $n = 30^2$, $band = 4$, $\sigma = 2$, $\xi = 0.1\%$			
OPTIMAL	$(\tilde{x}_{\mathtt{b}})_{1:900}$	0.0238	0.4008
	$(\tilde{x}_{\mathtt{b}})_{1:126}$	0.0437	0.3866
	$(\tilde{x}_{\mathtt{b}})_{127:468}$	0.0298	
	$(\tilde{x}_{\mathtt{b}})_{469:798}$	0.0173	
	$(\tilde{x}_{\mathtt{b}})_{799:900}$	0.0263	
`l_corner`	$(\tilde{x}_{\mathtt{b}})_{1:900}$	0.0171	0.4198
	$(\tilde{x}_{\mathtt{b}})_{1:126}$	0.0371	0.4258
	$(\tilde{x}_{\mathtt{b}})_{127:468}$	0.0146	
	$(\tilde{x}_{\mathtt{b}})_{469:798}$	0.0125	
	$(\tilde{x}_{\mathtt{b}})_{799:900}$	0.0943	



(a) $H_{\mathtt{b}}x_{\mathtt{h}} + \eta$. $\xi = 0.02\%$.

(b) CGLS yields error 0.2334.

(c) CGLS on 500 partitions yields error 0.1810.



(d) PCGLS with $D_{\tilde{x}_{\mathtt{h}}^{(1)}}$, $\tilde{x}^{(0)} = x_{\mathtt{h}}$, yields error 0.0408.

(e) PCGLS with $D_{\tilde{x}_{\mathtt{h}}^{(1)}}$, $\tilde{x}^{(0)} = b_{\mathtt{h}}$, yields error 0.2101.

(f) PCGLS with $D_{\tilde{x}_{\mathtt{h}}^{(2)}}$ on 500 partitions yields error 0.1656.

Figure 6.16.: OPTIMAL reconstruction of cut-out of the image `hubble-original` from [54] blurred with the 2D Gaussian operator (6.10) using bandwidth 5, $\sigma = 3$, and $\xi = 0.02\%$. Reconstruction of (c) and (f) is performed with Algorithm ROIP.

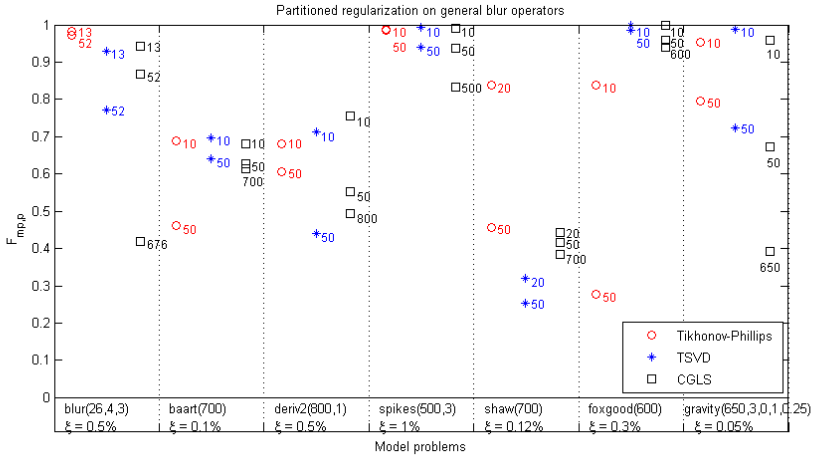


Figure 6.17.: Measure of the error metric $F_{mp,p}$ among problems from [68]. The number of partitions used is shown besides each data point. $F_{mp,p} = 0$ yields the exact solution x. $F_{mp,p} = 1$ means no improvement with respect to an unpartitioned reconstruction.

solution x. For CGLS we provide a reconstruction for the maximum number of possible partitions. We measure the improvement by using p partitions compared to the reference solution $\tilde{x}_1$ reconstructed with one partition, i.e., we use the error metric

$$F_{mp,p} = \frac{\|x - \tilde{x}_p\|_2}{\|x - \tilde{x}_1\|_2},$$

where mp are the model problems illustrated in Figure 6.17. The minimum value $F_{mp,p} = 0$ is optimal because $\tilde{x}_p = x$. A measure of $F_{mp,p} = 1$ means no improvement by reconstructing $\tilde{x}_p$ with p partitions.

Following Figure 6.17, every partitioned reconstruction is more accurate compared to $\tilde{x}_1$. For an increasing number of partitions it is possible to further improve on the solution. However, the improvement is limited by the dimension of the underlying model problem, i.e., the maximum number of partitions (cf. the results for CGLS). Note that for some test scenarios TPR, TSVD, or CGLS may yield strong differences in the improvement for the same p such as for `shaw` or `foxgood`. ●

6.5. Tikhonov-Phillips Regularization with Operator Dependent Seminorms

We introduce a new variant of TPR (see Section 2.5.2), where the penalty term depends on the kernel H. The basic idea is to obtain a penalty term which is able to discriminate between the signal and the noise subspace, that is, which is able to better evaluate whether the reconstructed solution belongs more to the signal than to the noise. In particular, instead of using the classical Tikhonov-Phillips functional in standard form $\|Hx-b\|_2^2+\alpha^2\|x\|_2^2$ (2.32) or its generalized representation $\|Hx - b\|_2^2 + \alpha^2\|Lx\|_2^2$ (2.35), where L is a fixed operator which measures the oscillations such as the first or the second derivative (2.36), we propose to solve

$$\begin{aligned} \min_x \left\| \begin{pmatrix} H \\ \alpha L_H \end{pmatrix} x - \begin{pmatrix} b \\ 0 \end{pmatrix} \right\|_2^2 &= \min_x \left\{ \|Hx - b\|_2^2 + \alpha^2 \|L_H x\|_2^2 \right\} \\ &\Leftrightarrow \quad (H^T H + \alpha^2 L_H^T L_H)x = H^T b, \end{aligned} \tag{6.22}$$

where L_H is an H-dependent linear operator which acts like the identity on the noise subspace and like the null matrix on the signal subspace, i.e., which acts like a high-pass filter in the Fourier context. Since the signal and noise subspace depend on H—and on the error vector η (2.29)—an H-dependent linear operator is expected to work better than a fixed operator, when used in the penalty term. Here, $\|L_H x\|_2$ represents a seminorm (see Section 2.5.2) as L_H usually has a nontrivial null space.

Our approach aims at deriving such operators which directly include H and thus use spectral information of the underlying problem. The modified regularization method (6.22) allows larger values of α while improving the reconstruction. That is because it acts separately on the frequency subspaces allowing to put more weight to the noise domain, in contrast to TPR in standard form where the uniform shift has influence on the whole spectrum. In general, a regularizing seminorm should have the property that it leads to a regularizing effect for the noise subspace relative to small eigenvalues, but behaves like zero on the signal

subspace relative to large eigenvalues. In the limiting case, a seminorm should behave like 1 on the noise part providing much emphasis on the penalty term while it should tend to 0 for the signal part having most emphasis on the model fit. Using the original operator H, we can define such operator dependent seminorms $\|L_H x\|_2$, where $K_H := L_H^T L_H$ is positive semidefinite, in the following three different ways.

6.5.1. Seminorm $\|L_{H,k}x\|_2$ for General Operators

Let $\sigma_{\max}$ be the maximum singular value supposed to be available from $H \overset{\text{SVD}}{=} U\Sigma V^T$ or being an approximation resulting from a couple of Arnoldi iterations. For a fixed $k \in \{1,2,3,\ldots\}$ we introduce the matrix

$$K_{H,k} = L_{H,k}^T L_{H,k} := \left(I - \frac{H^T H}{\sigma_{\max}^2}\right)^k$$

which is symmetric positive semidefinite with a normalized eigenspectrum $0 \le K_{H,k} \le 1$. It is a polynomial in $H^T H$ of the general form $p_k(x) = (1-x)^k$ with $0 \le x \le 1$. Via the power k, it is possible to steer the "L"-shape of $p_k(x)$ which is illustrated in Figure 6.18 (a). For small x, corresponding to singular vectors relative to noise, the function $p_k(x)$ tends to 1, for large k approaching the ordinate. For $x > 0$ and near 1, representing the signal subspace, $p_k(x)$ is close to zero with a zero of order k at $x = 1$. Including $K_{H,k}$, we can modify TPR in standard form (2.32) to the general representation

$$\left(H^T H + \alpha^2 L_{H,k}^T L_{H,k}\right) x = \left[H^T H + \alpha^2 \left(I - \frac{H^T H}{\sigma_{\max}^2}\right)^k\right] x = H^T b. \tag{6.23}$$

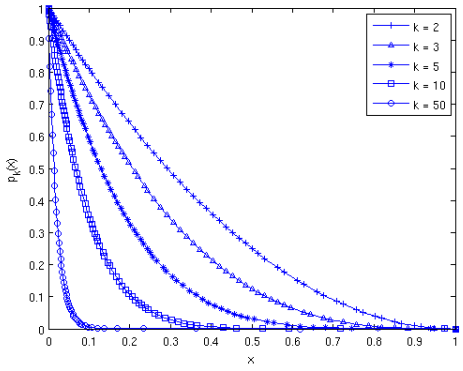
Especially for $k = 1$, (6.23) can be written in the form

$$\left[H^T H + \alpha^2 \left(I - \frac{H^T H}{\sigma_{\max}^2}\right)\right] x = \left[\left(1 - \frac{\alpha^2}{\sigma_{\max}^2}\right) H^T H + \alpha^2 I\right] x = H^T b,$$

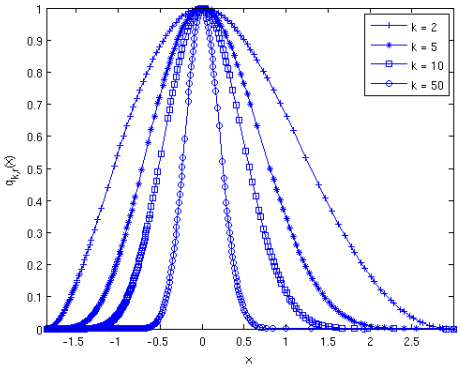
equivalent to standard TPR for a slightly modified problem. Using the SVD of H in (6.23), we observe

$$\begin{aligned} x &= V\left[\Sigma^2\left(1 - \frac{\alpha^2}{\sigma_{\max}^2}\right) + \alpha^2\right]^{-1} \Sigma U^T b \qquad \text{that is,} \\ x &= V\,\mathrm{diag}(\Phi_1,\Phi_2,\ldots,\Phi_n) U^T b \qquad \text{with} \qquad \Phi_i = \frac{\sigma_i^2}{\sigma_i^2 + \alpha^2(1-\sigma_i^2)} \end{aligned} \tag{6.24}$$

being the filter factors for a normalized spectrum with $\sigma_{\max} = 1$. See Figure 6.19 (a) which illustrates the difference to the filter factors of classical TPR (2.34) for $k = 0$. For increasing values of k more SVD components contribute to x with full strength.

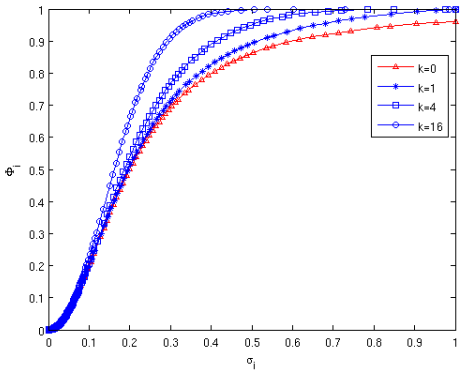


(a) Polynomial $p_k(x) = (1-x)^k$.

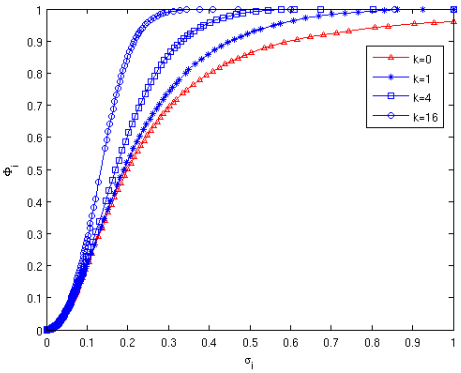


(b) Polynomial $q_{k,r}(x)$ (6.25).

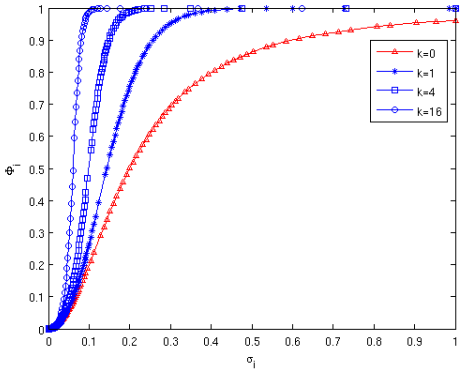
Figure 6.18.: Polynomials for increasing values of k. (b) shows $q_{k,r}(x)$ for the model problem `shaw` [68] where $[\lambda_{\min},\lambda_{\max}] = [-1.86,2.99]$ and r is chosen according to (6.29).



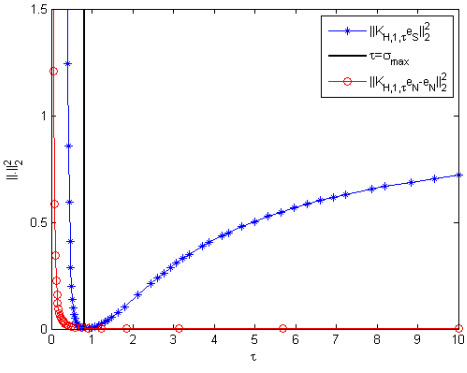
(a) $\Phi_i = \frac{\sigma_i^2}{\sigma_i^2+\alpha^2(1-\sigma_i^2)^k}$, $\alpha = 0.2$.



(b) $\Phi_i = \frac{\sigma_i^2}{\sigma_i^2+\alpha^2(1+2\sigma_i)^k(1-\sigma_i)^{2k}}$, $\alpha = 0.2$.



(c) $\Phi_i = \frac{\sigma_i^2}{\sigma_i^2+\alpha^2(1+10\sigma_i)^k(1-\sigma_i)^{10k}}$, $\alpha = 0.2$.



(d) Conditions (6.30) for $K_{H,1,\tau}$ on problem `heat(300,5.0)`.

Figure 6.19.: (a) corresponds to $\|L_{H,k}x\|_2$. (b) and (c) correspond to $\|L_{H,k,r}x\|_2$ with $\lambda_{\max} = 1$ and respectively $\lambda_{\min} = -\frac{1}{2}$ and $\lambda_{\min} = -\frac{1}{10}$. (d) corresponds to $\|L_{H,k,\tau}x\|_2$.

6.5.2. Seminorm $\|L_{H,k,r}x\|_2$ for Symmetric Indefinite Operators

For symmetric indefinite operators H, where the smallest and largest eigenvalue satisfy $\lambda_{\min} \ll 0$ and $\lambda_{\max} \gg 0$, respectively, it can be helpful to use a seminorm $\|L_{H,k,r}x\|_2$ defined by the matrix

$$K_{H,k,r} = L_{H,k,r}^T L_{H,k,r} := \left[\left(I - \frac{H}{\lambda_{\min}}\right)\left(I - \frac{H}{\lambda_{\max}}\right)\right]^k \left[I + \frac{k}{r}\left(\frac{1}{\lambda_{\min}} + \frac{1}{\lambda_{\max}}\right)H\right]^r.$$

$K_{H,k,r}$ can be analyzed via the polynomial

$$q_{k,r}(x) = \left[\left(1 - \frac{x}{\lambda_{\min}}\right)\left(1 - \frac{x}{\lambda_{\max}}\right)\right]^k \left[1 + \frac{k}{r}\left(\frac{1}{\lambda_{\min}} + \frac{1}{\lambda_{\max}}\right)x\right]^r. \tag{6.25}$$

Again, $k \in \{1,2,3,\ldots\}$ and $r \in \{0,1,2,3,\ldots\}$. This definition makes sense only if the maximum or the minimum eigenvalue is not close to zero. See Figures 6.19 (b) and 6.19 (c) which exemplary illustrate filter factors for $\lambda_{\max} = 1$ and respectively $\lambda_{\min} = -\frac{1}{2}$ and $\lambda_{\min} = -\frac{1}{10}$. We obtain similar behavior compared to the filter factors in (6.24).

The first two terms ensure that $q_{k,r}(x)$ has zeros of order k at the boundaries of $[\lambda_{\min},\lambda_{\max}]$. The additional third term $[\cdot]^r$ in $K_{H,k,r}$ guarantees a regularizing effect near zero, i.e., $q_{k,r}(0) = 1$, without destroying the monotonic behavior. For this purpose, the parameters k and r have to satisfy conditions to avoid negative or very large values and additional maxima of $q_{k,r}(x)$ in $[\lambda_{\min},\lambda_{\max}]$. The polynomial $q_{k,r}(x)$ is positive if

$$\begin{aligned} 1 + \frac{k}{r}\left(\frac{1}{\lambda_{\min}} + \frac{1}{\lambda_{\max}}\right)x \geq 0 \quad &\Leftrightarrow \quad \begin{cases} 1 + \frac{k}{r}\left(\frac{1}{\lambda_{\min}} + \frac{1}{\lambda_{\max}}\right)\lambda_{\min} \geq 0 \\ 1 + \frac{k}{r}\left(\frac{1}{\lambda_{\min}} + \frac{1}{\lambda_{\max}}\right)\lambda_{\max} \geq 0 \end{cases} \\ \Leftrightarrow \begin{cases} \frac{|\lambda_{\min}|}{\lambda_{\max}} \leq 1 + \frac{r}{k} = \frac{k+r}{k} \\ \frac{\lambda_{\max}}{|\lambda_{\min}|} \leq 1 + \frac{r}{k} = \frac{k+r}{k} \end{cases} \quad &\Leftrightarrow \quad \frac{k}{k+r} \leq \frac{\lambda_{\max}}{|\lambda_{\min}|} \leq \frac{k+r}{k}. \end{aligned} \tag{6.26}$$

The zeros z_1, z_2 of $q'_{k,r}(x)$ do not lie inside $[\lambda_{\min},\lambda_{\max}]$ if the following inequalities hold:

$$\lambda_{\max} < z_1 = -\frac{r\lambda_{\min}\lambda_{\max}}{k(\lambda_{\min} + \lambda_{\max})} < \lambda_{\min} \quad \text{and} \tag{6.27}$$

$$\lambda_{\max} < z_2 = \frac{(k+r)(\lambda_{\min}^2 + \lambda_{\max}^2) + 2k\lambda_{\min}\lambda_{\max}}{(\lambda_{\min} + \lambda_{\max})(r+2k)} < \lambda_{\min}. \tag{6.28}$$

By choosing

$$s := \max\left(\frac{|\lambda_{\min}|}{\lambda_{\max}}, \frac{\lambda_{\max}}{|\lambda_{\min}|}\right) \quad \text{and} \quad r := \lceil (s-1)k \rceil \quad \Rightarrow \quad s \leq 1 + \frac{r}{k}, \tag{6.29}$$

all conditions (6.26),(6.27), and (6.28) are satisfied. Figure 6.18 (b) shows $q_{k,r}(x)$ for increasing values of k for the spectrum of H resulting from the model problem `shaw` from [68]. The shape of $q_{k,r}(x)$ slightly resembles a Gaussian bell curve. For increasing values of k it falls towards the abscissa tending to the δ-function.

6.5.3. Seminorm $\|L_{H,k,\tau}x\|_2$ for General Nonsymmetric Operators

A third seminorm can be defined for the general nonsymmetric case via

$$K_{H,k,\tau} := \left[\left(I - \frac{H^T}{\tau}\right)\left(I - \frac{H}{\tau}\right)\right]^k,$$

where we have to make sure that $K_{H,k,\tau} \geq 0$ in order to have a related seminorm $\|L_{H,k,\tau}x\|_2$ with $K_{H,k,\tau} = L_{H,k,\tau}^T L_{H,k,\tau}$. This is satisfied for even k or τ chosen large enough. In general, we can estimate τ in order to obtain positivity. For $k = 1$ and exploiting $\|H + H^T\|_2 \leq 2\|H\|_2$, we observe

$$x^T K_{H,1,\tau} x = x^T x - 2\frac{x^T H x}{\tau} + \frac{x^T H^T H x}{\tau^2} \geq x^T \left(I - \frac{H + H^T}{\tau}\right) x \geq 0 \quad \text{for} \quad \tau > \|H + H^T\|_2.$$

Furthermore, τ has to be chosen in such a way that for the signal subspace the regularization is turned off, while for the noise subspace it is turned on. Again, we can heuristically model the signal and noise subspace by the probing vectors e_S and $e_N = n^{-\frac{1}{2}}(1,-1,1,-1,...)^T$, respectively, with τ chosen such that

$$\|K_{H,k,\tau}e_S\|_2^2 \ll 1 \qquad \text{and} \qquad \|K_{H,k,\tau}e_N - e_N\|_2^2 \ll 1. \tag{6.30}$$

In other words, we suggest to choose τ such that the seminorm acts as 0 on the signal subspace and like the identity I on the noise subspace. By plotting the polynomials in τ—resulting from $\|K_{H,k,\tau}e_S\|_2^2$ and $\|K_{H,k,\tau}e_N - e_N\|_2^2$—we can visually choose τ according to a value which satisfies the conditions (6.30). In many cases we observed $\tau = \sigma_{\max}$ yielding improved reconstructions. Moreover, $K_{H,k,\tau} > 0$ in this case. See Figure 6.19 (d) where the norms in (6.30) are plotted for $k = 1$ for the `heat(300,5.0)` problem [68]. Note that for $H = H^T > 0$ the choice $\tau = \sigma_{\max}$ leads to a seminorm different from $K_{H,k}$.

While seminorms incorporating derivative operators such as L_1 or L_2 (2.36) lead to an improved reconstruction if the signal is smooth, the introduced operator dependent seminorms are not restricted to the structure of the signal. In order to obtain polynomials of higher degree it is possible to use the scaling and squaring in the matrix power. Thus, degrees of 2^t will only require $\log_2(2^t) = t$ multiplications. We want to emphasize the equivalence of TPR with seminorm $\|L_H x\|_2^2$ and TPR applied to the preconditioned operator HL_H^{-1}. Furthermore, TPR in Banach space with l^p-norm penalty term $\|x\|_p^p$ is related to TPR with seminorm penalty term $\|L_H x\|_2^2$ by considering L_H formally as a data dependent diagonal matrix $L_H = \text{diag}(x_1^2, \ldots, x_n^2)$ [91], i.e.,

$$\left\|L_H^{\frac{p-2}{2}} x\right\|_2^2 = \sum_j x_j^{p-2} x_j^2 = \sum_j x_j^p = \|x\|_p^p \qquad \text{with} \qquad 1 < p < +\infty.$$

6.5.4. Numerical Experiments

We focus on 1D ill-posed problems from Regularization Tools [68] and thus on Fredholm integral equations of the first kind (2.28). Still, we assume that the entries of η in (2.29) are random numbers resulting from the same Gaussian distribution with zero mean and

standard deviation $\xi \in \mathbb{R}$. Hence, we use white noise of different magnitude and perform all computations on normalized values. Our direct regularization method is TPR.

Concerning the regularization parameter α we illustrate the maximum obtainable improvement for a perfect estimator. We compute it via the MATLAB function `fminbnd` which attempts to find a local minimizer in a given interval. We only consider reconstructions as reasonable where $\alpha \in [0,10^3]$. Thus we fade out reconstructions for which `fminbnd` hits the interval boundary ($\alpha = 10^3$) and highlight them via †. We refine the search by setting the termination tolerance to `TolX` $= 10^{-9}$. See Listing 6.1 for a brief illustration how we incorporate a Tikhonov-Phillips function handle in `fminbnd` in order to obtain the optimal α. Note that `fminbnd` is based on golden section search and parabolic interpolation and may provide only a local solution [108], i.e., the estimation may still not be optimal for certain problems. Hence, we measure the quality of the reconstructions via the relative reconstruction error (RRE) $\|\tilde{x} - x\|_2/\|x\|_2$, where x is the exact signal and $\tilde{x}$ denotes the reconstruction which is obtained for `opalpha` according to Listing 6.1. All our test problems have a fixed problem size $n = 300$. We emphasize reconstructions from TPR in standard form with values marked in bold style and curves in solid red color.

Listing 6.1: Computing α yielding the optimal reconstruction.

```
option=optimset('TolX',1e−9);
maxbnd = 1000;
[opalpha,opval,ex] = fminbnd(@(alpha) Tikhonov_Phillips(H,b,x,alpha),0,maxbnd,option);
```

Numerical Experiment 6.18 : the general case with test problems `baart` and `heat`. We are interested in using the seminorms $\|L_{H,k}x\|_2$ and $\|L_{H,k,\tau=\sigma_{\max}}x\|_2$ for the general nonsymmetric case. The test problem `baart` is an integral equation according to (2.28) with

$$H(s,t) = e^{s\cos(t)}, \qquad g(s) = \frac{2\sin(s)}{s}, \quad \text{and} \quad s \in [0,\tfrac{\pi}{2}],\ t \in [0,\pi].$$

Here, we obtained degradation with smoothing norms using derivative operators according to (2.36) and therefore do not use them in our results. In contrast to this, smoothing norms yield highly improved reconstructions for the test problem `heat` which models the inverse heat equation. The ill-conditioning of H can be steered by the parameter `kappa`: we choose `kappa` $= 5$ and obtain

$$H(s,t) = h(s-t), \qquad h(t) = \frac{t^{-\frac{3}{2}}}{10\sqrt{\pi}} e^{-\frac{1}{100t}}, \quad \text{and} \quad s,t \in [0,1],$$

with $\kappa_2(H) = 1.77\text{E}{+}17$. We affect the right-hand side in both examples with white noise of magnitude 10%, 5%, 1%, and 0.1%. As polynomial degrees we choose $k = 2^0,2^1,\ldots,2^5$. Besides the results in the Tables 6.12 and 6.13, we illustrate two additional information for $\xi = 1\%$ and $\xi = 5\%$ in the Figures 6.20 and 6.21. While fixing the polynomial degree at $k = 16$ we plot α versus the RRE and show the associated reconstructions for the optimal regularization parameter.

Any choice of α provides an improved reconstruction using operator dependent seminorms as their curves remain below the one corresponding to TPR in standard form. Incorporat-

ing smoothing norms for the `heat` problem yields highly improved reconstructions as well. Among many of our experiments we obtain improved results using the seminorm $\|L_{H,k,\tau}x\|_2$ when the heuristic conditions (6.30) are satisfied. See also Figure 6.19 (d). However, this is not a necessary condition. Compare the value 2.7281 in Table 6.11 with the results in Table 6.13. This also reflects that the second condition in (6.30) is a heuristic. ●

Table 6.11.: Values of (6.30) for $\|L_{H,k,\tau=\sigma_{\max}}x\|_2$ and problem `heat(300,5.0)`. $\xi = 5\%$.

Norm \ Degree	$k=4$	$k=8$	$k=16$
$\|K_{H,k,\tau=\sigma_{\max}}e_S\|_2^2$	0.0007	0.0005	0.0009
$\|K_{H,k,\tau=\sigma_{\max}}e_N - e_N\|_2^2$	0.0763	0.3949	2.7281

Numerical Experiment 6.19 : the symmetric indefinite case with test problem `shaw`. We illustrate the behavior of the seminorms $\|L_{H,k,r}x\|_2$ and $\|L_{H,k}x\|_2$ for the test problem `shaw` where the spectrum of H satisfies the necessary conditions $\lambda_{\min} = -1.86 \ll 0$ and $\lambda_{\max} = 2.99 \gg 0$. See the characteristics of the polynomial $q_{k,r}(x)$ for this problem in Figure 6.18 (b). The scenario setting is the same as in Numerical Experiment 6.18 except that we fix the polynomial degree at $k = 32$ for $\xi = 5\%$ in Figure 6.22. Thus, we obtain better distinguishable curves. Following Table 6.14, for noise of large magnitude the seminorm $\|L_{H,k,r}x\|_2$ yields improved results while it does not degrade the solution for small ones. In comparison, the seminorm $\|L_{H,k}x\|_2$ yields similar behavior with slightly weaker improvement. ●

Numerical Experiment 6.20 : further problems from Regularization Tools**.** In order to give an overall summary we provide results for further problems and right-hand sides. In Table 6.15 we incorporate $\|L_{H,k}x\|_2$ and $\|L_{H,k,\tau}x\|_2$ for signals perturbed by noise of magnitude 10%, 5%, and 1%. For model problems with smooth right-hand side we additionally provide results incorporating smoothing norms (2.36). As expected, they yield much improvement in such cases—see the `i_laplace(n,2)` problem. While here the seminorm $\|L_{H,k}x\|_2$ does not affect the reconstruction quality among all magnitudes of noise, it improves the solution for problems with a non-smooth right-hand side, e.g., `foxgood` or `gravity`. The difference compared to $\|Ix\|_2$ becomes more distinct for noise of larger magnitude. Incorporating the seminorm $\|L_{H,k,\tau}x\|_2$ is sensitive both to k and ξ. On the one hand we obtain improvement for the problem `phillips` for $\xi = 1\%$ or for `gravity` with $\xi = 10\%$ and $\xi = 5\%$. On the other hand, we observe a degradation for the problems `foxgood` or `deriv2`.

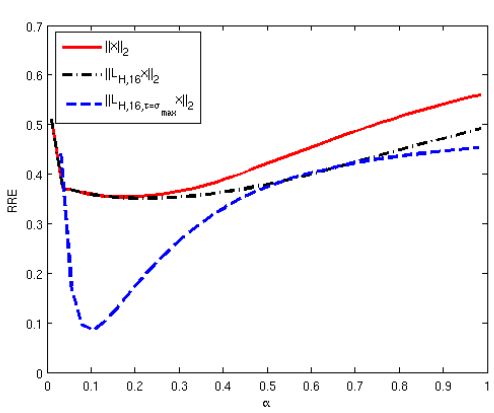
Finally, we are interested in the behavior using different right-hand sides for each model problem listed in Table 6.16. Besides those available from Regularization Tools we additionally use the following right-hand sides:

$$
\begin{aligned}
(x_1)_i &:= \sin\left(\tfrac{i\pi}{n}\right), && (x_2)_i := \sin\left(\tfrac{i\pi}{n}\right),\ (x_2)_{i=100,i=200,i=300} = 10\sin\left(\tfrac{i\pi}{n}\right),\\
(x_3)_i &:= i, && (x_4)_i := \frac{\left(i-\lfloor\frac{n}{2}\rfloor\right)^2}{\left(\lceil\frac{n}{2}\rceil\right)^2}, && (x_5)_i := \tfrac{i}{n} + \sin\left(\tfrac{\pi(i-1)}{n}\right),\\
(x_6)_i &:= (0,\ldots,0)^T,\ (x_6)_{i=100,i=200,i=300} = 5,
\end{aligned}
$$

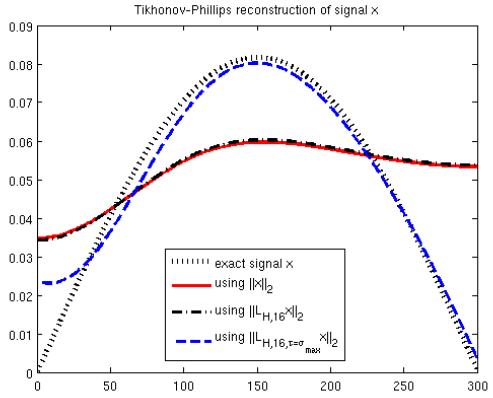
where $i = 1,\ldots,n$. As Regularization Tools provides several right-hand sides for some model problems such as `i_laplace` can be invoked with 4 different ones, the overall number varies between 7 and 10. We plot the improvement obtained from the two seminorms

Table 6.12.: RRE for optimal α in TPR of test problem `baart(300)`.

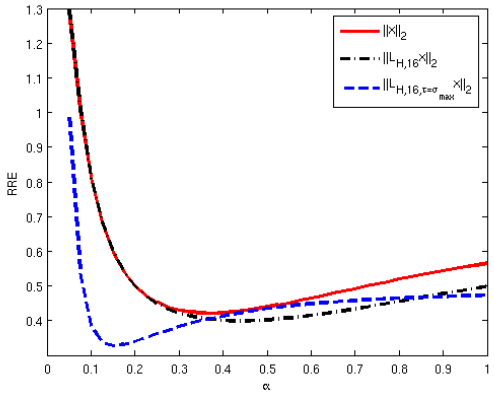
Norm \ Noise	$\xi = 10\%$	$\xi = 5\%$	$\xi = 1\%$	$\xi = 0.1\%$
$\|Ix\|_2$	**0.4638**	**0.4211**	**0.3535**	**0.1275**
$\|L_{1,\sigma}x\|_2$	0.4631	0.4201	0.3534	0.1275
$\|L_{4,\sigma}x\|_2$	0.4562	0.4153	0.3528	0.1275
$\|L_{8,\sigma}x\|_2$	0.4476	0.4095	0.3522	0.1275
$\|L_{16,\sigma}x\|_2$	0.4320	0.3999	0.3510	0.1275
$\|L_{32,\sigma}x\|_2$	0.4071	0.3826	0.3491	0.1275
$\|L_{1,\tau=\sigma_{\max}}x\|_2$	0.4890	0.4839	0.3825	0.1401
$\|L_{4,\tau=\sigma_{\max}}x\|_2$	0.4887	0.4887	0.3119	0.1186
$\|L_{8,\tau=\sigma_{\max}}x\|_2$	0.4928	0.3744	0.1269	0.0639
$\|L_{16,\tau=\sigma_{\max}}x\|_2$	0.4928	0.3272	0.0837	0.0631
$\|L_{32,\tau=\sigma_{\max}}x\|_2$	0.4928	0.4895	0.0595	0.0646



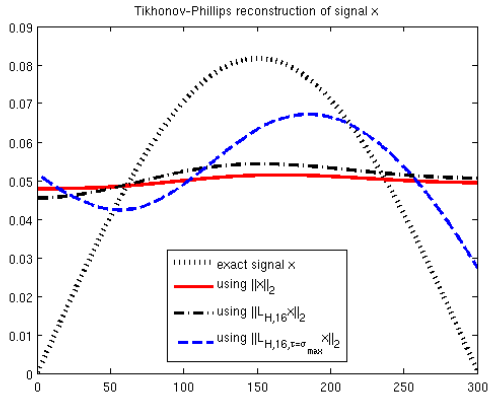
(a) RRE depending on α for $\xi = 1\%$.



(b) Optimal reconstructions for $\xi = 1\%$.



(c) RRE depending on α for $\xi = 5\%$.

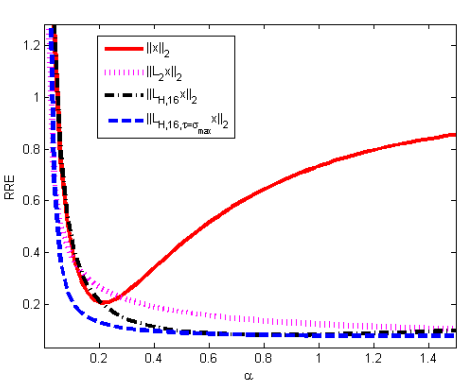


(d) Optimal reconstructions for $\xi = 5\%$.

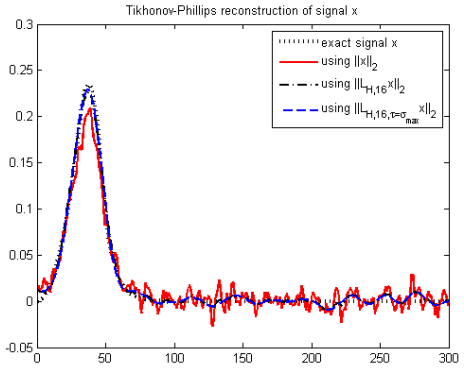
Figure 6.20.: Reconstruction of problem `baart(300)` using different seminorms of degree $k = 16$.

Table 6.13.: RRE for optimal α in TPR of problem `heat(300,5.0)`.

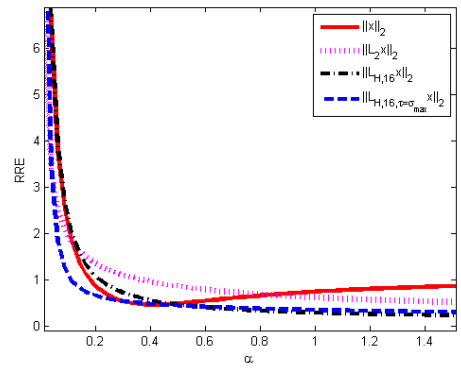
Norm \ Noise	$\xi = 10\%$	$\xi = 5\%$	$\xi = 1\%$	$\xi = 0.1\%$
$\|Ix\|_2$	**0.6041**	**0.4554**	**0.2042**	**0.0540**
$\|L_1 x\|_2$	0.3789	0.2416	0.0869	0.0197
$\|L_2 x\|_2$	0.3545	0.2138	0.0771	0.0172
$\|L_{H,1} x\|_2$	0.5141	0.3847	0.1726	0.0462
$\|L_{H,4} x\|_2$	0.4261	0.2986	0.1270	0.0343
$\|L_{H,8} x\|_2$	0.3789	0.2456	0.0986	0.0269
$\|L_{H,16} x\|_2$	0.3373	0.2080	0.0812	0.0211
$\|L_{H,32} x\|_2$	0.3207	0.2013	0.0785	0.0179
$\|L_{H,1,\tau=\sigma_{\max}} x\|_2$	0.4661	0.3386	0.1483	0.0400
$\|L_{H,4,\tau=\sigma_{\max}} x\|_2$	0.3671	0.2358	0.0923	0.0251
$\|L_{H,8,\tau=\sigma_{\max}} x\|_2$	0.3388	0.2083	0.0780	0.0197
$\|L_{H,16,\tau=\sigma_{\max}} x\|_2$	0.3266	0.2100	0.0769	0.0172
$\|L_{H,32,\tau=\sigma_{\max}} x\|_2$	†	†	0.0789	0.0171



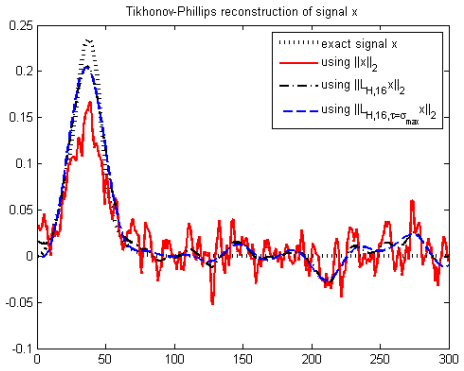
(a) RRE depending on α for $\xi = 1\%$.



(b) Optimal reconstructions for $\xi = 1\%$.



(c) RRE depending on α for $\xi = 5\%$.

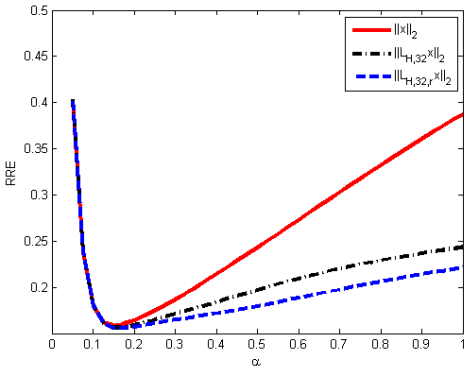


(d) Optimal reconstructions for $\xi = 5\%$.

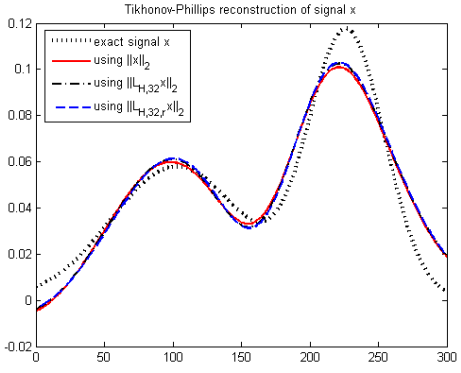
Figure 6.21.: Reconstruction of problem `heat(300,5.0)` using different seminorms with $k = 16$.

Table 6.14.: RRE for optimal α in TPR of problem `shaw(300)`.

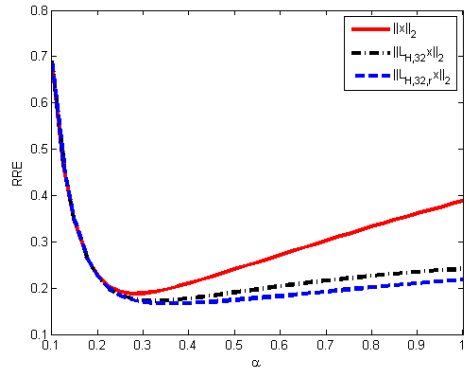
Norm \ Noise	$\xi = 10\%$	$\xi = 5\%$	$\xi = 1\%$	$\xi = 0.1\%$
$\|Ix\|_2$	**0.2195**	**0.1878**	**0.1584**	**0.0990**
$\|L_{H,1}x\|_2$	0.2182	0.1868	0.1582	0.0990
$\|L_{H,4}x\|_2$	0.2125	0.1836	0.1578	0.0990
$\|L_{H,8}x\|_2$	0.2064	0.1805	0.1574	0.0990
$\|L_{H,16}x\|_2$	0.1989	0.1766	0.1569	0.0990
$\|L_{H,32}x\|_2$	†	0.1721	0.1564	0.0990
$\|L_{H,64}x\|_2$	0.1829	0.1672	0.1558	0.0990
$\|L_{H,1,r}x\|_2$	0.2180	0.1862	0.1581	0.0990
$\|L_{H,4,r}x\|_2$	0.2069	0.1806	0.1574	0.0990
$\|L_{H,8,r}x\|_2$	0.1985	0.1762	0.1569	0.0990
$\|L_{H,16,r}x\|_2$	0.1900	0.1714	0.1563	0.0990
$\|L_{H,32,r}x\|_2$	0.1817	0.1665	0.1558	0.0990
$\|L_{H,64,r}x\|_2$	0.1793	0.1652	0.1556	0.0990



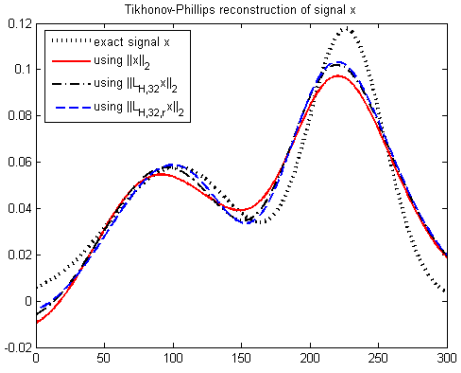
(a) RRE depending on α for $\xi = 1\%$.



(b) Optimal reconstructions for $\xi = 1\%$.



(c) RRE depending on α for $\xi = 5\%$.



(d) Optimal reconstructions for $\xi = 5\%$.

Figure 6.22.: Reconstruction of problem `shaw(300)` using different seminorms of degree $k = 32$.

Table 6.15.: RRE of problems from [68] using TPR.

Problem	Norm \ Noise	$\xi = 10\%$	$\xi = 5\%$	$\xi = 1\%$
`gravity(n,3,0,1,`$\frac{1}{4}$`)`	$\|Ix\|_2$	**0.2243**	**0.1832**	**0.1185**
	$\|L_{H,1}x\|_2$	0.2074	0.1735	0.1179
	$\|L_{H,4}x\|_2$	0.1863	0.1611	0.1173
	$\|L_{H,32}x\|_2$	0.1419	0.1342	0.1193
	$\|L_{H,1,\tau=\sigma_{\max}}x\|_2$	0.1845	0.1601	0.1179
	$\|L_{H,4,\tau=\sigma_{\max}}x\|_2$	0.1512	0.1408	0.1201
	$\|L_{H,8,\tau=\sigma_{\max}}x\|_2$	0.1414	0.1340	†
	$\|L_{H,16,\tau=\sigma_{\max}}x\|_2$	†	0.1525	0.1219
`foxgood(n)`	$\|Ix\|_2$	**0.1097**	**0.0727**	**0.0276**
	$\|L_{H,1}x\|_2$	0.1141	0.0744	0.0276
	$\|L_{H,4}x\|_2$	0.1123	0.0728	0.0268
	$\|L_{H,16}x\|_2$	0.1057	0.0668	0.0237
	$\|L_{H,64}x\|_2$	0.0867	0.0488	0.0145
	$\|L_{H,1,\tau=\sigma_{\max}}x\|_2$	0.1242	0.0834	0.0324
	$\|L_{H,8,\tau=\sigma_{\max}}x\|_2$	0.2117	0.1640	0.0792
	$\|L_{H,32,\tau=\sigma_{\max}}x\|_2$	0.3062	0.2988	0.2964
`deriv2(n,3)`	$\|Ix\|_2$	**0.1882**	**0.1483**	**0.0983**
	$\|L_{H,1}x\|_2$	0.1226	0.1209	0.1203
	$\|L_{H,4}x\|_2$	0.1217	0.1179	0.0951
	$\|L_{H,8}x\|_2$	0.1218	0.1182	0.0943
	$\|L_{H,32}x\|_2$	0.1225	0.1205	0.0897
	$\|L_{H,1,\tau=\sigma_{\max}}x\|_2$	0.2758	0.2059	0.1135
	$\|L_{H,4,\tau=\sigma_{\max}}x\|_2$	0.8962	0.7636	0.3852
	$\|L_{H,8,\tau=\sigma_{\max}}x\|_2$	†	1.0000	0.9977
`i_laplace(n,2)`	$\|Ix\|_2$	**0.8297**	**0.8206**	**0.8116**
	$\|L_1x\|_2$	0.1771	0.1461	0.1248
	$\|L_2x\|_2$	0.1491	0.1163	0.0954
	$\|L_{H,1}x\|_2$	0.8306	0.8208	0.8116
	$\|L_{H,4}x\|_2$	0.8306	0.8207	0.8116
	$\|L_{H,32}x\|_2$	0.8308	0.8200	0.8114
	$\|L_{H,1,\tau=\sigma_{\max}}x\|_2$	0.8976	0.8882	0.8820
	$\|L_{H,4,\tau=\sigma_{\max}}x\|_2$	0.9222	0.9222	0.9222
	$\|L_{H,32,\tau=\sigma_{\max}}x\|_2$	0.9900	0.9898	0.9915
`phillips(n)`	$\|Ix\|_2$	**0.2610**	**0.1787**	**0.0811**
	$\|L_{H,1}x\|_2$	0.2221	0.1566	0.0785
	$\|L_{H,4}x\|_2$	0.1918	0.1399	0.0781
	$\|L_{H,32}x\|_2$	0.1954	0.1358	0.1097
	$\|L_{H,1,\tau=\lambda_{\max}}x\|_2$	0.1955	0.1417	0.0770
	$\|L_{H,4,\tau=\lambda_{\max}}x\|_2$	0.1931	0.1358	0.0740
	$\|L_{H,32,\tau=\lambda_{\max}}x\|_2$	†	0.2333	0.0533

Table 6.16.: Test problems from REGULARIZATION TOOLS used in Figures 6.23 and 6.24.

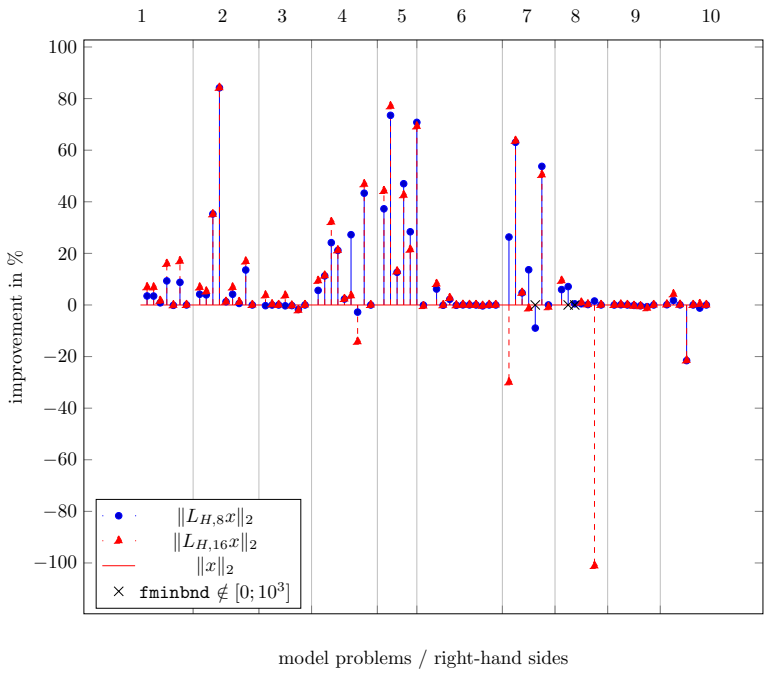
No.	Problem	No.	Problem
1	`baart(n)`	6	`i_laplace(n,[1,2,3,4])`
2	`deriv2(n,[1,2,3])`	7	`phillips(n)`
3	`foxgood(n)`	8	`shaw(n)`
4	`gravity(n,[1,2],0,1,0.5)` `gravity(n,3,0,1,0.25)`	9	`spikes(n,2)`
5	`heat(n,5)`	10	`wing(n,1/3,2/3)`

$\|L_{H,k}x\|_2$ and $\|L_{H,k,\tau}x\|_2$ for $\xi = 10\%$ in Figure 6.23 and for $\xi = 1\%$ in Figure 6.24. Note that in each test problem we first plot the improvement for the right-hand sides from REGULARIZATION TOOLS followed by the improvement for $x_1, \ldots, x_6$. We choose $\tau = \sigma_{\max}$ and $\tau = \lambda_{\max}$ but only plot the better result of both, even when the conditions (6.30) are violated. We summarize the results seminorm dependently:

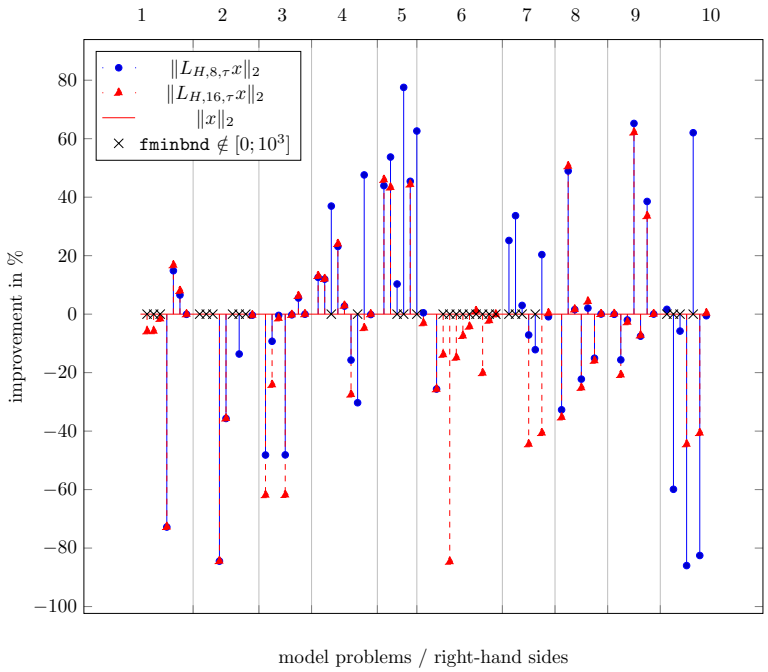
- $\|L_{H,k}x\|_2$: this seminorm provides the most robust behavior, i.e., yields improvement for the most examples and operator spectra while it does not destroy the reconstruction in the remaining problems. Nevertheless, the improvement might be small for certain test problems or smaller compared to the other seminorms; see the `baart` or the `shaw` scenario. Note that even for the indefinite spectrum of the `shaw` problem we obtain an improvement.
- $\|L_{H,k,r}x\|_2$: in case that H is symmetric, the more the indefinite eigenspectrum is separated from zero, and the better $-\lambda_{\min} \approx \lambda_{\max}$, the higher the improvement should be when applying this seminorm. See the `shaw` test problem in the Numerical Example 6.19, where the usage of $\|L_{H,k,r}x\|_2$ leads to best results. As we always choose r according to (6.29) the operator $K_{H,k,r}$ is well-defined. On the other hand, the usage of $\|L_{H,k,r}x\|_2$ will eventually destroy the reconstruction if $\lambda_{\min} \approx 0$ or $\lambda_{\max} \approx 0$, i.e., if H is practically singular.
- $\|L_{H,k,\tau}x\|_2$: for certain general operators, this seminorm may yield better improvement than $\|L_{H,k}x\|_2$. However, one has to choose τ with caution. For nonsymmetric H, τ has to be chosen such that $K_{H,k,\tau} \geq 0$ and that it has a regularizing effect only on the noise subspace. For example, we achieve best results using $K_{H,k,\tau=\sigma_{\max}}$ for the problems `baart` and `heat`. Here, $K_{H,k,\tau=\sigma_{\max}} \geq 0$. Although for `baart` the conditions (6.30) are not satisfied for larger k, we still observe improvement. This reflects the heuristic choice of modeling the subspaces by e_S and e_N. For symmetric H such as in the model problems `phillips` or `gravity 3`, $K_{H,k,\tau=\lambda_{\max}}$ is well-defined and yields improved results too.
- Smoothing norm $\|Lx\|_2$: for model problems containing a smooth right-hand side this well-known seminorm provides highly improved reconstructions. E.g., while reconstructing the `i_laplace(n,2)` or `i_laplace(n,4)` model problems, none of the operator dependent seminorms lead to improved results.

●

Interpretation of the results. We considered the impact of incorporating spectral information of H and therewith use operator dependent seminorms in the Tikhonov-Phillips regularization. Depending on the definiteness and the location of the spectral values, an

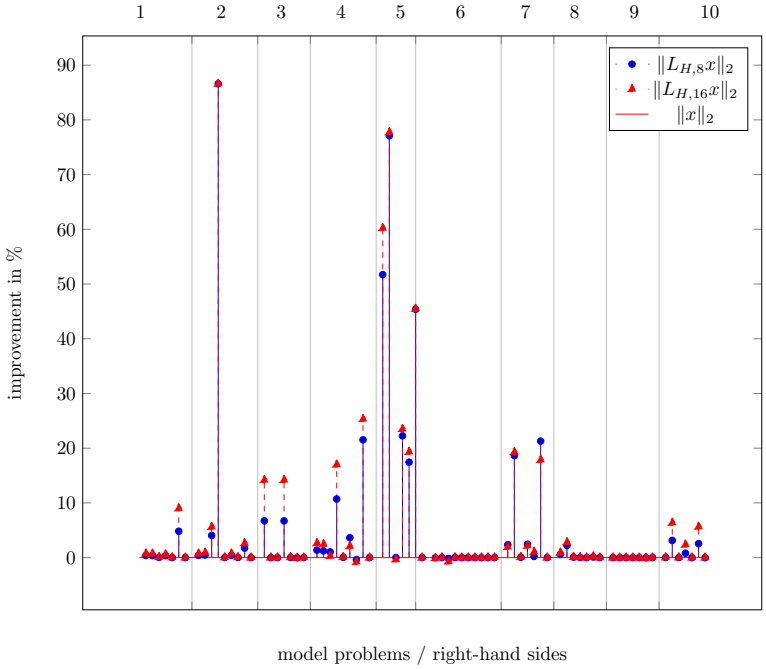


(a) Seminorm $\|L_{H,k}x\|_2$.

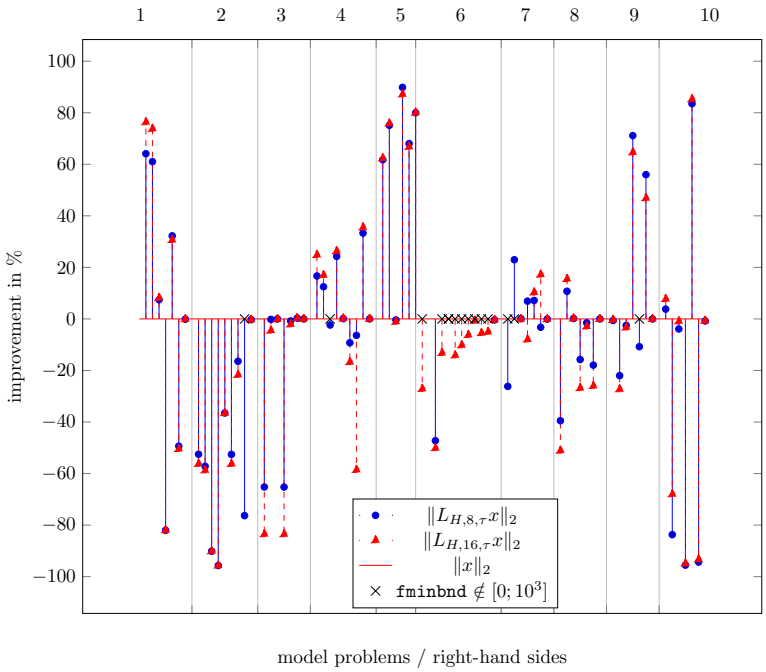


(b) Seminorm $\|L_{H,k,\tau}x\|_2$.

Figure 6.23.: Percental improvement using the seminorms $\|L_{H,k}x\|_2$ and $\|L_{H,k,\tau}x\|_2$ for the polynomial degrees $k = 8$, $k = 16$ and the noise level $\xi = 10\%$. The improvement is measured with reference to standard form TPR. For each of the 10 columns—test problems of Table 6.16—there are several measurements according to the right-hand sides from Regularization Tools and $x_1, \ldots, x_6$.



(a) Seminorm $\|L_{H,k}x\|_2$.



(b) Seminorm $\|L_{H,k,\tau}x\|_2$.

Figure 6.24.: Percental improvement using the seminorms $\|L_{H,k}x\|_2$ and $\|L_{H,k,\tau}x\|_2$ for the polynomial degrees $k = 8$, $k = 16$ and the noise level $\xi = 1\%$. The improvement is measured with reference to standard form TPR. For each of the 10 columns—test problems of Table 6.16—there are several measurements according to the right-hand sides from Regularization Tools and $x_1, \ldots, x_6$.

appropriate seminorm, corresponding to a polynomial with certain properties, will have a regularizing effect on the noise subspace but no action on the signal subspace. The improvement of the solution can be enhanced by increasing values of the polynomial degree. Here, the computation should be based on the scaling and squaring according to the matrix power to keep it efficient. As a brief summary we point out:

- The seminorm $\|L_{H,k}x\|_2$ yields robust behavior meaning that it will not destroy the solution if not appropriate for an underlying problem.
- For indefinite operators satisfying $\lambda_{\min} \ll 0$ and $\lambda_{\max} \gg 0$ the seminorm $\|L_{H,k,r}x\|_2$ is an alternative to obtain improved reconstructions.
- If τ is chosen carefully to satisfy the mentioned conditions, the seminorm $\|L_{H,k,\tau}x\|_2$ will produce improved results for certain problems as well.
- If the signal is known to be smooth, smoothing norms will yield distinct improvement.

Among several of our test problems we observed that the larger the magnitude of the noise, the more the improvement will be when applying the seminorms. Note that operator dependent seminorms rely on the assumption that a priori knowledge of the extremal spectral values is available. If not, however, approximations can be obtained via a couple of Arnoldi iterations.

6.6. Data Based Regularization Matrices for the Tikhonov-Phillips Regularization

We are interested in combining data based regularization with differential operators L_k (2.36), where $k = 1, 2$, in the general form of TPR. Following Section 2.5.2, discrete smoothing norms will improve the reconstruction if the underlying signal is smooth because the penalty term $\|L_k x\|_2$ will have no effect on the smooth signal part but provoke large seminorms on the oscillating part corresponding to noise. Observing the exemplary smooth vector $(1,1,\ldots,1)^T$ in the noise-free case, a smoothing norm has no effect on the signal part as

$$L_k(1,1,\ldots,1)^T = (0,0,\ldots,0)^T. \tag{6.31}$$

Note that in [117] a seminorm for TPR is proposed which is constructed by solving $\min_L \|L\tilde{x}\|_2$ for a prescribed structure of L and an approximate solution $\tilde{x}$, e.g., $\tilde{x}_I$ resulting from TPR in standard form.

Here, we propose a combination of data based regularization with differential operators in (2.35) by defining the penalty term

$$\|L_k D_{\tilde{x}}^{-1} x\|_2, \qquad \text{where} \qquad D_{\tilde{x}} := \operatorname{diag}(|\tilde{x}_1|,\ldots,|\tilde{x}_n|) \tag{6.32}$$

similar to (6.18) and $\tilde{x}$ is the best available approximation constructed via TPR using $L = I$ or $L = L_k$, denoted as $\tilde{x}_I$ and $\tilde{x}_{L_k}$, respectively. In the case that

$$|\tilde{x}_i| < \varepsilon_{\mathrm{D}} \qquad \text{we set} \qquad (D_{\tilde{x}})_{ii} = \varepsilon_{\mathrm{D}} \qquad \text{with} \qquad 0 < \varepsilon_{\mathrm{D}} \ll 1, \text{ e.g.,} \qquad \varepsilon_{\mathrm{D}} \in \mathcal{O}(\eta).$$

The seminorm satisfies the condition of acting only on the noise subspace leaving the solution's signal part unchanged. Similar to (6.31), we observe for $k = 1, 2$ that

$$L_k \left(D_{\tilde{x}}^{-1}\tilde{x}\right) = L_k(1,1,\ldots,1)^T = (0,0,\ldots,0)^T.$$

Following Section 6.3.3, we may apply an outer iterative process of constructing $D_{\tilde{x},s}$ based on an available reconstruction $\tilde{x}^{(s-1)}$. I.e., we construct $D_{\tilde{x},1}$ from the best available solution $\tilde{x}^{(0)}$, use (6.32) in (2.35) to obtain the improved signal $\tilde{x}^{(1)}$, construct $D_{\tilde{x},2}$, and so forth. In most cases $s = 3$ is sufficient. Note that in contrast to the data based regularization methods introduced in Section 6.3.2 the number of steps is robust, i.e., no stopping criterion is necessary as usually the solution does not change for larger values of s.

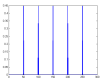
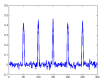
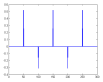
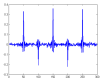
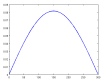
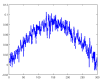
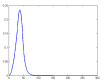
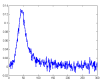
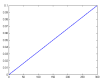
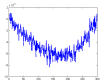
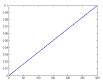
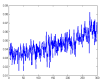
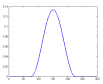
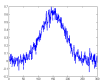
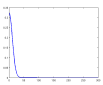
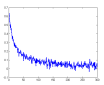
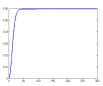
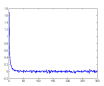
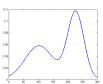
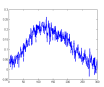
Numerical Experiment 6.21 Our experimental procedure is similar to the one in Section 6.5.4: we compute the optimal $\alpha \in [0,5000]$ using `fminbnd` and fix the size of each problem in Table 6.17 to $n = 300$. Note that for the `prolate` kernel from [108] we use two different signals: a pulse sequence with positive and negative discontinuities and a smooth sine arch.

Following Table 6.18, we obtain distinct improvement when using $L = D_{\tilde{x},s}^{-1}$ for signals satisfying the prerequisites for data based regularization (cf. problem 1 and 2). For smooth signals, where discrete smoothing norms yield significant improvement, the usage of $L = L_k D_{\tilde{x},s}^{-1}$ may additionally improve the reconstruction (see problems 3–7). However, there are also problems such as problem 10 where no improvement can be obtained for neither smoothing norms nor our proposed penalty term (6.32). Note that problem 9 is the inverse Laplace transformation with right-hand side $b(s) = \frac{1}{s(2s+1)}$ which has a singularity at $s = 0$ while the solution $x(t) = 1 - e^{-\frac{t}{2}}$ and thus $x(0) = 0$. This could be the reason for the slight degradation when using data based regularization. ●

Table 6.17.: Problems from Regularization Tools [68] and Matlab [108] used in Table 6.18.

No.	Problem	No.	Problem
1	`blur1D(300,3,2)`	6	`foxgood(300)`
2	`gallery('prolate',300,0.34)`	7	`phillips(300)`
3	`gallery('prolate',300,0.34)`	8	`i_laplace(300,1)`
4	`heat(300,5)`	9	`i_laplace(300,2)`
5	`deriv2(300,1)`	10	`shaw(300)`

Table 6.18.: RRE $\|x - \tilde{x}\|_2/\|x\|_2$ for problems listed in Table 6.17. The exact solution is x while $\tilde{x}$ denotes the reconstruction computed by using the given regularization matrix.

Problem No.	x	$Hx + \eta$	Regularization Matrix	White Noise $\xi = 10\%$	$\xi = 1\%$	$\xi = 0.1\%$
1			I	**0.9165**	**0.5576**	**0.1850**
			L_1	0.9387	0.5668	0.1896
			$L_1 D_{\tilde{x},3}^{-1}$, $\tilde{x}^{(0)} = \tilde{x}_{L_1}$	0.7029	0.1742	0.0258
			$D_{\tilde{x},3}^{-1}$, $\tilde{x}^{(0)} = \tilde{x}_I$	0.3937	0.0159	0.0015
2			I	**0.8956**	**0.5786**	**0.5638**
			L_1	0.9624	0.5783	0.5637
			$L_1 D_{\tilde{x},3}^{-1}$, $\tilde{x}^{(0)} = \tilde{x}_{L_1}$	0.8200	0.0810	0.0082
			$D_{\tilde{x},3}^{-1}$, $\tilde{x}^{(0)} = \tilde{x}_I$	0.5652	0.0285	0.0028
3			I	**0.8204**	**0.1485**	**0.0176**
			L_2	0.1608	0.0208	0.0027
			$L_2 D_{\tilde{x},3}^{-1}$, $\tilde{x}^{(0)} = \tilde{x}_{L_2}$	0.1539	0.0209	0.0027
4			I	**0.6041**	**0.2042**	**0.0540**
			L_2	0.3545	0.0771	0.0172
			$L_2 D_{\tilde{x},3}^{-1}$, $\tilde{x}^{(0)} = \tilde{x}_{L_2}$	0.2224	0.0400	0.0100
5			I	**0.5625**	**0.4343**	**0.2480**
			L_2	0.0853	0.0085	0.0009
			$L_2 D_{\tilde{x},3}^{-1}$, $\tilde{x}^{(0)} = \tilde{x}_{L_2}$	0.0593	0.0059	0.0006
6			I	**0.1097**	**0.0276**	**0.0074**
			L_2	0.0980	0.0098	0.0010
			$L_2 D_{\tilde{x},3}^{-1}$, $\tilde{x}^{(0)} = \tilde{x}_{L_2}$	0.0876	0.0088	0.0009
7			I	**0.2610**	**0.0811**	**0.0271**
			L_1	0.2401	0.0777	0.0266
			$L_1 D_{\tilde{x},3}^{-1}$, $\tilde{x}^{(0)} = \tilde{x}_{L_1}$	0.2104	0.0650	0.0268
8			I	**0.2937**	**0.2127**	**0.1724**
			L_1	0.3190	0.6478	0.6529
			$L_1 D_{\tilde{x},3}^{-1}$, $\tilde{x}^{(0)} = \tilde{x}_{L_1}$	0.1403	0.0282	0.0085
9			I	**0.8297**	**0.8116**	**0.7883**
			L_2	0.1491	0.0954	0.0917
			$L_2 D_{\tilde{x},3}^{-1}$, $\tilde{x}^{(0)} = \tilde{x}_{L_2}$	0.1576	0.1302	0.1309
10			I	**0.2195**	**0.1584**	**0.0990**
			L_2	0.5713	0.5401	0.5381
			$L_2 D_{\tilde{x},3}^{-1}$, $\tilde{x}^{(0)} = \tilde{x}_{L_2}$	0.5819	0.5484	0.5459

Chapter 7

Parallel Implementations of Sparse Approximate Inverses

This chapter deals with the technical realization of sparse approximate inverses introduced in Chapter 3. Their inherent parallelism allows to obtain highly scalable code for modern high performance clusters. Including this work, the following packages are state of the art and provide implementations of sparse approximate inverses based on Frobenius norm minimization and K-condition number minimization, respectively:

- `spai-3.2` [57]: Barnard's et al. [10, 11] parallel implementation of the SPAI preconditioner. Based on $\mathcal{J}(M) = \mathcal{J}(I)$, a SPAI is computed according to Section 3.1.3.
- `mspai-1.2` [87]: Our sequential and parallel implementation of the (M)S(P)AI preconditioner based on the Sections 3.1.1, 3.1.3, and 3.1.4.
- `fspai-1.1` [89]: Our sequential and parallel implementation of the FS(P)AI preconditioner according to Section 3.2.1 and 3.2.3, respectively.
- ParaSails [36]: Chow's parallel implementation of the (F)SAI approach using patterns of powers of sparsified A according to Section 3.1.2 and 3.2.2.

Designed for distributed memory architectures, all implementations realize the communication between the processing elements (PEs) of a cluster environment via the *message passing interface* (MPI) [111]. Note that `spai-3.2` and ParaSails are written in `C` while `mspai-1.2` and `fspai-1.1` use `C++` which allows generic implementations.

The pattern update mechanism of the first three codes requires a sophisticated communication concept such that the data exchange preserves the inherent parallelism of the algorithms. Therefore, we first describe the underlying and similar parallelization concept of these implementations. The subsequent two sections focus on `mspai-1.2` and `fspai-1.1`. We discuss the enhancements of the new `mspai-1.2` code compared to its previous versions: a generic dictionary approach to avoid expensive but redundant computations as well as memory optimization techniques. The last part introduces our new FSPAI implementation `fspai-1.1` for SPD/HPD systems. The description of its features and its basic code design is followed by notes on the implementation of the τ_{jk} computation. We also transfer `mspai-1.2`'s dictionary approach to FSPAI and illustrate the effect of avoiding redundant Cholesky factorizations. Motivated by the gained speedup of using sparse QR decompositions via CSparse in `mspai-1.2`, we analyze the effect of sparse Cholesky decompositions using CXSparse.

In both sections we end up with a runtime optimized new implementation of the MSPAI and FSPAI preconditioner, respectively, and illustrate that our enhancements do not affect the scalability of the codes. For this purpose, we provide strong scaling results on two

different architectures. Moreover, by constructing preconditioners for large-sized systems using a large number of PEs, we provide first insight into the scaling behavior for massively parallel settings. In the SPD case, we compare `fspai-1.1` with the ParaSails approach implemented in the Hypre library. Here, we provide results both for the setup of the preconditioner and its application within PCG.

7.1. Parallelization Concept

The parallelization strategy of the `mspai-1.2` and `fspai-1.1` implementation is based on that of the SPAI implementation `spai-3.2` [10, 11] but enhanced depending on the functionality of the package. The basic concept of all three implementations can be outlined as follows.

7.1.1. Data Distribution

The implementations use a data decomposition approach and uniformly scatter the columns of C/A and $\mathcal{J}(M)/\mathcal{J}(L)$ across the PEs such that the number of columns a PE is responsible for differs by at most one. They do not allow for a cyclic assignment of the columns. Thus, each PE stores a submatrix of consecutive columns in the CSC format (see Section 2.2.1) which allows for an efficient column-wise access and is the obvious choice for algorithms which offer an independent consideration of the preconditioner columns. Note that in case that C/A is symmetric, requests for rows of the matrix, e.g., to identify the sets $\mathcal{N}_\ell$ required to construct $\tilde{\mathcal{J}}_k$ (3.12) within the pattern updates, resemble column requests as $A_j(\mathcal{J})^T = A_j(\mathcal{J})$.

The initialization of the (local) data structures represents the *preprocessing stage*. Besides the CSC format, additional information is exchanged between the PEs so as to allow an efficient inter-process communication in the *computation stage*. For instance, the `pe` array represents the column mapping across the PEs: accessing it immediately specifies the PE which contains the requested column of C/A or $\mathcal{J}(M)/\mathcal{J}(L)$ in its local memory. During the setup of M or L, PEs will require data which is not locally available if the underlying environment is a distributed memory architecture. Hence, an efficient communication strategy becomes necessary which prevents slowing down the inherent parallelism of the algorithms.

7.1.2. One-Sided Communication

Early published parallel implementations on sparse approximate inverses [10, 11] using MPI were restricted to routines available in the standard at that time. As one-sided communication was not available it was implemented artificially using a kind of abstract client/server model. Although the current MPI standard [111] offers routines for one-sided communication, both the `mspai-1.2` and the `fspai-1.1` implementation make use of the communication server (communication handler) technique introduced by Barnard et al. [11] because it proves to be highly scalable on distributed memory environments. See the scalability results in the Sections 7.2.3 and 7.3.5. Our generic codes required to convert the communication model into a `C++` template based [131] form. Depending on the functionality of our packages, we enhanced the polling technique with further service requests.

The PEs run totally asynchronously with no barrier until M or L are constructed. Therefore, if a PE requires remote data, it sends a conditional message tag before falling into a non-blocking wait state until the request arrives. These requests are handled by the communication server that occasionally probes for such requests. There are several requests handled by the communication server:

1. Another PE requires a column of C/A which is not part of its work chunk and not in its local hash table (see Section 7.1.3). In `mspai-1.2`, there is also an additional event for requesting target columns of B.
2. Another PE requires a column of the current pattern $\mathcal{J}$ which is not in its work chunk. Note that in `mspai-1.2` there is an additional event for requesting columns of a prescribed upper bound pattern (see Section 3.1.2).
3. Another PE requests for an index of a column of M/L which is not preconditioned yet.
4. Another PE requests for storing a column of M/L.
5. A PE has worked off his local work chunk and informs the master/server PE that it has finished. Note that this PE may still work on chunks of other PEs which are not finished yet.
6. The master/server PE informs all client PEs that the requested approximate inverse M/L has been computed.

The events 2.-4. are part of the load balancing mechanism.

7.1.3. Latency Hiding

To circumvent possibly large latencies of distributed memory architectures, `mspai-1.2` and `fspai-1.1` mask the latency by using asynchronous communication and overlapping computation, similar to `spai-3.2`. Two overlapping regions are implemented:

1. Every PE polls the other PEs at the beginning of each pattern update such that it can receive remote data during the computation of every column of M/L. The routine `Communicate()` implements this polling mechanism.
2. While waiting for remotely (and asynchronously) requested data, a PE is able to handle other client requests. Refer to Figure 7.1 for an exemplary illustration.

An effective way to hide latency is to avoid unnecessary communication. While the preconditioner's column computations run independently of each other, it depends on $\mathcal{J}$ whether there is demand for a column of C/A which the PE is not responsible for. By imposing $\mathcal{J}(A^T)$ or $\mathcal{J}(A)$ as initial pattern, e.g., such a situation may occur already during the setup of a SAI (static approximation). Therefore, columns which are not available on the local node a priori have to be copied from other PEs. To avoid redundant requests, all copied data are cached in a dictionary implemented in form of a generic hash table of fixed (but changeable) size using 5 levels of linear probing [102]. As each node holds a decomposition dictionary of its own, dictionary entries are computed and stored multiple times on different nodes. Searching in the dictionary comes along without any communication and data exchange.

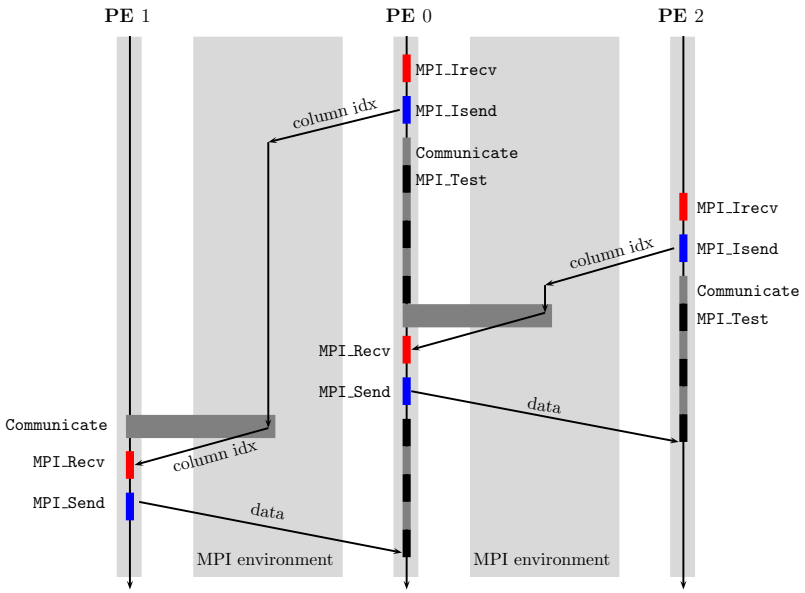


Figure 7.1.: Latency hiding through PE 0. While waiting for remote data from PE 1, client requests from other PEs (here PE 2) are manageable.

7.1.4. Load Balancing

Because the pattern of the inverse is not known a priori, the computational work for a PE is not predictable. This makes a static load balancing impossible. Consequently, although each PE owns approximately the same number of columns, there are likely to arise differences in the computational work among the PEs, depending both on the pattern of the inverse and the initially imposed pattern $\mathcal{J}(M)/\mathcal{J}(L)$. Therefore, to balance the load dynamically, a work stealing strategy is implemented in the following way: after constructing its own preconditioner part, the PE polls the other PEs—consecutively from PE number one—asking for columns which are not processed yet (see event 3. in Section 7.1.2). If it obtains a column index k, it remotely requests the data, constructs the approximate inverse M_k/L_k, and sends the solution back to the responsible PE. Finally, the load balancing mechanism is triggered again. Our performance results show that this work stealing strategy leads to proper scaling behavior of the implementations, also for massively parallel scenarios.

7.2. MSPAI Implementation `mspai-1.2`

In this section we aim to elaborate on the main changes from `mspai-1.0` [99, 128] to `mspai-1.2` as we also make use of them in our new `fspai-1.1` implementation; see Section 7.3. In a summary, our new MSPAI implementation provides the following improvements.

- We enhanced the (M)S(P)AI caching strategy by reducing the overhead resorting to stored QR factorizations. In this context, we also investigated the trade-off between possible key collisions and the costs for computing unique keys. See Section 7.2.1 on this.

- We added (M)S(P)AI hashing in order to analyze the behavior using a dictionary of unbounded size to store the factorizations and LS solutions. See Section 7.2.1.
- We optimized the overall runtime performance of the package. See Section 7.2.2.
- We optimized the IO for generating the pattern data in case that $\mathcal{J}(C)$ was requested. Instead of passing the system matrix file twice, we now generate the data internally from the created `Matrix` object. This becomes especially noticeable for large matrices beyond a size of 1E+05.
- We improved the user friendliness, e.g., we did several bug fixes and enhanced the exception concept.

Refer to the `ChangeLog` file in the `mspai-1.2` package [87] for a brief overview of all changes.

7.2.1. Avoiding Redundant QR Factorizations

Constructing a S(P)AI or an MS(P)AI requires to solve the occurring LS problems by QR factorizations; see (3.7) and (3.24). As $\#\text{flops}_{\text{QR}} \in \mathcal{O}(n^3)$, we have to expect that this is the most expensive part during the static and dynamic setup stage. Indeed, profiling[16] experiments of our implementation confirmed a bottleneck in the LAPACK [1] routine `dgeqrf` being responsible for the QR factorization. On the other hand, matrices with a structured sparsity pattern and reoccurring values—also in the target matrix B—entail the solution of many identical LS problems throughout the computation. Here, the corresponding QR factorizations are computed multiple times. Hence, an obvious idea is to avoid these redundancies and to exploit the reoccurring patterns.

The joint work with Kallischko [99, 128] introduced a generic caching approach holding the intermediate results. The `mspai-1.2` implementation comes along with enhancements on this strategy which are also integrated in our `fspai-1.1` implementation (see Section 7.3.2):

1. We additionally provide a hashing algorithm which resembles the (M)S(P)AI caching approach but implements the dictionary as a generic hash table, instead of a fixed-size array. We describe the generic dictionary approach in the following part. Note that this hash table has nothing in common with the hash table used to store remotely requested data in order to avoid communication. Numerical Experiment 7.1 compares both dictionary realizations on several matrices of increasing dimension.
2. As it is a central task in the dictionary approach, we analyzed the performance and robustness of the subalgorithm responsible for the key computation. See our comments in the next subsection.
3. The `mspai-1.0` implements the caching approach via a fixed-sized array incorporating a *last recently used* (LRU) mechanism. We replaced this data structure by a dynamic `C++ std::list` structure [131] which implements a doubly-linked list. Now, the front of the cache is the last recently used element. By doing so, we were able to merge the priority update mechanism with the search method for finding already computed QR factorizations. These two modifications yield slight speedups of up to 5%.

16 Profiling of our codes was predominantly done with VALGRIND's [129] CACHEGRIND tool and its visualization front-end KCACHEGRIND [143]. In certain cases, we additionally used the sampling profiler GPROF [52].

Dictionary Approach in `mspai-1.2`

After computing a QR factorization, the algorithm stores the resulting factors $\hat{Q}$, $\hat{R}$, and additional information in a dictionary. For the subsequent $\hat{C}_k$, the code looks up the result in the dictionary. If it is not computed yet, the algorithm computes the factorization and notes it down. Although, thus, computations are eliminated, naively searching in the dictionary will almost for certain slow down the application. In (7.1) we introduce the hash function key$(\cdot)$ resembling an integer hash key [102] for the dictionary entries. Each entry in the dictionary equals a 5-tuple $(\hat{C}_k, \hat{Q}, \hat{R}, k_{\hat{B}_k}, \hat{M}_k)$. With $\hat{C}_i = \hat{C}_j \Rightarrow \text{key}(\hat{C}_i) = \text{key}(\hat{C}_j)$, we apply a two-step check:

1. The search for a possible entry thus becomes possible via a bitwise comparison of the keys. If the key $k_{\hat{C}_k}$ is not stored yet, the algorithm terminates the search and starts the QR factorization.
2. If a matching key is found, the entries of the current $\hat{C}_k$ and the $\hat{C}_k^{\text{dict}}$ from the dictionary are compared bitwise. This second check decreases the chance of extracting wrong data in case of occurring key collisions. As long as the dimension of the various $\hat{C}_k$ is low—which is usually the case—this additional step remains cheap.

If all comparisons fail, the factorization is computed (wort case). Otherwise, it is additionally checked whether the key of the right-hand side, denoted as $k_{\hat{B}_k}$, is the same. If so, the solution $\hat{M}_k$ can be extracted from the dictionary and we are finished (best case). In the average case we save at least the expensive QR factorization. See Algorithm 14 for the generic dictionary approach in MSPAI. Note that the comparison and the factorization can run in parallel.

Computation of the Dictionary Keys

The performance improvement of the dictionary approach all depends on the key computation. The key has to be simple to compute, and, for a given number of different $\hat{C}_k$, the number of identical keys (key collisions) should be small. This is a classical hash key challenge where one long bit stream consisting of the values $\hat{c}_{ij}$ of $\hat{C}_k$ and their i, j indices act as input sequence. In our case, the central task is to provide a hash function

$$\text{key} : \mathbb{C}^{p_k \times q_k} \mapsto \mathbb{N}_0^+ \quad (\text{or } \mathbb{R}^{p_k \times q_k} \mapsto \mathbb{N}_0^+) \qquad \text{with} \qquad \text{key}(\hat{C}_k) = k_{\hat{C}_k} \tag{7.1}$$

which comes along with two problems:

1. We have to provide a mapping from double precision floating point values to integer values, i.e., a function

$$\text{subkey} : \mathbb{C} \mapsto \mathbb{N}_0^+ \quad (\text{or } \mathbb{R} \mapsto \mathbb{N}_0^+) \qquad \text{with} \qquad \text{subkey}(\hat{c}_i) = k_{\hat{c}_i}. \tag{7.2}$$

2. The position of the values occurring in $\hat{C}_k$ must be taken into account to compute key$(\hat{C}_k)$. Here, we focus on the column-wise linearization in an array $[\hat{c}_1, \hat{c}_2, \ldots, \hat{c}_{p_k}, \hat{c}_{p_k+1}, \ldots, \hat{c}_{p_k \cdot q_k}]$, used to represent $\hat{C}_k$ in our implementations.

For our purpose, an (unsigned) 32-bit key proved of sufficient range. We solve problem 1 by computing a subkey (7.2) for a 64 bit `double` value $\hat{c}_i$ in the following way: we combine its lower and upper 32-bit representation via an exclusive OR and return the lower 32 bits.

Algorithm 12: Key computation for $\hat{C}_k \in \mathbb{R}^{p_k \times q_k}$.

Input : $\hat{C}_k$ given as array $[\hat{c}_1, \hat{c}_2, \ldots, \hat{c}_{p_k \cdot q_k}]$.
Output: The key $k_{\hat{C}_k} = \text{key}(\hat{C}_k)$.

$k_{\hat{C}_k} \leftarrow 0$
$k_{\text{mix}} \leftarrow \text{subkey}(0.13)$
for $i \leftarrow 1$ **to** $p_k \cdot q_k$ **do**
  $k_{\hat{C}_k} \leftarrow k_{\hat{C}_k} \cdot 31 + k_{\text{mix}} + \text{subkey}(\hat{c}_i)$
end

Algorithm 13: Key computation for $\hat{C}_k \in \mathbb{C}^{p_k \times q_k}$.

Input : $\hat{C}_k$ given as array $[\hat{c}_1, \hat{c}_2, \ldots, \hat{c}_{p_k \cdot q_k}]$.
Output: The key $k_{\hat{C}_k} = \text{key}(\hat{C}_k)$.

$k_{\hat{C}_k} \leftarrow 0$
$k_{\text{mix}} \leftarrow \text{subkey}(0.13)$
for $i \leftarrow 1$ **to** $p_k \cdot q_k$ **do**
  $k_{\hat{C}_k} \leftarrow k_{\hat{C}_k} \cdot 31 + k_{\text{mix}} + \text{subkey}\left[\text{subkey}(\mathfrak{Re}[\hat{c}_i]) \oplus \text{subkey}(\mathfrak{Im}[\hat{c}_i])\right]$
end

Algorithm 14: MSPAI dictionary approach implemented in `mspai-1.2`.

Input : $\hat{C}_k \in \mathbb{C}^{p_k \times q_k}$, $\hat{B}_k \in \mathbb{C}^{p_k}$.
Output: The solution $\hat{M}_k \in \mathbb{C}^{q_k}$.

$k_{\hat{C}_k} \leftarrow \text{key}(\hat{C}_k)$
$k_{\hat{B}_k} \leftarrow \text{key}(\hat{B}_k)$
no_collision $\leftarrow$ **false**
if $k_{\hat{C}_k} \in$ *dictionary* **then**
  $\left(\hat{C}_k^{\text{dict}}, \hat{Q}^{\text{dict}}, \hat{R}^{\text{dict}}, k_{\hat{B}_k}^{\text{dict}}, \hat{M}_k^{\text{dict}}\right) \leftarrow \text{dictionary}\left[k_{\hat{C}_k}\right]$
  if $\hat{C}_k = \hat{C}_k^{dict}$ **then**
    no_collision $\leftarrow$ **true**
  else
    no_collision $\leftarrow$ **false**
  end
end
if *no_collision* **then**
  if $k_{\hat{B}_k} = k_{\hat{B}_k}^{dict}$ **then**
    $\hat{M}_k \leftarrow \hat{M}_k^{\text{dict}}$ {best case}
  else
    $\hat{M}_k \leftarrow (\hat{R}^{\text{dict}})^{-1}(\hat{Q}^{\text{dict}})^T \hat{B}_k$ {average case}
  end
else
  $\hat{Q}$,$\hat{R} \leftarrow \text{qr}(\hat{C}_k)$ {worst case}
  $\hat{M}_k \leftarrow \hat{R}^{-1}\hat{Q}^T \hat{B}_k$
  $\text{dictionary}\left[k_{\hat{C}_k}\right] \leftarrow \left(\hat{C}_k, \hat{Q}, \hat{R}, k_{\hat{B}_k}, \hat{M}_k\right)$
end

By combining all $p_k \cdot q_k$ resulting subkeys according to Algorithm 12 or Algorithm 13, we solve 2. Note that 31 and k_{mix} are used to mix the bit representation of the current key. In the complex case we additionally use an exclusive OR between the real and imaginary part of $\hat{c}_i$. We also tested variants for the key computation such as to replace $k_{\hat{C}_k}$ by subkey($k_{\hat{C}_k}$) in line 4 of Algorithm 12 and 13. However, combined with the mentioned comparison by components, our introduced form of both algorithms leads to a better ratio between computational costs and non-occurring key collisions.

Numerical Experiment 7.1 We study the optimization technique of avoiding redundant QR factorizations for several matrices from MATRIX MARKET [116], matrices resulting from computational fluid dynamics [99], and Laplacian matrices artificially generated on our own. Besides the standard 2D and 3D Laplacians, named LAPLACE2D_1* and LAPLACE3D_1*, we construct generalized 2D Laplacian matrices using a 13-point stencil. We denote them as LAPLACE2D_2*. Our results for constructing M result from one core of a node of the INFINICLUSTER environment D.2. We do not incorporate targeting and probing conditions, i.e., $C = A$ and $B = I$. Hence, we give the speedup of SAI and SPAI with respect to a standard dictionary-free realization. Refer to the environment specific Table D.2 in Appendix D for the used compiler optimization flags.

Following Table 7.1, the caching approach proves to be robust for almost all matrices, i.e., the algorithm is seldom slower than an implementation without a dictionary. The more identical decompositions occur—column four displays the maximum number of different patterns in the static case, i.e., the maximum size of the hash table—the faster the implementation; the memory overhead hereby is determined by the number and memory footprint of the individual dictionary entries.

An LRU scheme with a prescribed maximum dictionary size and a fixed upper threshold of fill-ins bounds the memory requirements of the implementation. This reduction to a fixed set of dictionary entries does not lead to a significant performance breakdown for our test matrices—the breakdowns in the measurements are due to small experiment sizes where measurement noise pollutes the figures. Besides the results listed, we also compared the runtimes for different cache sizes. It turned out that highly structured matrices benefit from a big cache that was able to hold all decompositions (cf. SPAI in Table 7.1). Unstructured matrices could not take advantage of any dictionary where the algorithm's performance suffers from the additional key computations. In an improved version, one can deploy both the QR factorization and the key computation to a thread of their own, and make them compete to deliver the decomposition. This multi-core parallelization is switched off here. All the insights also hold for the dynamic case incorporating pattern updates. ●

The dictionary modification with one dictionary per computing node hereby fits perfectly into the parallelization concept mentioned in Section 7.1. See also our scalability results in [86] on this. However, using a dictionary has two side effects:

- Both the dictionary entries and the remote columns fetched throughout the previous QR factorizations are cached locally and thus increase the memory consumption per node. Prescribing an upper memory footprint of these caches, we ensure that no node of the used cluster runs out of memory: whenever the cache size exceeds the given threshold, we clear it. In our experiments, we always chose the threshold such that M, the dictionary, and the remote cache fit into the local memory of each node. The memory side effect hence is controllable and deterministic.

Table 7.1.: Performance measurements by avoiding redundant LS problems and/or QR factorizations based upon LAPACK in `mspai-1.2`. All results are normalized with respect to the standard S(P)AI—also invokable by `mspai-1.2`—without optimizations, i.e., they give speedups. `mspai-1.2`'s dictionary is realized either as hash table or as an LRU cache. In the static case, the cache size is fixed to 60 elements with $\bar{\Upsilon}_{0,0}$ and $\mathcal{J}(M) = \mathcal{J}(A)$. In the dynamic case, the cache size is adjusted to a heuristic size, the start pattern $\mathcal{J}(M) = \mathcal{J}(I)$, and $\varepsilon_{\text{MSPAI}} = 0.01$.

Matrix		SAI			SPAI			
		Hash table		LRU cache		Hash table	LRU cache	
Name	n	Speedup	Entries	Speedup	Setting	Speedup	Speedup	Cache size
LAPLACE2D_1O10	100	1.28	24	1.28	$\bar{\Upsilon}_{6,5}$	0.98	1.06	60
LAPLACE2D_ 2O10	100	0.88	24	0.86	$\bar{\Upsilon}_{6,5}$	0.82	0.84	60
ORSIRR_2	886	0.76	776	0.85	$\bar{\Upsilon}_{10,6}$	0.80	0.83	20
LAPLACE3D_1O10	1,000	1.90	125	1.89	$\bar{\Upsilon}_{12,8}$	1.04	1.08	1000
PORES_2	1,224	0.64	1221	0.77	$\bar{\Upsilon}_{12,8}$	0.74	0.80	60
OLM2000	2,000	1.87	5	1.80	$\bar{\Upsilon}_{10,8}$	4.67	4.70	500
LAPLACE2D_1O50	2,500	2.97	24	2.90	$\bar{\Upsilon}_{10,8}$	3.07	3.07	1000
FIDAP029	2,870	0.67	2692	0.80	$\bar{\Upsilon}_{12,10}$	0.72	0.85	20
RDB3200L	3,200	2.51	50	2.45	$\bar{\Upsilon}_{10,8}$	3.01	3.00	1000
LAPLACE2D_2O60	3,600	2.16	81	2.15	$\bar{\Upsilon}_{8,8}$	2.29	2.29	800
CFD_SMALL	4,096	2.21	145	2.22	$\bar{\Upsilon}_{8,6}$	1.22	1.24	1000
CFD_LARGE	8,192	2.13	385	2.15	$\bar{\Upsilon}_{6,6}$	1.19	1.19	800
LAPLACE2D_1O100	10,000	2.99	24	2.83	$\bar{\Upsilon}_{5,8}$	2.20	2.19	60
LAPLACE2D_2O100	10,000	2.36	81	2.35	$\bar{\Upsilon}_{5,8}$	1.49	1.48	1000
LAPLACE3D_1O22	10,648	2.35	125	2.34	$\bar{\Upsilon}_{7,8}$	1.80	1.79	1000
LAPLACE2D_1O317	100,489	3.03	24	2.88	$\bar{\Upsilon}_{5,5}$	2.03	2.04	60
LAPLACE2D_2O317	100,489	2.31	81	2.20	$\bar{\Upsilon}_{3,4}$	1.26	1.26	60
LAPLACE3D_1O47	103,823	2.53	125	2.47	$\bar{\Upsilon}_{4,5}$	1.33	1.32	1000
LAPLACE2D_1O1000	1,000,000	3.06	24	2.97	$\bar{\Upsilon}_{6,8}$	2.26	2.22	1000
LAPLACE2D_2O1000	1,000,000	2.15	81	2.18	$\bar{\Upsilon}_{6,8}$	1.62	1.61	1000
LAPLACE3D_1O100	1,000,000	2.39	125	2.38	$\bar{\Upsilon}_{6,8}$	1.81	1.80	1000

- On the other hand, the individual dictionaries typically hold some entries redundantly as the dictionaries act independently of each other, i.e., multiple nodes compute the same entries. As a result, the averaged dictionary hit rate, i.e., the global number of successful dictionary searches on all nodes divided by the global total number of dictionary accesses, decreases. This decay increases the bigger the dictionary is and measurements for different dictionary sizes thus converge to each other.

In a summary, a simpler cache-type realization with an LRU update strategy works as well and comes along with fixed memory requirements. These are important for huge-sized problems where storing all the intermediate results exceeds the memory.

7.2.2. Memory Optimizations

Runtime and profiling tests with `spai-3.2`, `mspai-1.0`, and `mspai-1.1` come up with two bottlenecks during the computation of preconditioners for matrices larger than 1E+05. Both of them are connected with the initialization of memory depending on the size of the underlying problem. Because most of the overall preconditioner setup time is spent in the (re)allocation of memory, e.g., $\gtrsim 90\%$ for many settings, optimizing these parts is in great demand. Note that the following optimizations are catered to save runtimes of the codes, i.e., they can be considered only indirectly as memory footprint optimizations.

Optimizing the Shadow Computation

A key part in the construction of the approximate inverse in all implementations is the computation of the shadow. In SPAI and MSPAI the shadow becomes necessary to compute $\mathcal{I}_k$ (3.4) and $\tilde{\mathcal{J}}_k$ (3.12) while in FSPAI to determine $\hat{\mathcal{J}}_k$ (3.49). In order to avoid a linear searching for each occurring row index in each column, all implementations incorporate fast operations at the bit level. For this purpose, we initialize a sufficiently large temporary buffer `bitvec`—of type `unsigned int`—which will contain the bit representation of the row indices. In case of using a 32-bit architecture, the 32-bit range of each position in `bitvec` is used to represent 32 different row indices. Each bit is used as a placeholder for the corresponding row index, e.g., for the row index 17 the 17^{th} bit of `bitvec[0]` is set to true. Hence, by computing the position `pos` and the shift factor `bit`—according to Listing 7.1—the test whether a row index is already contained in `bitvec` can be performed using bitwise operations according to Listing 7.2.

The unoptimized versions mentioned above initialize `bitvec` as a zero array of size m, where m is the number of rows of A/C, each time right before computing the shadow. Since this reinitialization process is performed multiple times within each update step of each column, the individual initializations accumulate and build a bottleneck. The `mspai-1.2` omits this in the following ways:

1. On 32-bit architectures the minimum number of necessary positions in `bitvec` is not m but $(m \gg 5) + 1$, which is $\frac{m}{32} + 1$. However, this solution only shifts the problem to higher dimensions. Preconditioning matrices of a dimension larger than 1E+06 will suffer again from huge memory initializations.

2. It is unnecessary to reset `bitvec` completely in every iteration. It is sufficient to initialize it once at the beginning—before computing all columns of M with a size

of $\frac{m}{32}+1$—and to reset only those positions which were used in the previous pattern update iteration. Dealing with sparse matrices implies that only very few positions have to be reset. To store the used `bitvec` positions, a second reset array `reset_vec` is initialized once of size m. A counter `reset_len` specifies the first `reset_len` positions within `reset_vec` which contain the positions of `bitvec` to be resetted. See line 15 in Listing 7.1.

Note that we do not apply bitwise operations for `reset_vec` similar to `bitvec` as this turned out to be more expensive. Hence, due to the shifting (line 7 in Listing 7.1), it is possible that `reset_vec` contains redundant values, e.g., both 1 >> 5 and 2 >> 5 lead to position 0 in `bitvec`. However, it is cheaper to initialize `reset_vec` once at the beginning with full size m and reset some positions multiple times in a single loop.

Listing 7.1: Part of computation of int. al. the shadow $\mathcal{I}_k$ in `mspai-1.2` and $\hat{\mathcal{J}}_k$ (3.49) in `fspai-1.1`.

```
1  // Get column Aj/Cj, j ∈ J_k
2  ...
3  for (int idx = 0; idx < col_len; idx++) {
4      // Get the row index value from Aj/Cj
5      r_idx = col_idcs_buf[idx];
6      // log(n) bits of int, 32 bit architecture
7      pos = r_idx >> 5;
8      // bit shift factor
9      bit = r_idx % 32;
10     if (!Bit_Test( bitvec[pos], bit )) {
11         Set_Bit( &bitvec[pos], bit );
12         // Add row index to shadow
13         I->idcs[i_len++] = r_idx;
14         // Add reset position
15         reset_vec[reset_len++] = pos;
16     }
17 }
18 ...
```

Listing 7.2: Routines used in Listing 7.1.

```
1  int Bit_Test(unsigned int bv, int bit) {
2      return ( bv & (1 << bit) );
3  }
4
5  void Set_Bit(unsigned int *bv, int bit) {
6      *bv |= (1 << bit);
7  }
```

Optimizing the τ_{jk} Computation

In order to obtain the τ_{jk} and hence the several ρ_{jk} (3.15), the computation of $r_k^T C_j$ becomes necessary. For that purpose, two strategies mostly differing in the memory usage and the addressing of arrays are obvious:

1. A strategy which manages with less indirect back and forth addressing by working on full back mapped arrays. Two arrays of size m are initialized once at the beginning. First, the array `residual_backmapped` to which the residual values are back mapped from `residual` which is the reduced vector $\hat{r}_k$ (see Remark 3.2 #2). Thus, `residual_backmapped` holds the residual values on the positions indicated by the shadow $\mathcal{I}_k$. Second, the array `Cj` to which the nonzeros of C_j are copied during each τ_{jk} computation.

2. A strategy which manages without memory initializations, working completely on the reduced variant $\hat{r}_k^T C_j(\mathcal{I}_k)$. This variant comes along with many indirect addressings and value comparisons between `residual`, the shadow `I`, and the values c_{ij} accessed via `C->c_lines->C[i][j]`. Note that `c_lines` is the underlying `C++` class structure implementing the slightly modified CSC format.

The above mentioned implementations make only use of Strategy 1 which becomes a bottleneck for large problems, similar to the one in the shadow computation. However, profiling experiments with different matrices and different sparsity showed that the setup time rather depends on the sparsity of C_j and the size of the shadow than on m. For dense updates, in which both the nnz in C_j and the shadow $\mathcal{I}_k$ are large, nonzero Strategy 2 is outperformed by Strategy 1. The choice

$$\text{nnz}(C_j) \cdot |\mathcal{I}_k| \leq 1200 \tag{7.3}$$

revealed to be a reasonable heuristic bound to switch between the two strategies and let the computation run into the specific branch by means of the current sparsity and thus size of upcoming arrays. Hence, independently from m, if (7.3) is satisfied, we invoke Strategy 2, otherwise Strategy 1. See the Listings 7.3 and 7.4 for a brief illustration of the implementation. For large matrices and sparse updates, i.e., for a small shadow, it is more efficient to merely work on the reduced variant. In case of dense updates, initializing memory has a better runtime performance than putting many memory accesses into execution. Note that our experiments only cover a few architectures and the `gcc` compiler [51], i.e., there may occur differences in the runtimes between the two strategies on other environments. Moreover, it is possible that a heuristic choice different to (7.3) may produce more efficient results.

Numerical Experiment 7.2 We are interested in a runtime comparison between our unoptimized version `mspai-1.1`, the `spai-3.2`, and `mspai-1.2` using the memory optimizations described above. We choose Environment D.2 and construct the approximate inverse for the matrices TMT_UNSYM [44] and LAPLACE2D_2O317 which is a generalized 2D Laplacian matrix of dimension $n = 317^2$, assembled by a 13-point stencil. Our MSPAI implementations neither use probing conditions and sparse decompositions nor QR updates. The initial pattern $\mathcal{J}(M) = \mathcal{J}(I)$. Hence, all implementations have an equivalent setting and compute the same SPAI $M \approx A^{-1}$.

Following Table 7.2, the larger the dimension of A, the higher the speedup of `mspai-1.2`. Related to `spai-3.2` and `mspai-1.1`, e.g., for the matrix TMT_UNSYM a speedup of about 114 relative to `mspai-1.1` for 64 cores can be observed. Note that such speedup observations were also affirmed by Fourestey et al. [49] and Sawyer et al. [126] from the CSCS[17]. Moreover, the memory optimizations do not affect the scalability of the implementation. See Section 7.2.3. ●

7.2.3. Strong Scalability

There are two common ways measuring the parallel performance of parallel implementations. In a *strong scaling* scenario the number of cores grows while the problem size remains fixed. In a *weak scaling* the problem size grows with the number of cores such that the amount of work per core remains constant. From an algorithmic point of view, sparse approximate inverses offer an ideal starting point to achieve high scalability due to their

17 Swiss National Supercomputing Center; see http://www.cscs.ch/about_us/staff/index.html?d=16.

Listing 7.3: Heuristic switch in `mspai-1.2`'s τ_{jk} computation depending on the size of the shadow and the nnz in column C_j. Note the code snippet of the routine `Compute_Numerator()` in Listing 7.4. The code is slightly modified to improve the readability.

```
1  ...
2  if((len * I->len) <= 1200) { // Strategy 2.
3      // (r^T * C_j)^2 in nonzero variant, backmapped = false
4      num = Compute_Numerator(residual, C, Cj, j, len, I, false);
5  } else {
6      if(!first_set) { // Strategy 1.
7          res_len = C->m;
8          // Create back mapped residual vector
9          residual_backmapped = new T[res_len];
10         memset(residual_backmapped, 0, res_len * sizeof(T));
11         for (int i = 0; i < I->len; i++)
12             residual_backmapped[I->idcs[i]] = residual[i];
13         Cj = new T[res_len];
14         memset(Cj, 0, res_len * sizeof(T));
15         first_set = true;
16     }
17     memcpy( Cj, C->c_lines->C[j], len * sizeof(T) );
18     // (r^T * C_j)^2 in full variant, backmapped = true
19     num = Compute_Numerator(residual_backmapped, C, Cj, j, len, I, true);
20 }
21 ...
```

Listing 7.4: Main part of routine `Compute_Numerator()` used in Listing 7.3. The code is slightly modified to improve the readability.

```
1  ...
2  if (backmapped)
3      for (int r_idx = 0; r_idx < len; r_idx++)
4          num += residual[C->c_lines->col_idcs[j][r_idx]] * Cj[r_idx];
5  else {
6      for (int i = 0; i < len; i++) {
7          r_idx = C->c_lines->col_idcs[j][i];
8          for (int I_pos = 0; I_pos < I->len; I_pos++) {
9              I_idx = I->idcs[I_pos];
10             if (r_idx < I_idx)
11                 break;
12             if (I_idx == r_idx)
13                 num += residual[I_pos] * C->c_lines->C[j][i];
14         }
15     }
16 }
17 num *= num;
18 ...
```

Table 7.2.: Runtime comparisons between `mspai-1.2`, `spai-3.2`, and `mspai-1.1`, specified in seconds on Environment D.2. The QR factorizations are computed in dense mode using LAPACK without QR updates. No probing conditions are used and $B = I$, i.e., all implementations have an equivalent initial setting and construct the same SPAI.

Matrix	Setting	Cores	`mspai-1.2`	`spai-3.2`	`mspai-1.1`
LAPLACE2D_2O317 $n = 100{,}489$	$\bar{\Upsilon}_{2,10}$ $\varepsilon_{\text{SPAI}} = 0.3$	1	5.77	60.14	75.36
		2	2.98	34.32	40.87
		4	1.53	23.95	22.37
		8	0.93	12.65	12.60
		16	0.68	7.00	6.22
		32	0.97	6.08	6.86
		64	1.57	4.03	4.43
TMT_UNSYM $n = 917{,}825$	$\bar{\Upsilon}_{2,5}$ $\varepsilon_{\text{SPAI}} = 0.3$	1	20.24	> 609	> 1178
		2	10.63	> 609	> 1178
		4	5.43	> 609	> 1178
		8	3.05	609.92	1178.54
		16	1.91	280.11	617.68
		32	1.72	168.72	406.77
		64	2.04	130.73	234.09

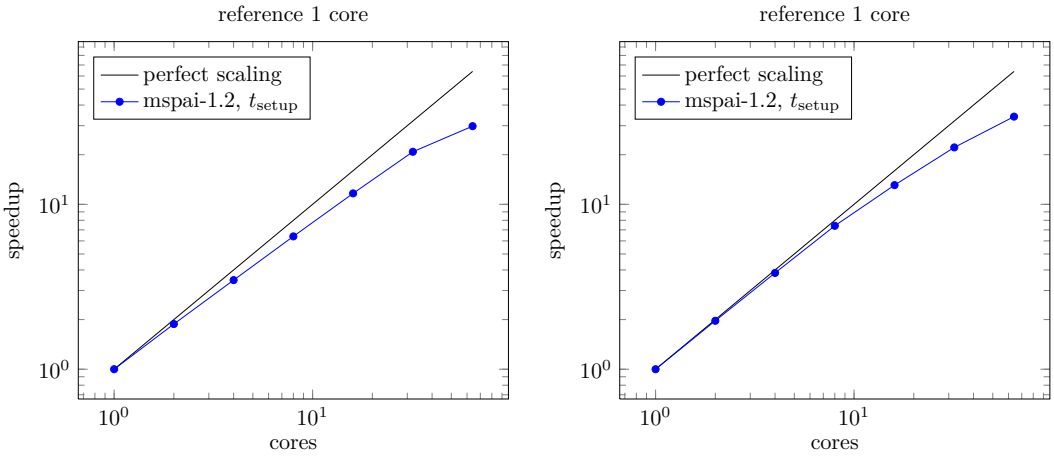
inherent parallelism. In combination with a sophisticated parallelization concept, present implementations such as PARASAILS [36], `spai-3.2` [10], or `mspai-1.1` [86, 99, 128] indeed show outstanding parallel scalability. However, the performance measurements are usually performed only for a low number of PEs; in many cases the maximum number is 128. The purpose of our following experiment is to analyze the strong scalability of `mspai-1.2` using a large number of PEs.

Numerical Experiment 7.3 We construct SPAI preconditioners for four matrices from [44]. We choose the real unsymmetric matrix E40R0100 and the complex unsymmetric matrix KIM1 to be processed on the INFINICLUSTER environment D.2. For the large matrices ATMOSMODJ, resulting from atmospheric modeling, and MEMCHIP, resulting from circuit simulation, we use SHAHEEN's BLUE GENE/P environment D.1. On both clusters we make use of the VN mode exploiting all 4 cores per node. Refer to the captions of Figure 7.2 and Figure 7.3 for the size and nnz of the matrices as well as for the chosen `mspai-1.2` settings. Note that t^c_{setup} denotes the setup time of an MSPAI on c cores (PEs). We use the standard SPAI and LAPACK based setup without any dictionary approach and no QR updates.

Our results show high scalability of the `mspai-1.2` code also for processing large data in massively parallel settings. The superlinear speedup in Figure 7.3 (a) likely arises due to cache effects. Note that we observe constant work load to distribute the data among the PEs, i.e., the overall scalability of `mspai-1.2` is not affected by the data distribution in the preprocessing stage; see Remark #2 in Section 7.2.4. ●

7.2.4. Remarks

1. *Although $\hat{C}_k$ is small compared to C and, typically, much denser, it still exposes a sparse pattern if the current pattern is sufficiently sparse. In [86] we also study the*



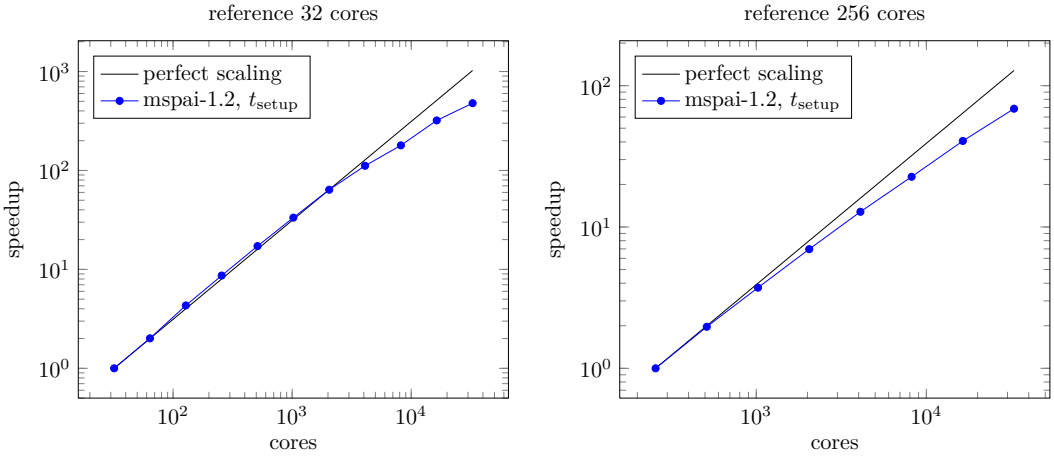
(a) Matrix E40R0100. $n = 17{,}281$, nnz = 553,562.

(b) Matrix KIM1. $n = 38{,}415$, nnz = 933,195.

Figure 7.2.: Strong scaling of setup time for SPAI using `mspai-1.2` on the InfiniCluster Environment D.2 in VN mode. Settings / Results for subfigures:

(a) $\bar{\Upsilon}_{12,5}$, $\varepsilon_{\text{SPAI}} = 1\text{E}{-}03$, $\mathcal{J}(M) = \mathcal{J}(I)$ / $t^{1}_{\text{setup}} = 157.48$ sec., $t^{64}_{\text{setup}} = 5.29$ sec.

(b) $\bar{\Upsilon}_{8,5}$, $\varepsilon_{\text{SPAI}} = 1\text{E}{-}03$, $\mathcal{J}(M) = \mathcal{J}(I)$ / $t^{1}_{\text{setup}} = 259.01$ sec., $t^{64}_{\text{setup}} = 7.61$ sec.



(a) Matrix ATMOSMODJ. $n = 1{,}270{,}432$, nnz = 8,814,880. The superlinear speedup is likely caused by cache effects.

(b) Matrix MEMCHIP. $n = 2{,}707{,}524$, nnz = 13,343,948.

Figure 7.3.: Strong scaling of setup time for SPAI using `mspai-1.2` on Shaheen's Blue Gene/P Environment D.1 in VN mode. Settings / Results for subfigures:

(a) $\bar{\Upsilon}_{5,5}$, $\varepsilon_{\text{SPAI}} = 0.01$, $\mathcal{J}(M) = \mathcal{J}(I)$ / $t^{32}_{\text{setup}} = 1383.10$ sec., $t^{32768}_{\text{setup}} = 2.89$ sec.

(b) $\bar{\Upsilon}_{10,5}$, $\varepsilon_{\text{SPAI}} = 0.05$, $\mathcal{J}(M) = \mathcal{J}(I)$ / $t^{256}_{\text{setup}} = 1472.49$ sec., $t^{32768}_{\text{setup}} = 21.43$ sec.

idea to make the algorithm benefit from a sparse QR factorization implementation for $\hat{C}_k$. Yet, for sufficiently dense matrices, sparse QR factorization approaches come along with a lower Mflop rate and a higher overhead if one compares them to QR factorizations working on full matrices. As we do not know the structure of all the occurring matrices a priori, we cannot restrict ourselves to one hardwired type of QR factorization. Instead, we examine $\hat{C}_k$'s nnz compared to p_k (3.4) and q_k (3.3) to choose an appropriate implementation for each computation. If $\frac{\text{nnz}(\hat{C}_k)}{p_k q_k} < \varepsilon_{QR}$, we invoke a sparse QR factorization based upon the used CSC format. We use CSparse *[41], since this package implements Householder transformations (see Section 2.1.3) using sparse data structures. In case of denser submatrices, we convert the internal representation of $\hat{C}_k$ into a full matrix and pass it to standard* Lapack/Atlas *[4] routines. Runtime tests with a large set of different matrices revealed $\varepsilon_{QR} = 15\%$ being a meaningful default value in our implementation; see also [99]. Note that this value can be modified by the user and depends on the actual machine and the library implementation used.*

2. *The* `spai-3.2`*,* `mspai-1.2`*, and* `fspai-1.1` *do not yet implement a parallel (MPI) file I/O concept. Hence, no speedup can be observed in parallel scenarios for passing the matrix file or writing the constructed preconditioner to the file system.*

3. *Fourestey et al. [49] analyzed* `mspai-1.2`*'s thread safety. A single race condition occurs by* `bit_vec` *and* `reset_vec`*. Making those arrays local inside the shadow computation* `Get_I_Set()` *leads to a fully thread safe code. Following preliminary results, the unoptimized memory consumption of the threaded implementation affects its scalability, also depending on the underlying architecture. While gaining good scalability among 24 threads (two AMD Magny-Cores) for Stokes and Oseen problems, breakdowns occur after 8 threads for the tested Laplacians and advection-diffusion-reaction problems.*

4. *There is also a project dealing with a SAI implementation for NVIDIA®'s CUDA Sparse Linear Algebra (CUSP) [14] library investigating its effect—especially the one of (dense) QR factorizations—on graphics processing units (GPUs) [49, 126]. In order to increase the Mflop rate, the authors elaborate on a sophisticated memory allocation strategy. Note that additionally gained speedup can be invested in denser start patterns for SAI in order to obtain preconditioners of higher quality. Research in this context will give insight in how SAI may efficiently exploit the GPU architecture which might become part of the multilevel parallelism on the path to exascale computing; see our Closing Remark 8.5.*

7.3. FSPAI Implementation `fspai-1.1`

7.3.1. General Notes

The `fspai-1.1` code is a generic `C++` implementation of the FSPAI algorithm according to Section 3.2.3 and Algorithm 8. Refer to the `fspai-1.1.pdf` manual on the web page for a detailed description of all options of the shell-based interface. The implementation unifies a stand-alone sequential implementation free from MPI directives and a parallel implementation designed for distributed memory architectures using MPI for the communication between the PEs. According to the single source principle, `fspai-1.1` uses compile-time

polymorphism for the environment classification and runtime polymorphism [131] for the various algorithms and the field type classification of the linear problems. We comment on `fspai-1.1`'s code design in the subsection below.

Features of `fspai-1.1`

- The generic implementation allows to compute sparse approximate inverse preconditioners both for SPD and HPD systems. See Remark 3.6 #2.
- It is possible to prescribe an arbitrary start pattern $\mathcal{J}(L)$. The options `-diag` and `-L` automatically construct $\mathcal{J}(L) = \mathcal{J}(I)$ and $\mathcal{J}(L) = \mathcal{J}(\text{low}(A))$, respectively. Further arbitrary a priori choices can be used by passing a pattern file in MATRIX MARKET format [27].
- Based on the dictionary approach in `mspai-1.2`, the implementation provides a similar caching and hashing approach to avoid redundant Cholesky factorizations. See Section 7.3.2 on this.
- It is possible to decide whether the Cholesky factorizations should be done in full mode using LAPACK/ATLAS or in sparse mode using CXSPARSE [42]. Note that this option is also possible for HPD systems. See Section 7.3.3 on this.
- After setting up an FSPAI L, it is possible to invoke CG and/or PCG with L to solve the system with right-hand side $b = (1,\ldots,1)^T$ or b being a random vector. Passing arbitrary right-hand sides to the solver is planned for a next release. For SPD systems, our parallel version provides the possibility to invoke the highly scalable parallel PCG solver from the HYPRE [95] library. Note that our sequential version implements a generic PCG solver also applicable to HPD systems.
- For the parallel version, the user may invoke `fspai-1.1` with options to solve the system with PCG using a PARASAILS [36] or an EUCLID preconditioner which is a parallel ILU preconditioner. See the HYPRE manual for more details and references on EUCLID. Both preconditioners will be constructed via the HYPRE library. This enables us to make quality comparisons between FSPAI and the mentioned preconditioners; see Section 7.3.4.
- The package comes along with a multiplicity of parameters which allow to indirectly control the density and quality of the approximate inverse L, similar to `mspai-1.2`. See also Section 3.2.3 or our manual `fspai-1.1.pdf` [89]. Note that parameters for (P)CG are available as well and both L and the solution vector x can be written to files.
- The `fspai-1.1` provides runtime optimized code, i.e., it implements both the improvements of our `mspai-1.2` code introduced in Section 7.2.2 and those mentioned below.
- In contrast to `mspai-1.2`, `fspai-1.1` was designed following the single source principle, also to simplify its maintainability. The package provides full functionality enabling to use it both in a sequential environment without an MPI implementation (library) and in a parallel environment using MPI for the inter-process communication.

Note that Cholesky updates (see Section 3.2.3) and FSPAI's generalization to the block case (Chapter 4) are planned to be implemented in a next release. In `fspai-1.1`, all pointwise subproblems according to (3.41)–(3.43) are Cholesky factorized in full dimension.

Code Design

For clarity reasons, we migrated the environment specific code parts into the different subfolders named `sequential` and `parallel`. This is the only place where compile-time polymorphism is used. The following classes are affected:

- `ENV_Handler` implements the environment specific routines such as initializing the environment or delivering the (MPI) communicator.
- `Comm_Handler` implements the generic parallelization concept described in Section 7.1.
- `PCG` implements the PCG algorithm. We implemented a sequential version according to Algorithm 1. For the parallel version we provide a highly scalable PCG solver using the implementation from the HYPRE library. For that purpose, a conversion to HYPRE's internal data structures and a modified preconditioner handle became necessary. Note that further solver classes such as CR (Figure 2.5) can be easily added to the package.

The classes `Double_Handler` and `Complex_Handler` represent interfaces handling real and complex matrix input, respectively. They are derived from the abstract base class `Type_Base_Handler`. From outside, i.e., in the `main` routine, the abstract `type_handle` initializes and invokes all template specializations. As unknown a priori, the field specific type handler is instantiated at runtime.

The various FSPAI approaches are modelled by the template method pattern [50]. The generic base class `Fspai_Base` implements the FSPAI skeleton which merely consists of a single loop over the preconditioner columns the PE is accountable for. The following derived classes provide implementations of the virtual method `Fspai_Column()` which is responsible for computing one column of L_k in the approach-specific way:

- `Fspai_Unrestrained` implements the standard FSPAI according to Algorithm 8. The subproblems are solved by using the LAPACK/ATLAS [1, 4] library.
- `Fspai_Sparse_Decomp` implements the standard FSPAI in which the subproblems are solved by the CXSPARSE library [42].
- `Fspai_Caching` implements the dictionary approach in Algorithm 15 using a generic fixed-size cache.
- `Fspai_Hashing` implements the dictionary approach in Algorithm 15 using a generic hash table.

As the computation on the data level is similar for all approaches, the derived classes manage the algorithmic realization only on a higher level. The kernels for the computation are concentrated in the class `Fspai_Sub` and are accessible by the corresponding inherited member from `Fspai_Base`. See Figure 7.4 for the simplified cut-out of `fspai-1.1`'s UML[18] based class diagram which describes the relations between the classes modelling the template method pattern. The instantiation of the specific FSPAI subalgorithm is performed at runtime by the class `Switch_Algorithm`. The chosen code design enables to add new FSPAI algorithms by simply providing derived classes implementing the method `FSPAI_Column()`.

18 unified modeling language (UML); see http://www.uml.org/.

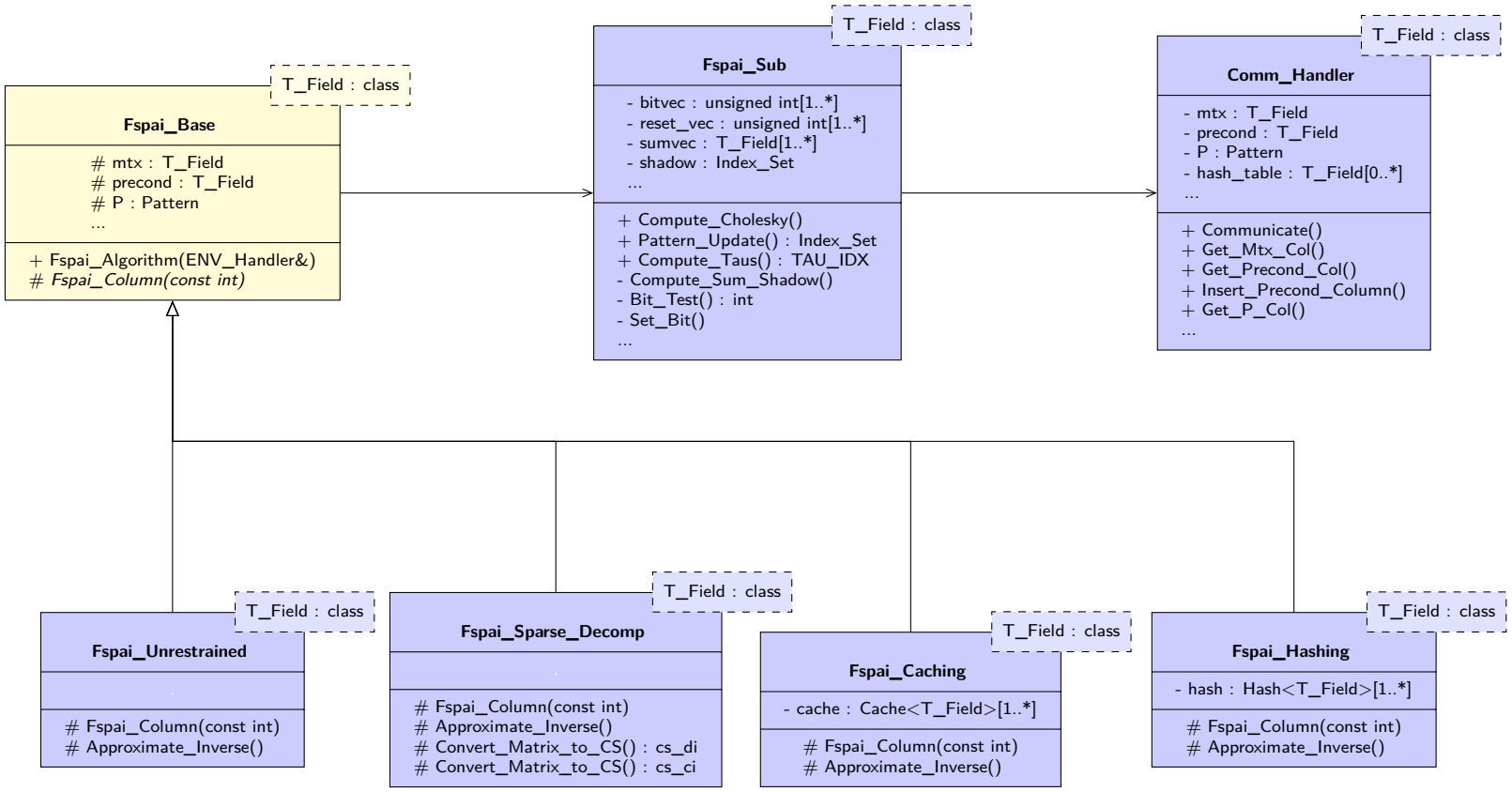


Figure 7.4.: Simplified UML based cut-out of `fspai-1.1`'s class diagram illustrating the template method pattern [50] which is used for the different algorithmic realizations.

τ_{jk} Computation

The optimized shadow computation described in Section 7.2.2 and Listing 7.1 is also implemented in `fspai-1.1` to obtain $\hat{\mathcal{J}}_k$. However, for the computation of the various $\tau_{jk} = a_{jj}^{-1}[A_j(\mathcal{J}_k)^T L_k(\mathcal{J}_k)]^2$ for $j \in \hat{\mathcal{J}}_k$, we use a different technique than the one introduced in Section 7.2.2 for `mspai-1.2`. Following our profiling experiments, an optimized τ_{jk} computation—catered to FSPAI—becomes possible by exploiting the symmetry of A and the triangular sparsity of L.

We integrate the computation of all nominator products $\zeta_{jk} := A_j(\mathcal{J}_k)^T L_k(\mathcal{J}_k)$ into the routine `Compute_Sum_Shadow()`. It iterates over the necessary columns A_j with $j \in \mathcal{J}_k$ in order to find the shadow and thus also the index set $\hat{\mathcal{J}}_k$. The array `sumvec`, which is of dimension n—the number of rows of A—stores at j^{th} position the value ζ_{jk}. As A is symmetric, a column-wise computation of ζ_{jk} is possible; this fits into the overall CSC processing scheme of `fspai-1.1` and thus also into its shadow computation. Hence, for an arbitrary column k, we obtain all ζ_{jk}, $j \in \hat{\mathcal{J}}_k \cup \mathcal{J}_k$ by a GAXPY[19] based computation in the submatrix $A_{k:n,k:n}$, i.e.,

$$\texttt{sumvec[}k,\ldots,n\texttt{]} := \begin{pmatrix} \zeta_{kk} \\ \vdots \\ \zeta_{nk} \end{pmatrix} = \sum_{j \in \mathcal{J}_k} (L_k)_j \begin{pmatrix} a_{kj} \\ \vdots \\ a_{nj} \end{pmatrix}. \tag{7.4}$$

See also Figure 7.5 for an exemplary illustration of (7.4). We point out the following facts:

1. By definition, L has a triangular structure. For the computation of the shadow and the ζ_{jk}, we only have to consider the indices $j \geq k$. Hence, it is possible to skip the computation of $\zeta_{1,k}, \ldots, \zeta_{k-1,k}$.

2. Following the definition of $\hat{\mathcal{J}}_k$ (3.49), our computation according to (7.4) also computes the unnecessary ζ_{jk} for $j \in \mathcal{J}_k$. See Figure 7.5, where only $\zeta_{6,2}$ and $\zeta_{8,2}$ are necessary for $\hat{\mathcal{J}}_2$. According to our experiments and profiling results, the computation of the additional ζ_{jk} revealed to be faster than using additional if-statements avoiding them. Likely, this is because if-else-branches on modern processing units are known to be slower than simple (arithmetic) calculations. However, there is plenty of other possible reasons such as scheduling of instructions by the compiler or compiler dependent optimization barriers. Concerning this matter, a detailed analysis at the routine's assembly level would reveal further insight, but is beyond the scope of this work. The `fspai-1.1` implements the approach in (7.4) by including the kernel computation

   ```
   sumvec[r_idx] = sumvec[r_idx] + (col_buf[idx]*Lk[j]);
   ```

   after line 16 in Listing 7.1.

Finally, the τ_{jk} for the indices $j \in \hat{\mathcal{J}}_k$ are obtained by $\frac{\texttt{sumvec[}j\texttt{]}^2}{a_{jj}}$. Note that we apply a similar initialization and resetting strategy for `sumvec` like the one introduced for `bitvec` in Section 7.2.2: `sumvec` is initialized only once at the beginning and immediately after computing a τ_{jk} value, ζ_{jk} is set to zero, i.e., there is no need for a resetting array in this case. This mechanism is necessary in order to guarantee a correct initial state for the next pattern update.

19 Generalized SAXPY (GAXPY) represents a series of SAXPYs regarding the same vector.

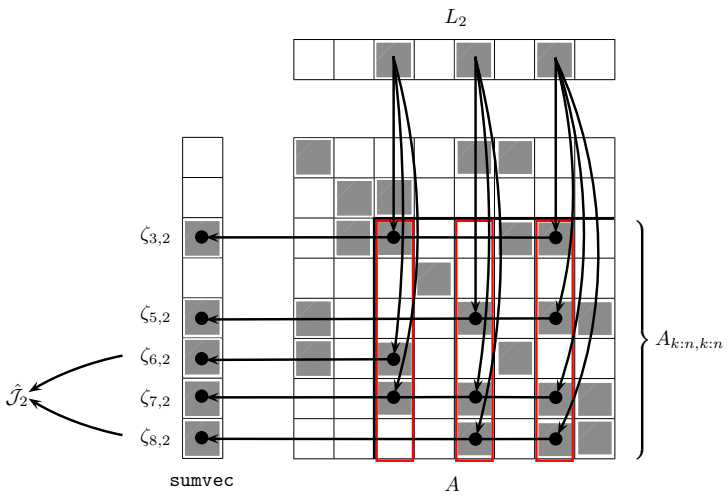


Figure 7.5.: Exemplary illustration of how the ζ_{jk} are computed and stored in `sumvec` according to (7.4) during the computation of $\hat{\mathcal{J}}_k$. The black dots symbolize the product and adding operations, respectively.

7.3.2. Avoiding Redundant Cholesky Factorizations

We transfer the dictionary approach introduced in Section 7.2.1 to `fspai-1.1`. Redundant computations during the setup stage are likely to exist because the underlying matrices are symmetric. Following Algorithm 15, these can be avoided in a similar way compared to Algorithm 14. Besides the different computation for the preconditioner columns, we only have to save one Cholesky factor—here denoted as $L_{\hat{A}_k}$—yielding dictionary entries being 4-tuples $(\hat{A}_k, L_{\hat{A}_k}, k_{A_k(\tilde{\mathcal{J}}_k)}, L_k)$.

Numerical Experiment 7.4 We are interested in the speedup incorporating the dictionary approach in FSAI and FSPAI. We choose matrices from [44] and Laplacian matrices generated on our own. For the hash table implementation we measure the number of occurring best, average, and worst cases, according to Algorithm 15. Note that if $|\tilde{\mathcal{J}}_k| = 0$, the solution of the current preconditioner column is immediately obtained by $l_{kk} = \frac{1}{\sqrt{a_{kk}}}$, not invoking any dictionary mechanism, i.e., the sum of all three numbers must not be equal to the dimension n of the system in the static case. Table 7.3 lists the results for constructing an FSAI with $\mathcal{J}(L) = \mathcal{J}(\text{low}(A))$. Here, the LRU cache has a fixed size of 60. Table 7.4 shows the results for the dynamic case where the cache size is adjusted to a heuristic value. All tests were performed on Environment D.5.

Similar to the dictionary approach in `mspai-1.2`, we obtain improvement for matrices where a huge amount of subproblems (best case) or at least reduced systems $A(\tilde{\mathcal{J}}_k,\tilde{\mathcal{J}}_k)$ (average case) are identical. However, compared to `mspai-1.2`'s dictionary approach, we obtain weaker improvements which rests in the following facts: First, compared to the QR factorization, the Cholesky factorization comes along with a lower flop rate; see Section 2.1. Second, the FS(P)AI preconditioner has only triangular structure, i.e., we can expect that the occurring subproblems are smaller compared to MS(P)AI—approximately by a factor 2. Consequently, less expensive computations can be avoided, deteriorating the cost-benefit ratio between the dictionary's overhead and the saving of computational costs.

Algorithm 15: FSPAI dictionary approach implemented in `fspai-1.1`.

Input : $\hat{A}_k := A(\tilde{\mathcal{J}}_k,\tilde{\mathcal{J}}_k) \in \mathbb{C}^{|\tilde{\mathcal{J}}_k|\times|\tilde{\mathcal{J}}_k|}$.
Output: The solution $L_k \in \mathbb{C}^n$.

$k_{\hat{A}_k} \leftarrow \text{key}[\hat{A}_k]$
$k_{A_k(\tilde{\mathcal{J}}_k)} \leftarrow \text{key}[A_k(\tilde{\mathcal{J}}_k)]$
no_collision $\leftarrow$ **false**
if $k_{\hat{A}_k} \in$ *dictionary* **then**
  $\left(\hat{A}_k^{\text{dict}}, L_{\hat{A}_k}^{\text{dict}}, k_{A_k(\tilde{\mathcal{J}}_k)}^{\text{dict}}, L_k^{\text{dict}}\right) \leftarrow \text{dictionary}\left[k_{\hat{A}_k}\right]$
  if $\hat{A}_k = \hat{A}_k^{dict}$ **then**
    no_collision $\leftarrow$ **true**
  else
    no_collision $\leftarrow$ **false**
  end
end
if *no_collision* **then**
  if $k_{A_k(\tilde{\mathcal{J}}_k)} = k_{A_k(\tilde{\mathcal{J}}_k)}^{dict}$ **then**
    $L_k \leftarrow L_k^{\text{dict}}$ {best case}
  else
    $y_k \leftarrow \hat{A}_k^{-1} A_k(\tilde{\mathcal{J}}_k)$ using $L_{\hat{A}_k}^{\text{dict}}$ {average case}
    $l_{kk} \leftarrow \left(a_{kk} - A_k(\tilde{\mathcal{J}}_k)^T y_k\right)^{-\frac{1}{2}}$
    $L_k(\tilde{\mathcal{J}}_k) \leftarrow -l_{kk} y_k$
  end
else
  $L_{\hat{A}_k} \leftarrow \text{chol}(\hat{A}_k)$ {worst case}
  $y_k \leftarrow \hat{A}_k^{-1} A_k(\tilde{\mathcal{J}}_k)$ using $L_{\hat{A}_k}$
  $l_{kk} \leftarrow \left(a_{kk} - A_k(\tilde{\mathcal{J}}_k)^T y_k\right)^{-\frac{1}{2}}$
  $L_k(\tilde{\mathcal{J}}_k) \leftarrow -l_{kk} y_k$
  $\text{dictionary}\left[k_{\hat{A}_k}\right] \leftarrow \left(\hat{A}_k, L_{\hat{A}_k}, k_{A_k(\tilde{\mathcal{J}}_k)}, L_k\right)$
end

We recommend to use the dictionary approach in FS(P)AI only if the underlying matrices are heavily structured, i.e., if a high proportion of best case situations is likely to occur, e.g., for Laplacian matrices. For matrices with a large amount of worst case situations, performance breakdowns must be expected. See the matrices GYRO, OLAFU, and CT20STIF. ●

7.3.3. Sparse Cholesky Factorizations

Motivated by the gained speedup using sparse QR factorizations in `mspai-1.2` (see Remark #1 in Section 7.2.4), we also provide an FSPAI implementation using sparse Cholesky factorizations. Because LAPACK based computations require dense storage schemes, we provide a mapping from `fspai-1.1`'s CSC format to full arrays. Hence, the submatrices $A(\mathcal{J}_k,\mathcal{J}_k)$, e.g., are stored in an array of size $|\mathcal{J}_k|^2$ instead of size $\text{nnz}\,(A(\mathcal{J}_k,\mathcal{J}_k))$.

Table 7.3.: Performance measurements of avoiding redundant computations according to Algorithm 15 in `fspai-1.1`. All results give speedups with respect to standard FSAI ($\bar{\Upsilon}_{0,0}$). The dictionary is realized either as hash table or as LRU cache of fixed size 60. $\mathcal{J}(L) = \mathcal{J}(\text{low}(A))$.

Matrix		Hash table				LRU cache
Name	n	Best case	Avg. case	Worst case	Speedup	Speedup
BCSSTK16	4,884	1735	21	3053	0.94	1.07
LAPLACE2D_1O100	10,000	9997	0	2	1.00	1.00
LFAT5000	19,994	9992	9995	4	1.67	1.67
JNLBRNG1	40,000	39004	198	797	1.50	1.50
QA8FM	66,127	64446	771	886	1.68	1.58
APACHE1	80,800	80303	394	102	1.67	1.67
LAPLACE2D_1O317	100,489	100486	0	2	1.56	1.47
LAPLACE3D_1O47	103,823	103819	0	3	1.43	1.36
OLAFU	16,146	0	2	16143	0.69	0.83
GYRO	17,361	0	5	17171	0.74	0.81
CT20STIF	52,329	300	154	51360	0.69	0.82

Table 7.4.: Performance measurements of avoiding redundant computations according to Algorithm 15 in `fspai-1.1`. All results give speedups with respect to standard FSPAI. `fspai-1.1`'s dictionary is realized either as hash table or as an LRU cache. Start pattern is $\mathcal{J}(L) = \mathcal{J}(I)$.

Matrix		Setting		Hash table				LRU cache	
Name	n			Best case	Avg. case	Worst case	Speedup	Size	Speedup
BCSSTK16	4,884	$\bar{\Upsilon}_{8,6}$	$\varepsilon_{\text{FSPAI}} = 1\text{E}{-}03$	5777	43	10769	0.95	20	0.97
LAPLACE2D_1O100	10,000	$\bar{\Upsilon}_{8,5}$	$\varepsilon_{\text{FSPAI}} = 1\text{E}{-}03$	78629	196	70	1.53	100	1.44
LFAT5000	19,994	$\bar{\Upsilon}_{15,5}$	$\varepsilon_{\text{FSPAI}} = 1\text{E}{-}03$	214575	84900	75	1.56	200	1.46
JNLBRNG1	40,000	$\bar{\Upsilon}_{10,5}$	$\varepsilon_{\text{FSPAI}} = 1\text{E}{-}04$	374257	11204	8960	1.50	80	1.46
QA8FM	66,127	$\bar{\Upsilon}_{8,5}$	$\varepsilon_{\text{FSPAI}} = 1\text{E}{-}04$	298931	5377	4332	1.22	500	1.17
APACHE1	80,800	$\bar{\Upsilon}_{10,6}$	$\varepsilon_{\text{FSPAI}} = 1\text{E}{-}02$	101813	30198	77	1.29	300	1.26
LAPLACE2D_1O317	100,489	$\bar{\Upsilon}_{8,5}$	$\varepsilon_{\text{FSPAI}} = 1\text{E}{-}03$	799720	630	70	1.56	100	1.49
LAPLACE3D_1O47	103,823	$\bar{\Upsilon}_{8,5}$	$\varepsilon_{\text{FSPAI}} = 1\text{E}{-}03$	515892	46	98	1.37	200	1.34
OLAFU	16,146	$\bar{\Upsilon}_{8,5}$	$\varepsilon_{\text{FSPAI}} = 1\text{E}{-}03$	0	39	78268	0.86	100	0.93
GYRO	17,361	$\bar{\Upsilon}_{8,5}$	$\varepsilon_{\text{FSPAI}} = 1\text{E}{-}03$	0	283	76504	0.84	100	0.90
CT20STIF	52,329	$\bar{\Upsilon}_{8,5}$	$\varepsilon_{\text{FSPAI}} = 1\text{E}{-}02$	2139	1152	98599	0.89	100	0.91

Dealing with sparse matrices, it is possible that the resulting subproblems (3.41) are sparse as well. Therefore, the choice of providing computational kernels which are geared towards taking the sparsity into account is obvious. For this purpose, `fspai-1.1` provides the derived class `Fspai_Sparse_Decomp` (see Figure 7.4) which uses CXSparse; a sparse matrix library by Davis [43] operating on the nonzeros.

There are two basic differences between the class `Fspai_Unrestrained`—which uses Lapack—and `Fspai_Sparse_Decomp`:

1. Both algorithms invoke only one library specific routine to solve (3.41) via Cholesky factorization. For Lapack this is `dposv()`/`zposv()` and for CXSparse `cs_di_cholsol()`/`cs_ci_cholsol()`, depending on whether the (reduced) matrix is real or complex. We refer to the total time spent in the specific routine by $t_{\text{Chol-solve}}$. Note that we use a minimum degree preordering of $A(\mathcal{J}_k,\mathcal{J}_k)$ (see Section 2.2.1) in case that its fill level is below 10%, otherwise $P = I$, which yields slight speedups. Here, we adjusted the empirically constituted fill level tolerance of 7% for QR factorizations proposed in [99]. The time for the preordering is not included in $t_{\text{Chol-solve}}$, but is negligible.

2. In order to use CXSparse, we have to convert the reduced matrix $A(\mathcal{J}_k,\mathcal{J}_k)$ into its `cs_ci`/`cs_di` `triplet` matrix format. We refer to this additional time step as t_{convert}.

Numerical Experiment 7.5 For arbitrary chosen matrices from [44], we analyze the runtime performance of both libraries when constructing an FSPAI. In all settings CG does not converge in 5000 iterations for $\varepsilon_{\text{CG}} = 1\text{E}{-}06$. We set $\varepsilon_{\text{FSPAI}} = 0$ and measure `fspai-1.1`'s setup time t_{setup} as well as the timings introduced above on Environment D.4. Table 7.5 lists all results. Note that mhd1280b is an HPD matrix while qc2534 is only complex symmetric, i.e., (P)CG is likely not to converge in this case.

In contrast to `mspai-1.2`, where we usually obtain speedup for real matrices using CSparse [86, 99], we observe performance breakdowns in `fspai-1.1`, but only for the SPD case. To find a possible explanation we additionally tried the following scenarios while still measuring $t_{\text{Chol-solve}}$ as mentioned above:

- We additionally performed both the experiments in Table 7.5 as well as further experiments on the architectures available at the Environments D.2 and D.5. The routine `cs_di_cholsol()` still performed worse than `dposv()`.
- In order to exclude differences between CSparse and CXSparse, we modified the code to be linkable against CSparse. The breakdowns for SPD systems still occurred with similar magnitude.
- We did detailed profiling of the code but were unable to locate bottlenecks being responsible for the breakdowns.
- A modification of the fill level—including switching it off—for the minimum degree ordering did not yield improvement.
- Experiments with various matrices of different sparsity also did not yield any changes.

Following our observations, we speculate that the Lapack routine for the solution of SPD systems using a Cholesky factorization is highly optimized. However, in order to obtain more insight, a detailed benchmarking of the library implementations is necessary. One can also implement further derived classes according to Figure 7.4 using other sparse matrix

Table 7.5.: Comparison between LAPACK and CXSPARSE for solving the occuring SPD/HPD subsystems (3.41) during the construction of an FSPAI. For all matrices chosen from [44], CG does not converge in 5000 iterations for $\varepsilon_{\text{CG}} = 1\text{E}{-}06$. Note that $t_{\text{Chol-solve}}$ denotes the total time to solve all subsystems using a Cholesky factorization. For `Fspai_Unrestrained` the LAPACK routine `dposv()`/`zposv()` is used. For `Fspai_Sparse_Decomp` CXSPARSE's routine `cs_di_cholsol()`/`cs_ci_cholsol()` is used. The tolerance $\varepsilon_{\text{FSPAI}} = 0$.

Matrix	Density(A) %	Setting	Library	t_{PCG} (iter)	t_{setup}	t_{convert}	$t_{\text{Chol-solve}}$
NASA2910, $n = 2910$, nnz(A) = 174,296	2.06 %	$\mathcal{J}(L) = \mathcal{J}(I)$, $\bar{\Upsilon}_{10,5}$	LAPACK	0.14 (103)	2.11	–	0.73
			CXSPARSE	0.14 (103)	2.87	0.35	1.14
OLAFU, $n = 16{,}146$, nnz(A) = 1,015,156	0.39 %	$\mathcal{J}(L) = \mathcal{J}(I)$, $\bar{\Upsilon}_{5,5}$	LAPACK	7.33 (1109)	2.83	–	0.74
			CXSPARSE	7.30 (1111)	3.96	0.37	1.52
SHIPSEC8, $n = 114{,}919$, nnz(A) = 3,303,553	0.03 %	$\mathcal{J}(L) = \mathcal{J}(I)$, $\bar{\Upsilon}_{3,5}$	LAPACK	19.89 (506)	6.36	–	1.39
			CXSPARSE	19.78 (506)	9.34	0.73	3.42
T2DAH_E, $n = 11{,}445$, nnz(A) = 176,117	0.13 %	$\mathcal{J}(L) = \mathcal{J}(I)$, $\bar{\Upsilon}_{5,5}$	LAPACK	0.03 (6)	0.82	–	0.35
			CXSPARSE	0.03 (6)	1.36	0.18	0.70
SMT, $n = 25{,}710$, nnz(A) = 3,749,582	0.57 %	$\mathcal{J}(L) = \mathcal{J}(I)$, $\bar{\Upsilon}_{5,5}$	LAPACK	5.82 (397)	7.26	–	1.22
			CXSPARSE	5.86 (397)	9.18	0.58	2.54
BCSSTK36, $n = 23{,}052$, nnz(A) = 1,143,140	0.22 %	$\mathcal{J}(L) = \mathcal{J}(I)$, $\bar{\Upsilon}_{5,5}$	LAPACK	6.39 (714)	3.42	–	1.04
			CXSPARSE	6.38 (714)	5.01	0.48	2.11
MHD1280B, $n = 1280$, nnz(A) = 22,778	1.39 %	$\mathcal{J}(L) = \mathcal{J}(I)$, $\bar{\Upsilon}_{15,5}$	LAPACK	0.004 (3)	2.59	–	2.20
			CXSPARSE	0.004 (3)	1.58	0.23	0.83
QC2534, $n = 2534$, nnz(A) = 463,360	7.22 %	$\mathcal{J}(L) = \mathcal{J}(\text{low}(A))$, $\bar{\Upsilon}_{5,5}$	LAPACK	†	13.22	–	8.74
			CXSPARSE	†	10.24	1.81	3.13

packages which provide routines for the solution of sparse linear systems; see [43] for a list and references. This could also give further evidence for the efficiency of the two storage formats in FSPAI. Note that even for an optimized conversion to the `triplet` format—e.g., by embedding it within the routine responsible for the reduction of A to $A(\mathcal{J}_k,\mathcal{J}_k)$—the setup times would still be larger due to higher costs in the routine providing solutions of the subproblems according to $t_{\text{Chol-solve}}$. ●

7.3.4. Quality and Performance Analysis

We are interested in the quality and the performance of the `fspai-1.1` code. As there is no reference high-level distributed memory implementation of the FSPAI algorithm, we resort to a comparison with MATLAB and HYPRE's PARASAILS preconditioner.

Numerical Experiment 7.6 : quality comparison with MATLAB. We compare the preconditioners resulting from our MATLAB and `fspai-1.1` implementation by investigating the norm and spectral properties of the preconditioned system L^TAL as well as (P)CG's convergence for the right-hand side $b = (1,\ldots,1)^T$. By ξ we additionally provide the variance of all eigenvalues of L^TAL around the expected value 1 according to Corollary 3.1. We use `fspai-1.1`'s `wp` option to write L into a file in MATRIX MARKET format. By importing the preconditioner in MATLAB we are able to analyze its properties in comparison with a pure MATLAB FSPAI solution. We choose arbitrary FSPAI settings and matrices from [44]. Note that for the scenario with matrix EX10HS, the value of $\det(L^TAL)$ becomes smaller than the machine precision and is rounded to zero, i.e., we do not provide the K-condition number in this case.

Following the results in Table 7.6, both preconditioned systems are of the same quality. For some matrices the properties slightly differ after the fourth decimal place. The difference in PCG's convergence for some of the chosen scenarios likely results from round-off errors and a still moderate spectral condition number $\kappa_2(L^TAL)$. ●

Numerical Experiment 7.7 : performance comparison with PARASAILS. For a comparison between `fspai-1.1` and the PARASAILS preconditioner we observe the setup and solver timings while focusing on two scenarios:

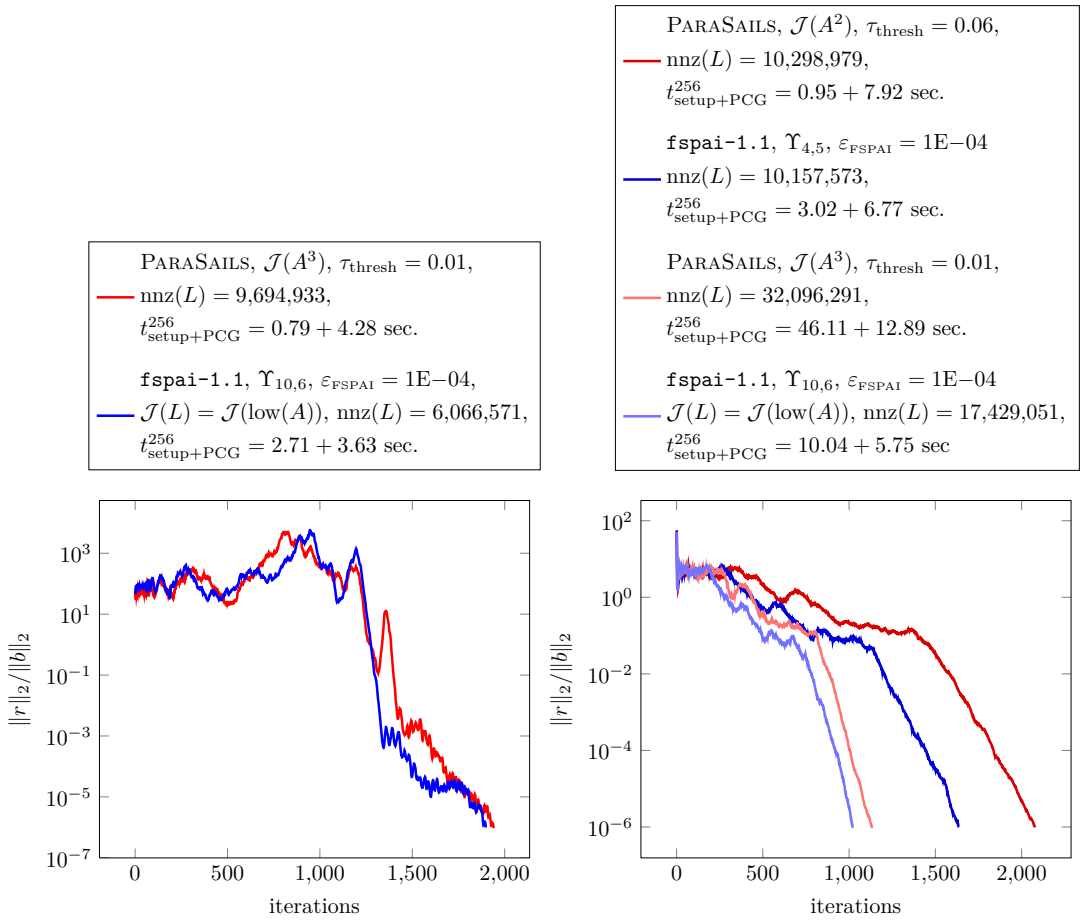
1. We adjust both preconditioner construction settings such that PCG converges approximately in the same number of iterations. Afterward, we analyze both the $\text{nnz}(L)$ and the resulting convergence behavior. The less entries L contains, the faster its application in the solver will be. Note that such a comparison must be treated with caution: it is possible that several preconditioners of different density lead to a comparable number of iterations. However, the resulting convergence curves of our scenarios resemble each other which indicates the plausibility of our density comparisons.

2. We adjust the setup settings such that both preconditioners contain a similar nnz and analyze the quality of L in terms of resulting PCG iterations.

We choose the matrices S3DKT3M2, PWTK, and CT20STIF from [44]. For all matrices, CG does not converge in 5000 iterations for $\varepsilon_{\text{CG}} = 1\text{E}{-}06$. See the captions and legends of the Figures 7.6 and 7.7 for the matrix properties, the chosen `fspai-1.1` and PARASAILS settings as well as the resulting timings and densities. Note that we do not use any postfiltering for the preconditioners, i.e., also for PARASAILS we set the drop tolerance $\tau_{\text{filter}} = 0.0$.

Table 7.6.: Quality comparison of `fspai-1.1` and our MATLAB FSPAI implementation. The PCG tolerance $\varepsilon_{\text{CG}} = 1\text{E}{-}09$ and $b = (1,1,\ldots,1)^T$. No convergence in 5000 (P)CG iterations is marked as †. All matrices are taken from [44].

Implementation	Setting	nnz(L)	$\kappa_2(L^TAL)$	$K(L^TAL)$	ξ	$\|L_AL - I\|_F$	PCG iters.
	MSC01440, $n = 1{,}440$, $\kappa_2(A) = 3.306\text{E}{+}06$, $\|L_A - I\|_F = 1.649\text{E}{+}04$, CG †						
MATLAB	$\Upsilon_{5,5}$, $\varepsilon_{\text{FSPAI}} = 0.01$, $\mathcal{J}(L) = \mathcal{J}(I)$	18,467	2.426E+02	1.120	0.107	4.842E+02	139
`fspai-1.1`		18,467	2.422E+02	1.120	0.107	4.842E+02	139
	MHD1280B (HPD), $n = 1{,}280$, $\kappa_2(A) = 4.749\text{E}{+}12$, $\|L_A - I\|_F = 3.374\text{E}{+}01$, CG †						
MATLAB	$\bar{\Upsilon}_{2,5}$, $\varepsilon_{\text{FSPAI}} = 0.01$, $\mathcal{J}(L) = \mathcal{J}(I)$	5,197	3.416	1.011	0.021	1.530E+03	20
`fspai-1.1`		5,197	3.416	1.011	0.021	1.530E+03	20
	BCSSTK11, $n = 1{,}473$, $\kappa_2(A) = 2.212\text{E}{+}08$, $\|L_A - I\|_F = 2.485\text{E}{+}05$, CG †						
MATLAB	$\bar{\Upsilon}_{3,5}$, $\varepsilon_{\text{FSPAI}} = 0.1$, $\mathcal{J}(L) = \mathcal{J}(I)$	2,772	3.066E+04	1.416	0.270	5.502E+02	722
`fspai-1.1`		2,772	3.066E+04	1.416	0.270	5.502E+02	726
	BCSSTK19, $n = 817$, $\kappa_2(A) = 1.340\text{E}{+}11$, $\|L_A - I\|_F = 9.792\text{E}{+}07$, CG †						
MATLAB	$\bar{\Upsilon}_{5,5}$, $\varepsilon_{\text{FSPAI}} = 0.01$, $\mathcal{J}(L) = \mathcal{J}(I)$	12,164	1.319E+05	1.223	0.148	1.310E+04	457
`fspai-1.1`		12,164	1.319E+05	1.223	0.148	1.310E+04	400
	EX10HS, $n = 2{,}548$, $\kappa_2(A) = 5.484\text{E}{+}11$, $\|L_A - I\|_F = 5.658\text{E}{+}04$, CG †						
MATLAB	$\Upsilon_{8,5}$, $\varepsilon_{\text{FSPAI}} = 0.001$, $\mathcal{J}(L) = \mathcal{J}(I)$	53,769	2.634E+07	–	0.415	1.649E+05	4060
`fspai-1.1`		53,769	2.634E+07	–	0.415	1.649E+05	4091
	NOS7, $n = 729$, $\kappa_2(A) = 2.375\text{E}{+}09$, $\|L_A - I\|_F = 1.273\text{E}{+}04$, CG converges in 4455 iterations						
MATLAB	$\bar{\Upsilon}_{5,5}$, $\varepsilon_{\text{FSPAI}} = 0.01$, $\mathcal{J}(L) = \mathcal{J}(\text{low}(A))$	2,973	2.368E+07	1.049	0.036	1.197E+01	50
`fspai-1.1`		2,973	2.368E+07	1.049	0.036	1.197E+01	52
	PLBUCKLE, $n = 1{,}282$, $\kappa_2(A) = 1.283\text{E}{+}06$, $\|L_A - I\|_F = 1.789\text{E}{+}04$, CG converges in 2164 iterations						
MATLAB	$\Upsilon_{0,0}$, $\varepsilon_{\text{FSPAI}} = 0.01$, $\mathcal{J}(L) = \mathcal{J}(\text{low}(A))$	15,963	1.141E+03	1.078	0.075	4.916E+01	115
`fspai-1.1`		15,963	1.141E+03	1.078	0.075	4.916E+01	115

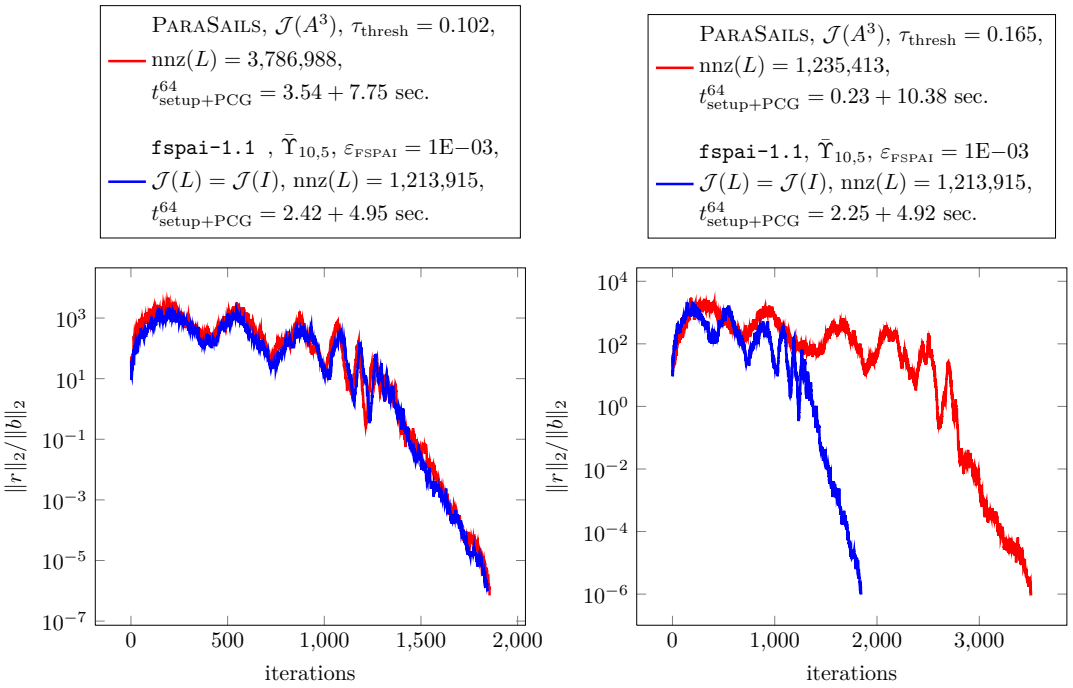


(a) Matrix s3dkt3m2. Static analyzis of cylindrical shell. $n = 90{,}449$. nnz$(A) = 3{,}686{,}223$.

(b) Matrix pwtk. Pressurized wind tunnel. $n = 217{,}918$. nnz$(A) = 11{,}524{,}432$.

Figure 7.6.: PCG convergence for the two matrices given in (a), (b) from [44] with $b = (1, \ldots, 1)^T$ and $\varepsilon_{\text{CG}} = 1\text{E}{-}06$. CG does not converge in 5000 iterations for both scenarios. Used environment is Shaheen's Blue Gene/P D.1 with 256 cores in SMP mode. In subfigure (b) the dark curves (two right-aligned ones) give the convergence behavior for an approximately equal nnz(L). The light curves (two left-aligned ones) illustrate the impact on the nnz(L) for an approximately equal number of iterations.

The results reflect our expectations: FSPAI's well-directed index choosing criterion yields much sparser preconditioners of similar quality compared to ParaSails. See, e.g., Figure 7.6 (b) and Figure 7.7 (a) where ParaSails' pattern heuristic requires respectively twice and triply as much nnz as FSPAI in order to obtain similar convergence quality. Depending on the chosen setting, the setup costs using `fspai-1.1` can but must not be more expensive than using ParaSails. However, the high density of ParaSails's FSAI preconditioner usually results in higher solver costs. See, for instance, the light plots in Figure 7.6 (b) where $t^{256}_{\text{total}} = 59.0$ seconds for ParaSails and $t^{256}_{\text{total}} = 15.8$ seconds for `fspai-1.1`. On the other hand, depending on the scenario, Chow's pattern heuristic may also yield convergence in less time compared to FSPAI; see the dark plots in Figure 7.6 (b).



(a) Impact on nnz(L) for approximately equal number of PCG iterations.

(b) Convergence for approximately equal nnz(L).

Figure 7.7.: PCG convergence for engine block stifness matrix CT20STIF [44] with $n = 52{,}329$ and nnz(A) = 2,600,295. Right-hand side $b = (1,\ldots,1)^T$ and $\varepsilon_{\text{CG}} = 1\text{E}{-}06$. CG does not converge in 5000 iterations. Used environment is the InfiniCluster D.2 with 64 cores in VN mode.

In a summary, FSPAI's pattern heuristic can be more expensive but will lead to sparse preconditioners which are cheap to apply in iterative solvers and consume less memory. When using ParaSails to construct an FSAI, it is possible to obtain convergence at low setup costs. However, there may be problems where the inverse can not be approximated well by $\mathcal{J}(L) = \mathcal{J}(\text{low}(\tilde{A}^k))$ for low powers of k. In order to obtain convergence, an increasing value of k will produce dense patterns which produce high setup and solver costs—possibly much higher than compared to `fspai-1.1`. ●

7.3.5. Strong Scalability

Similar to `mspai-1.2` in Section 7.2.3, we analyze the strong scalability of the `fspai-1.1` code in the following experiment.

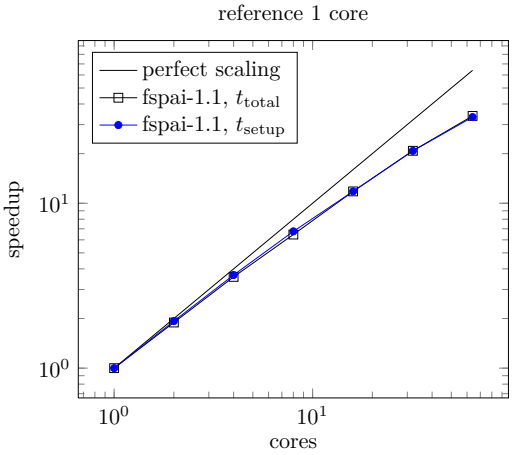
Numerical Experiment 7.8 We illustrate both the setup and total timings $t_{\text{total}} = t_{\text{setup}} + t_{\text{PCG}}$ of `fspai-1.1` on the InfiniCluster D.2 and Shaheen's Blue Gene/P Environment D.1. For our chosen matrices from [44] CG does not converge in 5000 iterations in order to satisfy $\varepsilon_{\text{CG}} = 1\text{E}{-}06$. Refer to the captions of Figure 7.8 and Figure 7.9 for the size and the nnz of the matrices as well as for the chosen `fspai-1.1` settings.

Note the scalability results in Table 7.7 corresponding to Figure 7.8 (b) and Figure 7.9 (b). For the matrix FLAN_1565 convergence can be obtained also for smaller update settings than for $\Upsilon_{40,5}$. However, we use a large setting in order to obtain a meaningful ratio between setup and communication costs.

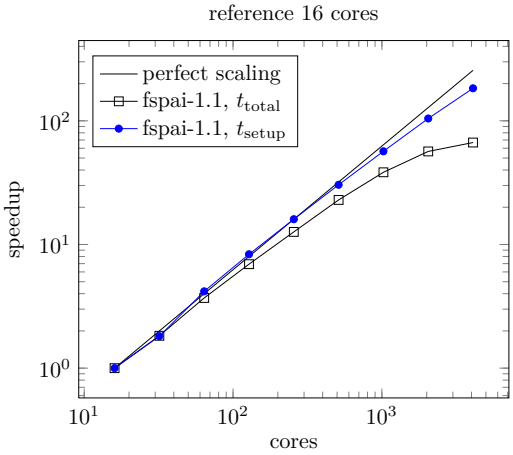
The parallelization concept and FSPAI's inherent parallelism yield high scalability, also for a large number of PEs. In comparison, HYPRE's (P)CG scalability is slightly weaker. Especially for a large number of cores (P)CG's communication costs become larger; cf. t_{PCG} in Table 7.7. Note that we observe similar strong scalability when executing `fspai-1.1` in SMP mode on the clusters. ●

Table 7.7.: Timing results corresponding to Figure 7.8 (b) and Figure 7.9 (b).

STOCF-1465, $\Upsilon_{20,5}$, $\varepsilon_{\text{FSPAI}} = 5\text{E}{-}04$					FLAN_1565, $\Upsilon_{40,5}$, $\varepsilon_{\text{FSPAI}} = 1\text{E}{-}05$				
Cores	t_{setup}	t_{PCG}	t_{total}	Iter.	Cores	t_{setup}	t_{PCG}	t_{total}	Iter.
16	458.8	346.5	805.3	1812	256	1014.3	104.3	1118.6	1471
32	253.1	187.8	440.8	1807	512	515.6	52.5	568.1	1492
64	109.8	108.2	218.0	1799	1024	262.4	29.5	291.8	1470
128	55.1	61.1	116.2	1789	2048	135.3	15.1	150.4	1492
256	28.7	35.1	63.7	1789	4096	71.1	8.4	79.5	1492
512	15.1	20.0	35.1	1795	8192	38.6	6.1	44.7	1471
1024	8.1	12.9	21.0	1795	16384	22.0	6.0	28.0	1471
2048	4.4	9.9	14.2	1803	32768	12.8	7.6	20.4	1471
4096	2.5	9.6	12.1	1807	65536	8.5	6.9	15.4	1471



(a) Matrix M_T1. $n = 97{,}578$, nnz = 9,753,570.

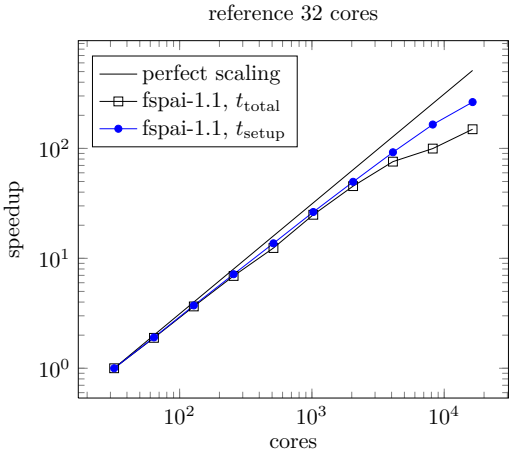


(b) Matrix StocF-1465. $n = 1{,}465{,}137$, nnz = 21,005,389.

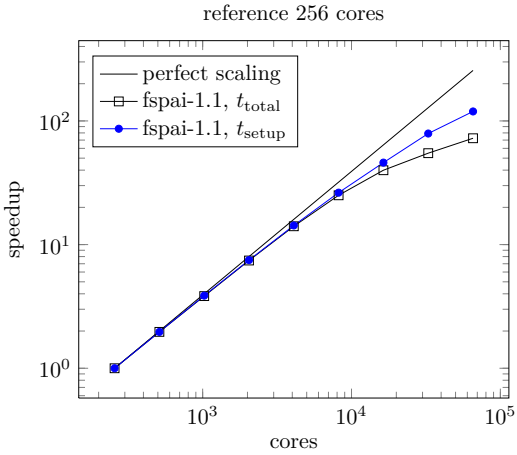
Figure 7.8.: Strong scaling of `fspai-1.1`. Settings / Results for the subfigures:

(a) Environment D.2 (VN mode), $\bar{\Upsilon}_{15,5}$, $\varepsilon_{\text{FSPAI}} = 1\text{E}{-}04$ / PCG conv. in 1486 iter., $t^{1}_{\text{setup + PCG}} = 213.76 + 239.38$ sec., $t^{64}_{\text{setup + PCG}} = 6.42 + 6.96$ sec.

(b) Environment D.1 (VN mode), $\Upsilon_{20,5}$, $\varepsilon_{\text{FSPAI}} = 5\text{E}{-}04$ / PCG conv. in 1807 iter., $t^{16}_{\text{setup + PCG}} = 458.82 + 346.50$ sec., $t^{4096}_{\text{setup + PCG}} = 2.50 + 9.56$ sec..



(a) Matrix Hook_1498. $n = 1{,}498{,}023$, nnz = 59,374,451.



(b) Matrix Flan_1565. $n = 1{,}564{,}794$, nnz = 114,165,372.

Figure 7.9.: Strong scaling of `fspai-1.1` on Environment D.1 with VN mode. Settings / Results for subfigures:

(a) $\Upsilon_{30,5}$, $\varepsilon_{\text{FSPAI}} = 1\text{E}{-}04$ / PCG converges in 1076 iterations, $t^{32}_{\text{setup + PCG}} = 754.63 + 184.90$ sec., $t^{16384}_{\text{setup + PCG}} = 2.86 + 3.43$ sec.

(b) $\Upsilon_{40,5}$, $\varepsilon_{\text{FSPAI}} = 1\text{E}{-}05$ / PCG converges in 1471 iterations, $t^{256}_{\text{setup + PCG}} = 1014.28 + 104.26$ sec., $t^{65536}_{\text{setup+PCG}} = 8.50 + 6.94$ sec.

Chapter 8

Conclusions and Perspectives

We summarize the main results and give a collection of promising approaches motivating for further research. For the sake of clarity, we merge thematically associated conclusions and perspectives in own sections slightly differing from the overall outline of the thesis. A final remark closes the discussion.

8.1. Variants of Sparse Approximate Inverses

1. Pointing out the challenge of prescribing sparsity patterns for SAI, we illustrated that the commonly used heuristic $\mathcal{J}(A)$ may produce preconditioners with zero columns although A is regular. We introduced the heuristic choice $\mathcal{J}(A^T)$ avoiding this case.
2. We refined the classical matrix based probing technique of MSPAI by introducing mask probing which allows to impose column specific probing conditions on the preconditioner, maintaining all its advantages. We utilized this approach to derive efficient smoothing and regularizing preconditioners; see 13. and 14.
3. As a semi-adaptive (M)SPAI variant, we introduced a multistep successive MSPAI (MMSPAI) algorithm where further improvement is only performed on those columns which do not yet satisfy a specified tolerance. Our comparisons with an MSP based preconditioner indicate lower setup costs and—due to much sparser preconditioners—lower costs when applying in solvers such as BiCGSTAB. In this context, we also analyzed the effect of different pattern heuristics imposed on the rear part which requires improvement. Usually, a dense pattern like $\mathcal{J}({A^{(i)}}^T A^{(i)})$ lets all columns satisfy the tolerance after a small number of steps at low setup costs.
4. We mentioned computing complex FSPAIs for HPD systems which involves complex conjugation and complex square roots but no algorithmic changes. Our generic FSPAI implementation (see 9.) also allows to setup factorized preconditioners for this type of systems. Furthermore, we presented theoretical properties of FS(P)AI related to its stopping criterion. Exemplary, we showed that FSPAI's index choosing criterion may produce preconditioners which yield an arbitrary large K-condition number although they satisfy the prescribed stopping criterion. However, this is only of theoretical interest as this case rarely occurs in practice.
5. Based on 4., we derived an exact version of FSPAI (EFSPAI) where a pattern index is chosen according to its exact reduction of the K-condition number. It resembles FSPAI in all but the quality metric to determine profitable indices. Referring to parallelization, we described a hierarchy of Schur component updates which reduce EFSPAI's higher computational costs. Our results indicate that EFSPAIs are of slightly higher

quality than FSPAIs—related to the K-condition number—but require higher setup costs which, in general, cannot be recovered by lower PCG costs.

6. We derived a block version of FSPAI (BFSPAI) whose formulae resemble the pointwise case and thus preserve parallelism, robustness, and adaptive capturing of promising blocks. Depending on the block size, BFSPAIs are of similar or higher quality on the one hand but also involve higher PCG costs on the other hand. Preliminary results indicate lower setup costs motivating for the application of BFSPAI, especially in block SPD/HPD problems. However, further results based on high-level implementations are required to approve these observations. Based on 5., we additionally transferred the exact reduction metric to the block case in Appendix C, i.e., we derived the exact BFSPAI (EBFSPAI).

Perspectives:

- Similar to the pointwise case, block sparse approximate inverses require an efficient initial block sparsity pattern. A graph based approach might be a promising way to determine block structures leading to efficient preconditioners, possibly without processing all elements of the matrix.
- Using Newton or Broyden-type methods to solve nonlinear equations usually leads to a sequence of linear systems of equations $A^{(i)}x = b^{(i)}$, where $i = 0, 1, \ldots$, and the system matrices mildly differ from step to step by the known defect matrices $E^{(i)}$. While setting up preconditioners for each of the occurring linear systems is too costly, using the same preconditioner for a sequence of linear systems—denoted as *freezing*—often yields poor convergence. In this context, Tebbens and Tůma proposed to compute a preconditioner only once at the beginning and to reuse it over several systems by updating it. This showed to be efficient especially in matrix-free environments [134]. The costs for computing SAIs $M^{(i+1)} \approx (A^{(i)} + E^{(i)})^{-1}$ for each system via

$$\min_{\mathcal{J}(M^{(i+1)}) \in \mathcal{J}} \|(A^{(i)} + E^{(i)})M^{(i+1)} - I\|_F$$

  are likely to be higher than to compute a SAI $M^{(0)} \approx {A^{(0)}}^{-1}$ only for the first system and to update it. Here, MS(P)AI probing could help to determine modifications of improved quality possibly implying fewer preconditioner updates.
- Motivated by Remark 3.5, it would be interesting to provide an FSPAI preconditioner for saddle point problems.
- Motivated by modified IC preconditioners, an open point is to transfer MSPAI's probing concept to FS(P)AI, i.e., to propose the modified FSPAI (MFSPAI).

8.2. Implementations of Sparse Approximate Inverses

7. With the enhanced (M)SPAI implementation `mspai-1.2`, we presented a generic dictionary approach implemented as a hash table and improved LRU cache mechanism of fixed size. For structured matrices, leading to many identical subproblems, an access to previously computed results may speed up the setup up to a factor of 3. A cache of moderate size reaches similar speedup compared to the hash table but comes along with the advantage of a bounded memory consumption.

8. Furthermore, we elaborated on optimizations with respect to memory allocations which slowed down the construction of preconditioners for large matrices. Compared to earlier versions as well as to `spai-3.2`, our new code has a significant faster setup of high dimensional preconditioners, preserving its scalability. In this context, we analyzed `mspai-1.2`'s strong scaling for the first time in massively parallel scenarios on the Blue Gene/P environment. The inherent parallelism of (M)SPAIs and the effective parallelization concept introduced by Barnard et al. [10, 11] were approved.

9. With `fspai-1.1`, we introduced a new FSPAI implementation. We described its underlying generic code design and gave insight into the technical realization of the costly τ_{jk} computation.

10. We transferred `mspai-1.2`'s dictionary approach to `fspai-1.1` to avoid redundant computations occurring for structured SPD/HPD systems. However, in the factorized case, the gained speedup is lower with a factor of around 1.5. That is, because less expensive computations can be saved: the Cholesky factorization has a lower flop rate and the preconditioner requires only a triangular structure to be filled.

11. Motivated by the gained speedup of sparse QR factorizations in `mspai-1.2`, we analyzed the effect of performing sparse Cholesky factorizations in `fspai-1.1`. Here, we obtained improvement only for the complex case. The construction of SPD preconditioners using Lapack is about twice as fast as with CXSparse.

12. We provided quality comparisons with Matlab which indicate the correct preconditioner computation of `fspai-1.1`. Furthermore, we did runtime and quality comparisons with ParaSails on the Blue Gene/P environment. Our implementation as well as the resulting FSPAI quality show at the very least competitive behavior depending on the chosen setting. Similar to 8., experiments both involving the setup and PCG time illustrate the high scalability of the implementation and support the used parallelization concept also in the factorized case.

Perspectives:

- Motivated by the preliminary results between pointwise and blockwise sparse approximate inverse preconditioners, there is great demand in realizing the B(M)SPAI and/or BFSPAI based on `mspai-1.2` and/or `fspai-1.1`. In connection to this, a proper preprocessing strategy to identify a meaningful block sparsity pattern should be implemented as well. Extensive (parallel) tests would give meaningful insight into the block approaches.

- Exploiting the template method pattern, one could add EFSPAI to `fspai-1.1` in order to obtain more reliable results when comparing both algorithms.

- Similar to using QR updates in `mspai-1.2` [99, 128], one could implement Cholesky updates in `fspai-1.1`, based on Section 3.2.3. Especially for large values of α in $\bar{\Upsilon}_{\alpha,\beta}$, this should reduce the setup costs. A combination of the dictionary approaches and the QR/Cholesky updates could additionally speedup the setup stage for structured matrices.

- In order to simplify the maintainability of `mspai-1.2`, one should provide a single source similar to `fspai-1.1`. Moreover, there is demand for providing the codes as libraries or integrate them in other packages such as PETSc [9].

- One could integrate a float and refinement approach for the solution of the subproblems. First compute the QR or Cholesky factorization in single precision arithmetic and afterward estimate its condition number on R and L, respectively. If it is too large, an additional decomposition in double precision is triggered, otherwise the single precision solution is iteratively refined. This could be especially interesting on architectures where float computations are much faster, e.g., on IBM®'s Cell architecture.

8.3. Smoothing with Sparse Approximate Inverses

13. For different discretizations of the 1D and 2D Laplace operator, we derived optimal smoothers for multigrid methods using generating functions. By translating the occurring minimization conditions into masks and global probing conditions, we derived smoothing MSAI preconditioners. We also analyzed global probing conditions with action on the underlying system and applied them to systems with varying coefficients. Depending on the chosen subspace, it is possible to reach nearly optimal smoothing quality. Moreover, the choice of the probing weight is robust, i.e., for increasing values the subspace dependent optimum is reached.

Perspectives:

- The smoothing quality of SAI and SPAI was analyzed in [133] and [29], respectively. In connection to SPD problems, it is an open point how effective FSAI and FSPAI will be when applied as smoothers.
- More experiments on systems with varying coefficients and the solution of real problems using MS(P)AI smoothers in MG methods will deliver further insight into their smoothing quality.
- Motivated by Theorem 3.1 in [99], one could try to derive an analytic expression for an MSAI M_ρ, incorporating any of the introduced smoothing conditions $\mathsf{S3}_{\mathsf{L}*}$ or $\mathsf{S3}_{\mathsf{G}*}$, e.g., starting with the 1D operator A_1 (5.1). Consequently, it would be possible to analytically compute $M_\infty := \lim_{\rho\to\infty} M_\rho$ and its μ_{smooth}. Based on this, one could try to derive probing conditions which yield optimal improvement, e.g., solving $M_\infty \overset{!}{=} \mathsf{M3}_{\text{opt}}$.

8.4. Improving the Reconstruction of Discrete Ill-Posed Problems

14. We analyzed MSAI's regularization property as preconditioner in CG(LS). Focusing on structured operators, we used a similar methodology as in 13.: we derived optimal regularizing preconditioners and transferred the occurring minimization conditions into local and global probing conditions for MSAI. Besides artificial problems, we also applied the approach to problems from Regularization Tools [68]. For an increasing regularization weight, the application of the resulting preconditioners in PCG tend to CGLS. Lower weights usually lead to sharper convergence curves where the optimum of similar quality is reached in less iterations. For certain ill-posed problems the application in PCGLS yields improved results, also depending on the preconditioner's

adjustment relative to boundaries or discontinuities of the signal. Here, the modification of MSAI's probing subspace with respect to the underlying signal structure leads to strong improvement.

15. Based on smoothing and B-splines, we proposed an L-curve approach for the estimation of discrete regularization parameters. Our comparative study with other L-curve algorithms and the discrepancy principle—using CGLS as reconstruction method—indicates competitive behavior.

16. Motivated by 14., we introduced preconditioners which incorporate available signal data. In this context, we introduced data based regularization methods which can be embedded in an outer iteration in order to obtain further improvement of the reconstruction. For weakly blurred signals containing discontinuities and zero components, strong improvement is achievable.

17. Furthermore, we considered a piecewise reconstruction of the signal, which can be embedded in the outer iteration approach, as well. As no residual-free method for the estimation of the regularization parameter is available, our results are only of theoretical interest. Applied to general blur operators and discrete ill-posed problems, all considered regularization methods (TPR, TSVD, CGLS) show improvement for an increasing number of partitions.

18. For TPR in general form, we derived seminorms which incorporate the spectral data of the operator. Extensive 1D results show that the approach yields robust behavior: it is independent of the signal structure and any choice of the regularization parameter leads to improved or similar reconstruction quality compared to TPR in standard form.

19. Finally, we proposed a combination of smoothing norms and data based regularization for TPR regularization in general form. For smooth signals the positive effect of finite difference regularization matrices can be improved. For discrete deconvolution problems strong improvement can be gained similar to 16.

Perspectives:

- Certain ill-posed problems in image or signal deblurring are characterized by dense matrices; such matrices cannot always be approximated by sparse or banded ones. It is an open point how MS(P)AI behaves in this case, also when used with a dense pattern.
- Using the proposed seminorm approaches for TPR, one could further investigate the visual effects for 2D signals such as images.
- One could provide regularization matrices for TPR which incorporate both the spectral data of the operator and available signal data, e.g., combine data based regularization with operator dependent seminorms.
- If the perturbation norm $\|e\|_2$ or the deviation ξ is unknown, a robust stopping criterion for the Algorithms 10 (ROI) and 11 (ROIP) should be available. Otherwise, deterioration of the solution becomes possible if the (observed) signal does not satisfy the necessary prerequisites.
- In order to make partitioned regularization feasible—at least in terms of a better reconstruction—an accurate (and residual-free) estimation method for regularization parameters is of demand.

8.5. Closing Remark

Sparse approximate inverses are representative in the class of inherently parallel preconditioners which become of increasing importance on the path to exascale computing and beyond. Still, the effect of applying these preconditioners in real large-scale scenarios is sparely explored. In sequential environments, (variants of) ILU and IC are typically preferred mainly because of lower setup costs yielding lucrative quality. Block sparse approximate inverses indicate to be an alternative—at least for block problems—due to robustness and significantly reduced setup costs compared to their pointwise adaptive variants. Preliminary results—signifying comparable quality to ILU and IC [84]—motivate for technical realizations and further experiments to confirm these prospects. In this context, the quest for algorithms producing efficient a priori block patterns is an open point. The MMSPAI and EFSPAI preconditioners are variants maintaining the advantages of sparse approximate inverses. However, they should only be preferred to the classical Spai-like preconditioners under certain conditions such as the need for a moderately improved SAI at lower setup costs than fully augmented multistep preconditioners or a guaranteed minimization of the K-condition number.

A fundamental step from petascale to exascale computing involves a paradigm change for the parallel implementation of algorithms. Emerging high performance clusters will contain hundreds of millions of cores yielding peak performance in a multilevel parallelism [49, 126]:

1. In order to minimize expensive data movement, merely the nodes of the cluster will communicate via MPI, i.e., there will be only one MPI task per node.
2. Using OpenMP, shared memory parallelization will additionally avoid data movement between the cores on each node.
3. A bulk of the power consumption of a PE is a result of its required memory management. With the upcoming number of simultaneously working PEs, power consumption becomes an apparent bottleneck. Vectorization performed by accelerators such as GPUs might be a remedy to maximize flops per Watt.

Consequently, parallel software will require new hybrid memory approaches in order to exploit all levels of parallelism efficiently. Moreover, algorithms which are inherently parallel become the basic prerequisite for exascalable implementations. Block versions of sparse approximate inverses seem to fit into all these requirements. However, Barnard et al.'s polling mechanism must be rethought in this new context. Note that projects such as the one mentioned in Remark #4 in Section 7.2.4 will deliver insight into promising techniques which might fit into one level of the multi-parallel concept.

MG or iterative regularization methods have a different action on the low respectively high frequency part of the spectrum and benefit from smoothing and regularizing preconditioners which have a different behavior on these subspaces. If the probing subspace is chosen properly, the desired smoothing or regularizing effect can be achieved with MS(P)AI. Especially in connection to structured systems, it is possible to show that MS(P)AI may yield nearly optimal results. Transferring the probing approach to natural ill-posed problems yields at least one of the positive effects which are demanded from regularizing preconditioners. However, the choice of the subspace is more sensitive than for smoothing preconditioners.

It is known that the solution of (discrete) ill-posed inverse problems is problem dependent. This typically leads to different reconstruction techniques which yield improvement only

under certain conditions. Moreover, it seems that the quality of the reconstruction may also depend on whether the structure of the signal is taken into account or not, at least if the solution's domain and the domain of the observed data is not fundamentally different. Constructing preconditioners for iterative regularization methods which are effective for a large number of problems remains even more challenging than for the noise-free case. Here, a different treatment of the reconstruction process on the signal and noise subspace is difficult to combine, especially for a setup in a global frequency sense. Note that any modelling of the high and low frequency subspace remains a heuristic sensitive to the underlying problem and the magnitude of the noise. Moreover, noise that is not white (or related to it) requires a different treatment, in general.

Appendix A

Mathematical Details

Many of the mathematical details used in this work and presented in the following can be found in Horn and Johnson [78].

A.1. Derivatives

- For $X \in \mathbb{R}^{n\times n}$ and $A \in \mathbb{R}^{n\times n}$, we observe

$$\frac{\partial}{\partial X}\operatorname{trace}(X^T A) = \operatorname{trace}(A^T X) = A. \tag{A.1}$$

Proof A (block) component-wise derivation yields

$$\frac{\partial}{\partial X_{ij}}\operatorname{trace}(X^T A) = \frac{\partial}{\partial X_{ij}}\sum_k \left[X(.,k)^T A(.,k)\right] = \frac{\partial}{\partial X_{ij}}\sum_k\sum_l (X_{lk}A_{lk}) = A_{ij}. \quad \blacksquare$$

- Let $X \in \mathbb{R}^{n\times n}$ and $A \in \mathbb{R}^{n\times n}$, then

$$\frac{\partial}{\partial X}\operatorname{trace}(X^T AX) = AX + A^T X = (A + A^T)X. \tag{A.2}$$

Proof We can write the trace in (block) component wise form

$$\operatorname{trace}(X^T AX) = \sum_k (X^T AX)_{kk} = \sum_k\sum_l (X^T A)_{kl}X_{lk} = \sum_k\sum_l\sum_m X_{mk}A_{ml}X_{lk}.$$

Via the derivation of a component X_{ij}, we receive the solution of the derivative

$$\frac{\partial}{\partial X_{ij}}\operatorname{trace}(X^T AX) = \sum_l A_{il}X_{lj} + \sum_m X_{mj}A_{mi},$$

which in matrix notation can be written as $AX + A^T X = (A + A^T)X$. $\blacksquare$

- For $X \in \mathbb{R}^{n\times n}$ holds

$$\frac{\partial}{\partial X}\det(X) = \det(X)X^{-T}. \tag{A.3}$$

Proof As $\operatorname{adj}(X) = \operatorname{cof}(X)^T$, we obtain

$$X^{-1} = \det(X)^{-1}\operatorname{adj}(X) = \det(X)^{-1}\operatorname{cof}(X)^T \qquad \Leftrightarrow \qquad \operatorname{cof}(X) = \det(X)X^{-T}.$$

Using the cofactor expansion of $\det(X)$ along the i^{th} row, we obtain

$$\det(X) = \sum_j X_{ij} \operatorname{cof}(X_{ij}),$$

where $\operatorname{cof}(X_{ij})$ is the (i,j) cofactor of X. The partial derivative of $\det(X)$ with respect to the k^{th} component in the i^{th} row of X yields

$$\frac{\partial}{\partial X_{ik}} \det(X) = \frac{\partial}{\partial X_{ik}} \sum_j X_{ij} \operatorname{cof}(X_{ij}) = \operatorname{cof}(X_{ik}).$$

Hence, we can write the full derivative as

$$\frac{\partial}{\partial X} \det(X) = \operatorname{cof}(X) = \det(X) X^{-T}. \qquad \blacksquare$$

A.2. Inequalities

- For $x_1, \ldots, x_n$ being nonnegative with $x_k \in \mathbb{R}$, the inequality of the geometric and arithmetic mean reads as

$$\sqrt[n]{\prod_{k=1}^{n} x_k} \leq \frac{1}{n} \sum_{k=1}^{n} x_k. \tag{A.4}$$

- For $x \in \mathbb{R}$, $x \geq -1$, and $n \in \mathbb{N}$, Bernoulli's inequality holds with

$$(1+x)^n \geq 1 + nx. \tag{A.5}$$

- For SPD $A = \begin{pmatrix} a_{ii} & a_{ij} \\ a_{ij} & a_{jj} \end{pmatrix}$, we observe

$$\frac{a_{ij}^2}{a_{ii} a_{jj}} < 1. \tag{A.6}$$

Proof $\det(A) = a_{ii} a_{jj} - a_{ij}^2 > 0 \quad \Leftrightarrow \quad \frac{a_{ij}^2}{a_{ii} a_{jj}} < 1.$ $\blacksquare$

- For SPD $A \in \mathbb{R}^{n \times n}$ holds

$$\operatorname{trace}(A) > 0. \tag{A.7}$$

- For SPD $A \in \mathbb{R}^{n \times n}$ and SPD $B \in \mathbb{R}^{n \times n}$, we have

$$\operatorname{trace}(AB) > 0. \tag{A.8}$$

Proof Let $A = L^T L$ be the Cholesky decomposition of A. Then the inequality

$$\operatorname{trace}(AB) = \operatorname{trace}(L^T L B) = \operatorname{trace}(L B L^T) \overset{(A.7)}{>} 0$$

is valid. $\blacksquare$

- For SPD $A \in \mathbb{R}^{n\times n}$ and positive semidefinite $B \in \mathbb{R}^{n\times n}$, we observe

$$\operatorname{trace}(AB) \geq 0 \tag{A.9}$$

- For SPD $B \in \mathbb{R}^{n\times n}$ holds

$$\det(B)^{\frac{1}{n}} \leq \frac{1}{n}\operatorname{trace}(B). \tag{A.10}$$

Proof Considering

$$\operatorname{trace}(B) = \sum_k \lambda_k(B) \qquad \text{and} \qquad \det(B) = \prod_k \lambda_k(B),$$

we can apply (A.4). ■

- For SPD $A \in \mathbb{R}^{n\times n}$ and SPD $B \in \mathbb{R}^{n\times n}$, we observe

$$\det(AB) \leq \left[\frac{1}{n}\operatorname{trace}(AB)\right]^n. \tag{A.11}$$

Proof Let $A = L^T L$ be the Cholesky decomposition of A. Then we obtain

$$\begin{aligned}\det(AB) &= \det(L^T LB) = \det(LBL^T) \overset{(A.10)}{\leq} \left[\frac{1}{n}\operatorname{trace}(LBL^T)\right]^n \\ &= \left[\frac{1}{n}\operatorname{trace}(L^T LB)\right]^n = \left[\frac{1}{n}\operatorname{trace}(AB)\right]^n.\end{aligned}$$

■

Appendix B

Theoretical Properties of BFS(P)AI

The properties of FS(P)AI introduced in Section 3.2.4 can be predominantly transferred to the block case introduced in Section 4.2 and 4.3. Consider L to be a BFSAI solution for a given block pattern $\mathcal{J}(L)$ and $\mathcal{J}_k = \tilde{\mathcal{J}}_k \cup \{k\}$.

Lemma B.1 *Based on $\mathcal{J}_k$, the BFSAI solution L_k satisfies*

$$A(\mathcal{J}_k,\mathcal{J}_k)L_k(\mathcal{J}_k) = \begin{pmatrix} S_{kk}L_{kk} \\ 0 \end{pmatrix} \in \mathbb{R}^{|\mathcal{J}_k|\times b}.$$

Proof Can be obtained by following proof in Lemma 3.1 trivially generalized to block indices. ■

Theorem B.1 *The weighted Frobenius (energy) norm of a BFSAI solution L_k satisfies*

$$\|L_k\|_A = \sqrt{\text{trace}(L_k^T A L_k)} = \sqrt{b}.$$

Proof

$$L_k^T A L_k = L_k(\mathcal{J}_k)^T A(\mathcal{J}_k,\mathcal{J}_k)L_k(\mathcal{J}_k) \overset{\text{Lemma } B.1}{=} L_k(\mathcal{J}_k)^T \begin{pmatrix} S_{kk}L_{kk} \\ 0 \end{pmatrix} = L_{kk}^T S_{kk} L_{kk} \overset{(4.8)}{=} I_b.$$

Hence, we obtain $\|L_k\|_A = \sqrt{\text{trace}(L_k^T A L_k)} = \sqrt{I_b} = \sqrt{b}$. ■

Corollary B.1 *For every BFSAI solution L holds $\frac{1}{n}\text{trace}(L^T A L) = 1$.*

Proof

$$\frac{1}{n}\text{trace}(L^T A L) = \frac{1}{n}\sum_{k=1}^{n_b}\text{trace}(L_k^T A L_k) \overset{\text{Theorem } B.1}{=} \frac{1}{n}\sum_{k=1}^{n_b} b = \frac{n_b b}{n} = 1.$$ ■

Theorem B.2 *If all $\tau_{jk} = 0$, the corresponding block column L_k is the exact inverse of the k^{th} block column of the Cholesky factor of A, i.e., if*

$$\forall k : \tau_{jk} = 0 \quad \textit{for all} \quad j \in \{k+1,\ldots,n_b\} \setminus \tilde{\mathcal{J}}_k, \qquad \textit{then} \qquad L^T A L = I.$$

Proof Let $\tilde{\mathcal{I}}_k = \{k+1,\ldots,n_b\}$. Following (4.8), $(L^T A L)_{kk} = I_b$. For the non-diagonal block components $L_i^T A L_k$, we assume $i > k$ without loss of generality. Therefore, $\mathcal{J}_i \subset \tilde{\mathcal{I}}_k$. Based on the assumption, we observe

$$\forall j \in \tilde{\mathcal{I}}_k \setminus \tilde{\mathcal{J}}_k : \tau_{jk} = \|A_j^T L_k\|^2_{A_{jj}^{-1}} = 0 \quad \Leftrightarrow \quad A_j^T L_k = 0 \quad \overset{\text{Lemma } B.1}{\Rightarrow} \quad A(\tilde{\mathcal{I}}_k,.)L_k = 0.$$

With $\mathcal{J}_i \subset \tilde{\mathcal{I}}_k$, the equality

$$L_i^T A L_k = L_i(\tilde{\mathcal{I}}_k)^T A(\tilde{\mathcal{I}}_k,.)L_k = 0$$

is valid, verifying that all non-diagonal blocks are zero blocks. ■

Corollary B.2 *For a full block sparsity pattern $\mathcal{J}$ the BFSAI solution L is the exact inverse of the Cholesky factor of A, i.e., $L^T AL = I$.*

Proof With $\{k+1,\ldots,n_b\} \setminus \tilde{\mathcal{J}}_k = \emptyset$, this follows immediately from Theorem B.2. ■

Corollary B.3 *For $\varepsilon_{BFSPAI} = 0$, $\alpha \geq n$, and $\beta \geq 1$, BFSPAI will compute the exact inverse $L = L_A^{-1}$.*

Proof Without any stopping criterion, BFSPAI will augment $\mathcal{J}_k$ with block indices j unless all $\tau_{jk} = 0$. Now we apply Theorem B.2. ■

Appendix C

Exact Block FSPAI: EBFSPAI

Similar to EFSPAI in Section 3.2.5, it is possible to derive the exact reduction factor for the K-condition number in BFSPAI when augmenting $\tilde{\mathcal{J}}_k$ with a block index j. Considering the definitions

$$\tilde{\mathcal{J}}_k^{+\mathrm{j}} := \tilde{\mathcal{J}}_k \cup \{j\} \qquad \text{and} \qquad \mathcal{I}_k^{-\mathrm{j}} := \mathcal{I}_k \setminus \{j\} \qquad \text{with} \qquad \mathcal{I}_k = \{k,\dots,n_b\} \setminus \tilde{\mathcal{J}}_k,$$

and S_{old} denoting the Schur complement of $A(\tilde{\mathcal{J}}_k,\tilde{\mathcal{J}}_k)$ in $A_{k:n,k:n}$, we can write the new Schur complement S_{new} of $A(\tilde{\mathcal{J}}_k^{+\mathrm{j}},\tilde{\mathcal{J}}_k^{+\mathrm{j}})$ in $A_{k:n,k:n}$ as

$$S_{\text{new}} = A(\mathcal{I}_k^{-\mathrm{j}},\mathcal{I}_k^{-\mathrm{j}}) - A(\tilde{\mathcal{J}}_k^{+\mathrm{j}},\mathcal{I}_k^{-\mathrm{j}})^T A(\tilde{\mathcal{J}}_k^{+\mathrm{j}},\tilde{\mathcal{J}}_k^{+\mathrm{j}})^{-1} A(\tilde{\mathcal{J}}_k^{+\mathrm{j}},\mathcal{I}_k^{-\mathrm{j}}).$$

Using block Cholesky updates according to (3.52), we can compute S_{new} by incorporating the known decomposition L^TL for the original set $\tilde{\mathcal{J}}_k$. This leads to

$$\begin{aligned} S_{\text{new}} &= A(\mathcal{I}_k^{-\mathrm{j}},\mathcal{I}_k^{-\mathrm{j}}) - A(\tilde{\mathcal{J}}_k^{+\mathrm{j}},\mathcal{I}_k^{-\mathrm{j}})^T \left(\tilde{P}^T\tilde{L}^T\tilde{L}\tilde{P}\right)^{-1} A(\tilde{\mathcal{J}}_k^{+\mathrm{j}},\mathcal{I}_k^{-\mathrm{j}}) \\ &= A(\mathcal{I}_k^{-\mathrm{j}},\mathcal{I}_k^{-\mathrm{j}}) - Y^TY, \qquad \text{where} \qquad Y = \tilde{L}^{-T}\tilde{P}A(\tilde{\mathcal{J}}_k^{+\mathrm{j}},\mathcal{I}_k^{-\mathrm{j}}). \end{aligned}$$

With the Cholesky factor L in $\tilde{L}$, we solve $\tilde{L}^TY = \tilde{P}A(\tilde{\mathcal{J}}_k^{+\mathrm{j}},\mathcal{I}_k^{-\mathrm{j}})$ to obtain Y:

$$\begin{pmatrix} \mathrm{chol}[(S_{\text{old}})_{jj}]^T & A_j(\tilde{\mathcal{J}}_k)^TL^{-1} \\ 0 & L^T \end{pmatrix} \begin{pmatrix} Y_1 \\ Y_2 \end{pmatrix} = \begin{pmatrix} A(j,\mathcal{I}_k^{-\mathrm{j}}) \\ A(\tilde{\mathcal{J}}_k,\mathcal{I}_k^{-\mathrm{j}}) \end{pmatrix}$$

$$\Leftrightarrow \quad Y_2 = L^{-T}A(\tilde{\mathcal{J}}_k,\mathcal{I}_k^{-\mathrm{j}}) \qquad \text{and} \tag{C.1}$$

$$\begin{aligned} Y_1 &= \mathrm{chol}[(S_{\text{old}})_{jj}]^{-T}\left[A(j,\mathcal{I}_k^{-\mathrm{j}}) - A_j(\tilde{\mathcal{J}}_k)^TL^{-1}L^{-T}A(\tilde{\mathcal{J}}_k,\mathcal{I}_k^{-\mathrm{j}})\right] \\ &= \mathrm{chol}[(S_{\text{old}})_{jj}]^{-T}S_{\text{old}}(j,\mathcal{I}_k^{-\mathrm{j}}). \end{aligned} \tag{C.2}$$

Therewith, the new Schur complement can be computed as

$$\begin{aligned} S_{\text{new}} &= A(\mathcal{I}_k^{-\mathrm{j}},\mathcal{I}_k^{-\mathrm{j}}) - Y_1^TY_1 + Y_2^TY_2 \\ &\overset{(C.1)}{\overset{(C.2)}{=}} A(\mathcal{I}_k^{-\mathrm{j}},\mathcal{I}_k^{-\mathrm{j}}) - A(\tilde{\mathcal{J}}_k,\mathcal{I}_k^{-\mathrm{j}})^T \underbrace{L^{-1}L^{-T}}_{=\,A(\tilde{\mathcal{J}}_k,\tilde{\mathcal{J}}_k)^{-1}} A(\tilde{\mathcal{J}}_k,\mathcal{I}_k^{-\mathrm{j}}) - \\ &\qquad - S_{\text{old}}(j,\mathcal{I}_k^{-\mathrm{j}})^T\,\mathrm{chol}[(S_{\text{old}})_{jj}]^{-1}\,\mathrm{chol}[(S_{\text{old}})_{jj}]^{-T}S_{\text{old}}(j,\mathcal{I}_k^{-\mathrm{j}}) \\ &\overset{\text{Def. } 3.3}{=} S_{\text{old}}(\mathcal{I}_k^{-\mathrm{j}},\mathcal{I}_k^{-\mathrm{j}}) - S_{\text{old}}(j,\mathcal{I}_k^{-\mathrm{j}})^T(S_{\text{old}})_{jj}^{-1}S_{\text{old}}(j,\mathcal{I}_k^{-\mathrm{j}}). \end{aligned} \tag{C.3}$$

Because the numerator of $K(L^TAL)$ (3.36) remains constant—according to Corollary B.1—and the denominator only depends on L_{kk}, we have $(L_{\text{old}})_{jj} = (L_{\text{new}})_{jj}$ for $j \neq k$. Hence, the new K-condition number can be written as

$$\begin{aligned}
K_{\text{new}} &\overset{\text{Corollary } B.1}{=} \frac{1}{\det(A)^{\frac{1}{n}} \prod_{j=1}^{n_b} \det[(L_{\text{new}})_{jj}]^{\frac{2}{n}}} \\
&= \frac{1}{\det[(L_{\text{new}})_{kk}]^{\frac{2}{n}} \det(A)^{\frac{1}{n}} \prod_{\substack{j=1 \\ j\neq k}}^{n_b} \det[(L_{\text{old}})_{jj}]^{\frac{2}{n}}} \\
&\overset{\substack{\det(X)^{-1}=\det(X^{-1}) \\ \det(XY)=\det(X)\det(Y)}}{=} \frac{\left[\det[(L_{\text{new}})_{kk}^{-1}] \det[(L_{\text{new}})_{kk}^{-1}]\right]^{\frac{1}{n}}}{\det(A)^{\frac{1}{n}} \prod_{\substack{j=1 \\ j\neq k}}^{n_b} \det[(L_{\text{old}})_{jj}]^{\frac{2}{n}}} \\
&\overset{\substack{\det(X)=\det(X^T) \\ (4.10)}}{=} \frac{\det[(S_{\text{new}})_{kk}]^{\frac{1}{n}}}{\det(A)^{\frac{1}{n}} \prod_{\substack{j=1 \\ j\neq k}}^{n_b} \det[(L_{\text{old}})_{jj}]^{\frac{2}{n}}}.
\end{aligned} \tag{C.4}$$

Following (C.3), the diagonal blocks of the Schur complement are updated via

$$\begin{aligned}
(S_{\text{new}})_{kk} &= (S_{\text{old}})_{kk} - (S_{\text{old}})_{jk}^T (S_{\text{old}})_{jj}^{-1} (S_{\text{old}})_{jk} \\
&= (S_{\text{old}})_{kk} \Big[I_b - \underbrace{(S_{\text{old}})_{kk}^{-1} (S_{\text{old}})_{jk}^T (S_{\text{old}})_{jj}^{-1} (S_{\text{old}})_{jk}}_{=:\ \mu_{jk}} \Big] = (S_{\text{old}})_{kk} (I_b - \mu_{jk}).
\end{aligned}$$

Consequently, the relationship between K_{old} and K_{new} (C.4)—corresponding to $\tilde{\mathcal{J}}_k$ and $\tilde{\mathcal{J}}_k^{+j}$, respectively—can be described by

$$\begin{aligned}
K_{\text{new}} &= \frac{\det[(S_{\text{old}})_{kk}]^{\frac{1}{n}} \det(I_b - \mu_{jk})^{\frac{1}{n}}}{\det(A)^{\frac{1}{n}} \prod_{\substack{j=1 \\ j\neq k}}^{n} \det[(L_{\text{old}})_{jj}]^{\frac{2}{n}}} = K_{\text{old}} \det(I_b - \mu_{jk})^{\frac{1}{n}} \qquad \text{with} \\
\mu_{jk} &:= (S_{\text{old}})_{kk}^{-1} (S_{\text{old}})_{jk}^T (S_{\text{old}})_{jj}^{-1} (S_{\text{old}})_{jk}
\end{aligned}$$

being the exact reduction factor when adding a block index j to the block sparsity pattern $\mathcal{J}_k$. It corresponds to the heuristic factor τ_{jk} in (4.15). Similar to Lemma 3.2 for EFSPAI, we can formulate the corresponding

Lemma C.1 *The heuristic reduction factor τ_{jk} (4.15) in the block case can be written as*

$$\tau_{jk} = \text{trace}[(S_{old})_{kk}^{-1} (S_{old})_{jk}^T A_{jj}^{-1} (S_{old})_{jk}].$$

Proof Observing

$$\begin{aligned}
A_j^T L_k &= A_j(\mathcal{J}_k)^T L_k(\mathcal{J}_k) = \begin{pmatrix} A_{kj} \\ A_j(\tilde{\mathcal{J}}_k) \end{pmatrix}^T \begin{pmatrix} L_{kk} \\ -A(\tilde{\mathcal{J}}_k, \tilde{\mathcal{J}}_k)^{-1} A_k(\tilde{\mathcal{J}}_k) L_{kk} \end{pmatrix} \\
&= \left[A_{jk} - A_j(\tilde{\mathcal{J}}_k)^T A(\tilde{\mathcal{J}}_k, \tilde{\mathcal{J}}_k)^{-1} A_k(\tilde{\mathcal{J}}_k) \right] L_{kk} = (S_{\text{old}})_{jk} \, \text{chol}[(S_{\text{old}})_{kk}]^{-1},
\end{aligned}$$

we can write the τ_{jk} as

$$\tau_{jk} = \text{trace}(L_k^T A_j A_{jj}^{-1} A_j^T L_k) = \text{trace}(A_j^T L_k L_k^T A_j A_{jj}^{-1})$$

$$= \operatorname{trace}[(S_{\text{old}})_{jk}\,\operatorname{chol}[(S_{\text{old}})_{kk}]^{-1}\,\operatorname{chol}[(S_{\text{old}})_{kk}]^{-T}(S_{\text{old}})_{jk}^T A_{jj}^{-1}]$$
$$= \operatorname{trace}[(S_{\text{old}})_{jk}(S_{\text{old}})_{kk}^{-1}(S_{\text{old}})_{jk}^T A_{jj}^{-1}] = \operatorname{trace}[(S_{\text{old}})_{kk}^{-1}(S_{\text{old}})_{jk}^T A_{jj}^{-1}(S_{\text{old}})_{jk}]. \quad \blacksquare$$

Theorem C.1 *The heuristic reduction factor τ_{jk} in the block case (4.15) satisfies*

$$0 \le \tau_{jk} < b.$$

Proof Based on Lemma C.1, the first inequality is trivial as A and S are positive definite. For the second one, we derive

$$\tau_{jk} = \operatorname{trace}[(S_{\text{old}})_{kk}^{-1}(S_{\text{old}})_{jk}^T A_{jj}^{-1}(S_{\text{old}})_{jk}] = b - \operatorname{trace}\left[I_b - (S_{\text{old}})_{kk}^{-1}(S_{\text{old}})_{jk}^T A_{jj}^{-1}(S_{\text{old}})_{jk}\right]$$
$$= b - \operatorname{trace}\left[(S_{\text{old}})_{kk}^{-1}\left[(S_{\text{old}})_{kk} - (S_{\text{old}})_{jk}^T A_{jj}^{-1}(S_{\text{old}})_{jk}\right]\right] \overset{(A.8)}{<} b. \quad \blacksquare$$

Lemma C.2 *The inequality* $\operatorname{trace}\left[(S_{old})_{jj}A_{jj}^{-1}\right] \le b$ *holds.*

Proof

$$\operatorname{trace}\left[(S_{\text{old}})_{jj}A_{jj}^{-1}\right] = \operatorname{trace}\left[I_b - A_j(\tilde{\mathcal{J}}_k)^T A(\tilde{\mathcal{J}}_k,\tilde{\mathcal{J}}_k)^{-1}A_j(\tilde{\mathcal{J}}_k)A_{jj}^{-1}\right]$$
$$= b - \operatorname{trace}\left[A_j(\tilde{\mathcal{J}}_k)^T A(\tilde{\mathcal{J}}_k,\tilde{\mathcal{J}}_k)^{-1}A_j(\tilde{\mathcal{J}}_k)A_{jj}^{-1}\right] \overset{(A.9)}{\le} b. \quad \blacksquare$$

Lemma C.3 *The inequality* $\det\left[(S_{old})_{jj}^{-1}A_{jj}\right] \le 1$ *holds.*

Proof Using the Woodbury matrix identity [53], we observe that

$$(S_{\text{old}})_{jj}^{-1} = \left[A_{jj} - A_j(\tilde{\mathcal{J}}_k)^T A(\tilde{\mathcal{J}}_k,\tilde{\mathcal{J}}_k)^{-1}A_j(\tilde{\mathcal{J}}_k)\right]^{-1}$$
$$= A_{jj}^{-1} - A_{jj}^{-1}A_j(\tilde{\mathcal{J}}_k)^T\left[A(\tilde{\mathcal{J}}_k,\tilde{\mathcal{J}}_k) + A_j(\tilde{\mathcal{J}}_k)A_{jj}^{-1}A_j(\tilde{\mathcal{J}}_k)^T\right]^{-1}A_j(\tilde{\mathcal{J}}_k)A_{jj}^{-1}$$

and thus

$$(S_{\text{old}})_{jj}^{-1}A_{jj} = I_b - A_{jj}^{-1}A_j(\tilde{\mathcal{J}}_k)^T\left[A(\tilde{\mathcal{J}}_k,\tilde{\mathcal{J}}_k) + A_j(\tilde{\mathcal{J}}_k)A_{jj}^{-1}A_j(\tilde{\mathcal{J}}_k)^T\right]^{-1}A_j(\tilde{\mathcal{J}}_k). \tag{C.5}$$

We obtain the estimate

$$\det\left[(S_{\text{old}})_{jj}^{-1}A_{jj}\right] \overset{(A.11)}{\le} \left[\frac{1}{b}\operatorname{trace}[(S_{\text{old}})_{jj}^{-1}A_{jj}]\right]^b$$
$$\overset{(C.5)}{=} \frac{1}{b^b}\left\{b - \operatorname{trace}\left[A_{jj}^{-1}A_j(\tilde{\mathcal{J}}_k)^T\left[A(\tilde{\mathcal{J}}_k,\tilde{\mathcal{J}}_k) + A_j(\tilde{\mathcal{J}}_k)A_{jj}^{-1}A_j(\tilde{\mathcal{J}}_k)^T\right]^{-1}A_j(\tilde{\mathcal{J}}_k)\right]\right\}^b$$
$$\overset{(A.9)}{\le} \frac{1}{b^b}b^b = 1. \quad \blacksquare$$

Applying the above lemmas, we can state

Theorem C.2 *The exact and heuristic reduction factor τ_{jk} (Lemma C.1) satisfy the relation*

$$\det(I_b - \mu_{jk}) \le \left(1 - \frac{\tau_{jk}}{b}\right)^b.$$

Proof Applying Sylvester's determinant theorem [53], we can transform the left-hand side into

$$\begin{aligned}
\det\left[I_b - (S_{\text{old}})_{kk}^{-1}(S_{\text{old}})_{jk}^{T}(S_{\text{old}})_{jj}^{-1}(S_{\text{old}})_{jk}\right] &= \det\left[I_b - (S_{\text{old}})_{jk}(S_{\text{old}})_{kk}^{-1}(S_{\text{old}})_{jk}^{T}(S_{\text{old}})_{jj}^{-1}\right] \\
&= \det\left\{\left[(S_{\text{old}})_{jj} - (S_{\text{old}})_{jk}(S_{\text{old}})_{kk}^{-1}(S_{\text{old}})_{kj}^{T}\right]A_{jj}^{-1}\right\}\det\left[A_{jj}(S_{\text{old}})_{jj}^{-1}\right] \\
\overset{(S_{\text{old}})_{jk}\overset{\text{symmetry}}{=}(S_{\text{old}})_{kj}^{T}}{=} &\det\left\{\Big[\underbrace{(S_{\text{old}})_{jj} - (S_{\text{old}})_{kj}^{T}(S_{\text{old}})_{kk}^{-1}(S_{\text{old}})_{jk}}_{\text{SPD}}\Big]A_{jj}^{-1}\right\}\det\left[A_{jj}(S_{\text{old}})_{jj}^{-1}\right].
\end{aligned} \tag{C.6}$$

Based on the representation (C.6), we observe the estimate

$$\begin{aligned}
&\det\left[I_b - (S_{\text{old}})_{kk}^{-1}(S_{\text{old}})_{jk}^{T}(S_{\text{old}})_{jj}^{-1}(S_{\text{old}})_{jk}\right] \\
\overset{(A.11)}{\leq} &\frac{1}{b^b}\,\text{trace}\left\{\left[(S_{\text{old}})_{jj} - (S_{\text{old}})_{kj}^{T}(S_{\text{old}})_{kk}^{-1}(S_{\text{old}})_{jk}\right]A_{jj}^{-1}\right\}^b \det[A_{jj}(S_{\text{old}})_{jj}^{-1}] \\
\overset{\det(XY)=\det(YX)\ \text{Lemma } C.3}{\leq} &\frac{1}{b^b}\,\text{trace}\left\{\left[(S_{\text{old}})_{jj} - (S_{\text{old}})_{kj}^{T}(S_{\text{old}})_{kk}^{-1}(S_{\text{old}})_{jk}\right]A_{jj}^{-1}\right\}^b \\
= &\frac{1}{b^b}\left\{\text{trace}\left[(S_{\text{old}})_{jj}A_{jj}^{-1}\right] - \text{trace}\left[(S_{\text{old}})_{kj}^{T}(S_{\text{old}})_{kk}^{-1}(S_{\text{old}})_{jk}A_{jj}^{-1}\right]\right\}^b \\
\overset{\text{Lemma } C.2}{\leq} &\frac{1}{b^b}\left\{b - \text{trace}\left[(S_{\text{old}})_{kj}^{T}(S_{\text{old}})_{kk}^{-1}(S_{\text{old}})_{jk}A_{jj}^{-1}\right]\right\}^b \\
\overset{\text{Lemma } C.1}{=} &\left(1 - \frac{\tau_{jk}}{b}\right)^b.
\end{aligned}$$

■

Corollary C.1 *The heuristic reduction factor τ_{jk} and the exact reduction factor μ_{jk} induce the following relationship between the K-condition numbers for the block case*

$$K_{new} = \det(I_b - \mu_{jk})^{\frac{1}{n}} K_{old} \leq \left(1 - \frac{\tau_{jk}}{b}\right)^{\frac{1}{n_b}} K_{old} \leq \left(1 - \frac{\tau_{jk}}{n}\right) K_{old}.$$

Proof The first inequality holds due to Theorem C.2 and $n_b = \frac{n}{b}$. We obtain the second inequality applying Bernoulli's inequality (A.5):

$$\left(1 - \frac{\tau_{jk}}{n}\right)^{n_b} \geq 1 - \frac{\tau_{jk}}{b} \quad \Leftrightarrow \quad 1 - \frac{\tau_{jk}}{n} \geq \left(1 - \frac{\tau_{jk}}{b}\right)^{\frac{1}{n_b}}.$$

■

Appendix D

Experimental Environments

D.1. Parallel Environment Blue Gene/P on Shaheen

Hardware	Blue Gene/P racks	16
	Nodes per rack	1024
	Processors per node	4
	Total number of cores	65536
	Processor type	Quad-core PowerPC 450
	Core clock speed	0.850 GHz
	L1 cache	32 KB
	L2 cache	2 MB
	L3 cache	8 MB
	RAM per node	4 GB
	Disk subsystem	see URL
	Network	see URL
Operating system	2.6.16.60-0.74.7-ppc64 ppc64 GNU/Linux	
Library versions	MPI	GNU/4.1.2-mpich2-1.3 IBM
	Lapack	3.2.1
	CSparse	2.2.3
	CXSparse	2.2.6
	Hypre	2.7
	UFConfig	3.7.0
Compiler version	IBM XL C/C++ Advanced Edition for Blue Gene/P 9.0	
Used compiler optimization flags	`-O3 -qarch=450d -qtune=450 -qsmp=omp`	
URL	http://www.hpc.kaust.edu.sa/documentation/shaheen/	

D.2. Parallel Environment InfiniCluster

Hardware	Nodes	36
	Processors per node	4
	Total number of cores	128
	Currently available cores	120
	Processor type	AMD® Opteron® 850
	Core clock speed	2.391 GHz
	L1 cache	16 KB
	L2 cache	1 MB
	RAM per node	8 GB
	Disk subsystem	2x 73 GB SCSI
	Network	4x Infiniband & Gigabit Ethernet
Operating system	2.6.24-23-server x86_64 GNU/Linux	
Library versions	OpenMPI	1.3
	Lapack_Atlas	3.0
	CSparse	2.2.0
	CXSparse	2.2.5
	UFConfig	3.7.0
Compiler version	Gcc	4.2.4
Used compiler optimization flags	`-O3 -Wstrict-aliasing -fstrict-aliasing -fshort-enums -finline-limit=10000000000 -ffast-math -param large-function-growth=10000000000 -param max-inline-insns-single=10000000000`	
URL	http://www.lrr.in.tum.de/Par/arch/infiniband/	

D.3. Sequential Environment Workstation 1

Hardware	Processor type	Intel® Pentium® D
	Cores	2
	Core clock speed	2.810 GHz
	L1 cache	16 KB
	L2 cache	1 MB
	RAM	3 GB
Operating system	2.6.32-37-generic x86_64 GNU/Linux	

D.4. Sequential Environment Workstation 2

Hardware	Processor type	Intel® Core™ 2 Duo E6600
	Cores	2
	Core clock speed	2.400 GHz
	L1 cache	32 KB
	L2 cache	4 MB
	RAM	2 GB
	Hard disc capacity	250 GB
Operating system	2.6.20-16-generic i686 GNU/Linux	
Library versions	Lapack	3.1.1
	Atlas	3.6.0
	CSparse	2.2.0
Compiler version	Gcc	4.1.2
Used compiler optimization flags	`-O3`	

D.5. Sequential Environment Notebook

Hardware	Processor type	Intel® Core™ 2 Duo P8700
	Cores	2
	Core clock speed	2.527 GHz
	L1 cache	128 KB
	L2 cache	3 MB
	RAM	4 GB
	Hard disc capacity	250 GB
Operating system	2.6.31-gentoo-r1 i686 GNU/Linux	
Library versions	Lapack	3.1.1-r1
	Atlas	3.8.0
	CSparse	2.2.3
	CXSparse	2.2.5
	UFConfig	3.7.0
Compiler version	Gcc	4.3.4
Used compiler optimization flags	`-O3`	

Bibliography

[1] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK Users' Guide*, SIAM, 3rd ed., 1999.

[2] R. S. Anderssen and P. M. Prenter, *A formal comparison of methods proposed for the numerical solution of first kind integral equations*, J. Austral. Math. Soc. (Series B), 22 (1981), pp. 488–500.

[3] P. Arbenz, O. Chinellato, M. H. Gutknecht, and M. Sala, *Software for numerical linear algebra* , ETH, Eidgenössische Technische Hochschule Zürich, Institut für Wissenschaftliches Rechnen, Zürich, 2006. http://e-collection.library.ethz.ch/view/eth:28724 (May 7th 2012).

[4] Atlas, *Automatically tuned linear algebra software, home page.* http://math-atlas.sourceforge.net/ (April 11th 2012).

[5] O. Axelsson, *Iterative solution methods*, Cambridge University Press, 1994.

[6] O. Axelsson and I. E. Kaporin, *On the sublinear and superlinear rate of convergence of conjugate gradient methods*, Numer. Algorithms, 25 (2000), pp. 1–22.

[7] ——, *Optimizing two-level preconditionings for the conjugate gradient method*, in Large-Scale Scientific Computing, Third International Conference, LSSC 2001, Sozopol, Bulgaria, June 6-10, 2001, Revised Papers, S. Margenov, J. Wasniewski, and P. Y. Yalamov, eds., vol. 2179, Springer, 2001, pp. 3–21.

[8] O. Axelsson and B. Polman, *A robust preconditioner based on algebraic substructuring and two-level grids*, in Robust Multigrid Methods (W. Hackbusch, ed.), Notes Numer. Fluid Mech., 23 (1988), pp. 1–26.

[9] S. Balay, J. Brown, K. Buschelman, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith, and H. Zhang, *PETSc Web page*, 2012. http://www.mcs.anl.gov/petsc (June 14th 2012).

[10] S. T. Barnard, L. M. Bernardo, and H. D. Simon, *An MPI implementation of the SPAI preconditioner on the T3E*, Int. J. High Perf. Comput. Appl., 13 (1999), pp. 107–123.

[11] S. T. Barnard and R. L. Clay, *A portable MPI implementation of the SPAI preconditioner in ISIS++*, Proceedings of the 8th SIAM Conference on Parallel Processing for Scientific Computing, Minneapolis, MN, (1997).

[12] S. T. Barnard and M. J. Grote, *A block version of the SPAI preconditioner*, in Proceedings of the 9th SIAM conference on Parallel Processing for Scientific Computing, San Antonio, TX, (1999).

[13] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. Van der Vorst, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition*, SIAM, 1994.

[14] N. Bell and M. Garland, *Cusp: Generic parallel algorithms for sparse matrix and graph computations*. http://cusp-library.googlecode.com (June 12th 2012), version 0.3.0, 2012.

[15] M. W. Benson and P. O. Frederickson, *Iterative solution of large sparse linear systems arising in certain multidimensional approximation problems*, Utilitas Mathematica, 22 (1982), pp. 127–140.

[16] M. Benzi, *Preconditioning techniques for large linear systems: A survey*, J. Comput. Phys., 182 (2002), pp. 418–477.

[17] M. Benzi, J. K. Cullum, and M. Tůma, *Robust approximate inverse preconditioning for the conjugate gradient method*, SIAM J. Sci. Comput., 22 (2000), pp. 1318–1332.

[18] M. Benzi, R. Kouhia, and M. Tůma, *Stabilized and block approximate inverse preconditioners for problems in solid and structural mechanics*, Comput. Methods Appl. Mech. Engrg., 190 (2001), pp. 6533–6554.

[19] M. Benzi, J. Marín, and M. Tůma, *A two-level parallel preconditioner based on sparse approximate inverses*, Iterative Methods in Scientific Computation IV, D. R. Kincaid and A. C. Elster, eds., IMACS Series in Computational and Applied Mathematics, 5 (New Brunswick, NJ, 1999), pp. 167–178.

[20] M. Benzi, C. D. Meyer, and M. Tůma, *A sparse approximate inverse preconditioner for the conjugate gradient method*, SIAM J. Sci. Comput., 17 (1996), pp. 1135–1149.

[21] M. Benzi and M. Tůma, *A sparse approximate inverse preconditioner for nonsymmetric linear systems*, SIAM J. Sci. Comput., 19 (1998), pp. 968–994.

[22] ——, *A comparative study of sparse approximate inverse preconditioners*, Appl. Numer. Math., 30 (1999), pp. 305–340.

[23] ——, *A parallel solver for large-scale Markov chains*, Appl. Numer. Math, 41 (2002), pp. 135–153.

[24] ——, *A robust incomplete factorization preconditioner for positive definite matrices*, Num. Lin. Alg. Appl., 10 (2003), pp. 385–400.

[25] Å. Björck, *Numerical Methods for Least Squares Problems*, SIAM, 1996.

[26] L. S. Blackford, J. Demmel, J. Dongarra, I. Duff, S. Hammarling, G. Henry, M. Heroux, L. Kaufman, A. Lumsdaine, A. Petitet, R. Pozo, K. Remington, and R. C. Whaley, *An updated set of basic linear algebra subprograms (BLAS)*, ACM Trans. Math. Soft., 28 (2002), pp. 135–151.

[27] R. F. Boisvert, R. Pozo, and K. A. Remington, *The Matrix Market Exchange Formats: Initial Design*, U. S. Department of Commerce, National Institute of Standards and Technology, Computing and Applied Mathematics Laboratory, NISTIR 5935, (1996), pp. 1–14.

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

[29] O. Bröker, M. J. Grote, C. Mayer, and A. Reusken, *Robust parallel smoothing for multigrid via sparse approximate inverses*, SIAM J. Sci. Comput., 23 (2001), pp. 1396–1417.

[30] J. Bräckle, *A Block Version of the Factorized Sparse Approximate Inverse Preconditioner (FSPAI)—Theory and Implementation*, Technische Universität München, Fakultät für Informatik, Fakultät für Mathematik, Diploma thesis, 2011.

[31] D. Calvetti, B. Lewis, and L. Reichel, *GMRES-type methods for inconsistent systems*, Lin. Alg. Appl., 316 (2000), pp. 157–169.

[32] J. L. Castellanos, S. Gómez, and V. Guerra, *The triangle method for finding the corner of the L-curve*, Appl. Numer. Math., 43 (2002), pp. 359–373.

[33] T. F. Chan, W. P. Tang, and W. L. Wan, *Wavelet sparse approximate inverse preconditioners*, BIT, 37 (1997), pp. 644–660.

[34] T. F. C. Chan and T. P. Mathew, *The interface probing technique in domain decomposition*, SIAM J. Matrix Anal. Appl., 13 (1992), pp. 212–238.

[35] K. Chen, *Matrix Preconditioning Techniques and Applications*, Cambridge University Press, 2005.

[36] E. Chow, *Parallel sparse approximate inverse (least-squares) preconditioner (ParaSails) home page.* https://computation.llnl.gov/casc/parasails/parasails.html (June 14th 2012).

[37] ——, *A priori sparsity patterns for parallel sparse approximate inverse preconditioners*, SIAM J. Sci. Comput., 21 (2000), pp. 1804–1822.

[38] ——, *Parallel implementation and practical use of sparse approximate inverse preconditioners with a priori sparsity patterns*, Int. J. High Perf. Comput. Appl., 15 (2001), pp. 56–74.

[39] J. Chung, J. G. Nagy, and D. P. O'Leary, *A weighted-GCV method for Lanczos-hybrid regularization*, Electronic Transactions on Numerical Analysis, 28 (2008), pp. 149–167.

[40] E. Cuthill and J. McKee, *Reducing the bandwidth of sparse symmetric matrices*, in Proceedings of the 24th national conference, ACM '69, New York, NY, USA, 1969, ACM, pp. 157–172.

[41] T. A. Davis, *Concise sparse package (CSparse) home page.* http://www.cise.ufl.edu/research/sparse/CSparse/ (May 24th 2012).

[42] ——, *Extended concise sparse package (CXSparse) home page.* http://www.cise.ufl.edu/research/sparse/CXSparse/ (May 24th 2012).

[43] ——, *Direct Methods for Sparse Linear Systems*, SIAM, 2006.

[44] T. A. Davis and Y. Hu, *The university of Florida sparse matrix collection*, ACM Trans. Math. Softw., 38 (2011), pp. 1:1–1:25.

[45] S. Demko, W. F. Moss, and P. W. Smith, *Decay rates for inverses of band matrices*, Math. Comp., 43 (1984), pp. 491–499.

[46] P. Deuflhard and A. Hohmann, *Numerical Analysis in Modern Scientific Computing. An Introduction*, Springer, 2nd ed., 2003.

[47] J. J. Dongarra, J. Du Croz, I. S. Duff, and S. Hammarling, *Algorithm 679: A set of level 3 basic linear algebra subprograms*, ACM Trans. Math. Soft., 16 (1990), pp. 18–28.

[48] J. J. Dongarra, I. S. Duff, D. C. Sorensen, and H. A. Van der Vorst, *Numerical Linear Algebra for High Performance Computers*, SIAM, 1998.

[49] G. Fourestey, W. Sawyer, R. Popescu, and C. Vanini, *SPAI preconditioners for HPC applications*, Technische Universität München, SCCS Colloquium, personal communication, (2011).

[50] E. Gamma, R. Helm, R. E. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.

[51] GCC, *The GNU compiler collection home page*. http://gcc.gnu.org/ (June 14th 2012).

[52] GNU gprof, *The GNU profiler home page*. http://www.cs.utah.edu/dept/old/texinfo/as/gprof_toc.html (June 14th 2012).

[53] G. H. Golub and C. F. Van Loan, *Matrix Computations*, John Hopkins University Press, 3rd ed., 1996.

[54] R. C. Gonzalez and E. W. Richard, *Digital Image Processing, DIP3/e—Book Images Downloads*, Prentice Hall, 2008. http://www.imageprocessingplace.com/DIP-3E/dip3e_book_images_downloads.htm (June 7th 2012).

[55] N. I. M. Gould and J. A. Scott, *Sparse approximate-inverse preconditioners using norm-minimization techniques*, SIAM J. Sci. Comput., 19 (1998), pp. 605–625.

[56] U. Grenander and G. Szegö, *Toeplitz Forms and Their Applications*, Chelsea Publishing, 2nd ed., 1984.

[57] M. J. Grote, S. T. Barnard, O. Bröker, and M. Hagemann, *Sparse approximate inverses (SPAI) home page*.
http://www.computational.unibas.ch/software/spai (June 14th 2012).

[58] M. J. Grote and T. Huckle, *Parallel preconditioning with sparse approximate inverses*, SIAM J. Sci. Comput., 18 (1997), pp. 838–853.

[59] M. Hagemann and O. Schenk, *Weighted matchings for preconditioning symmetric indefinite linear systems*, SIAM J. Sci. Comput, 28 (2006), pp. 403–420.

[60] M. Hanke, *Conjugate Gradient Type Methods for Ill-Posed Problems*, Longman Scientific & Technical, 1995.

[61] ——, *Iterative regularization techniques in image reconstruction*, in Proceedings of the Conference Mathematical Methods in Inverse Problems for Partial Differential Equations. Mt. Holyoke, Springer, 1998, pp. 35–52.

[62] M. Hanke and P. C. Hansen, *Regularization methods for large-scale problems*, Surv. Math. Ind., 3 (1993), pp. 253–315.

[63] M. Hanke and J. Nagy, *Restoration of atmospherically blurred images by symmetric indefinite conjugate gradient techniques*, Inverse Problems, 12 (1996), pp. 157–173.

[64] M. Hanke, J. G. Nagy, and R. J. Plemmons, *Preconditioned iterative regularization for ill-posed problems*, in Numerical Linear Algebra and Scientific Computing, L. Reichel, A. Ruttan, and R. S. Varga, eds., de Gruyter, Berlin, 1993, pp. 141–163.

[65] P. C. Hansen, *The truncated SVD as a method for regularization*, BIT, 27 (1987), pp. 534–553.

[66] ——, *The discrete picard condition for discrete ill-posed problems*, BIT, 30 (1990), pp. 658–672.

[67] ——, *Analysis of discrete ill-posed problems by means of the L-curve*, SIAM Review, 34 (1992), pp. 561–580.

[68] ——, *Regularization tools: A Matlab package for analysis and solution of discrete ill-posed problems*, Numer. Algorithms, 6 (1994), pp. 1–35.

[69] ——, *Rank-Deficient and Discrete Ill-posed Problems: Numerical Aspects of Linear Inversion*, SIAM, 1998.

[70] ——, *Discrete Inverse Problems: Insight and Algorithms*, SIAM, 2010.

[71] P. C. Hansen and T. K. Jensen, *Smoothing-norm preconditioning for regularizing minimum-residual methods*, SIAM J. Matrix Anal. Appl., 29 (2006), pp. 1–14.

[72] ——, *Noise propagation in regularizing iterations for image deblurring*, Electronic Transactions on Numerical Analysis, 31 (2008), pp. 204–220.

[73] P. C. Hansen, T. K. Jensen, and G. Rodriguez, *An adaptive pruning algorithm for the discrete L-curve criterion*, J. Comp. Appl. Math., 198 (2007), pp. 483–492.

[74] P. C. Hansen, M. E. Kilmer, and R. H. Kjeldsen, *Exploiting residual information in the parameter choice for discrete ill-posed problems*, BIT, 46 (2006), pp. 41–59.

[75] P. C. Hansen and D. P. O'Leary, *The use of the L-curve in the regularization of discrete ill-posed problems*, SIAM J. Sci. Comput., 14 (1993), pp. 1487–1503.

[76] M. R. Hestenes and E. Stiefel, *Methods of conjugate gradients for solving linear systems*, J. Res. Natl. Bur. Stand., 49 (1952), pp. 409–436.

[77] R. M. Holland, A. J. Wathen, and G. J. Shaw, *Sparse approximate inverses and target matrices*, SIAM J. Sci. Comput., 26 (2005), pp. 1000–1011.

[78] R. A. Horn and C. R. Johnson, *Matrix Analysis*, Cambridge University Press, 1985.

[79] T. Huckle, *Efficient computation of sparse approximate inverses*, Num. Lin. Alg. Appl., 5 (1998), pp. 57–71.

[80] ——, *Approximate sparsity patterns for the inverse of a matrix and preconditioning*, Appl. Numer. Math., 30 (1999), pp. 291–303.

[81] ——, *Factorized sparse approximate inverses for preconditioning and smoothing*, Selçuk Journal of Applied Mathematics, 1 (2000), pp. 63–70.

[82] ——, *Factorized sparse approximate inverses for preconditioning*, The Journal of Supercomputing, 25 (2003), pp. 109–117.

[83] ——, *Compact fourier analysis for designing multigrid methods*, SIAM J. Sci. Comput., 31 (2008), pp. 644–666.

[84] ——, *Constructing sparse approximate inverse block preconditioners*, 83rd Annual Meeting of the International Association of Applied Mathematics and Mechanics (GAMM 2012), personal communication, (2012).

[85] T. Huckle and A. Kallischko, *Frobenius norm minimization and probing for preconditioning*, Int. J. Comput. Math., 84 (2007), pp. 1225–1248.

[86] T. Huckle, A. Kallischko, A. Roy, M. Sedlacek, and T. Weinzierl, *An efficient parallel implementation of the MSPAI preconditioner*, Parallel Comput., 36 (2010), pp. 273–284.

[87] T. Huckle, A. Kallischko, and M. Sedlacek, *Modified sparse approximate inverse (MSPAI) home page*. http://www5.in.tum.de/wiki/index.php/MSPAI (June 14th 2012).

[88] T. Huckle and S. Schneider, *Numerische Methoden*, Springer, 2006.

[89] T. Huckle and M. Sedlacek, *Factorized sparse approximate inverse (FSPAI) home page*. http://www5.in.tum.de/wiki/index.php/FSPAI (June 14th 2012).

[90] ——, *Smoothing and regularization with modified sparse approximate inverses*, Journal of Electrical and Computer Engineering, Volume 2010 of Iterative Signal Processing in Communications, (2010), pp. 1–16. Article ID 930218.

[91] ——, *Data based regularization for discrete deconvolution problems*, BIT, (2011). Submitted (in review), preprint on http://www5.in.tum.de/persons/huckle/DataBased.pdf.

[92] ——, *SPAI (SParse Approximate Inverse)*, in Encyclopedia of Parallel Computing, Springer, 2011, pp. 1867–1870.

[93] ——, *Data based regularization matrices for the Tikhonov-Phillips regularization*, Proceedings of the 83rd Annual Meeting of the International Association of Applied Mathematics and Mechanics, PAMM Proc. Appl. Math. Mech., accepted, (2012).

[94] ——, *Tikhonov-Phillips regularization with operator dependent seminorms*, Numer. Algorithms, 60 (2012), pp. 339–353.

[95] HYPRE, *A library of high performance preconditioners, home page*. http://acts.nersc.gov/hypre (June 12th 2012).

[96] M. Jacobsen and T. K. Jensen, *Matlab object-oriented regularization tools (MOOre Tools) home page*, Technical University of Denmark, 2006. http://www2.imm.dtu.dk/~pch/MOOReTools/ (June 14th 2012).

[97] C. Janna and M. Ferronato, *Adaptive pattern research for block FSAI preconditioning*, SIAM J. Sci. Comput., 33 (2011), pp. 3357–3380.

[98] C. Janna, M. Ferronato, and G. Gambolati, *A block FSAI-ILU parallel preconditioner for symmetric positive definite linear systems*, SIAM J. Sci. Comput., 32 (2010), pp. 2468–2484.

[99] A. Kallischko, *Modified Sparse Approximate Inverses (MSPAI) for Parallel Preconditioning*, PhD thesis, Technische Universität München, 2008.

[100] C. Kanzow, *Numerik linearer Gleichungssysteme: Direkte und iterative Verfahren*, Springer, 2005.

[101] I. E. Kaporin, *New convergence results and preconditioning strategies for the conjugate gradient method*, Num. Lin. Alg. Appl., 1 (1994), pp. 179–210.

[102] D. E. Knuth, *The Art of Computer Programming, Volume 3: Sorting and Searching*, Addison-Wesley, 2nd ed., 1998.

[103] L. Yu. Kolotilina, A. A. Nikishin, and A. Yu. Yeremin, *Factorized sparse approximate inverse preconditionings. IV: simple approaches to rising efficiency*, Num. Lin. Alg. Appl., 6 (1999), pp. 515–531.

[104] ——, *Factorized sparse approximate inverse preconditionings. III. Iterative construction of preconditioners*, J. Math. Sciences, 101 (2000), pp. 3237–3254.

[105] L. Yu. Kolotilina and A. Yu. Yeremin, *Factorized sparse approximate inverse preconditionings I. Theory*, SIAM J. Matrix Anal. Appl., 14 (1993), pp. 45–58.

[106] ——, *Factorized sparse approximate inverse preconditionings II: solution of 3D FE systems on massively parallel computers*, Int. J. High Speed Comput., 7 (1995), pp. 191–215.

[107] B. S. Lazarov and O. Sigmund, *Factorized parallel preconditioner for the saddle point problem*, Int. J. Numer. Meth. Biomed. Engng, 27 (2009), pp. 1398–1410.

[108] Matlab, *v7.11.0 (R2010b)*, The MathWorks® Inc., Natick, MA, USA, 2010. http://www.mathworks.com/ (June 12th 2012).

[109] A. Meister, *Numerik linearer Gleichungssysteme*, Vieweg, 3rd ed., 2008.

[110] T. Minka, *The Lightspeed* Matlab *Toolbox v2.6 home page*. http://research.microsoft.com/en-us/um/people/minka/software/lightspeed/ (April 11th 2012).

[111] MPI Forum, *Message passing interface (MPI) forum home page*. http://www.mpi-forum.org/ (June 8th 2012).

[112] J. G. Nagy, *Accelerating convergence of iterative image restoration algorithms*, in Proceedings of the Advanced Maui Optical and Space Surveillance Technologies (AMOS) Conference, Maui, Hawaii, S. Ryan and the Maui Economic Development Board, eds., 2007, pp. 1–10.

[113] J. G. Nagy and D. P. O'Leary, *Restoring images degraded by spatially variant blur*, SIAM J. Sci. Comput., 19 (1998), pp. 1063–1082.

[114] J. G. Nagy, K. Palmer, and L. Perrone, *Iterative methods for image deblurring: A Matlab object-oriented approach*, Numer. Algorithms, 36 (2004), pp. 73–93.

[115] J. G. Nagy, R. J. Plemmons, and T. C. Torgersen, *Iterative image restoration using approximate inverse preconditioning*, IEEE Transactions on Image Processing, 5 (1996), pp. 1151–1162.

[116] NIST, *Matrix Market home page*. A service of the Mathematical and Computational Sciences Division of the Information Technology Laboratory of the National Institute of Standards and Technology, http://math.nist.gov/MatrixMarket/.

[117] S. Noschese and L. Reichel, *Inverse problems for regularization matrices*, Numer. Algorithms, in press, (2012), pp. 1–11.

[118] C. C. Paige and M. A. Saunders, *Solution of sparse indefinite systems of linear equations*, SIAM J. Numer. Anal., 12 (1975), pp. 617–629.

[119] ——, *LSQR: an algorithm for sparse linear equations and sparse least squares*, ACM Trans. Math. Softw., 8 (1982), pp. 43–71.

[120] D. L. Phillips, *A technique for the numerical solution of certain integral equations of the first kind*, J. Assoc. Comp. Mach., 9 (1962), pp. 84–97.

[121] A. Quarteroni, R. Sacco, and F. Saleri, *Numerical Mathematics*, Springer, 2nd ed., 2007.

[122] G. Rodriguez and D. Theis, *An algorithm for estimating the optimal regularization parameter by the L-curve*, Rendiconti di Matematica, 25 (2005), pp. 69–84.

[123] Y. Saad, *Iterative Methods for Sparse Linear Systems*, SIAM, 2nd ed., 2003.

[124] Y. Saad and H. A. van der Vorst, *Iterative solution of linear systems in the 20th century*, J. Comput. Appl. Math., 123 (2000), pp. 1–33.

[125] Y. Saad and J. Zhang, *BILUM: block versions of multielimination and multilevel ILU preconditioner for general sparse linear systems*, SIAM J. Sci. Comput., 20 (1997), pp. 2103–2121.

[126] W. Sawyer, G. Fourestey, R. Popescu, and C. Vanini, *A GPU implementation of the SPAI preconditioner for cardiovascular simulation*, 83rd Annual Meeting of the International Association of Applied Mathematics and Mechanics (GAMM 2012), personal communication, (2012).

[127] R. Schaback and H. Werner, *Numerische Mathematik*, Springer, 4th ed., 1992.

[128] M. Sedlacek, *Effiziente parallele Implementierung des MSPAI Präkonditionierers unter Verwendung von Caching-Strategien und QR-Updates*, Diploma thesis, Fakultät für Informatik, Technische Universität München, 2008.

[129] J. Seward, N. Nethercode, and J. Weidendorfer, *Valgrind 3.3—Advanced Debugging and Profiling for GNU/Linux Applications*, Network Theory Limited, 2008. Valgrind home page at http://valgrind.org/ (June 8th 2012).

[130] C. Siefert and E. de Sturler, *Probing methods for saddle-point problems*, Electronic Transactions on Numerical Analysis, 22 (2006), pp. 163–183.

[131] B. Stroustrup, *The C++ Programming Language (Special Edition)*, Addison-Wesley, 3rd ed., 2000.

[132] W.-P. Tang, *Toward an effective sparse approximate inverse preconditioner*, SIAM J. Matrix Anal. Appl., 20 (1999), pp. 970–986.

[133] W.-P. Tang and W. L. Wan, *Sparse approximate inverse smoother for multigrid*, SIAM J. Matrix Anal. Appl., 21 (2000), pp. 1236–1252.

[134] J. D. Tebbens and M. Tůma, *Preconditioner updates for solving sequences of linear systems in matrix-free environment*, Num. Lin. Alg. Appl., 17 (2010), pp. 997–1019.

[135] A. N. Tikhonov, *Solution of incorrectly formulated problems and regularization method*, Sov. Math. Dokl., 4 (1963), pp. 1035–1038. English translation of Dokl. Akad. Nauk. SSSR 151 (1963), 501-504.

[136] Tom'sHardware, *Intel® Core™ i7-990X SiSoftware Sandra 2011 processor arithmetic performance test*, Bestofmedia Group, 2012. http://www.tomshardware.com/reviews/core-i7-990x-extreme-edition-gulftown,2874-6.html (June 14th 2012).

[137] L. N. Trefethen and D. Bau III, *Numerical Linear Algebra*, SIAM, 1997.

[138] U. Trottenberg, C. Oosterlee, and K. Schüller, *Multigrid*, Academic Press, 2001.

[139] H. A. Van der Vorst, *Bi-CGSTAB: a fast and smoothly converging variant of bi-cg for the solution of non-symmetric linear systems*, SIAM J. Sci. Stat. Comp., 13 (1992), pp. 631–644.

[140] K. Wang, O. Lawlor, and L. V. Kale, *The nonsingularity of sparse approximate inverse preconditioning and its performance based on processor virtualization*, tech. rep., Department of Computer Science, University of Illinois at Urbana-Champaign, 2005.

[141] K. Wang and J. Zhang, *MSP: a class of parallel multistep successive sparse approximate inverse preconditioning strategies*, SIAM J. Sci. Comput., 24 (2003), pp. 1141–1156.

[142] K. Wang, J. Zhang, and C. Shen, *A class of parallel multilevel sparse approximate inverse preconditioners for sparse linear systems*, J. Scalable Computing: Practice and Experience, 7 (2006), pp. 93–106.

[143] J. Weidendorfer, *Kcachegrind, a profile data visualization tool, home page.* http://kcachegrind.sourceforge.net/html/Home.html (June 14th 2012).

[144] A. Yu. Yeremin and A. A. Nikishin, *Factorized-sparse-approximate-inverse preconditionings of linear systems with unsymmetric matrices*, J. Math. Sciences, 121 (2004), pp. 2448–2457.

[145] J. Zhang, *Sparse approximate inverse and multilevel block ILU preconditioning techniques for general sparse matrices*, Appl. Numer. Math., 35 (2000), pp. 67–86.

Index