Technische Universität München
Zentrum Mathematik

Modified Sparse Approximate Inverses (MSPAI) for Parallel Preconditioning

Alexander Kallischko

Vollständiger Abdruck der von der Fakultät für Mathematik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Die Dissertation wurde am 15.11.2007 bei der Technischen Universität eingereicht und durch die Fakultät für Mathematik am 18.2.2008 angenommen.

Abstract

The solution of large sparse and ill-conditioned systems of linear equations is a central task in numerical linear algebra. Such systems arise from many applications like the discretization of partial differential equations or image restoration. Herefore, Gaussian elimination or other classical direct solvers can not be used since the dimension of the underlying coefficient matrices is too large and Gaussian elimination is an $\mathcal{O}\left(n^3\right)$ algorithm. Iterative solvers techniques are an effective remedy for this problem. They allow to exploit sparsity, bandedness, or block structures, and they can be parallelized much easier. However, due to the matrix being ill-conditioned, convergence becomes very slow or even not be guaranteed at all. Therefore, we have to employ a preconditioner.

The sparse approximate inverse (SPAI) preconditioner is based on Frobenius norm minimization. It is a well-established preconditioner, since it is robust, flexible, and inherently parallel. Moreover, SPAI captures meaningful sparsity patterns automatically. The derivation of pattern update criteria for complex valued systems of equations, the reformulation and extension of a nonsingularity result, and the investigation of SPAI's regularization qualities are our first original contributions to research. Furthermore, we investigate the effect of a fill-in-reducing graph algorithm for pattern updates in FSPAI, which is the factorized variant of SPAI. Additionally, a theoretical result for the SPAI and FSPAI of M-matrices is presented.

As the main contribution to ongoing research, we develop the new modified sparse approximate inverse preconditioner MSPAI. On the one hand, this is a generalization of SPAI, because we extend SPAI to target form. This allows us also to compute explicit matrix approximations in either a factorized or unfactorized form. On the other hand, this extension enables us to add some further, possibly dense, rows to the underlying matrices, which are then also taken into account during the computation. These additional constraints for the Frobenius norm minimization generalize the idea of classical probing techniques, which are restricted to explicit approximations and very simple probing constraints. By a weighting factor, we force the resulting preconditioner to be optimal on certain probing subspaces represented by the additional rows. For instance, the vector of all ones leads to preservation of row sums, which is quite important in many applications as it reflects certain conservation laws. Therefore, MSPAI probing can also be seen as a generalization to the well-known modified preconditioners such as modified incomplete LU or modified incomplete Cholesky. Furthermore, we can improve Schur complement approximations, which are the original application area of classical probing. Given factorized preconditioners can also be improved relative to a probing subspace. For symmetric linear systems, new symmetrization techniques are introduced. The effectiveness of MSPAI probing is proven by many numerical examples such as matrices arising from domain decomposition methods and Stokes problems.

Besides the theoretical development of MSPAI probing, an efficient implementation is presented. We investigate the use of a linear algebra library for sparse least squares problems in combination with QR updates and compare it to standard dense methods. Furthermore, we implement a caching strategy which helps to avoid redundant QR factorizations especially for the case of highly structured matrices. The support for maximum sparsity patterns rounds up our implementation. Various tests reveal significantly lower runtimes compared to the original implementation of SPAI.

Zusammenfassung

Die Präkonditionierung großer, dünnbesetzter und schlecht konditionierter linearer Gleichungssysteme ist eine der zentralen Aufgaben der numerischen linearen Algebra. Solche Systeme treten in vielen Anwendungen wie z.B. der Diskretisierung partieller Differentialgleichungen oder der Bildrekonstruktion auf. Gauß Elimination oder andere klassische direkte Verfahren für allgemeine Gleichungssysteme sind dafür nicht mehr geeignet, weil die Koeffizientenmatrizen viel zu hohe Dimensionen erreichen und z.B. der Gauß Algorithmus eine Komplexität von $\mathscr{O}\left(n^3\right)$ aufweist. Abhilfe schaffen hier iterative Löser. Diese ermöglichen es, spezielle Strukturen in Matrizen wie Dünnbesetztheit und Band- und Blockstrukturen effizient auszunutzen. Außerdem ist die Parallelisierung unproblematischer. Trotzdem kann Konvergenz bei zu schlechter Kondition der Koeffizientenmatrix nur sehr langsam oder auch gar nicht erreichbar sein. In diesem Fall bietet sich die Möglichkeit der Präkonditionierung.

Der SPAI (sparse approximate inverse) Präkonditionierer berechnet dünnbesetzte Näherungen der Inversen einer Matrix und basiert auf Frobenius-Norm-Minimierung. Durch seine Robustheit, Flexibilität und intrinsische Parallelität stellt er ein gängiges Verfahren zur Präkonditionierung großer linearer Gleichungssysteme dar. Außerdem verfügt SPAI über die Möglichkeit, eine vorgegebene Besetztheitsstruktur (sparsity pattern) automatisch um sinnvolle Einträge zu erweitern, um die Approximation der Inversen zu verbessern. Die ersten Beiträge dieser Arbeit zur aktuellen Forschung behandeln die Herleitung von Kriterien zur Erweiterung des Besetztheitsmusters im Fall komplexwertiger Gleichungssysteme, die Neuformulierung und Erweiterung eines Regularitätsbeweises, sowie die Untersuchung, inwiefern sich SPAI als Regularisierungspräkonditionierer bei Bildrekonstruktionsproblemen eignet. Weiterhin wird die Auswirkung eines fill-in-reduzierenden Graphenalgorithmus auf die Erweiterungsschritte im Besetztheitsmuster des FSPAI Präkonditionierers untersucht, der faktorisierten Variante von SPAI für symmetrisch positiv definite Matrizen. Außerdem wird eine theoretische Aussage über SPAI und FSPAI für M-Matrizen bewiesen.

Der größte Beitrag dieser Arbeit besteht aus der Entwicklung des MSPAI (modified sparse approximate inverse) Präkonditionierers. Zum einen stellt MSPAI durch die Erweiterung auf Targetform eine Verallgemeinerung von SPAI dar. Dadurch lassen sich sowohl inverse als auch explizite Approximation, sowohl in faktorisierter, als auch unfaktorisierter Form berechnen. Andererseits können durch diese Erweiterung weitere (womöglich dichtbesetzte) Zeilen an die zugrunde liegenden Matrizen angehängt werden, die dann ebenfalls bei der Berechnung des MSPAI mit berücksichtigt werden. Diese zusätzlichen Nebenbedingungen an die Frobenius-Norm-Minimierung verallgemeinern auch das Prinzip des klassischen Probings. Mittels eines Gewichtsfaktors wird erreicht, dass der entstehende Präkonditionerer auf bestimmten Unterräumen optimal agiert. Beispielsweise führt der Vektor, der dicht mit 1 besetzt ist, zur Erhaltung der Zeilensummen, was in vielen Anwendungen sehr wichtig ist, spiegelt es schließlich diverse Erhaltungssätze wider. In dieser Hinsicht kann MSPAI auch als Verallgemeinerung der Klasse der modifizierten Präkonditionierer wie z.B. der modifizierten unvollständigen LU-Zerlegung angesehen werden. Zusätzlich lassen sich durch MSPAI auch Näherungen von Schur Komplementen verbessern, worin die ursprüngliche Anwendung des klassischen Probings besteht. Auch von anderen Methoden erzeugte faktorisierte Approximationen lassen sich mittels MSPAI und Probing-Nebenbedingungen in ihrer Qualität verbessen. Für lineare Gleichungssysteme mit symmetrischer Koeffizientenmatrix werden überdies neue Techniken zur Symmetrisierung eingeführt. Die Wirksamkeit von MSPAI wird an einigen numerischen Beispielen wie Gebietszerlegung und Stokes-Problemen nachgewiesen.

Neben der theoretischen Entwicklung des MSPAI thematisiert diese Arbeit auch eingehend dessen effiziente Implementierung. Zunächst werden dünnbesetzte Verfahren zur Lösung von Least-Squares-Problemen in Verbindung mit QR-Updates untersucht und mit den bisherigen Standardmethoden verglichen. Außerdem wird eine Caching-Strategie eingeführt, die dabei hilft, redundante QR-Zerlegungen im Fall hochstrukturierter Matrizen einzusparen. Schließlich rundet die Unterstützung maximaler Oberpattern für die Dünnbesetztheitsstrukturen die Implementierung ab. Laufzeittests beweisen die Effektivität dieser Maßnahmen in Form von deutlich reduzierten Laufzeiten im Vergleich mit der aktuellen Standardimplementierung von SPAI.

Contents

Introduction

Mathematicians are not faced with the problem of solving systems of linear equations for only the last two centuries — this task is much older and, to our knowledge, the first solution technique goes back to the third century. Liu Hui, e.g. ($\approx$ 220 AD to $\approx$ 280 AD) was a chinese mathematician who tried to solve mathematical problems in everyday life. His main contribution are comments to "The nine books on Arithmetic" [70], which he published in 263 AD. There, he suggested 3.14 as an approximation for π, derived formulas for the volumes of tetrahedrons, pyramids, cones, and other geometric primitives. He was also the first to describe a method for solving systems of linear equations. It is the method today known as Gaussian elimination.

Without computers or calculators, the solution of large systems still was impossible. In the 1880s, it took about 7 weeks to solve a linear system $Ax = b$ of dimension $n = 29$. Hereby, people had to use logarithm tables. In 1951, it still took over 3 hours to solve a system of dimension 17 [56]. On an Intel Centrino with 1600 MHz, MATLAB nowadays needs 0.6 seconds to compute the solution for $n = 1000$. Despite this enormous raise in computation power, Gaussian elimination is only suitable for systems up to $n \approx 10000$, since it is an $\mathcal{O}\left(n^3\right)$ algorithm. In order to solve much larger systems, iterative methods were developed. They can also take into account sparsity or certain band or block structures, which makes them usually highly efficient and the methods of choice today. These iterative approaches enable us to solve the large linear systems which arise in many numerical applications such as the discretization of partial differential equations or image restoration problems.

Even these highly elaborate iterative techniques can be prohibitively slow if the coefficient matrix A of the system is ill-conditioned. Then, either convergence can not be achieved at all or a huge number of steps becomes necessary. In order to accelerate the solution process, preconditioning techniques have been investigated since the early 1970s. Hereby, the linear system is transformed into an equivalent one with much lower condition number leading to considerably lower iteration counts and thus reduced solution time. In Scientific Computing, it is common knowledge that the choice of a good preconditioner is even more important than the actual choice of the iterative solver.

Many preconditioning techniques were developed so far [25]. Among the most robust and flexible ones is the class based on Frobenius norm minimization. Extending this approach, Grote and Huckle [48] developed the sparse approximate inverse (SPAI) preconditioner which is able to update a given start sparsity pattern automatically. Furthermore, SPAI is inherently parallel, since its computation splits up into independent least squares problems, one for each column. There is also the factorized variant FSPAI [62] for symmetric positive definite matrices. Another important group of preconditioners are the modified incomplete factorizations [4, 49]. They yield preconditioners which preserve the row sums, i.e. the action on the vector of all ones. In many applications, this leads to a significantly improved convergence behavior. Another type of preconditioners, which are constructed such that they are optimal with respect to some given subspace, is classical probing [24]. However, modified factorizations and classical probing are very limited in the choice of probing subspaces. Moreover, they are difficult to parallelize. Our main contribution to ongoing research is therefore the extension of SPAI to more general approximations and the development of the modified sparse approximate inverse (MSPAI) preconditioner. MSPAI is still based on Frobenius norm minimization, but allows us to add any number and any type of probing information to the preconditioner. Hence, we can optimize it subject to any subspace which seems adequate for the actual system of equations. Inherent parallelism and adaptivity of the sparsity pattern are preserved. Besides many numerical test results proving MSPAI's effectiveness in terms of faster convergence, we also present an efficient implementation of MSPAI.

Chapter 1 will provide the numerical basics to face the topic. We start off with the classic Gaussian elimination, the solution of triangular systems, and the Cholesky factorization for symmetric positive definite systems. We mention these methods as they are also the fundamentals behind some frequently employed preconditioning techniques such as the incomplete LU factorization. The second section gives the mathematical background for the term *condition number* of a linear system of equations. The matrices we are focussing on in this thesis usually have a rather low number of nonzero entries compared to their dimension. Such matrices are called *sparse matrices*, and we present some standard techniques for storing and using them efficiently. In this context, we also explain the use of graph algorithms operating on the matrix's adjacency graph related to the sparsity structure. We lay special focus on the approximate minimum degree algorithm and present some new results based on this approach in Chapter 2. Afterwards, the emphasis will be on iterative solvers. We explain both a few basic standard methods and give insight into Krylov subspace approaches. The most well-known representative of this class is the conjugate gradient method. The last section describes preconditioning techniques starting with forward preconditioners such as incomplete LU factorization. This is followed by a short overview about inverse preconditioners, which directly leads to the main topic of this thesis.

Chapter 2 is devoted to the sparse approximate inverse preconditioner (SPAI) by Grote and Huckle [48] and its factorized variant FSPAI [62] for symmetric positive definite matrices. The underlying Frobenius norm minimization splits up to the solution of independent least squares problems, one for each column in the preconditioner. This property also gives SPAI a great inherent parallelism which is nowadays a topic of constantly increasing importance. We also explain in detail the use of the QR decomposition to solve the least squares problems. After the discussion of the meaningful choice of sparsity patterns for the SPAI preconditioner including the idea of maximum sparsity patterns, we also provide some theoretical properties of SPAI. Furthermore, SPAI can not only be computed for a fixed sparsity pattern, but it can also start with an arbitrary sparsity structure and then automatically enlarge it by identifying and adding the most appropriate entries reducing the error. The same holds for FSPAI, where we compute an approximation to the Cholesky factor of the inverse. This is also done column-wise and hence completely in parallel. Like SPAI, FSPAI can perform pattern updates and thereby increase its approximation quality automatically. This chapter already contains some pieces of original research. We derive a formulation of the SPAI pattern update criteria for complex valued systems of equations. Up to now, this was only done for the real valued case. We give a more compact reformulation of a nonsingularity result for SPAI and extend it. Furthermore, we investigate SPAI's regularization properties in image restoration. To our knowledge, this is also a new field of application. For FSPAI pattern updates, we investigate the effect of an approximate minimum degree preordering.

Finally, we state a theorem considering SPAI and FSPAI for M-matrices, a class of matrices frequently arising from the discretization of partial differential equations.

Chapter 3 starts off with the description of the classical probing technique and the group of modified preconditioners. Both have in common that they do not only yield explicit approximations to the system matrix, but also include additional information in the preconditioner, hereafter referred to as *probing information.* Typically, probing and modified-type preconditioners are restricted to the conservation of row sums in the preconditioner. Moreover, classical probing can only compute very sparse approximations with simple structures, since people have to invert them in every iteration of the iterative solver. Additionally, these methods are hard to parallelize. To overcome these restrictions and drawbacks, we generalize, in the third section, the class of Frobenius norm minimization based preconditioners to target form [57]. Furthermore, we extend the idea from $n \times n$ to rectangular $m \times n$ matrices. Thus, we gain inherent algorithmic parallelism due to the Frobenius norm minimization. Moreover, we can add several rows to the input matrices which contain the probing information in form of probing vectors. Here is the next gain in generality: we can add as many probing vectors of any type we consider as meaningful for the actual application. We also benefit from the other advantages, which we have already seen for SPAI. The computation is not restricted to a special sparsity pattern, and we can perform automatic pattern updates. Since it is based on modified preconditioners, we call this formulation *modified sparse approximate inverse* (*MSPAI*). With this, we can compute inverse approximations like SPAI, adding probing information. However, we can also get direct approximations of a matrix, for instance a sparser one which acts on certain subspaces as the original matrix does. For both inverse or direct approximations, we also derive formulations of MSPAI for factorized approximations. This allows improving any given factorized preconditioner by adding probing information. Another application is the MSPAI probing of Schur complements, which is the application domain classical probing was designed for. Which concrete probing vectors can or should be chosen for MSPAI is discussed in Section 3.4. For this purpose, we provide a theoretical basis, i.e. a purely structural condition considering probing vectors' sparsity structure and the sparsity structure of the preconditioner. The final section in this chapter derives several symmetrization techniques, since matrices resulting from Frobenius norm minimization processes are usually unsymmetric. However, for symmetric and even symmetric positive definite input matrices, we want to have a preconditioner reflecting these properties. We present one method based on Frobenius norm minimization and another approach involving two consecutive iteration steps. A symmetrization technique for factorized preconditioners concludes the chapter.

Chapter 4 gives insight to a topic which is crucial for every numerical algorithm: its efficient implementation. MSPAI can only be employed and tested efficiently if we provide a fast code. We review the use of sparse methods for the solution of least squares problems and combine them with the idea of QR updates [17, 46]. The latter arise in SPAI-type computations involving pattern updates. An experiment will also evaluate the potential of single precision QR methods for our purpose. Section 4.2 introduces the idea of a caching algorithm for MSPAI computation. It is based on the observation that in highly structured matrices, we perform many redundant factorizations. With a caching strategy holding already computed factorizations, we try to exploit these redundancies to reduce the overall computation time. After that, we investigate the benefit from a maximum sparsity pattern [61] in parallel computing environments — an idea which was already discussed theoretically in Chapter 2. A profound comparison of the current standard SPAI implementation SPAI 3.2 and our MSPAI 1.0 code rounds up the chapter. For each single improvement, we provide detailed runtime test results. To our knowledge, the use of sparse methods, the combination with QR updates, the caching approach, and the maximum sparsity pattern were not investigated before for SPAI-like preconditioners.

Chapter 5 contains numerical examples, whereas the previous chapter focused on efficient implementation. First, we demonstrate the application of sparsification steps in an example application arising from statics' simulation. The main system of equations there is dense. Hence, in order to compute an FSPAI preconditioner for it, we determine a thinner sparsity pattern. The second section gives more examples for the effect of approximate minimum degree graph algorithms on FSPAI pattern updates. We mention positive semidefinite matrices arising from Navier-Stokes discretizations. The concrete application of all MSPAI variants and symmetrization techniques is presented in Section 5.3. We start with the employment of all variants in domain decomposition problems. Then, we investigate MSPAI's use in preconditioning Stokes problems. Sparse spectrally equivalent approximations of dense matrices are also used for several tests. A comparison of incomplete modified factorizations and factorized MSPAI probing is also presented. Finally, we demonstrate the effect of consecutive probing steps.

A conclusion summarizing the results of this thesis closes the discussion.

Chapter 1

Numerical Solution of Systems of Linear Equations

This introductory chapter is devoted to both the classical direct methods for solving systems of linear equations and also more recent iterative algorithms. We start off with the probably most common algorithm, the Gaussian algorithm and its specialization for symmetric positive definite (*spd*) problems, the Cholesky decomposition. We mention these techniques, because methods such as forward and backward substitution or the Gaussian algorithm itself build the basis for some preconditioning approaches as well.

After that, we will take a closer look at the theoretical properties of linear systems and thus, the problems which may emerge when solving them. More precisely, we will focus on the term *condition number* in the context of matrices, which correspond to the linear systems.

With these basics in mind, we continue with an introduction to sparse matrices including common data structures and basic algorithms which exploit the sparsity. For larger problem sizes, direct methods are not efficient. Iterative methods are used, instead. We will explain some basic iterative techniques and also the most commonly used Krylov subspace solvers.

If the system matrix is very ill-conditioned, the convergence of iterative solvers can be prohibitively slow or even not be guaranteed at all. Preconditioning techniques often provide a widely used remedy for this problem. In the last section, we focus on a few well-known methods and conclude this chapter with a brief review of inverse-type preconditioners. This leads to the main topic of this work, namely the preconditioning of large sparse systems of equations using the sparse approximate inverse preconditioner SPAI.

Starting point of all the approaches presented here is the objective to find a solution of the system of n linear equations

$$\begin{array}{ccccccccc} a_{11}x_1 & + & a_{12}x_2 & + & \dots & + & a_{1n}x_n & = & b_1 \\ a_{21}x_1 & + & a_{22}x_2 & + & \dots & + & a_{2n}x_n & = & b_2 \\ \vdots & & \vdots & & \vdots & & \vdots & & \vdots \\ a_{n1}x_1 & + & a_{n2}x_2 & + & \dots & + & a_{nn}x_n & = & b_n \end{array} \tag{1.1}$$

as it arises for instance in the discretization of partial differential equations or image restoration problems. For the sake of a more compact notation, we will further denote (1.1) as

$$Ax = b, \tag{1.2}$$

where $A = (a_{ij})_{i,j=1,\dots,n} \in \mathbb{R}^{n\times n}$ is the coefficient matrix, the vector $b = (b_i)_{i=1,\dots,n} \in \mathbb{R}^n$ the right-hand side, and $x = (x_i)_{i=1,\dots,n} \in \mathbb{R}^n$ the solution we want to compute. Throughout the whole thesis, we will use MATLAB-style colon notation

$$y(k:\ell) \qquad \text{or} \qquad y_{k:\ell}$$

for vectors when we refer to the components y_i of a vector $y \in \mathbb{R}^n$ corresponding to the indices $i = k,\dots,\ell$, and

$$A(:,k)$$

for matrices, if we want to address the k-th column of the matrix A, respectively.

1.1 Gauss Algorithm and LU Factorization

Gaussian elimination is the most efficient method among the class of classical solution methods for systems of linear equations [35]. The idea is to transform the original general system (1.2) into upper triangular form. We can solve systems of equations with triangular coefficient matrix quite easily, as we will see in the first section. The LU decomposition allows the efficient solution of the system with many different right-hand sides. The presentation here follows mainly [35, 46], but can also be found in [5, 18, 51, 93]. For the theoretical background in linear algebra, we refer to [41].

1.1.1 Triangular Systems

In the special case of A having an either lower or upper triangular pattern, the solution process is much easier than in the general dense and unstructured form. By *pattern* we refer to the set of row and column indices, where A has entries which are nonzero. Therefore, A is called *lower triangular matrix*, if

$$A = \begin{pmatrix} \times & 0 & \cdots & 0 \\ \times & \times & \ddots & \vdots \\ \vdots & \ddots & \times & 0 \\ \times & \cdots & \times & \times \end{pmatrix} \qquad \text{i.e.} \qquad a_{ij} \begin{cases} = 0 & \text{if} \quad i < j \\ \neq 0 & \text{if} \quad i \geq j \end{cases},$$

and analogously *upper triangular matrix*, if

$$A = \begin{pmatrix} \times & \times & \cdots & \times \\ 0 & \times & \ddots & \vdots \\ \vdots & \ddots & \times & \times \\ 0 & \cdots & 0 & \times \end{pmatrix} \qquad \text{i.e.} \qquad a_{ij} \begin{cases} = 0 & \text{if} \quad i > j \\ \neq 0 & \text{if} \quad i \leq j \end{cases}.$$

A solution to a system such as (1.2) with lower triangular matrix A can be obtained quite easily using

$$x_i = \frac{1}{a_{ii}} \left(b_i - \sum_{k=1}^{i-1} a_{ik} x_k \right), \qquad i = 1,\dots,n, \tag{1.3}$$

starting with $x_1 = \frac{b_1}{a_{11}}$ and then computing $x_2, \dots, x_n$ one after another. This is called *forward substitution*. Similarly, one obtains the components of the solution vector x in the upper triangular case, but now in different order, beginning with $x_n = \frac{b_n}{a_{nn}}$. This *backward substitution* process continues with calculating the components $x_{n-1}, \dots, x_1$ via

$$x_i = \frac{1}{a_{ii}} \left(b_i - \sum_{k=i+1}^{n} a_{ik} x_k \right), \qquad i = n-1, \dots, 1.$$

The computational effort of both versions is

$$\mathcal{O}\left(n^2\right)$$

floating point operations (*flops*) and thus entails about the same amount of work like a matrix-vector product.

Forward and backward substitution are not only used in the context of direct solution of general dense systems of equations, but also in iterative solvers. If we have a factorized preconditioner of triangular form which approximates A explicitly, we apply it in the iterative solver using forward and backward substitution.

1.1.2 LU Decomposition by Gaussian Elimination

In order to use the substitution methods of the previous section, we first have to compute triangular factors of A. This transformation is performed column-wise, starting with matrix $A = A^{(1)}$, which is transformed into matrix $A^{(k)}$ $(k = 2, \dots, n-1)$ and so on until matrix $A^{(n)}$ has the desired upper triangular form. In the k-th elimination step, we transform the subdiagonal elements in column k to a vector of zeros by subtracting an appropriate multiple of row $k-1$ to the k-th row. Following [35], we can denote these elimination steps by multiplication of matrix $A^{(k)}$ from the left with a matrix

$$L_k = \begin{pmatrix} 1 & & & & \\ & \ddots & & & \\ & & 1 & & \\ & & -\ell_{k+1,k} & \ddots & \\ & & \vdots & & \ddots \\ & & -\ell_{n,k} & & & 1 \end{pmatrix},$$

where $\ell_{ik} = \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}}$, $i = k+1, \dots, n$. This matrix is called *Frobenius matrix*. Its inverse is given simply by changing the signs of the values ℓ_{ik}. So we have

$$L_{n-1} \cdot \ldots \cdot L_1 A =: U \qquad \Longleftrightarrow \qquad A = \underbrace{L_1^{-1} \cdot \ldots \cdot L_{n-1}^{-1}}_{=:L} U$$

with lower triangular matrix L and upper triangular matrix U. L has the special property that it has a main diagonal of all ones. After setting $y = Ux$, we obtain the solution by solving $Ly = b$ using forward substitution and then solving $Ux = y$ by backward substitution.

The dominant part of the solution process is the Gaussian elimination with applying the matrices L_k, $k = 1, \dots, n-1$ to A and then forming the lower triangular matrix L. The computational costs for that are

$$\#\text{flops} \doteq \frac{2}{3}n^3 = \mathscr{O}\left(n^3\right)$$

and thus outweigh the costs of the two substitution steps. Once having computed the LU decomposition of A, we can cheaply compute the solution for different right-hand sides b.

Pivoting

For nonsingular matrices A, the LU decomposition exists and is unique (for proofs see e.g. [35, 41, 46]). Nevertheless, there exist quite simple counterexamples with nonsingular A, where the Gaussian elimination fails. Consider the matrix

$$A = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix},$$

which has $\det(A) = -1 \neq 0$, but L_1 cannot be formed due to $\ell_{11} = \frac{a_{11}}{a_{11}}$ and $a_{11} = 0$. The element $a_{kk}^{(k)}$ in general is called *pivot element*. Gaussian elimination fails if the pivot element is zero, and in practical numerical implementations even if it is "too small". But the situation changes if the rows in A are swapped. So for general matrices, it may be necessary to permute rows in a way such that the entries with biggest absolute value are at the pivot position. This strategy is called *row pivoting* and involves another at most $\mathscr{O}\left(n^2\right)$ operations. On the other hand, one can also apply this strategy to the columns of A, thus getting *column pivoting* or a combination of both, which is then called *total pivoting*. The latter is of $\mathscr{O}\left(n^3\right)$ and therefore almost never used.

It can be shown theoretically (see [35]) that for every nonsingular matrix A there exists a permutation matrix P such that a triangular factorization

$$PA = LU$$

is possible. Note that this is a constructive proposition for the existence of the factorization and its feasibility via Gaussian elimination.

1.1.3 Cholesky Decomposition

If A is symmetric positive definite (*spd*), the Gaussian elimination process can be simplified exploiting these properties.

Theorem 1.1 *For every spd matrix A there exists a uniquely determined factorization of the form*

$$A = \bar{L}D\bar{L}^T,$$

where $\bar{L}$ is a unit lower triangular matrix (with ones on the main diagonal) and D a positive diagonal matrix.

Proof See [35]. $\square$

Since $D = \operatorname{diag}(d_i)$ has only positive entries, we can define $L := \bar{L}D^{\frac{1}{2}}$ and obtain the *Cholesky factorization* of A as

$$A = LL^T,$$

where $D^{\frac{1}{2}} = \operatorname{diag}\left(\sqrt{d_i}\right)$. Due to the positive definiteness and the inertia theorem of Sylvester, Gaussian elimination on A leads to a sequence of remainder matrices $A^{(k)}_{k:n,k:n}$, which are again spd. Therefore, no pivoting strategy is necessary for computing the Cholesky factorization. Through exploiting the symmetry of A, we arrive at an algorithm with

$$\#\text{flops} \doteq \frac{1}{3}n^3 = \mathscr{O}\left(n^3\right),$$

which is half the amount of Gaussian elimination in general matrices. After having computed the Cholesky factor L, the system is solved, again, by applying one step of forward substitution with L and one step of backward substitution with L^T.

1.1.4 Iterative Refinement

The representation of floating point numbers in computers is only accurate up to the so-called *machine precision.* So the results which we obtain from the implementation of numerical algorithms can be inaccurate or even worthless. Gaussian elimination can also be affected by these roundoff errors. To overcome this problem, one could compute the solution using a higher machine precision, which can be simulated with certain software packages and entails a huge additional effort.

Conversely, a rather cheap remedy for improving the accuracy of a computed solution is *iterative refinement.* Let x be the exact solution of the linear equation $Ax = b$ and x_0 the solution obtained by Gaussian elimination. Then, we can explicitly evaluate the *residual*

$$r_0 := b - Ax_0 = A(x - x_0) = A\Delta x_0,$$

which satisfies the *corrector equation*

$$A\Delta x_0 = r_0. \tag{1.4}$$

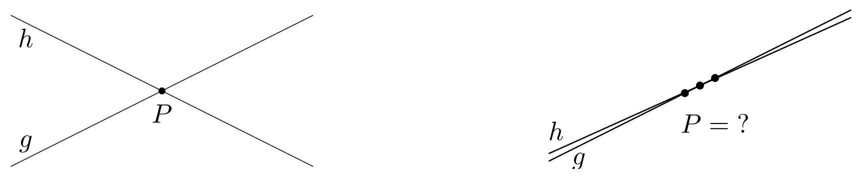
Because the LU decomposition of A is already given, (1.4) can be solved cheaply involving only a forward and a backward substitution step and we get the approximate correction $\tilde{\Delta} x_0$. The approximate solution

$$x_1 := x_0 + \tilde{\Delta} x_0$$

is then supposed to be more accurate than x_0. This refinement step can be repeated, until an "acceptable accuracy" is reached. Indeed, Skeel showed in [90] that for Gaussian elimination, one single step of iterative refinement is sufficient to gain an acceptably accurate solution. Moreover, if it is carried out using column pivoting, Gaussian elimination followed by one refinement step is backward stable.

1.2 Condition Number of Systems of Linear Equations

Due to the floating point representation of numbers in today's computer architectures, real numbers ($\mathbb{R}$) cannot be stored exactly in the sense of arbitrary precision unless they are integers. The accuracy of a floating point number is restricted to the machine precision ε. Despite these technical errors, which affect numerical computations, also errors resulting from the implementation of numerical algorithms have to be taken into account. For instance, the realization of the elementary operations $+, -, \cdot, /$ also leads to rounding errors in the result.



(a) Well-conditioned problem. (b) Rather ill-conditioned setting.

Figure 1.1: Two graphical examples for finding the intersection point P of two lines g and h.

A kind of measure for how strongly perturbations of input variables influence the result, independently of the choice of the algorithm, is given by the *condition number* of the problem. Figure 1.1 illustrates two settings for finding the intersection point P of two lines. When the angle between the two lines is large enough we can determine P quite easily. Conversely, if the lines are nearly parallel, it is hard to find the exact location of P. To define the term condition number precisely, we formulate our numerical problem (f, x), following [35], as a function which maps the set of input data into an output set

$$f : U \subset \mathbb{R}^n \to \mathbb{R}^m,$$

where U is an open subset of $\mathbb{R}^n$ and $x, \tilde{x} \in U$. Additionally, we define δ as the precision of which the input data is perturbed. Here, x is the "exact" input and $\tilde{x}$ the perturbed input. At first, we consider normwise precision, which can be formulated in an absolute

$$\|\tilde{x} - x\| \leq \delta$$

with $0 < \delta \in \mathbb{R}$ or in a relative way:

$$\frac{\|\tilde{x} - x\|}{\|x\|} \leq \delta.$$

The absolute output error is given by $f(\tilde{x}) - f(x)$, and the relative output error by $\frac{\|f(\tilde{x}) - f(x)\|}{\|f(x)\|}$, respectively. This leads to

Definition 1.1 (Absolute Normwise Condition Number) *The* absolute normwise condition number *of the problem (f, x) is the smallest number κ_{abs} such that*

$$\|f(\tilde{x}) - f(x)\| \ \dot{\leq}\ \kappa_{\text{abs}} \|\tilde{x} - x\| \qquad \textit{for} \quad \tilde{x} \to x.$$

Accordingly, the smallest number κ_{rel} such that

$$\frac{\|f(\tilde{x}) - f(x)\|}{\|f(x)\|} \ \dot{\leq}\ \kappa_{\text{rel}} \frac{\|\tilde{x} - x\|}{\|x\|} \qquad \textit{for} \quad \tilde{x} \to x$$

is called the relative normwise condition number *of (f, x).*

κ_{abs} and κ_{rel} are therefore measures on how strongly the absolute and relative errors increase. If a problem (f, x) has small condition numbers, it is said to be *well-conditioned*, otherwise *ill-conditioned.* Furthermore, we speak of an *ill-posed* problem, if such numbers do not exist or if $\kappa_{\cdot} = \infty$.

For differentiable f, the mean value theorem yields the condition numbers

$$\kappa_{\text{abs}} = \|f'(x)\| \qquad \text{and} \qquad \kappa_{\text{rel}} = \frac{\|x\|}{\|f(x)\|}\,\|f'(x)\|\,. \tag{1.5}$$

Hereby, $\|f'(x)\|$ is the operator norm of the Jacobian $f'(x) \in \mathbb{R}^{m\times n}$.

Componentwise Condition Number

Up to now, we have only defined the term condition number in a normwise sense. However, we can also define it in a componentwise way, starting with the absolute componentwise precision

$$|\tilde{x}_i - x_i| \le \delta, \qquad (i = 1, \ldots, n).$$

Consequently, the relative componentwise precision is defined as

$$\frac{|\tilde{x}_i - x_i|}{|x_i|} \le \delta, \qquad (i = 1, \ldots, n).$$

With that in mind, we arrive at the

Definition 1.2 (Relative Compononentwise Condition Number) *The* relative compononentwise condition number *of the problem (f, x) is the smallest number $\kappa_{\text{rel}} \ge 0$ such that*

$$\max_i \frac{|f_i(\tilde{x}) - f_i(x)|}{|f_i(x)|} \;\dot{\le}\; \kappa_{\text{rel}} \max_i \frac{|\tilde{x}_i - x_i|}{|x_i|} \qquad \textit{for} \quad \tilde{x} \to x, \qquad (i = 1, \ldots, n).$$

Just as in the normwise case before, we can exploit the differentiability with respect to x and the mean value theorem in order to obtain the result

$$\kappa_{\text{rel}} = \frac{\||f'(x)|\,|x|\|_\infty}{\||f(x)|\|_\infty}.$$

Condition Number of a Linear System $Ax = b$

For our purpose, the analysis of the conditioning of a system of linear equations $Ax = b$ with nonsingular A, we formulate (f, x) as

$$f : \mathbb{R}^n \to \mathbb{R}^n, b \mapsto f(b) := A^{-1}b.$$

In other words, we first regard the right-hand side vector b as the input. With $f'(b) = A^{-1}$ and (1.5), we obtain

$$\kappa_{\text{abs}} = \left\|A^{-1}\right\| \qquad \text{and} \qquad \kappa_{\text{rel}} = \frac{\|b\|}{\|A^{-1}b\|}\left\|A^{-1}\right\| = \frac{\|Ax\|}{\|x\|}\left\|A^{-1}\right\|.$$

Due to the submultiplicativity of the matrix norm $\|.\|$, we can estimate the relative condition number by

$$\kappa_{\text{rel}} \leq \|A\| \left\|A^{-1}\right\|. \tag{1.6}$$

In the next step, we keep b fixed and formulate f as a function in A:

$$f : \mathrm{Gl}(n) \subset \mathrm{Mat}_n(\mathbb{R}) \to \mathbb{R}^n, A \mapsto f(A) = A^{-1}b,$$

which is again differentiable but not linear anymore. For the derivative of f with respect to A, we obtain (see [35])

$$f'(A)C = -A^{-1}CA^{-1}b = -A^{-1}Cx \quad \text{for} \quad C \in \mathbb{R}^{n\times n}.$$

This leads directly to the condition numbers

$$\begin{aligned} \kappa_{\text{abs}} &= \|f'(A)\| = \sup_{\|C\|=1} \left\|A^{-1}Cx\right\| \leq \left\|A^{-1}\right\| \|x\|, \\ \kappa_{\text{rel}} &= \frac{\|A\|}{\|x\|} \|f'(A)\| \leq \|A\| \left\|A^{-1}\right\|. \end{aligned} \tag{1.7}$$

Because of (1.6) and (1.7), we define the *condition number of the matrix A* as

$$\kappa(A) := \|A\| \left\|A^{-1}\right\|. \tag{1.8}$$

$\kappa(A)$ contains the relative condition number for all right-hand sides $b \in \mathbb{R}^n$. So, the relative error in the solution x can be up to $\kappa(A)$ times the relative error in the input values A and b. In general, (1.8) can also be formulated as

$$\kappa(A) := \frac{\max\limits_{\|x\|=1} \|Ax\|}{\min\limits_{\|x\|=1} \|Ax\|},$$

which can additionally be determined for both singular and rectangular matrices A. The following three properties hold:

- $\kappa(A) \geq 1$,
- $\kappa(\alpha A) = \kappa(A) \qquad \forall\, \alpha \in \mathbb{R} \setminus \{0\}$,
- $A \neq 0$ is singular $\iff \kappa(A) = \infty$.

The definition of the condition number of A also depends on the choice of the underlying norm $\|.\|$, which is usually indicated via subscript:

$$\kappa_2(A) = \|A\|_2 \left\|A^{-1}\right\|_2 = \frac{\sigma_1(A)}{\sigma_n(A)},$$

where σ_1 denotes the greatest and σ_n the smallest singular value of A. Thus, $\kappa_2(A)$ can be interpreted as the elongation of the hyperellipsoid $\{Ax : \|x\|_2 = 1\}$. In the following chapters, we will refer to $\kappa_2(A)$ by $\kappa(A)$ or just κ.

Remark 1.1 *If we apply the concept of componentwise condition number to systems of linear equations, we get analogously (b regarded as input)*

$$\kappa_{\text{rel}} = \frac{\left\| |A^{-1}|\,|b| \right\|_\infty}{\left\| |A^{-1}b| \right\|_\infty} = \frac{\left\| |A^{-1}|\,|b| \right\|_\infty}{\left\| |x| \right\|_\infty}. \tag{1.9}$$

Skeel introduced this number in [89]. If we further apply the same idea of keeping b fixed and analyze the effect of perturbations in A now, it follows for the componentwise relative condition number:

$$\kappa_{\mathrm{rel}} = \frac{\left\| |f'(A)|\,|A| \right\|_\infty}{\left\| |f(A)| \right\|_\infty} = \frac{\left\| \left|A^{-1}\right|\,|A|\,|x| \right\|_\infty}{\left\| |x| \right\|_\infty}. \tag{1.10}$$

Altogether, using (1.9) and (1.10), we get the value for the condition number of the combined problem

$$\kappa_{\mathrm{rel}} = \frac{\left\| \left|A^{-1}\right|\,|A|\,|x| + \left|A^{-1}\right|\,|b| \right\|_\infty}{\left\| |x| \right\|_\infty} \le 2\frac{\left\| \left|A^{-1}\right|\,|A|\,|x| \right\|_\infty}{\left\| |x| \right\|_\infty}.$$

If we further set now $x = e = (1, \dots, 1)$, it follows that

$$\frac{1}{2}\kappa_{\mathrm{rel}} \le \frac{\left\| \left|A^{-1}\right|\,|A|\,|e| \right\|_\infty}{\left\| |e| \right\|_\infty} = \left\| \left|A^{-1}\right|\,|A| \right\|_\infty,$$

and we finally define

$$\kappa_C(A) := \left\| \left|A^{-1}\right|\,|A| \right\|_\infty$$

as Skeel's condition number. *$\kappa_C(A)$ satisfies the same properties as $\kappa(A)$ before. Additionally, for a diagonal matrix D, which implies a completely decoupled system, it holds $\kappa_C(D) = 1$. This result is intuitively much clearer than the potentially arbitrary large value of $\kappa(D)$.*

1.3 Sparse Matrices

So far, we did not take into account special structures of the system matrix A. The algorithms and theoretical results of the last few sections can be applied to any general matrix. But in many practical applications such as the numerical solution of partial differential equations (*PDEs*) the matrices which arise have some special properties like only a small number of nonzero entries compared to the dimension and they are highly structured. Such matrices are called *sparse* matrices. They enable us to construct numerical methods which profit extensively from the *sparsity structure*, both in terms of efficient storage in memory and in terms of fast algorithms and data structures. However, there is no strict definition whether a matrix is sparse or not. In general, a matrix is called sparse if the storage of only the nonzero entries and the application of specially adapted algorithms lead to a computational advantage compared to full (*dense*) storage. One little more formal characterization, which is commonly used, says that a matrix is sparse, if

$$\mathtt{nnz}(A) = \mathscr{O}(n),$$

where the number of nonzeros is denoted by `nnz` and n is the dimension of A. Another rule of thumb defines A to be sparse, if there is a number $p \ll n$ such that every row and column of A has at most p nonzero entries. A further term connected with the notion of sparsity is the *density* of a matrix, which is given by $\frac{\mathtt{nnz}}{n^2}$. The *pattern* of a matrix or vector consists of the indices (i, j) of the nonzero entries and thus, reflects the sparsity structure. We will denote it by $\mathscr{P}(A)$ or $\mathscr{P}(b)$ for a matrix A or a vector b, respectively. Moreover, $|\mathscr{P}(A)| = \mathtt{nnz}(A)$.

When a matrix has no or only quite "few" zero entries, it is called *dense*. In this thesis, we will also use the terms *sparser* and *denser*, or *thinner* and *thicker*, respectively, if we speak of a matrix which has less or more nonzero entries than another one.

1.3.1 Sparse Storage Schemes and Basic Matrix Operations

Following [83], we present here a few well-established storage schemes. One of the most straightforward ideas for storing a sparse matrix is the *coordinate format.* Hereby, the matrix entries are stored in triplets, one integer for the number of the row, one integer for the column index, and a double (or float) variable for the matrix entry. Altogether, one needs three arrays for storing all the nonzero entries, each of length `nnz`. For an example, consider the matrix

$$A = \begin{pmatrix} 1.1 & 0 & -2.2 & 0 & 0 \\ 0 & 3.3 & 4.4 & 0 & 0 \\ 0 & 0 & -5.5 & -6.6 & 0 \\ 7.7 & -8.8 & 0 & 9.9 & 0 \\ 0 & 0 & 0 & 0 & 10.1 \end{pmatrix}, \tag{1.11}$$

which has $n = 5$, $\texttt{nnz}(A) = 10$. In coordinate form, we store it in the three arrays

$$\begin{array}{lcl} \texttt{AA} & = & \{\ 1.1,\ -2.2,\ 3.3,\ 4.4,\ -5.5,\ -6.6,\ 7.7,\ -8.8,\ 9.9,\ 10.1\ \}, \\ \texttt{IR} & = & \{\ 1,\ 1,\ 2,\ 2,\ 3,\ 3,\ 4,\ 4,\ 4,\ 5\ \}, \\ \texttt{IC} & = & \{\ 1,\ 3,\ 2,\ 3,\ 3,\ 4,\ 1,\ 2,\ 4,\ 5\ \}. \end{array}$$

`AA` contains the values in either single or double precision and the integer arrays hold the row (`IR`) and column (`IC`) indices, respectively. So, the coordinate format requires storing $3 \cdot \texttt{nnz}(A)$ numbers. This is not optimal, but it has the advantage of a great flexibility when we want to insert or delete entries. Note that the values can be stored in completely arbitrary order, which can also be seen in the pseudo code implementation of Algorithm 1.1.

Algorithm 1.1 Matrix-vector product $c \leftarrow Ab$ in coordinate format.

```
Require: $b$,AA,IR,IC
1: $c \leftarrow 0$
2: for $j = 1 : \texttt{nnz}(A)$ do
3:     $c(\texttt{IR}(j)) \mathrel{+}= \texttt{AA}(j) \cdot b(\texttt{IC}(j))$
4: end for
```

Probably the most popular storage scheme is the *compressed sparse row* (*CSR*) scheme. It further reduces the memory consumption by avoiding redundancies in the index arrays. The matrix entries are stored row-wise. The first index array `JA` contains the column index of each corresponding matrix entry, i.e. the value $\texttt{AA}(j)$ is in column $\texttt{JA}(j)$ in A. A second integer array `IA` contains the start indices of the rows relative to `AA`. So, the values of the k-th row of A are stored in $\texttt{AA}(\texttt{IA}(k)), \dots, \texttt{AA}(\texttt{IA}(k+1))$. Matrix (1.11) from above would be saved in

$$\begin{array}{lcl} \texttt{AA} & = & \{\ 1.1,\ -2.2,\ 3.3,\ 4.4,\ -5.5,\ -6.6,\ 7.7,\ -8.8,\ 9.9,\ 10.1\ \}, \\ \texttt{JA} & = & \{\ 1,\ 3,\ 2,\ 3,\ 3,\ 4,\ 1,\ 2,\ 4,\ 5\ \}, \\ \texttt{IA} & = & \{\ 1,\ 3,\ 5,\ 7,\ 10,\ 11\ \}. \end{array}$$

`IA` contains as last entry $\texttt{IA}(1) + \texttt{nnz}$, which stands for the symbolic beginning of row $n+1$. Compared to coordinate form, the costs for storage are reduced to $2 \cdot \texttt{nnz}(A) + n + 1$. The implementation (see Algorithm 1.2) also entails indirect addressing, which leads to slower access times to the entries compared to direct addressing in a standard array. There are

Algorithm 1.2 Matrix-vector product $c \leftarrow Ab$ in CSR format.

```
Require: $b$,AA,JA,IA
1: $c \leftarrow 0$
2: for $i = 1 : n$ do
3:     for $j = \mathtt{IA}(i) : \mathtt{IA}(i+1) - 1$ do
4:         $c(i) += \mathtt{AA}(j) \cdot b(\mathtt{JA}(j))$
5:     end for
6: end for
```

many variations and extensions to the CSR format. The most common one is to store A column-wise, which yields the *compressed sparse column* (*CSC*) format.

If a numerical method requires fast and frequent access to diagonal entries, the main diagonal is stored explicitly in the *modified compressed sparse row* (*MSR*) format. Here, the main diagonal of A is stored in the first n entries of AA, followed by an empty entry $\mathtt{AA}(n+1)$ and then the row-wise ordered non-diagonal entries. The index arrays JA and IA are combined to one index array JA. The first $n+1$ entries are the pointers to the starting positions of the rows in AA. For $j = n+2, \ldots, \mathtt{nnz}+1$, $\mathtt{JA}(j)$ yields the column index of element $\mathtt{AA}(j)$:

$$\begin{array}{lcllllllllllll} \mathtt{AA} & = & \{ & 1.1, & 3.3, & -5.5, & 9.9, & 10.1, & *, & -2.2, & 4.4, & -6.6, & 7.7, & -8.8 \ \}, \\ \mathtt{JA} & = & \{ & 7, & 8, & 9, & 10, & 12, & 12, & 3, & 3, & 4, & 1, & 2 \ \}. \end{array}$$

The implementation only changes slightly compared to the original CSR method, see Algorithm 1.3.

Algorithm 1.3 Matrix-vector product $c \leftarrow Ab$ in MSR format.

```
Require: $b$,AA,JA
1: for $i = 1 : n$ do
2:     $c(i) = \mathtt{AA}(i) \cdot b(i)$
3:     for $j = \mathtt{JA}(i) : \mathtt{JA}(i+1) - 1$ do
4:         $c(i) += \mathtt{AA}(j) \cdot b(\mathtt{JA}(j))$
5:     end for
6: end for
```

The computational costs for the matrix-vector products in the three storage schemes presented here are all of order $\mathcal{O}(\mathtt{nnz}(A))$ and only differ in the access times due to the indirect addressing and possible caching side effects.

There are many more possibilities to store a sparse matrix efficiently. The actual application always determines which method is the best. For instance, it depends on whether access should be fastest column- or row-wise. The SPAI preconditioner is usually computed column-wise. Therefore, we need fast access to the columns of A and hence, we would choose the CSC format. Further details and a few more storage schemes can be found in [83].

1.3.2 The Graph Related to a Matrix

Graph theory provides quite powerful tools for representing the sparsity structure of a matrix by investigating the *adjacency graph* $\mathscr{G}$ related to a matrix A. In general, a graph $\mathscr{G}$ is defined by two sets: the set of *vertices*

$$\mathscr{V} = \{v_1, \dots, v_n\}$$

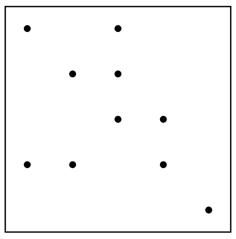
and the set of *edges*

$$\mathscr{E} \subseteq \mathscr{V} \times \mathscr{V}.$$

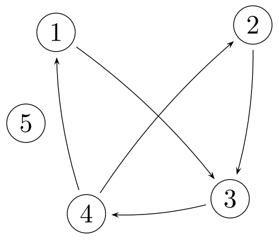
In the context of sparse matrices, the adjacency graph $\mathscr{G} = (\mathscr{V}, \mathscr{E})$ of $A \in \mathbb{R}^{n\times n}$ consists of n vertices corresponding to the n unknowns of the linear system and the set of edges $\mathscr{E}$, which represents the relation

$$\exists \text{ edge from node } i \text{ to node } j \quad \iff \quad a_{ij} \neq 0.$$

This graph is *directed* for general matrices and *undirected* in the case of a symmetric pattern. The latter are represented using undirected edges. As an example, both the sparsity structure and the corresponding adjacency graph of the example matrix (1.11) are depicted in Figure 1.2. The loops for the diagonal elements a_{ii}, which correspond to the edges (v_i, v_i) in



(a) Sparsity pattern of matrix (1.11).



(b) Adjacency graph of (1.11).

Figure 1.2: Sparsity pattern of a matrix and corresponding graph $\mathscr{G} = (\mathscr{V}, \mathscr{E})$.

$\mathscr{G}$, are usually omitted. In all discussed examples here, we assume that matrices have nonzeros in their diagonal. When we try to solve linear systems arising from the discretization of PDEs, the adjacency graph of the system matrix is often identical to the discretization mesh of the domain, where the PDE is solved.

There are many graph algorithms applied to matrices for different purposes. They all determine row and/or column permutations or certain partitionings of the original matrix. For instance, there are special fast algorithms for band matrices which clearly outperform general methods. The Cuthill-McKee or the reverse Cuthill-McKee algorithm [30] produce a permutation that minimizes the bandwidth of a matrix to make these fast methods applicable. Another issue is parallelization, namely partitioning a matrix in such a way that it can be solved in a parallel computing environment, minimizing the communication overhead. The hypergraph partitioning package hMetis by Karypis [72] can be used for such applications. For symmetric indefinite systems of equations, Hagemann and Schenk [50] presented a graph-based method which improves the solution process in direct sparse solvers.

The Gaussian elimination process often leads to a large amount of fill-in in L and U. This makes computations more costly and leads to higher memory usage. In this case, graph algorithms can help to find a permutation of A such that the fill-in is reduced. We briefly describe one of the most commonly used methods for the case of structurally symmetric matrices, the *Approximate Minimum Degree* (*AMD*) algorithm by Amestoy, Davis, and Duff [2].

1.3.3 Approximate Minimum Degree Reordering

One possibility to reduce fill-in during factorization is to consider the elimination graph. The elimination graph $\mathscr{G}^k = (\mathscr{V}^k, \mathscr{E}^k)$ represents the pattern of the remaining submatrix after the k-th elimination step. One obtains $\mathscr{G}^k$ from $\mathscr{G}^{k-1}$ by removing the k-th pivot vertex p and adding edges to $\mathscr{E}^{k-1}$ such that the vertices in $\mathscr{G}^{k-1}$ form a fully connected subgraph (called a *clique*). The edges which are added represent the new entries, namely the fill-in. In order to reduce this fill-in, it makes sense to choose a pivot vertex p with minimum degree. The *degree of a vertex* $v \in \mathscr{V}$ is defined as $|\{u \in \mathscr{V} : \{u, v\} \in \mathscr{E}\}|$, i.e. the number of vertices $u \in \mathscr{V}$ which are connected to v through an edge $e \in \mathscr{E}$. This method uses a non-optimal greedy heuristic. Moreover, note that according to [104], the optimal solution is NP complete.

The drawback of the employment of elimination graphs is the unpredictable number of added edges and thus, unpredictable memory usage. Therefore, a different graph model is used, the *quotient graph*. Its storage never exceeds the storage of the original graph and it exploits the economic storage of cliques which can simply be represented by its vertices. The vertices, which were already removed from the graph are called *elements*, the uneliminated *variables*. Variables v_i with the same degree are grouped into supervariables $\mathscr{S}_i$ with one principle variable. If a supervariable is selected to be eliminated, all variables in the supervariable are eliminated. For the elimination, we choose now the supervariable with minimum *external degree*, i.e. with minimum value $t_i - |\mathscr{S}| + 1$, where t_i is the degree of the principal variable in $\mathscr{S}$. The exact computation of the external degrees requires too much computational effort. In the AMD algorithm [2], we only estimate an upper bound for the external degree, which entails considerably fewer operations.

Considering the symmetric example matrix bcsstk14 [77] with dimension $n = 1806$, we observe a great reduction of `nnz` in the Cholesky factor. Without preordering, we have 190791 nonzeros, whereas using approximate minimum degree results in `nnz` $= 108947$. This is a reduction of over 42%. Hence, the storage consumes less memory and operations such as matrix-vector products or forward as well as backward substitutions are carried out much faster.

In Chapter 2, we will investigate the effect of AMD on the factorized sparse approximate inverse preconditioner (FSPAI). Furthermore, AMD is used in order to accelerate the solution of least squares problems within the computation of the sparse approximate inverse preconditioner (SPAI). For a more detailed review of graphs and graph algorithms, see [43, 83, 92].

1.4 Iterative Solution Methods

As seen in Section 1.1, the explicit solution of dense linear systems requires operations in the order of $\mathcal{O}\left(n^3\right)$. This is only applicable for problem sizes up to about $n = 10000$. For larger ones, Gaussian elimination cannot be efficient anymore and the solution process would take too much time. In such cases, *iterative methods* can help to reduce the costs by exploiting certain properties of the matrices and profit from sparsity structures and specially adapted efficient data structures.

All the methods presented here start with an initial guess x^0 for the solution of $Ax = b$ and compute a sequence of vectors $x^0 \to x^1 \to \ldots \to x^k$, until the error of k-th iterate is under a user-provided bound ε. Hereby, we distinguish between the *absolute residual*

$$r_{\text{abs}} := \left\|b - Ax^k\right\|_2 = \left\|r^k\right\|_2$$

and the *relative residual*

$$r_{\text{rel}} := \frac{\left\|b - Ax^k\right\|_2}{\left\|b - Ax^0\right\|_2} = \frac{\left\|r^k\right\|_2}{\left\|r^0\right\|_2}.$$

The vector $r^i := b - Ax^i$ is called *residual vector*. Often, the initial guess is chosen as vector of zeros. Then, the relative residual reduces to

$$r_{\text{rel}} = \frac{\left\|b - Ax^k\right\|_2}{\|b\|_2} = \frac{\left\|r^k\right\|_2}{\|b\|_2}.$$

1.4.1 Basic Algorithms

We start off reviewing the most basic iterative solution methods for systems of linear equations. Our representation is mainly influenced by [83]. The methods can also be found in many other books about numerical mathematics, e.g. [35, 46, 94]. The *Jacobi* iteration computes the next iterate by annihilating the i-th component r_i of the residual vector

$$r_i = \left(b - Ax^{k+1}\right)_i \overset{!}{=} 0 \qquad \Longleftrightarrow \qquad a_{ii}x_i^{k+1} = -\sum_{\substack{j=1\\ j\neq i}}^{n} a_{ij}x_j^k + b_i$$

which yields the compononentwise form of the Jacobi iteration

$$x_i^{k+1} = \frac{1}{a_{ii}}\left(b_i - \sum_{\substack{j=1\\ j\neq i}}^{n} a_{ij}x_j^k\right), \qquad i = 1, \ldots, n. \tag{1.12}$$

All the components x_i^{k+1} together form the next iterate x^{k+1}. Considering the matrix splitting $A = D - E - F$, where D is a diagonal matrix, E is strict lower triangular matrix and F is of strict upper triangular form, we can reformulate (1.12) as

$$x^{k+1} = D^{-1}(E + F)x^k + D^{-1}b. \tag{1.13}$$

By imposing an order in which the components of x are determined and thus, the components of the residual are annihilated, the *Gauss-Seidel iteration* also involves the components of x^{k+1}, which are already computed. Since the order is $i = 1, 2, \ldots, n$, the iteration is

$$x_i^{k+1} = \frac{1}{a_{ii}} \left(-\sum_{j=1}^{i-1} a_{ij} x_j^{k+1} - \sum_{j=i+1}^{n} a_{ij} x_j^k + b_i \right), \qquad i = 1, \ldots, n.$$

Again, this can be rewritten using $A = D - E - F$ from above:

$$x^{k+1} = (D - E)^{-1} F x^k + (D - E)^{-1} b. \tag{1.14}$$

In every Jacobi step (1.13), we only have to multiply with the inverse of a diagonal matrix, which results in a quite cheap scaling step. For a Gauss-Seidel step (1.14), however, we have to solve a system with lower triangular matrix $D - E$ using forward substitution (1.3). Choosing the reversed computation order $i = n, n-1, \ldots, 1$, we arrive at the *backward Gauss-Seidel iteration*:

$$(D - F) x^{k+1} = E x^k + b.$$

In general, the two methods presented so far are based on matrix splittings $A = M - N$ with $M = D$ for Jacobi, and $M = D - E$ for forward Gauss-Seidel. An iterative method of the form

$$M x^{k+1} = N x^k + b = (M - A) x^k + b$$

can be formulated for any splitting $A = M - N$, as long as M is nonsingular. Introducing a *relaxation parameter* ω such that

$$\omega A = (D - \omega E) - (\omega F + (1 - \omega) D),$$

we obtain, as corresponding iterative method, the so-called *Successive Over Relaxation* (*SOR*):

$$(D - \omega E) x^{k+1} = (\omega F + (1 - \omega) D) \, x^k + \omega b.$$

The actual choice of the relaxation parameter ω is strongly dependent on the spectral properties of A and therefore also on the underlying PDE, which has been discretized. Literature is full of optimal estimates for ω, for instance, see [83, 94]. Combining one step of SOR and one of backward SOR, we get the recurrence

$$x^{k+1} = G_\omega x^k + f_\omega,$$

with

$$\begin{aligned} G_\omega &= (D - \omega F)^{-1} (\omega E + (1 - \omega) D)(D - \omega E)^{-1} (\omega F + (1 - \omega) D) \\ f_\omega &= \omega (2 - \omega)(D - \omega F)^{-1} D (D - \omega E)^{-1} b. \end{aligned}$$

This combination is called *Symmetric Successive Over Relaxation* (*SSOR*).

The answer for the question, in which cases these basic methods presented here will converge, can partially be given by a quite general convergence theorem:

Theorem 1.2 *The fixed-point method $x^{k+1} = G x^k + f$ with $G \in \mathbb{R}^{n \times n}$ converges for each starting vector x^0 if and only if for the spectral radius $\rho(G) = \max_j |\lambda_j(G)|$ holds:*

$$\rho(G) < 1,$$

where the eigenvalues of G are denoted by λ_j.

See for instance [35] for the proof. There are numerous further proofs and deeper analyses of convergence for these basic iterative methods. Here, we restrict ourselves to only one for a quite important class of matrices.

Definition 1.3 (Diagonal dominant matrix) *A matrix A is called*

- weakly diagonally dominant *if*

$$|a_{ii}| \geq \sum_{\substack{i=1 \\ i\neq j}}^{n} |a_{ij}|\,, \qquad j = 1,\ldots,n$$

- strictly diagonally dominant *if*

$$|a_{ii}| > \sum_{\substack{i=1 \\ i\neq j}}^{n} |a_{ij}|\,, \qquad j = 1,\ldots,n$$

- irreducibly diagonally dominant *if A is irreducible and weakly diagonally dominant, where strict inequality holds for at least one j.*

Strictly and irreducibly diagonally dominant matrices are nonsingular, which follows immediately applying Gershgorin's theorem. Another consequence is the following theorem.

Theorem 1.3 *If A is a strictly diagonally dominant or irreducibly diagonally dominant matrix, then the associated Jacobi and Gauss-Seidel iterations converge for any initial guess x^0.*

For the proof, see [83].

Nowadays, the main application area of these basic methods is not anymore solving linear systems. They are mostly employed as so-called *smoothers* in multigrid methods [98]. There, the relaxed Gauss-Seidel and relaxed Jacobi are used for damping the high-frequency parts in the error on the fine (original) grid, whereas the computationally expensive operations are carried out on *coarser* grids with only few unknowns left.

1.4.2 Krylov Subspace Methods

The most frequently used and fastest iterative solvers can be found in the class of Krylov subspace methods. The most popular representative is the method of *conjugate gradients* (*cg*), which was developed by Hestenes and Stiefel in 1952 [55]. Although it is restricted to spd matrices, we will use it to present the basic notion of Krylov subspace methods.

The historical origin of the cg method lies in the minimization of the function

$$\phi(x) = \frac{1}{2}\langle x, Ax\rangle - \langle x, b\rangle\,, \tag{1.15}$$

which is equivalent to the solution of the linear system arising from the gradient

$$\Delta\phi(x) = Ax - b \overset{!}{=} 0.$$

Geometrically, Equation (1.15) describes a hyperparaboloid, of which we try to find the absolute minimum. Starting off at some "position" x^0, the initial guess for the minimum, the direction of steepest descent given by the negative value of the gradient

$$-\Delta\phi(x^k) = b - Ax^k = r^k.$$

r^k could be taken as search direction to approach the solution x. This leads to a sequence of *optimal line search* problems

$$\phi^k(\alpha_{k+1}) = \phi(x^k + \alpha_{k+1}x^k)$$

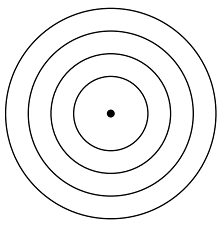
with solution

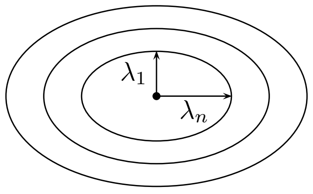
$$\alpha_{k+1} = \frac{\langle r_k, r_k\rangle}{\langle r_k, Ar_k\rangle}.$$

But from formula

$$\left|\phi(x^k) - \phi(x)\right| \leq \left(1 - \frac{1}{\kappa_2(A)}\right)\left|\phi(x^{k-1}) - \phi(x)\right|,$$

we can identify the condition number $\kappa_2(A)$ as a measure for the "distortion" of the level sets $\{x : \phi(x) = c\}$, $c \geq 0$ from spheres to ellipsoids. Precisely, the smallest eigenvalue $\lambda_1(A)$ and the largest eigenvalue $\lambda_n(A)$ are the lengths of the smallest and largest semiaxis of $\{\phi(x) = 1\}$, see Figure 1.3. Therefore, the higher the condition number $\kappa_2(A) = \frac{\lambda_n(A)}{\lambda_1(A)}$ becomes, the more the level sets differ from spheres and thus, many of the search directions of steepest descent run in parallel, and convergence can be extremely slow.


(a) Level sets for well-conditioned A.



(b) Elliptic shaped level sets due to big $\kappa_2(A)$.

Figure 1.3: Spectral condition number $\kappa_2(A)$ as measure for distortion of level sets.

A solution for this problem is the choice of different search directions. The cg method uses A-conjugate projections of r^k denoted by p^k, which means $\langle p^k, Ap^k\rangle = 0$. Many authors use the definition $\langle x, y\rangle_A := \langle x, Ay\rangle$ and the corresponding norm $\|x\|_A = \sqrt{\langle x, y\rangle_A}$, which is called *energy norm.* The subspace

$$U_k = \text{span}\{r^0, Ar^0, \ldots, A^{k-1}r^0\},$$

on which the error $x^k - x$ is projected, is called *Krylov subspace.* The first search direction is $p^0 = r^0$, and the following directions for the consecutive steps are computed using Gram-Schmidt orthogonalization. Therefore, the search directions build an orthogonal basis and thus, p^{k+1} can be obtained via $p^{k+1} = r^{k+1} + \beta_k p^k$. β_k is given by

$$\beta_{k+1} = \frac{\langle r^{k+1}, r^{k+1}\rangle}{\langle r^k, r^k\rangle}.$$

In order to minimize $\phi(x)$ in direction of p^k, the minimum value for α_k is given by

$$\alpha_k = -\frac{\langle r^k, r^k \rangle}{\langle p^k, Ap^k \rangle}.$$

Due to this construction process, the dimension of the underlying Krylov subspace is at most n and in precise arithmetic, the cg method converges in at most n iterations. The

Algorithm 1.4 Conjugate Gradient Method (cg).

```
Require: $A$, $b$, initial guess $x^0$, tolerance for relative residual TOL
 1: $p^0 = r^0 = b - Ax^0$
 2: for $k = 0 : k_{\max}$ do
 3:    $\alpha_k = -\frac{\langle r^k, r^k \rangle}{\langle p^k, Ap^k \rangle}$
 4:    $x^{k+1} = x^k - \alpha_k p^k$
 5:    $r^{k+1} = r^k - \alpha_k Ap^k$
 6:    if relative residual < TOL then
 7:       STOP
 8:    end if
 9:    $\beta_{k+1} = \frac{\langle r^{k+1}, r^{k+1} \rangle}{\langle r^k, r^k \rangle}$
10:    $p^{k+1} = r^{k+1} + \beta_{k+1} p^k$
11: end for
```

complete cg method is shown in Algorithm 1.4. The dominating part of the computational effort is the cost for the matrix-vector product Ap^k in each of the k iterations needed until convergence:

$$\#\text{flops}(cg) \doteq k \cdot \#\text{flops}(Ap^k).$$

So, when the number of iterations k is sufficiently small and the matrix-vector product exploits the sparsity structure, the computational cost for the cg method is $\mathcal{O}\left(n^2\right)$. The connection between convergence behavior and the condition number of the coefficient matrix A is drawn by the following estimate by Axelsson and Barker [6]:

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

The cg method will converge more rapidly as the condition number of the system matrix tends to 1.

For the much more general case of nonsymmetric matrices $A \neq A^T$, there exist also many algorithms. In this thesis, we will mainly present results using the cg method and, for general matrices, the BiCGSTAB [102] or GMRES [84] algorithm. The former is a stabilized extension of the conjugate gradient method and the latter is based on Arnoldi iterations. For detailed descriptions see [46, 83].

1.5 Preconditioning

The term *preconditioning* was initially introduced by Alan Turing in a paper published already in 1948 (see [99]). There, Turing describes preconditioning as a process to improve the condition number of systems of linear equations. But the term did not come into popular use until the early 1970s.

As we can see for instance in the convergence estimate of the cg algorithm, the condition number $\kappa(A)$ of the coefficient matrix A is crucial for the rate of decrease in the error, and hence also for the number of iterations an iterative solver will need. Thus, we try to transform $Ax = b$ into an equivalent system using a matrix $M \in \mathbb{R}^{n\times n}$ such that the overall condition number is reduced significantly. The matrix M is then called *preconditioner.*

Generally, the main goal is to minimize the overall solution time, which consists of the setup time for the preconditioner and the time needed by the solver. We will distinguish between *forward* and *inverse* preconditioners. Two main constraints emerge for the computation of both types:

- It should be possible to compute M fast and additionally, the application of M should also be as cheap as possible. For both types, forward and inverse, the latter imposes an upper bound for the number of nonzeros in M.

- The quality of the preconditioner in terms of reducing the overall condition number requires a higher computational effort and usually leads to a denser M.

Algorithm 1.5 Preconditioned Conjugate Gradient Method (pcg).

```
Require: $A$, $b$, $M$, initial guess $x^0$, tolerance for relative residual TOL
 1: $r^0 = b - Ax^0$
 2: $q^0 = Mr^0$
 3: for $k = 0 : k_{\max}$ do
 4:   $\alpha_k = -\frac{\langle r^k, r^k\rangle}{\langle q^k, Aq^k\rangle}$
 5:   $x^{k+1} = x^k - \alpha_k q^k$
 6:   $r^{k+1} = r^k - \alpha_k Aq^k$
 7:   if relative residual < TOL then
 8:     STOP
 9:   end if
10:   $\beta_{k+1} = \frac{\langle r^{k+1}, Mr^{k+1}\rangle}{\langle r^k, Mr^k\rangle}$
11:   $q^{k+1} = Mr^{k+1} + \beta_{k+1} q^k$
12: end for
```

The application of the preconditioner in an actual algorithm does of course not employ the explicit calculation of the matrix-matrix product AM, but is carried out by two consecutive matrix-vector products (in the inverse case). Algorithm 1.5 shows the *preconditioned conjugate gradient* method for an inverse preconditioner M. It involves one more matrix-vector product than the unpreconditioned cg method. Note that in this case, M has to be an spd matrix, as well. The algorithm described here deals with real matrices. The variation for the complex case is presented in detail in [25].

1.5.1 Forward Preconditioners

Forward type or *explicit* or *direct* preconditioners yield approximations to A such that $M \approx A$ and thus, $\kappa(AM^{-1}) \overset{!}{\ll} \kappa(A)$. The corresponding *left preconditioned* system then reads

$$M^{-1}Ax = M^{-1}b. \tag{1.16}$$

We perform *right preconditioning* by the substitution $y = Mx$ and thus solve

$$AM^{-1}y = b \qquad \Longrightarrow \qquad x = M^{-1}y.$$

The combination of these two methods leads to

$$M_1^{-1}AM_2^{-1}y = M_1^{-1}b \qquad \Longrightarrow \qquad x = M_2^{-1}y.$$

Examples for forward preconditioners were already shown implicitly in the previous section. The basic iterative solvers can be interpreted as preconditioned systems of linear equations of the form (1.16) with the preconditioners

$$\begin{aligned}
M_{\text{Jacobi}} &= D, \\
M_{\text{Gauss-Seidel}} &= D - E, \\
M_{\text{SOR}} &= \frac{1}{\omega}(D - \omega E), \\
M_{\text{SSOR}} &= \frac{1}{\omega(2-\omega)}(D - \omega E)D^{-1}(D - \omega F).
\end{aligned}$$

Another quite commonly used forward type class is the class of incomplete factorizations.

Incomplete Factorizations

One of the most well-known and intensively investigated group of forward preconditioners is the class of incomplete factorizations. The method is based on some heuristics such as the observation that in the inverse of a matrix, entries with relatively small absolute value occur at positions where the original matrix has zero entries. The simplest variant is the *incomplete LU factorization* (*ILU*). Here, an LU decomposition is performed, which is incomplete in the sense that only the entries for a small given sparsity pattern are computed. This is the *ILU(0)* method. The 0 states that no fill-in is allowed outside the pattern (0-level fill-in).

There are methods which allow more levels of fill-in (ILU(1), ILU(2), . . .) but are not robust enough and do not yield reliable convergence behavior. A more promising variation of the basic ILU(0) method is *ILU(p, τ)*. Here, we prescribe a fixed number p of nonzero elements and a threshold τ for dropping smaller elements.

Whereas ILU yields preconditioners for general matrices, the equivalent method for spd matrices is called *incomplete Cholesky (IC)*. Here, the basic idea is to perform an incomplete Cholesky decomposition and, again, there exist many variants for controlling the fill-in. For a fixed pattern, it is called *IC(0)*, accordingly.

A drawback of incomplete factorizations is instability which can lead to breakdown. Chow and Saad [28] reported failure either in the factorization process itself or during backward substitution. This can occur when ILU is applied to strongly nonsymmetric or indefinite systems. More recent results can be found in [20, 73]. We also want to mention that there exist incomplete factorization approaches not based on LU decomposition, but on orthogonal factorizations [8].

Another major disadvantage of forward type approaches might potentially be that in each step of an iterative solver, a linear system in M has to be solved. This must not be more costly than solving the original system. As a consequence, forward preconditioners have to be constructed in a way that allows easy and fast inversion of M, for instance diagonal or triangular form. This is a constraint, from which the second category, the inverse preconditioners, does not suffer.

1.5.2 Inverse Preconditioners

Inverse preconditioners are constructed from approximations to the inverse A^{-1} of the system matrix such that

$$\kappa(AM) \overset{!}{\ll} \kappa(A).$$

Just like in the forward case, they can be applied as *left preconditioners*

$$MAx = Mb,$$

or as *right preconditioners*

$$AMy = b, \qquad x = My$$

or even as a combination of both techniques

$$M_1AM_2y = M_1b, \qquad x = M_2y.$$

Due to the inverse approximation character of M, we do not need to apply M via solving a linear system, but by simply performing a matrix-vector product. Therefore, there are no restrictions to the sparsity pattern of M.

Benzi and Tůma [14] provide a comparative overview of sparse approximate inverse preconditioners. A large group thereof are *polynomial preconditioners*, see [23, 37, 69]. The idea is to approximate A^{-1} by a polynomial in A of low degree. These preconditioners are applied only via matrix-vector products, which makes this type well-suited for parallel computing environments. On the other hand, they are less effective than e.g. incomplete factorizations as far as reducing the condition number and hence the number of iterations is concerned. The additional matrix-vector products outweigh the reduction of the number of iterations.

Also in the class of inverse preconditioners, there are factorized approaches such as AINV (see [13, 15]) or FSAI by Kolotilina and Yeremin [74, 75]. AINV computes approximations

$$A^{-1} \approx ZD^{-1}W^T$$

with unit upper triangular matrices Z and W, as well as the diagonal matrix D. Z approximates U^{-1} and $W \approx L^{-T}$. The factors are computed using incomplete biconjugation, i.e. a conjugate Gram-Schmidt process. The vectors are thinned-out using a drop tolerance in order to assure sparsity. AINV does not require an imposed sparsity pattern. It is applicable to general sparse matrices. Robustness is only proven for special types of matrices such as H-matrices, see [12]. Also, parallelization of this method is a quite difficult task, involving graph algorithms for optimal preorderings.

Finally, there are two sparse approximate inverse preconditioners, which are able to capture the best sparsity pattern for M itself: on the one hand, the SPAI preconditioner and on the other hand its factorized variant, FSPAI. In contrast to the other approaches presented here, they are robust, yield competitive preconditioners, and are inherently parallel. Both form the fundamentals for the rest of this thesis. They are introduced in the following chapter.

Chapter 2

SPAI and FSPAI

Approximate inverse preconditioners are known to be robust methods for general problems $Ax = b$. This chapter introduces the idea of Frobenius norm minimization in order to obtain such approximate inverses and explains, how they are computed. The first section will describe the sparse approximate inverse (*SPAI*) preconditioner by Grote and Huckle [48] in detail. After having discussed the solution of the arising least squares (LS) problems, we will focus on SPAI's specialty, which lets SPAI clearly stand out of the other approximate inverse preconditioners. SPAI can autonomously identify new promising entries for the sparsity pattern of the preconditioner. The new entries are chosen such that the residual is minimized. We also present a formulation of criteria for pattern updates when the linear system $Ax = b$ is complex ($A \in \mathbb{C}^{n\times n}$, $x, b \in \mathbb{C}^n$). This was not yet topic in literature. This is followed by a discussion about sparsity patterns for inverse preconditioners in general containing an overview of recent approaches to effectively predict sparsity structures. After repeating some well-known theoretical properties of SPAI and reformulating a rather recent further result on the nonsingularity of SPAI, we conclude the section with an example application from image restoration. This use of SPAI as a regularizing preconditioner is also a new field of application.

Especially for symmetric positive definite (spd) problems, we want to have a preconditioner which preserves these properties. Therefore, the second part of this chapter explains the factorized sparse approximate inverse (*FSPAI*) preconditioner, which computes a triangular inverse factor of the coefficient matrix. For spd problems, it performs considerably better than the unfactorized SPAI. This will be demonstrated by an example simulation. The specialty of FSPAI is again the automatic pattern update ability. It enables us to use it as a black box algorithm. In contrast to SPAI, FSPAI is not invariant under permutations. The effect of minimum degree algorithms on many other matrix factorizations was extensively investigated by many authors. According to that, we present the influence of the AMD reordering on the FSPAI pattern update steps.

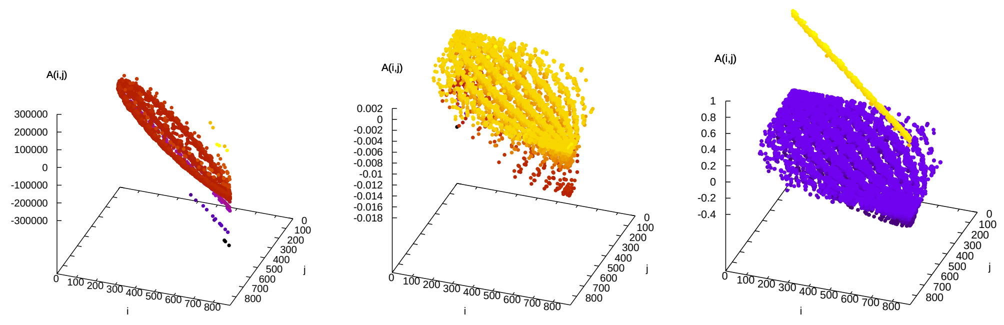
A new proof considering the SPAI and FSPAI of M-matrices, respectively, will round off the overview about approximate inverse preconditioners which automatically capture their sparsity pattern.

2.1 SPAI

Sparse approximate inverses $M \approx A^{-1}$ computed via Frobenius norm minimization were initially used for preconditioning by Benson and Frederickson [11]. The key idea is to construct a sparse matrix either from a sparse or even thinned out dense matrix A, so that the matrix-vector products in the iterative solver do not become too expensive. The minimization

$$\min_{M} \|AM - I\|_F^2 \tag{2.1}$$

with matrix $A \in \mathbb{R}^{n \times n}$ and n-dimensional unity matrix I leads to the sparse approximate inverse $M \in \mathbb{R}^{n \times n}$ with $M \approx A^{-1}$. Here, we concentrate on right preconditioning. For a left preconditioner, we only have to apply the formulation for the right preconditioner inserting A^T instead of A and we get an M so that $MA \approx I$. By Figure 2.1(c), we can see how AM approximates the identity: AM contains many very small entries near 0 (blue) and on its diagonal values near to 1 (yellow).



(a) A. (b) SPAI $M \approx A^{-1}$. (c) $AM \approx I$.

Figure 2.1: 3D matrix plots of example matrix orsirr2 after reverse Cuthill-McKee reordering, SPAI M and preconditioned AM. The yellow line in Figure 2.1(c) indicates the values near 1 on the main diagonal, whereas all other values are near 0.

The aspect which gives SPAI a clear advantage over other preconditioners as far as its implementation and calculation on modern supercomputers or clusters is concerned is its inherent parallelism. It is based on a property of the Frobenius norm that allows us to split it up into a sum of Euclidean norms

$$\begin{aligned} \|AM - I\|_F^2 &= \sum_{k=1}^{n} \|(AM - I)e_k\|_2^2 && (2.2) \\ &= \sum_{k=1}^{n} \|Am_k - e_k\|_2^2 && (2.3) \end{aligned}$$

where m_k and e_k denote the k-th columns of M and I, respectively. Each summand in (2.3) represents a least squares (LS) problem for exactly one column m_k of M and can be solved completely independently from the others:

$$\min_{m_k} \|Am_k - e_k\|_2, \qquad k = 1, \ldots, n. \tag{2.4}$$

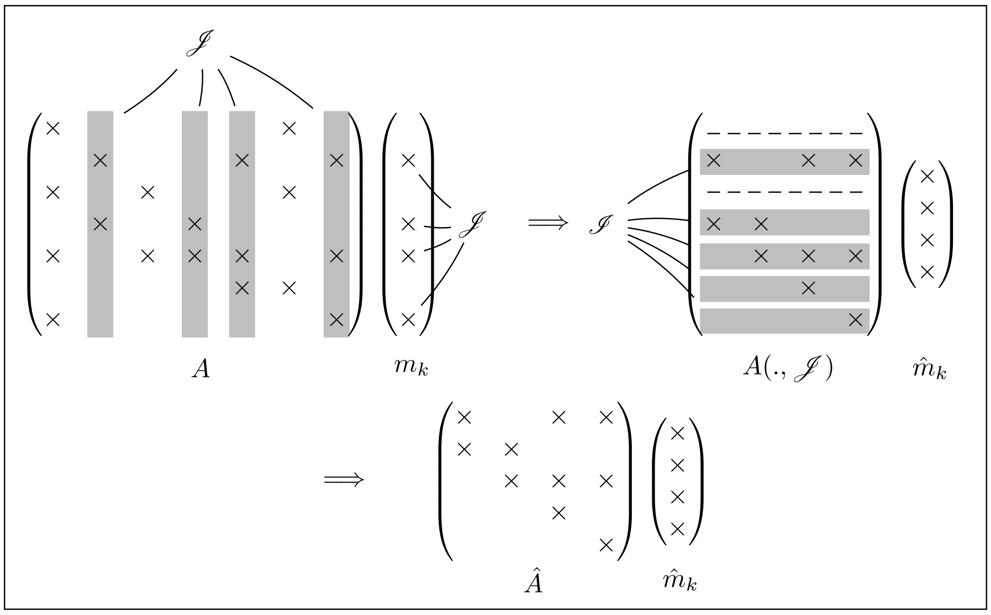


Figure 2.2: $\hat{A}$ is defined by the index sets $\mathscr{J}$ and the shadow $\mathscr{I}$ thereof (here is $\mathscr{J} = \{2,4,5,7\}$ and $\mathscr{I} = \{2,4,5,6,7\}$).

2.1.1 Solution of the Least Squares Problems

The degrees of freedom in the LS problems (2.4) are defined by the sparsity pattern of m_k. Let $\mathscr{J}$ be the set of indices where the pattern of m_k has nonzero entries

$$\begin{aligned}\mathscr{J} &:= \{j : m_k(j) \neq 0, j = 1,\dots,n\},\\ q &:= |\mathscr{J}|.\end{aligned}$$

Then $\mathscr{J}$ also contains the column indices of A which are needed (see Figure 2.2) when we evaluate a matrix-vector product with the reduced vector $m_k(\mathscr{J})$. The accordingly reduced matrix $A(.,\mathscr{J})$ usually has a lot of zero rows due to its sparsity. So we further denote the set of indices of nonzero rows in $A(.,\mathscr{J})$ with $\mathscr{I}$:

$$\begin{aligned}\mathscr{I} &:= \left\{ i : \sum_{j\in\mathscr{J}} |a_{ij}| \neq 0, i = 1,\dots,n \right\}, \\ p &:= |\mathscr{I}|.\end{aligned} \tag{2.5}$$

$\mathscr{I}$ is also called the *shadow of* $\mathscr{J}$. With the definitions

$$\begin{aligned}\hat{A} &:= A(\mathscr{I},\mathscr{J}) \in \mathbb{R}^{p\times q},\\ \hat{m}_k &:= m_k(\mathscr{J}) \in \mathbb{R}^{q\times 1},\\ \hat{e}_k &:= e_k(\mathscr{I}) \in \mathbb{R}^{p\times 1},\end{aligned}$$

the LS problems (2.4) are reduced to the much smaller problems

$$\min_{\hat{m}_k} \left\| \hat{A}\hat{m}_k - \hat{e}_k \right\|_2, \qquad k = 1,\dots,n, \tag{2.6}$$

with much smaller dimensions p and q compared to the original dimension n. Because of this, current implementations of such algorithms employ math libraries such as LAPACK [76] or ATLAS [3] for dense matrices to solve these minimizations. They are all based on the computation of the QR decomposition via Householder transformations, which is an $\mathcal{O}\left(n^3\right)$ algorithm. Therefore, the use of sparse methods will be examined in Chapter 4.

Remark 2.1 *Barnard and Grote also introduced a block SPAI variant in [9]. There, the LS problems are not formulated column-wise, but block-wise, thus using block QR methods. A is not partitioned by columns anymore, but by blocks. Moreover, the pattern update criteria change slightly. On the one hand, the block approach is faster to compute, but on the other hand the scalar SPAI leads to more accurate LS solutions and thus to a more effective preconditioner M.*

There are several other ways to solve (2.6) such as methods for normal equations, Gram-Schmidt or the Modified Gram-Schmidt method. If A is nonsingular, $\hat{A}$ has full column rank q. Thus we can use the QR decomposition of $\hat{A} = QR$, $Q \in \mathbb{R}^{p\times p}$, $R \in \mathbb{R}^{p\times q}$. This implies that the nonzero entries of m_k are computed as

$$\begin{aligned} \hat{c} &:= Q^T \hat{e}_k, \\ \hat{m}_k &= R^{-1}_{1:q,:} \hat{c}_{1:q}. \end{aligned}$$

Since we apply Q^T only once on one vector, we do not need to build Q explicitly, which would be computationally quite expensive. The action of Q^T on a vector can be calculated using the Householder vectors from the QR decomposition step. This will be described in detail in Chapter 4.

Remark 2.2 *If we prescribe a simple diagonal pattern, we can give an analytical solution for the n entries m_{kk}, $k = 1, \ldots, n$ of M. The LS problem for one column is*

$$\min_{m_{kk}} \left\| \hat{A} m_{kk} - e_k \right\|_2 . \tag{2.7}$$

Because of $m_{kk} \in \mathbb{R}$, $\hat{A}$ reduces to one single column $a_k \in \mathbb{R}^n$ and $e_k \in \mathbb{R}^n$. So, using the inner product, we have

$$\left\| a_k m_{kk} - e_k \right\|_2^2 = m_{kk}^2 \langle a_k, a_k \rangle - 2 m_{kk} \langle a_k, e_k \rangle + \langle e_k, e_k \rangle .$$

Differentiation with respect to m_{kk} yields the solution of (2.7)

$$\begin{aligned} 2 m_{kk} \langle a_k, a_k \rangle - 2 \langle a_k, e_k \rangle &\overset{!}{=} 0 \\ \Leftrightarrow \quad m_{kk} = \frac{\langle a_k, e_k \rangle}{\langle a_k, a_k \rangle} &= \frac{a_{kk}}{\left\| a_k \right\|_2^2}, \end{aligned}$$

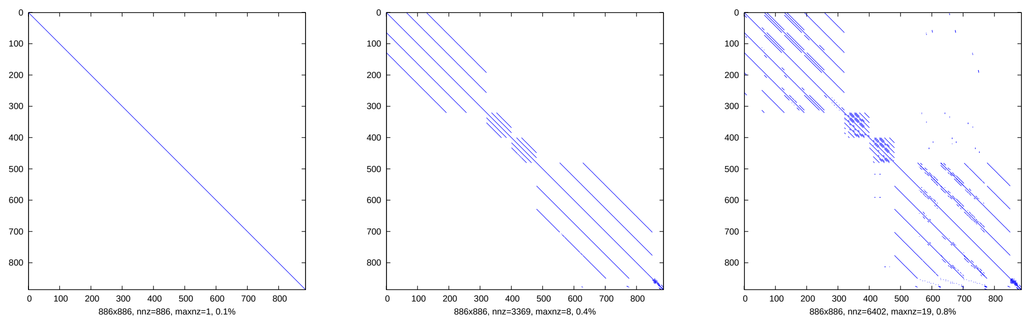
where a_{kk} is the diagonal element of the k-th column of A. M with diagonal pattern is called optimal diagonal SPAI *or* SPAI-0.

Remark 2.3 *Bröker, Grote, Mayer, and Reusken have shown in [22] that the optimal diagonal SPAI yields an attractive alternative to standard smoothers in multigrid methods. They observed a better smoothing behavior compared to a damped Jacobi, but it is completely parameter free. According to [22], a static SPAI with the pattern of A behaves mostly like a Gauss-Seidel smoother. Moreover, it has the advantage of inherent parallelism in its computation. Where local adaptivity is needed, it is possible to perform pattern updates automatically as it will be shown in the following section.*

As stated above, the dimension of $\hat{A}$ is defined by the sparsity pattern for M. The actual choice of these patterns will be subject of Section 2.1.3, whereas the automatic determination thereof is shown in the following.

2.1.2 Pattern Updates

Once a preconditioner M is computed as stated in the previous section for a given start pattern, SPAI has the special ability to update this sparsity structure autonomously by identifying the most profitable new indices. We explain this process by showing how the update of one single column m_k is computed. Figure 2.3 depicts an example for two update steps.



(a) Diagonal start pattern. (b) Pattern after 1 update step. (c) Pattern after 2 update steps.

Figure 2.3: Pattern update steps for matrix orsirr2 (see [77]) with parameters $\varepsilon = 0.2$ and a maximum of 5 new entries added in each update step.

At first, we calculate the residual vector r for the k-th column of M:

$$r = Am_k - e_k.$$

If r is a vector of all zeros, then m_k is already the k-th column of the exact inverse A^{-1}. We denote by $\mathscr{L}$ the set of indices of the nonzero entries in r:

$$\mathscr{L} := \{\ell : r(\ell) \neq 0\} \cup \{k\}.$$

Now, for every $\ell \in \mathscr{L}$ we can specify a set $\mathscr{N}_\ell$, which consists of the nonzero indices of the ℓ-th row in A

$$\mathscr{N}_\ell := \{j : a_{\ell j} \neq 0\}.$$

Only these column indices of the ℓ-th row are potential new candidates, because the others would vanish when performing the sparse matrix-vector product. The union $\tilde{\mathscr{J}}$ of the sets $\mathscr{N}_\ell$ $(\forall \ell \in \mathscr{L})$ yields the complete set of indices, which can be added in order to improve $\|r\|_2$:

$$\tilde{\mathscr{J}} := \bigcup_{\ell \in \mathscr{L}} \mathscr{N}_\ell$$

with $\mathscr{J} \cap \tilde{\mathscr{J}} = \emptyset$. For the identification of the indices, which lead to the most profitable reduction of $\|r\|_2$ we can solve the 1D minimization problem for every $j \in \tilde{\mathscr{J}}$

$$\min_{\mu_j} \|A(m_k + \mu_j e_j) - e_k\|_2 = \min_{\mu_j} \|r + \mu_j A e_j\|_2$$

and, using $Ae_j = a_j$, obtain the term

$$\min_{\mu_j} \|r + \mu_j a_j\|_2^2 = \mu_j^2 \|a_j\|_2^2 + 2\mu_j \langle r, a_j \rangle + \|r\|_2^2 .$$

After differentiation with respect to μ_j, we obtain

$$2\mu_j \|a_j\|_2^2 + 2 \langle r, a_j \rangle \overset{!}{=} 0$$

and finally the solution

$$\mu_j = -\frac{r^T a_j}{\|a_j\|_2^2}. \tag{2.8}$$

Instead of one-dimensional minimization, we could also use multivariate minimization as proposed by Gould and Scott in [47]. This approach sometimes recovers sparsity structures of the inverse in a better way than the univariate method, but suffers from higher computational costs. A comparison for banded matrix structures can be found in [62].

In order to see which μ_j lead to the smallest residual $\|r\|_2$, we can calculate the new residuals we would achieve when we add the index j to $\mathscr{J}$ and denote it by ρ_j:

$$\begin{aligned} \rho_j^2 := \|r + \mu_j a_j\|_2^2 &= \|r\|_2^2 - 2\left\langle r, \frac{r^T a_j}{\|a_j\|_2^2} a_j \right\rangle + \left\| \frac{r^T a_j}{\|a_j\|_2^2} a_j \right\|_2^2 \\ &= \|r\|_2^2 - 2\frac{r^T a_j}{\|a_j\|_2^2} r^T a_j + \frac{(r^T a_j)^2}{\|a_j\|_2^4} \|a_j\|_2^2 \\ &= \|r\|_2^2 - \frac{(r^T a_j)^2}{\|a_j\|_2^2}. \end{aligned} \tag{2.9}$$

Implementations of the SPAI algorithm always provide a parameter for the number of new indices that shall be added to the current pattern in one update step. SPAI then takes this number of indices corresponding to the smallest values of ρ_j. However, there are many other variants and heuristics for determining how many indices should be added. If, for instance, we try to obtain a preconditioner M which has only few nonzero entries, we only add few entries per step and do not perform many subsequent update steps. As another approach, we can compute the mean value $\bar{\rho}$ of all ρ_j and then only add the indices with $\rho_j \leq \bar{\rho}$. But also for this criterion, an upper bound for the number of indices must be imposed in order to ensure a certain low degree of sparsity. The most wide-spread and most commonly used SPAI implementation of Stephen Barnard (see [10]) works with this method. Our implementation of the MSPAI algorithm, see Chapters 3 and 4, chooses new indices in this way, too. For a more compact notation, we define:

Definition 2.1 (Update parameter Υ) *By $\Upsilon_{\alpha,\beta}$, we denote the parameter for SPAI-like algorithms such that a maximum of α pattern update steps is performed, each adding exactly β new indices to the pattern of one column (unless all positions in that column are already set). $\bar{\Upsilon}_{\alpha,\beta}$ indicates that only indices are added which correspond to the values $\rho_j \leq \bar{\rho}$.*

Let $\tilde{\mathscr{J}}$ be the set of new indices which are added to $\mathscr{J}$. Consequently, $\tilde{\mathscr{I}}$ is the potentially enlarged set of nonzero rows in $A(., \mathscr{J} \cup \tilde{\mathscr{J}})$. Then, we again have to solve the LS problem (2.6), but now with larger submatrix

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

where $\tilde{p} := |\tilde{\mathscr{I}}|$ and $\tilde{q} := |\tilde{\mathscr{J}}|$. Since we have already computed the QR factorization of $\hat{A}$, we do not have to perform a complete factorization of $\bar{A}$. Nonzero entries in rows $\tilde{\mathscr{I}}$ can only occur in columns $\tilde{\mathscr{J}}$. So we can find row and column permutations such that

$$\begin{aligned}\tilde{A} &= \begin{pmatrix} \hat{A} & A(\mathscr{I}, \tilde{\mathscr{J}}) \\ 0 & A(\tilde{\mathscr{I}}, \tilde{\mathscr{J}}) \end{pmatrix} \\ &= \begin{pmatrix} Q & \\ & I_{\tilde{p}} \end{pmatrix} \begin{pmatrix} R & Q_1^T A(\mathscr{I}, \tilde{\mathscr{J}}) \\ 0 & Q_2^T A(\mathscr{I}, \tilde{\mathscr{J}}) \\ 0 & A(\tilde{\mathscr{I}}, \tilde{\mathscr{J}}) \end{pmatrix} = \begin{pmatrix} Q & \\ & I_{\tilde{p}} \end{pmatrix} \begin{pmatrix} R & B_1 \\ 0 & B_2 \end{pmatrix}.\end{aligned}$$

The only remaining task now is to compute the QR decomposition of $B_2 \in \mathbb{R}^{(\tilde{p}+p-q)\times \tilde{q}}$. A detailed review of these QR updates can be found in Section 4.1.3.

As long as $\|r\|_2$ is bigger than a certain user provided tolerance ε, and the number of nonzeros of the current pattern is smaller than a second parameter, the update step is repeated until one of the criteria is fulfilled. The latter one assures, if chosen meaningful that M does not become too dense.

Remark 2.4 *In the case of complex valued problems $Ax = b$ with $A \in \mathbb{C}^{n\times n}$, $x, b \in \mathbb{C}^n$, the computation of M for a fixed pattern is just the same as in the real case. It only involves the use of complex LAPACK [76] methods for solving the LS problems. However, in the pattern update steps, the formulas for the computation of $\mu_j \in \mathbb{R}$ (2.8) and accordingly $\rho_j \in \mathbb{R}$ (2.9) differ slightly. Using the properties of the complex inner product, we get*

$$\begin{aligned}\|r + \mu_j a_j\|_2^2 &= |\mu_j|^2 \|a_j\|_2^2 + \langle r, \mu_j a_j\rangle + \langle \mu_j a_j, r\rangle + \|r\|_2^2 \\ &= \mu_j^2 \|a_j\|_2^2 + \bar{\mu}_j \langle r, a_j\rangle + \mu_j \overline{\langle r, a_j\rangle} + \|r\|_2^2 \\ &= \mu_j^2 \|a_j\|_2^2 + 2\mu_j \mathrm{Re}\left(r^H a_j\right) + \|r\|_2^2\end{aligned}$$

with minimum value

$$\mu_j = -\frac{\mathrm{Re}\left(r^H a_j\right)}{\|a_j\|_2^2}.$$

Using this to derive the formula for ρ_j, we arrive at

$$\begin{aligned}\rho_j^2 &= \|r\|_2^2 + \bar{\mu}_j \langle r, a_j\rangle + \mu_j \overline{\langle r, a_j\rangle} + \|\mu_j a_j\|_2^2 \\ &= \|r\|_2^2 - 2\frac{\mathrm{Re}\left(r^H a_j\right)}{\|a_j\|_2^2}\mathrm{Re}\left(r^H a_j\right) + \frac{\mathrm{Re}\left(r^H a_j\right)^2}{\|a_j\|_2^4}\|a_j\|_2^2 \\ &= \|r\|_2^2 - \frac{\mathrm{Re}\left(r^H a_j\right)^2}{\|a_j\|_2^2}.\end{aligned}$$

We use this criterion in our MSPAI 1.0 implementation (see Chapter 4) in case of complex matrices.

2.1.3 Sparsity Patterns

Due to the fact that for most even sparse matrices A the exact inverse A^{-1} is usually dense, the approximation of $M \approx A^{-1}$ is improved by allowing more entries in the sparsity pattern of M. In the case of SPAI, this can immediately be seen from (2.9), because the residual norm $\|r\|_2$ can only be reduced if the pattern update adds the j-th entry to the pattern. However, the computation time increases with the size of $\hat{A}$. So the sparsity pattern which we provide for the preconditioner M is crucial as far as approximation quality is concerned. Moreover, the time to compute it is equally important. Furthermore, a sparse approximate inverse preconditioner with more entries will lead to fewer iteration steps when it is employed within an iterative solver such as GMRES or the preconditioned conjugate gradient method (pcg). Each single iteration will take more time, because the matrix-vector products within one iteration become more expensive. So one has to balance these factors. We need patterns that are thick enough to produce sufficiently small residuals $\|AM - I\|_F$. Furthermore, the patterns should also be thin enough, so that the computation time of M itself and therefore the time needed by one iteration does not grow too high.

As we have seen in Section 2.1.2, SPAI has the special ability to start with an arbitrary sparsity pattern and update it adaptively with respect to certain optimality aspects. Nevertheless, these update steps can be quite costly, as for the identification of possible new entries (index set $\mathscr{N}_\ell$), we have to perform many set operations. Having identified the potential new entries, the computation of the reduction ρ_j of the residuals involves many inner products. We will address the problem of an improved implementation in Chapter 4 as we will be investigating the effect of sparse QR methods and caching on the runtimes of SPAI.

From the theoretical point of view, it makes sense to look for promising start patterns for this process in order to have less or no pattern updates at all. Thus, we put more effort on finding good start patterns for the Frobenius norm minimization. Based on the Neumann representation for the inverse A^{-1} of a matrix

$$(I - A)^{-1} = \sum_{j=0}^{\infty} A^j, \tag{2.10}$$

the powers of A can provide quite straightforward and promising nonzero structures for M. A drawback is that these patterns become quite dense as j increases. As a solution, it is possible to apply thinning-out steps and sparsify A^j again. These thresholds are very problem-dependent and can lead to singular matrices if chosen improperly. A prescaling step can help to make it easier to determine meaningful drop tolerances. We present in Section 5.1 an example for the successful employment of a sparsification strategy. To overcome the difficulties, Chow proposed in [27] a method based on the graph related to the sparsity structure of a matrix. The basic idea of his approach is to guarantee that in each column of the pattern still at least two entries remain. In terms of graph algorithms, this means that after sparsifying, there is at least a fixed number of edges to or from each vertex. In practice, the algorithm chooses, for each column, the diagonal entry and the indices corresponding to the two or more biggest values in the column. Chow then computes the powers of this matrix, which can be sparsified again and so on. Preconditioners (2.1) calculated with these patterns lead to convergence behavior in iterative solvers of equal quality as the ones computed by an adaptive SPAI. Usually, the former are denser and therefore the single iterations take more time, whereas the latter are chosen very carefully with respect to the residual minimization. On the other hand, this clearly takes more time.

Huckle presented in [61] the idea of providing a small bounded maximum pattern for SPAI, which combines the advantages of the two approaches. Starting point is the Neumann series (2.10) and the characteristic polynomial of a matrix (Cayley-Hamilton theorem). By that, we observe that the sparsity pattern $\mathscr{P}\left(A^{-1}\right)$ of the inverse A^{-1} is contained in the pattern of $(I+|A|)^{n-1}$:

$$\mathscr{P}\left(A^{-1}\right) \subseteq \mathscr{P}\left((I+|A|)^{n-1}\right).$$

Then, Huckle applies this argument to $B = A^T A$, which yields

$$\mathscr{P}\left(A^{-1}\right) \subseteq \mathscr{P}\left((|A|^T |A|)^{n-1} |A|^T\right).$$

It can also be advisable to include the pattern of A^T into the approximation of $\mathscr{P}\left(A^{-1}\right)$. Hence, we replace $(I+|A|)$ by $(I+|A|+|A|^T)$ and arrive at

$$\mathscr{P}\left((I+|A|+|A|^T)^{n-1} |A|^T\right)$$

as a good approximation for $\mathscr{P}\left(A^{-1}\right)$. These estimates are computed under the assumption that A has only nonnegative entries, but for general matrices, they hold as well. For the actual employment, we do not compute the algebraic powers of $(|A|^T |A|)^{n-1}$ or $(I+|A|+|A|^T)^{n-1}$ completely, but only up to a small $m \ll n-1$. This can then be done efficiently by the use of graph algorithms. So, the patterns we prescribe as maximum sparsity patterns are

$$\mathscr{P}\left((|A|^T |A|)^m |A|^T\right) \quad \text{or} \tag{2.11}$$

$$\mathscr{P}\left((I+|A|+|A^T|)^m |A|^T\right). \tag{2.12}$$

Huckle further shows that (2.11) is an upper bound for the pattern which can occur by computing the SPAI pattern updates. Moreover, in some cases, $\mathscr{P}\left((|A|^T |A|)^m |A|^T\right)$ and the pattern recovered by SPAI are identical. (2.11) can get quite dense when we increase m, whereas (2.12) allows a more sensitive increase in the number of nonzeros.

Two advantages immediately emerge from the use of such maximum patterns:

- The number of indices which have to be tested (the determination of the index sets $\mathscr{N}_\ell$) decreases significantly and therefore the computation time for the preconditioner.
- In parallel environments, we precisely know in advance which parts of A each processor will need at most. So we can distribute the data in the beginning, and no further (expensive) communication will be needed until the end of the computation, where the resulting matrix M is gathered from all the computing nodes. This should also lead to a considerably good load balancing in terms of CPU computation time, because the pattern estimates are quite precise upper bounds for the patterns found by the SPAI algorithm.

The maximum patterns can be applied for general unsymmetric matrices, as well as for triangular or symmetric ones. In the triangular case, one should not include A^T in the pattern, but use $\mathscr{P}\left(|A|^m\right)$ for small m, instead. For spd matrices, Huckle extends the notion of maximum patterns also for the FSPAI pattern update approach (see Section 2.2) applying the same arguments as for nonsymmetric problems.

The actual runtime improvements by the use of maximum sparsity patterns are investigated in Sections 4.3 and 4.4.

2.1.4 Theoretical Properties of SPAI

In [48], Grote and Huckle also present theoretical properties of SPAI, focussing on the spectrum of AM and the difference between A and M in certain norms.

Theorem 2.1 *Let $p = \max_{1\leq k\leq n}\{$number of nonzero elements of $r_k\}$. r_k is the residual vector of the k-th column and satisfies $\|r_k\|_2 < \varepsilon$. Then*

$$\begin{aligned} \|AM - I\|_F &\leq \sqrt{n}\varepsilon, \\ \|AM - I\|_2 &\leq \sqrt{n}\varepsilon, \\ \|AM - I\|_1 &\leq \sqrt{p}\varepsilon. \end{aligned}$$

Furthermore

$$\begin{aligned} \left\|M - A^{-1}\right\|_F &\leq \left\|A^{-1}\right\|_2 \sqrt{n}\varepsilon, \\ \left\|M - A^{-1}\right\|_2 &\leq \left\|A^{-1}\right\|_2 \sqrt{n}\varepsilon, \\ \left\|M - A^{-1}\right\|_1 &\leq \left\|A^{-1}\right\|_1 \sqrt{p}\varepsilon. \end{aligned}$$

Corollary 2.1 *If $\sqrt{p}\varepsilon < 1$, then M is nonsingular.*

Theorem 2.2 *Let $p = \max_{1\leq k\leq n}\{$number of nonzero elements of $r_k\}$. Then, the eigenvalues λ_k of AM are clustered at 1 and lie in a circle of radius $\sqrt{p}\varepsilon$. Furthermore, if $\sqrt{p}\varepsilon < 1$, then $\lambda_{\max}$ and $\lambda_{\min}$ satisfy*

$$\left|\frac{\lambda_{\max}}{\lambda_{\min}}\right| \leq \frac{1+\sqrt{p}\varepsilon}{1-\sqrt{p}\varepsilon}.$$

Theorem 2.3 *The singular values of AM are clustered at 1 and lie inside the interval $[1-\delta, 1+\delta]$, with $\delta = \sqrt{n}\varepsilon(2+\sqrt{n}\varepsilon)$. Furthermore, if $\delta < 1$, then the condition number of AM satisfies*

$$\kappa_2(AM) \leq \sqrt{\frac{1+\delta}{1-\delta}}.$$

The proofs to these theorems can be found in [48].

Another rather recent theoretical result was published by Wang, Lawlor, and Kale in [103]. It is a result concerning the nonsingularity of a computed SPAI M. Here we present both the theorem and the proof in a more compact reformulation:

Theorem 2.4 *Let M be the SPAI of a nonsingular matrix $A \in \mathbb{R}^{n\times n}$ and define $C := AM - I$ with $C = (c_{ij})_{i,j=1,\dots,n}$. The pattern of M shall include the main diagonal entries. If*

$$\sum_{k=1}^{n} |c_{kk}| < 1,$$

then

$$\|AM - I\|_F < 1,$$

and M is nonsingular.

Proof We regard one column $\hat{m}_k$ of the solution of the index reduced LS problem

$$\min_{\hat{m}_k} \left\|\hat{A}\hat{m}_k - \hat{e}_k\right\|_2^2,$$

which also satisfies the corresponding normal equation

$$\hat{A}^T \hat{A} \hat{m}_k = \hat{A}^T \hat{e}_k. \tag{2.13}$$

Setting $D := AM$ with $D = (d_{ij})_{i,j=1,\dots,n}$, we can observe that

$$\hat{m}_k^T \hat{A}^T \hat{e}_k = d_{kk}. \tag{2.14}$$

Multiplying (2.13) from the left with $\hat{m}_k^T$ and inserting (2.14) directly yields

$$\begin{aligned} d_{kk} &= \hat{m}_k^T \hat{A}^T \hat{e}_k = \hat{m}_k^T \hat{A}^T \hat{A} \hat{m}_k = \|d_k\|_2^2 \\ &= \sum_{j=1}^{n} d_{jk}^2. \end{aligned} \tag{2.15}$$

By construction, we obtain

$$d_{jk} = \begin{cases} c_{jk} & \text{for} \quad j \neq k \\ c_{jk} + 1 & \text{for} \quad j = k \end{cases}. \tag{2.16}$$

With (2.16) and (2.15), the following holds:

$$\begin{aligned} c_{kk} &= d_{kk} - 1 = \sum_{j=1}^{n} d_{jk}^2 - 1 = \sum_{\substack{j=1 \\ j\neq k}}^{n} d_{jk}^2 + d_{kk}^2 - 1 \\ &= \sum_{\substack{j=1 \\ j\neq k}}^{n} c_{jk}^2 + (c_{kk}+1)^2 - 1 = \sum_{\substack{j=1 \\ j\neq k}}^{n} c_{jk}^2 + c_{kk}^2 + 2c_{kk} - 1 + 1 \\ &= \sum_{j=1}^{n} c_{jk}^2 + 2c_{kk} \end{aligned}$$

and consequently

$$c_{kk} = -\sum_{j=1}^{n} c_{jk}^2. \tag{2.17}$$

Now, let $\sum_{j=1}^{n} |c_{kk}| < 1$. With (2.17) follows

$$\left| \sum_{k=1}^{n} c_{kk} \right| = \left| - \sum_{k=1}^{n} c_{kk} \right| = \left| \sum_{k=1}^{n} \sum_{j=1}^{n} c_{jk}^2 \right| = \|C\|_F^2 = \|AM - I\|_F^2 < 1.$$

According to [83], AM is then nonsingular, and thus M is nonsingular. $\square$

Remark 2.5 *We can check the condition $\sum_{j=1}^{n} |c_{kk}| < 1$ quite easily by computing*

$$\sum_{j=1}^{n} |c_{kk}| = \sum_{k=1}^{n} |d_{kk} - 1| = \sum_{k=1}^{n} |a_k m_k| - n$$

with a_k being the k-th row of A and m_k the k-th column of M. Since this involves the evaluation of n sparse dot products, it causes considerably less computational effort than evaluating the matrix-matrix product AM explicitly.

A few details in the proof of Theorem 2.4 allow us to formulate another thereoretical result on AM.

Corollary 2.2 *The diagonal entries d_{kk} $(k = 1, \ldots, n)$ of AM satisfy*

$$0 \le d_{kk} \le 1.$$

Proof Equation (2.15) implies $0 \le d_{kk}$. The second inequality follows by equations (2.16) and (2.17):

$$c_{kk} = d_{kk} - 1 \le 0 \qquad \Longleftrightarrow \qquad d_{kk} \le 1.$$

$\square$

Example 2.1 Consider matrix (1.11) from Section 1.3:

$$A = \begin{pmatrix} 1.1 & 0 & -2.2 & 0 & 0 \\ 0 & 3.3 & 4.4 & 0 & 0 \\ 0 & 0 & -5.5 & -6.6 & 0 \\ 7.7 & -8.8 & 0 & 9.9 & 0 \\ 0 & 0 & 0 & 0 & 10.1 \end{pmatrix}.$$

A static SPAI for A with pattern $\mathscr{P}(A)$ leads to an M such that

$$AM = D = \begin{pmatrix} \boxed{0.062} & & 0.424 & 0.124 & \\ & \boxed{0.314} & -0.162 & -0.247 & \\ 0.201 & -0.386 & \boxed{0.701} & -0.247 & \\ 0.134 & -0.257 & -0.061 & \boxed{0.835} & \\ & & & & \boxed{1.000} \end{pmatrix}$$

with diagonal entries $d_{kk} \in [0, 1]$ $(k = 1, \ldots, 5)$ as predicted by Corollary 2.2.

Since we will have a closer look on the effect of graph reordering algorithms on the factorized SPAI later on, we wish to mention that SPAI is permutation invariant. If permutation matrices P_1 and P_2 are applied to a matrix A, we get $P_1 A P_2$ and as a preconditioner the SPAI $P_2^T M P_1^T$ instead of M if $M \approx A^{-1}$.

2.1.5 Example Application: SPAI in Image Restoration

As a new field of application for SPAI we want to present its regularization property in image restoration. This was joint work with David Flade [42]. We address the problem of reconstructing blurred images which are additionally perturbed by noise. Blur can have many origins such as atmospheric turbulence blur in astronomical image processing or out-of-focus problems. Mathematically it is modelled using the so-called *point spread function* (*PSF*). The PSF describes the blur for every point of the image. If it is not given explicitly, it is determined in several experiments using point sources of light. For simplicity, we assume a PSF which blurs the points of an image independently from the position within the image, i.e. generating spatially invariant blur. Moreover, we assume the blur to be *isotropic* which means that it has the same effect in all directions. An example for an *anisotropic* distortion would be motion blur.

Following [80], the image deblurring problem is given by the linear system

$$Hx = g + \eta, \tag{2.18}$$

where H is the blur matrix, x the original "unblurred" image, which we want to reconstruct, g the given image, and η the additional noise. H is constructed using the PSF. For 1D reconstruction problems, x is a signal and the convolution $g = h * x$ with the blurring function

$$h = (\dots, 0, h_{-m}, h_{-m+1}, \dots, h_0, \dots, h_{m-1}, h_m, 0, \dots)^T$$

applied to

$$x = (\dots, x_{-m+1}, \dots, x_0, \dots, x_n, x_{n+1}, \dots, x_{n+m}, \dots)^T$$

yields the given blurred signal $g = (g_1, \dots, g_n)^T$. The entries g_j are then given by

$$g_j = \sum_{k=-\infty}^{\infty} h_{j-k} x_k.$$

So g is not only determined by $x_1, \dots, x_n$, but also by $x_{-m-1}, \dots, x_{-1}$ and $x_{n+1}, \dots, x_{n+m}$. Hence, the deblurring problem (2.18) in the 1D case can be formulated as the linear system

$$\begin{pmatrix} h_m & \dots & h_0 & \dots & h_{-m} & & & & \\ & h_m & & h_0 & & h_{-m} & & & \\ & & \ddots & \ddots & \ddots & \ddots & \ddots & & \\ & & & h_m & & h_0 & & h_{-m} & \\ & & & & h_m & \dots & h_0 & \dots & h_{-m} \end{pmatrix} \begin{pmatrix} x_{-m+1} \\ \vdots \\ x_1 \\ \vdots \\ x_n \\ x_{n+1} \\ \vdots \\ x_{n+m} \end{pmatrix} = \begin{pmatrix} g_1 \\ \vdots \\ g_n \end{pmatrix}. \qquad (2.19)$$

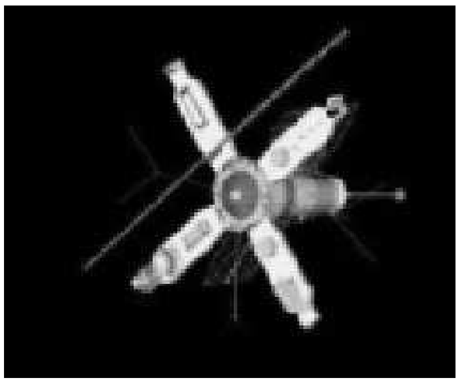
Since (2.19) is underdetermined, we make additional assumptions on the values $x_{-m+1}, \dots, x_0$ and $x_{n+1}, \dots, x_{n+m}$. This can be interpreted as the definition of boundary conditions on the problem. For our purposes, we impose Dirichlet boundary conditions, i.e. $x_{-m+1} = \dots = x_0 = x_{n+1} = \dots = x_{n+m} = 0$. Other choices are possible, such as Neumann or reflective boundary conditions, periodic or anti-reflective types. Due to the choice of the boundary conditions, the resulting blur matrix H falls into certain classes of structured matrices. In our Dirichlet case, we obtain a *Toeplitz* matrix. Tensor arguments yield the respective blur matrices for the 2D case.

The blur matrices H, which are constructed this way, are usually very ill-conditioned or even singular or indefinite. Additionally, the noise η on the blurred image g turns the setting into an ill-posed problem. Thus, already a small amount of noise completely corrupts solutions if they are computed in a straightforward way. The small eigenvalues of H amplify the noise level extremely and make the solution useless. To overcome this effect, we can either use regularization techniques or the regularizing properties of iterative solvers. Among the former is the quite well-known method of *Tikhonov regularization* [97], which transforms (2.18) into a minimization problem. We minimize

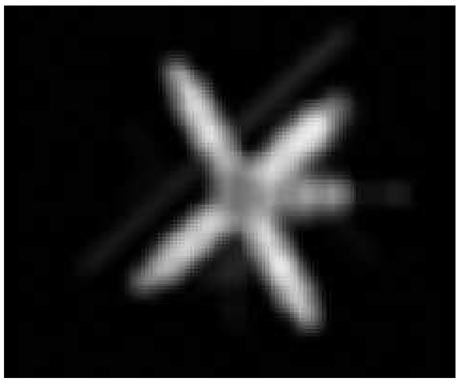
$$\|Hx - (g + \eta)\|_2^2 + \mu \|x\|_2^2$$

with respect to x. $\mu \geq 0$ is a small constant parameter and the solution is computed by solving

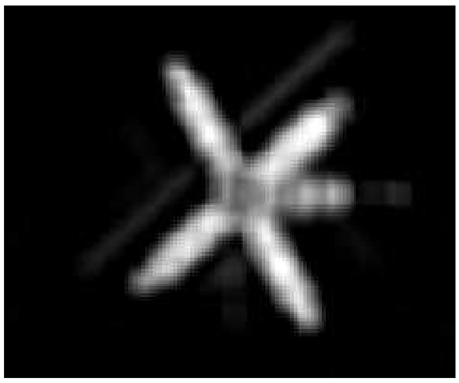
$$(H^T H + \mu I)x = H(g + \eta).$$



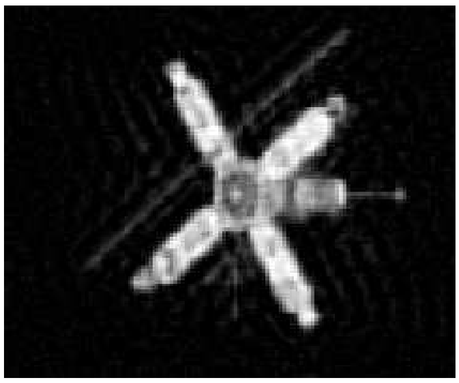
(a) Original image.



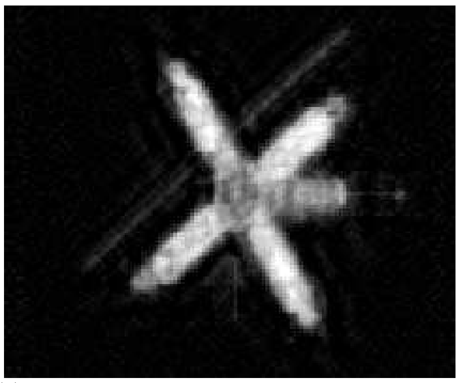
(b) Blurred image with noise of order 10^{-2}.



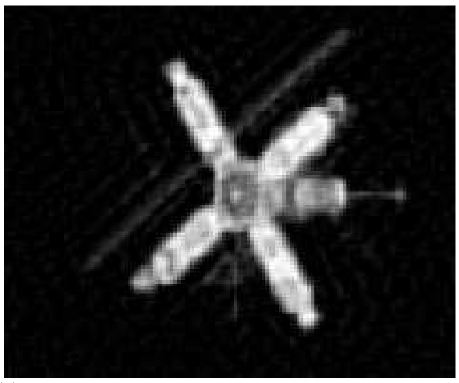
(c) Unpreconditioned BiCGSTAB result after 1 iteration.



(d) Best unpreconditioned BiCGSTAB result after 17 iterations.



(e) SPAI preconditioned BiCGSTAB after 1 iteration.



(f) Optimal preconditioned BiCGSTAB after 1 iteration.

Figure 2.4: Example for image deblurring.

μ has to be chosen carefully: large enough so that it has a regularization effect and small enough such that the difference to the original deblurring problem does not become too significant.

The second remedy is to exploit the inherent regularizing property of iterative solution methods (see, for instance [52, 54]). Additionally, they suit very well for large-scale problems and parallel computing environments. For spd blur matrices, we can use the cg method and in the general unsymmetric case BiCGSTAB, for instance. The usual observation in connection with ill-posed problems is a reduction of the residual in the beginning of the solution process. There, the residual is of larger magnitude than the noise, such that it does not affect the solution, yet. By stopping the solver at the point where the residual attains its minimum value, we can obtain acceptable results. If the solution process progresses on, the noise is taken into account more and more, and the solution deteriorates up to divergence in the end. This effect is called *semi-convergence* [53]. If we consider the singular value decomposition

$$H = U\Sigma V^T$$

with orthonormal matrices $U, V \in \mathbb{R}^{n\times n}$ and $\Sigma = \operatorname{diag}(\sigma_1, \ldots, \sigma_n)$, then $H^{-1} = V\Sigma^{-1}U^T$ holds, and the exact solution can be formulated as

$$x = H^{-1}(g+\eta) = \sum_{k=1}^{n} \left(\frac{u_k^T g}{\sigma_k} + \frac{u_k^T \eta}{\sigma_k} \right) v_k.$$

Since blur matrices are typically ill-conditioned, there are huge eigenvalue clusters around zero. For a small index k, we can assume $\sigma_k \gg \left|u_k^T \eta\right|$ so that the second summand containing the noise η vanishes. But for bigger indices k the situation changes and the summand $\frac{u_k^T\eta}{\sigma_k}$ becomes more significant and finally contaminates the solution. Therefore, it is important to have a good stopping criterion. There are some methods which analyze the so-called *L-curve* [68], i.e. the curve we get when we plot the points $(\log \left\|g - Hx^k\right\|_2, \log \left\|x^k\right\|_2)$ has usually the shape of an "L", and the sharp bend point then implies a good stopping index. For an overview of stopping criteria, e.g. see [68].

In order to accelerate the solution process, we can use a preconditioner. This leads to fewer iterations of the Krylov subspace method at the beginning. There it acts on the large eigenvalues which do not correspond to the noise part of the right-hand side (noise is usually connected with high frequencies). In the optimal case, a preconditioner has no effect on the smaller eigenvalues at all and therefore does not amplify the noise. SPAI is such a preconditioner, since it is more effective in treating large eigenvalues than small ones. In order to achieve $\min_M \|AM - I\|_F < 1$, SPAI primarily has to work on large eigenvalues. For small eigenvalues, $\|AM - I\|_F$ is already small for $M \approx 0$. The objective of our work was to investigate the regularization property of SPAI in this context. Our numerical tests revealed that both the structure and the density of the sparsity pattern prescribed for the SPAI are crucial for the quality of the result. If we choose a pattern with too many entries, the resulting SPAI M approximates the inverse H^{-1} of the blur matrix too exactly, i.e. the small eigenvalues, which correspond to the noise, are also clustered instead of being untouched. For carefully chosen sparser patterns, we get exactly the effect we need for regularization, namely a clustering of the large eigenvalues of HM and no changes in the small ones.

Figure 2.4 shows an example for the regularizing effect. Without preconditioning, we obtain a quite acceptable solution after 17 steps of BiCGSTAB (Figure 2.4(d)). In the preconditioned cases, semi-convergence was reached already after 1 iteration. Using SPAI as preconditioner (Figure 2.4(e)), the result is both of the same quality as the best unpreconditioned solution and also competitive to an alternative preconditioning method (Figure 2.4(f)). This reference method is a truncated version of the superoptimal circulant preconditioner presented in [36]. For a more detailed description of the test parameters and further results, we refer to [42].

2.2 FSPAI

The SPAI algorithm discussed in the previous section computes an approximate inverse preconditioner for general matrices A. Even if A is symmetric, the resulting SPAI preconditioner M will in general be unsymmetric. Hence, we cannot employ it in iterative solvers for symmetric linear systems of equations such as pcg. The consequence would be to solve the system with a method for unsymmetric systems, which have usually a much higher computational complexity. Symmetrization approaches could be a way out (see Chapter 3), but it cannot be assured that the resulting matrices are positive definite.

This section is devoted to a method which allows us to compute factorized sparse approximate inverses (*FSPAI*) for the special purpose of preconditioning symmetric positive definite (spd) matrices A. The FSPAI algorithm [62] is also able to find enlarged sparsity patterns and yields positive definite preconditioners.

Starting point is the spd matrix $A = L_A^T L_A$ with unknown Cholesky factor L_A. Our concern is to find a sparse approximate Cholesky factor L of the inverse

$$M = LL^T \approx A^{-1} \qquad \text{or} \qquad L \approx L_A^{-1}.$$

According to Kolotilina and Yeremin [75], L can be obtained via Frobenius norm minimization

$$\min_L \|L_A L - I\|_F$$

and then normalized such that $\operatorname{diag}\left(L^T A L\right) = I$. This minimization problem reduces to a linear system in a submatrix of A. Additionally, the sparsity pattern of L is restricted to lower triangular form by definition. So, L can be computed without actually knowing the Cholesky factor L_A.

Kaporin [71] presented another equivalent method by minimizing the *Kaporin functional*

$$K := \frac{\frac{1}{n}\operatorname{tr}\left(L^T A L\right)}{\det\left(L^T A L\right)^{\frac{1}{n}}}. \tag{2.20}$$

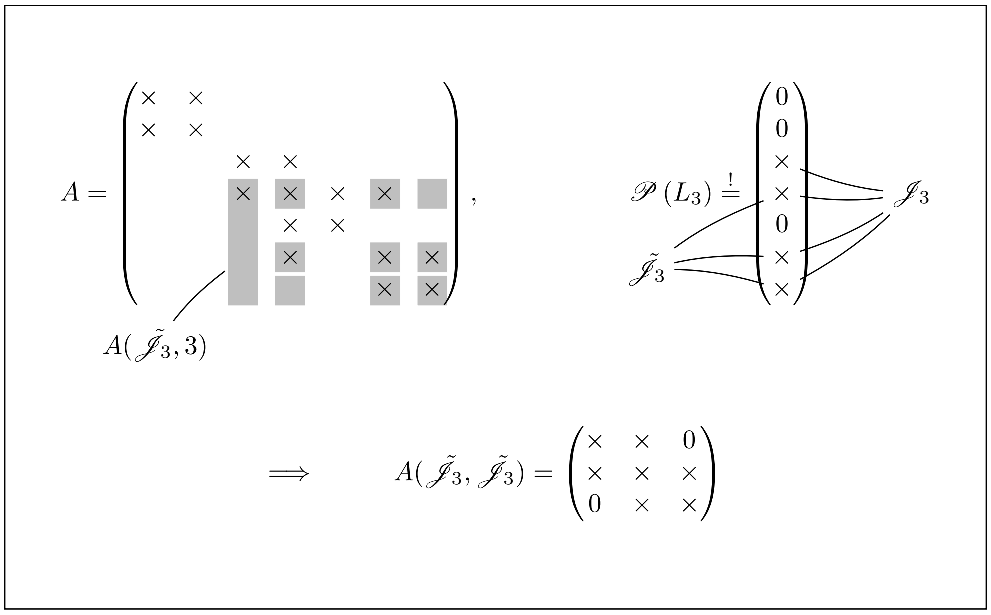

$A =$
$\mathscr{P}(L_3) \overset{!}{=}$
$\mathscr{J}_3$
$\tilde{\mathscr{J}}_3$
$A(\tilde{\mathscr{J}}_3, 3)$
$\Longrightarrow \quad A(\tilde{\mathscr{J}}_3, \tilde{\mathscr{J}}_3) = \begin{pmatrix} \times & \times & 0 \\ \times & \times & \times \\ 0 & \times & \times \end{pmatrix}$

Figure 2.5: $A(\tilde{\mathscr{J}}_k, \tilde{\mathscr{J}}_k)$ is defined by the index sets $\mathscr{J}_k$ and $\tilde{\mathscr{J}}_k$ (here is $\mathscr{J}_3 = \{3,4,6,7\}$ and $\tilde{\mathscr{J}}_k = \{4,6,7\}$).

2.2.1 Computation of FSPAI for Fixed Pattern

According to [62], we denote the k-th column of L by L_k and the allowed nonzero pattern in L_k by $\mathscr{J}_k$ (where $\tilde{\mathscr{J}}_k := \mathscr{J}_k \setminus \{k\}$), and get

$$\begin{aligned}
\frac{\operatorname{tr}\left(L^T A L\right)}{\det\left(L^T A L\right)^{\frac{1}{n}}} &= \frac{\sum_{k=1}^{n} L_k^T A L_k}{\det(A)^{\frac{1}{n}} \left(L_{11} L_{22} \cdots L_{nn}\right)^{\frac{2}{n}}} \\
&= \frac{\sum_{k=1}^{n} L_{kk}^2 A_{kk} + 2 L_{kk} L_k(\tilde{\mathscr{J}}_k)^T A(\tilde{\mathscr{J}}_k, k)}{\det(A)^{\frac{1}{n}} \left(L_{11} L_{22} \cdots L_{nn}\right)^{\frac{2}{n}}} \\
&\quad + \frac{L_k(\tilde{\mathscr{J}}_k)^T A(\tilde{\mathscr{J}}_k, \tilde{\mathscr{J}}_k) L_k(\tilde{\mathscr{J}}_k)}{\det(A)^{\frac{1}{n}} \left(L_{11} L_{22} \cdots L_{nn}\right)^{\frac{2}{n}}}.
\end{aligned} \tag{2.21}$$

We obtain the unknowns $L_k(\mathscr{J}_k)$ (see also Figure 2.5) through the derivatives of (2.21) with respect to $\tilde{\mathscr{J}}_k$ as

$$\begin{aligned}
L_k(\tilde{\mathscr{J}}_k) &= -L_{kk} A(\tilde{\mathscr{J}}_k, \tilde{\mathscr{J}}_k)^{-1} A(\tilde{\mathscr{J}}_k, k), \\
L_{kk}^2 &= \frac{1}{A_{kk} - A(\tilde{\mathscr{J}}_k, k)^T A(\tilde{\mathscr{J}}_k, \tilde{\mathscr{J}}_k)^{-1} A(\tilde{\mathscr{J}}_k, k)}.
\end{aligned}$$

Here we can see that L can be computed column-wise in parallel, since the unknowns for each column k are completely independent of each other. The actual solution takes three steps:

$$\begin{aligned} y_k &= A(\tilde{\mathscr{J}}_k, \tilde{\mathscr{J}}_k)^{-1} A(\tilde{\mathscr{J}}_k, k), && (2.22a) \\ L_{kk} &= \frac{1}{\sqrt{A_{kk} - A(\tilde{\mathscr{J}}_k, k)^T y_k}}, && (2.22b) \\ L_k(\tilde{\mathscr{J}}_k) &= -L_{kk} y_k. && (2.22c) \end{aligned}$$

The small submatrix $A(\tilde{\mathscr{J}}_k, \tilde{\mathscr{J}}_k)$ of A is symmetric positive definite, too, so we can solve the linear system (2.22a) using Cholesky decomposition. This is the dominating part of the computational effort, when we compute an L for one fixed pattern, since Cholesky factorization is $\mathscr{O}\left(n^3\right)$. For the FSPAI L, we also do not need an additional normalization step as in [75], because $\operatorname{diag}\left(L^T AL\right) = I$ by construction. Kolotilina [74] proposed several strategies for prescribing promising sparsity patterns in order to statically compute of a factorized sparse approximate inverse factor. These approaches can also be applied to static FSPAI or can give valuable start patterns for the dynamic case, which involves pattern updates.

Remark 2.6 *If we only prescribe a diagonal sparsity pattern for a static FSPAI, we obtain, by equation (2.22b), a diagonal matrix with entries $L_{kk} = 1/\sqrt{A_{kk}}$, $k = 1, \ldots, n$. Hence, diagonal FSPAI preconditioning $L^T AL$ is equivalent to a symmetric Jacobi preconditioner.*

Remark 2.7 *In [60], Huckle investigated FSPAI's smoothing properties for a finite difference discretization of a 2D elliptic PDE. FSPAI is much easier to parallelize than an ILU preconditioner employed as a smoother, whereas ILU showed the best smoothing results in that test.*

For large sparse spd matrices, FSPAI provides a much better preconditioning method than SPAI. The submatrices which occur in FSPAI are smaller and faster to factorize than the ones in the SPAI algorithm. Moreover, FSPAI can use Cholesky decomposition, which is faster by a factor of 4 than QR decomposition in the SPAI case. FSPAI also computes a triangular factor L, which halves the costs for storing compared to a SPAI with equal sparsity pattern. All these advantages can be seen in the following example.

Example 2.2 The matrix bcsstk14, which is taken from Matrix Market [77], has dimension $n = 1806$. The condition number is $\kappa(A) \approx 1.19 \cdot 10^{10}$ and $\texttt{nnz}(A) = 63454$. We compute a static SPAI M and a static FSPAI L with the pattern $\mathscr{P}(A)$ of the original matrix and the lower triangular part of $\mathscr{P}(A)$, respectively. This test has been performed in MATLAB on a 1600 MHz Intel Centrino laptop. Table 2.1 shows the simulation results. FSPAI needs only half the memory of SPAI and produces a better approximation as far as Frobenius norm minimization and condition number of the preconditioned system are concerned. The setup time for the SPAI is of more than factor 9 higher than for FSPAI. Moreover, we need to use BiCGSTAB for SPAI, because we cannot ensure positive definiteness, not even symmetry for M. On the one hand, this leads to more expensive iterations and, due to the lower approximation quality, also to more than twice as many iterations as the FSPAI. The total simulation time using FSPAI is less by a factor of more than 7.7 compared to SPAI. For this test, no pattern updates were performed.

Table 2.1: Comparison of SPAI and FSPAI for the test matrix bcsstk14. Solution of $Ax = b$ with random right-hand side b and a relative residual of 10^{-6} as stopping criterion in the iterative solver.

	SPAI	FSPAI
`nnz`	63454	32630
memory usage ([Bytes])	768748	398788
$\|AM - I\|_F$ resp. $\left\|L^T AL - I\right\|_F$	17.21	12.83
$\kappa(AM)$ resp. $\kappa(L^T AL)$	4.31e+03	4.02e+02
setup time for M/L ([s])	5.62	0.62
solver time for BiCGSTAB/pcg ([s])	1.26	0.27
total time ([s])	6.88	0.89
# iterations	115	54

A further example application for the use of FSPAI with a static pattern is given in Section 5.1. The sparsity pattern has been determined through prescaling and sparsification steps applied to the underlying matrix which arose in statics' simulation.

2.2.2 Pattern Updates

Analogously to SPAI, FSPAI's specialty lies in identifying new promising entries for an augmented sparsity pattern. The approaches in the two methods are quite similar and lead to a 1D minimization problem for each new index in the pattern (see [62]). We define

$$L_k^{(j)} := L_k + \lambda_j e_j$$

with the j-th unit vector e_j. Solving

$$\min_{\lambda_j} \frac{\frac{1}{n}\mathrm{tr}\left((L^T + \lambda_j e_k e_j^T)A(L + \lambda_j e_j e_k^T)\right)}{\det\left((L^T + \lambda_j e_k e_j^T)A(L + \lambda_j e_j e_k^T)\right)^{\frac{1}{n}}}$$

yields the minimum value

$$\lambda_j = -\frac{A_j^T L_k}{A_{jj}}.$$

Just as in the SPAI method, we observe the reduction of the functional (2.20)

$$\frac{\frac{1}{n}}{\det(A)^{\frac{1}{n}}\,(L_{11}\cdots L_{nn})^{\frac{2}{n}}} \cdot \tau_j$$

and identify a factor τ_j for each λ_j:

$$\tau_j = \frac{\left(A(j, \mathscr{J}_k)L_k(\mathscr{J}_k)\right)^2}{A_{jj}}.$$

The bigger the value of τ_j is, the more the Kaporin functional will be reduced, when the j-th entry is added to the pattern $\mathscr{J}_k$. We do not need to compute τ_j for every $j = 1, \ldots, n$. Graph algorithms working on the graph representing the sparsity structure of A enable us to find indices j such that

$$A(j, \mathscr{J}_k)L_k(\mathscr{J}_k) \neq 0.$$

Thus, the number of candidates for which we have to compute τ_j in general reduces significantly due to the sparsity of A. The candidates with the biggest τ_j-values can then be added to $\mathscr{J}_k$. In order to keep the density of L low, we can also add only the indices which correspond to the τ_j which are larger than the mean value of all τ_j. This criterion was already proposed for SPAI in Section 2.1.2. And again, just like for SPAI, we do not have to compute the Cholesky factorization of the enlarged $A(\tilde{\mathscr{J}}_k, \tilde{\mathscr{J}}_k)$, but we can perform Cholesky updates (see, for instance [46, 62]). The pattern update steps are carried out until the maximum value of all τ_j is lower than a user provided tolerance ε.

We can also identify new candidates for the pattern of L by multivariate minimization instead of the 1D ansatz above. Besides higher computational costs, we cannot observe a better recovery of the inverse of A as it was observed for SPAI. For a comparison, refer to [62].

2.2.3 Effect of Approximate Minimum Degree Reordering on FSPAI

The sparse approximate inverse preconditioner SPAI is invariant under permutations, as we have seen in Section 2.1. This is not the case for FSPAI. Here, graph reorderings do have influence on the result. In Section 1.3.3, we have already seen that reorderings of minimum-degree-type substantially decrease the height of the elimination tree in direct factorization methods. This leads to sparser matrix factors. Benzi and Tůma also proposed the use of minimum degree reorderings in [13] and performed a deeper analysis thereof in [15], considering the AINV preconditioner. For the AINV algorithm, Benzi observes a significant decrease in the nnz of the inverse factors. He refers to that as *inverse fill-in*. Sparsification steps during the biorthogonalization lead to this reduction and due to the preordering, more entries are dropped and set to zero than without preordering.

Table 2.2: Comparison of FSPAI for AMD preordered and unpreordered test matrices arising from Navier-Stokes equations. FSPAI was computed with $\varepsilon = 0.2$ and the pcg algorithm iterated up to a relative residual of 10^{-6} as stopping criterion.

	without preordering		AMD preordering		
n	#iter.	$\mathtt{nnz}(L)$	#iter.	$\mathtt{nnz}(\hat{L})$	$\frac{\mathtt{nnz}(\hat{L})}{\mathtt{nnz}(L)}$
216	20	2547	20	2454	0.96
512	27	7181	26	6467	0.90
1000	37	15631	42	13857	0.89
2197	54	37981	48	32935	0.87
4097	67	75397	68	64574	0.86

We concentrate on the approximate minimum degree (AMD) algorithm, which has already been described in Section 1.3.3. Our test matrices arise from a 3D discretization of the Navier-Stokes equations. The discretization is done on cartesian grids and preserves both energy and impulse. After a Chorin projection (see [26]), we obtain symmetric semidefinite matrices. The geometry is a channel which is divided into $n_x \cdot n_y \cdot n_z = n$ cells in the three spatial dimensions. Thus, the discretization matrices are of size n. This number grows rapidly as we refine the discretization grid in order to increase accuracy.

With these matrices of different dimensions, we investigate the effect of an AMD preordering step on the FSPAI pattern updates. As a parameter, we take a tolerance $\varepsilon = 0.2$ for the accuracy in each column. The start pattern is diagonal, and we add one new entry per column in each update step. As we can see in Table 2.2, the AMD preordering has only little influence on the iteration numbers of the pcg, because symmetric orderings are represented by orthonormal permutation matrices and do not affect the condition number. But in the last column, we see that the density of the FSPAI $\hat{L}$ in the preordered case is reduced up to nearly 15%, i.e. the tolerance ε is achieved after fewer update steps in each column. Thus, an AMD preordering leads to sparser factorized approximate inverses and therefore to lower computation times in each iteration of the solver. Additionally, the FSPAI computation time reduces due to the saving of pattern update steps. This observation coincides with Benzi's results on a reduced inverse fill-in.

The test matrices in Table 2.2 arise from simulations of an empty channel. We provide a more detailed review of our results involving different settings with obstacles in the channel in Section 5.2.

2.3 M-Matrices

To conclude this chapter, we want to mention a property, when the coefficient matrix A belongs to an important special class of matrices, the so-called *M-matrices*:

Definition 2.2 (M-Matrix) *A matrix $A \in \mathbb{R}^{n\times n}$ is called* M-matrix *if it is nonsingular, if non-positive values are exclusively off the main diagonal, i.e.*

$$a_{ij} \begin{cases} > 0 & \text{for} \quad i = j, \\ \leq 0 & \text{for} \quad i \neq j \end{cases}, \qquad i, j = 1, \ldots, n,$$

and if the inverse A^{-1} has only non-negative entries.

Moreover, for an M-matrix A and $x, y \in \mathbb{R}^n$ we have *inverse monotony*, i.e.

$$Ax \leq Ay \qquad \Longrightarrow \qquad x \leq y, \tag{2.23}$$

where the relation $\leq$ is taken componentwise. Using the properties of M-matrices, we can formulate the following theorem:

Theorem 2.5 *For an spd M-matrix $A \in \mathbb{R}^{n\times n}$, the FSPAI L has only non-negative entries, i.e.*

$$L \geq 0,$$

where $\geq$ is used componentwise.

Proof We restrict the proof to the k-th column L_k of L, $k = 1, \ldots, n$. According to Section 2.2.1, $\mathscr{J}_k$ denotes the set of indices in the prescribed pattern for L_k and $\tilde{\mathscr{J}}_k = \mathscr{J}_k \setminus \{k\}$. In the computation of L_k, we first solve $A(\tilde{\mathscr{J}}_k, \tilde{\mathscr{J}}_k)y_k = A(\tilde{\mathscr{J}}_k, k)$, see (2.22a). Since A is an M-matrix, the spd submatrix $A(\tilde{\mathscr{J}}_k, \tilde{\mathscr{J}}_k)$ is also an M-matrix. Moreover, $\tilde{\mathscr{J}}_k$ does not contain the diagonal index k. Hence, $A(\tilde{\mathscr{J}}_k, k)$ is a vector with only non-positive entries, and inverse monotony (2.23) induces

$$A(\tilde{\mathscr{J}}_k, \tilde{\mathscr{J}}_k)y_k = A(\tilde{\mathscr{J}}_k, k) \leq 0 \qquad \Longrightarrow \qquad y_k \leq 0.$$

From (2.22b), we see that $L_{kk} = 1/\sqrt{A_{kk} - A(\tilde{\mathscr{J}}_k, k)^T y_k} > 0$ and equation (2.22c)

$$L_k(\tilde{\mathscr{J}}_k) = -L_{kk}y_k \geq 0$$

completes the proof. $\square$

This property does not hold for SPAI, as the following counter-example shows. Consider the M-matrix

$$A = \begin{pmatrix} 10 & -1 & -4 & & \\ -1 & 10 & -1 & -4 & \\ -4 & -1 & 10 & -1 & -4 \\ & -4 & -1 & 10 & -1 \\ & & -4 & -1 & 10 \end{pmatrix}.$$

For the (static) SPAI M with tridiagonal pattern, the numerical solution we obtain is

$$M = \begin{pmatrix} 0.0859 & 0.0056 & & & \\ 0.0032 & 0.0859 & \boxed{-0.0028} & & \\ & 0.0035 & 0.0741 & 0.0035 & \\ & & \boxed{-0.0028} & 0.0859 & 0.0032 \\ & & & 0.0056 & 0.0859 \end{pmatrix},$$

which has two negative entries, whereas the exact inverse may only have non-negative components. However, the FSPAI L for the same pattern yields a completely non-negative solution

$$L = \begin{pmatrix} 0.3178 & & & & \\ 0.0318 & 0.3178 & & & \\ & 0.0318 & 0.3178 & & \\ & & 0.0318 & 0.3178 & \\ & & & 0.0318 & 0.3162 \end{pmatrix}.$$

Chapter 3

Modified SPAI

In this chapter, we will combine some well-known preconditioning techniques to the *Modified Sparse Approximate Inverse* (*MSPAI*). At first, we review the idea of probing, a rather simple approach to generate explicit matrix approximations with respect to some additional constraints such as preservation of the row sum. Another type of preconditioners with this property builds the class of modified factorizations. The disadvantages are the restriction to explicit factorizations on the one hand and the restriction to the vector of all ones $(1, \ldots, 1)^T$ as probing information on the other hand.

In order to overcome these restrictions, we generalize the SPAI-like Frobenius norm minimization from the previous chapter in Section 3.3 by extending it to target form [57] and then adding probing constraints. This modification then allows us to add an arbitrary number of arbitrary probing vectors to the preconditioner. By a weighting factor, we control how strongly they are taken into account during the computation. Therefore, this MSPAI probing can be regarded as both a generalization of the modified preconditioners and as a regularization of the classic probing technique. Moreover, our MSPAI method is still embarrassingly parallel, whereas parallelization is a quite demanding task for modified preconditioners in general due to their sequential nature.

We present formulations for explicit and inverse approximations, as well as factorized and unfactorized variants, and the application to Schur complements. The latter is of particular interest, since Chan and Mathew [24] developed the probing technique especially for preconditioning Schur complements which arise in domain decomposition methods. Furthermore, the factorized variants allow us to add probing information to any given factorization and presumably improve them.

Section 3.4 will then give answers to the question which probing vectors can or should be chosen in order to get improvements. After that, we provide some theoretical results in Section 3.5, namely an analytic solution for a special MSPAI setting which allows to state a purely structural requirement for the probing vectors and the prescribed sparsity pattern.

The final section deals with the problem of symmetrizing given preconditioners. The ones from Frobenius norm minimization are in general unsymmetric, so we need to symmetrize them in order to be able to use iterative solvers for symmetric coefficient matrices. We state both unfactorized and factorized methods addressing this task.

The whole chapter presents original research, and parts of it were published in [63].

3.1 Probing

The probing technique was introduced for preconditioning *interface matrices* in domain decomposition methods (see Chan and Mathew [24], Axelsson [7], or more recently Siefert and Sturler [88]). There, the interface matrix is a Schur complement S, whose condition number needs to be improved. As a preconditioner for S one defines e.g. a band matrix M that shows the right behavior on certain subspaces or probing vectors e_j. This results in $Me_j = Se_j$ $(j = 1, ..., k)$. As probing vectors one has to choose very special vectors, e.g. $e_1 = (1, 0, 0, 1, 0, 0, 1, ...)^T$, $e_2 = (0, 1, 0, 0, 1, 0, 0, 1, ...)^T$, and $e_3 = (0, 0, 1, 0, 0, 1, ...)^T$, which are collected in $E = (e_1, e_2, e_3)$. This choice results in simple equations for computing the preconditioner M:

$$M \cdot E = \begin{pmatrix} m_{11} & m_{12} & & & \\ m_{21} & m_{22} & m_{23} & & \\ & m_{32} & m_{33} & m_{34} & \\ & & \ddots & \ddots & \ddots \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ \vdots & \vdots & \vdots \end{pmatrix} = \begin{pmatrix} m_{11} & m_{12} & 0 \\ m_{21} & m_{22} & m_{23} \\ m_{34} & m_{32} & m_{33} \\ m_{44} & m_{45} & m_{43} \\ \vdots & \vdots & \vdots \end{pmatrix}. \tag{3.1}$$

From this equation one can read off the entries of M by comparison with the prescribed vectors $Me_j = Se_j$ such that

$$M \cdot E \overset{!}{=} S \cdot E.$$

Note that the resulting M will usually be unsymmetric and that the probing equation $Me_j = Se_j$ will not be satisfied in view of the band structure of M, e.g. the first component in Se_3 may be not zero. In the special case of only one probing vector and M being a diagonal matrix, we get $M = \operatorname{diag}(Se)$.

Axelsson and Polman have improved this approach by introducing the *method of action* [7], which leads to an spd preconditioner in some cases. The advantage of these methods is that without explicit knowledge of the problem it is possible to obtain a preconditioner that imitates the behavior of the original problem on certain subspaces. The disadvantages are that one has to choose special probing vectors that lead to an easy-to-solve system for M; furthermore, it is difficult to ensure special properties such as symmetry or positive definiteness.

3.2 Modified Incomplete Factorizations

The *modified incomplete factorizations* (see also Section 1.5.1) were introduced by Gustafsson [49] and by Axelsson [4] as an improvement of the ILU algorithm. In the ILU method, the Gaussian elimination process is restricted to a certain sparsity pattern. Therefore, new entries appearing on certain not allowed positions in U are deleted. The result is an approximate factorization $A = LU + R$, where R contains the neglected entries. The aim of the modified approach is again to imitate the action of the given matrix on a certain subspace. In this case, the subspace is generated by the vector of all ones. In PDE examples, this vector plays an important role in the sense that the discretized problem leads to a matrix where $A \cdot (1, 1, ..., 1)^T$ is zero up to the boundary conditions. Hence, this vector can be used as an approximation of the subspace to small eigenvalues. If we force the preconditioner to coincide with the given problem on this subspace, we expect to improve the condition number of the preconditioned system.

In order to maintain the action of the preconditioner relative to the vector of all ones, we try to preserve the row sum of the triangular factors in the ILU method. Therefore, in the *modified ILU* (*MILU*) algorithm, we do not delete the entries on not-allowed positions. Instead, we add them to the related main diagonal entry, thereby maintaining the row sum. For PDE problems such as the 2D Laplacian this MILU improves the condition number significantly [5]. As a disadvantage, this approach is restricted to the vector of all ones, and the method is not very robust for more general problems.

3.3 Generalized Form of Frobenius Norm Minimization and Probing

As we have seen in the previous sections, it is worthwhile to introduce a probing or modified approach for more general problems. With this intention, we consider a SPAI-like Frobenius norm minimization in the most general form. This new form allows the approximation of an arbitrary rectangular matrix B, replacing the identity matrix I in SPAI form $\|AM - I\|_F$ (compare the target matrix SPAI form in [57]). Furthermore, we can replace the matrix A by any rectangular matrix C. Here, B and C should be sparse matrices, but we also allow a small number of dense rows in C and B. The matrix M, which we want to compute, has a prescribed sparsity pattern (it is also possible to update the sparsity pattern dynamically, e.g. by a SPAI-like approach, but we do not consider this dynamic method here). Technically, this causes no difference in the implementation compared with the original SPAI algorithm. Thus, for given matrices $C, B \in \mathbb{R}^{m\times n}$, we want to find a matrix $M \in \mathbb{R}^{n\times n}$ by solving the MSPAI minimization

$$\min_M \|CM - B\|_F^2 . \tag{3.2}$$

Analogously to SPAI-type algorithms, we solve the minimization (3.2) completely in parallel, by still computing M column-wise. Another property inherited from SPAI is the ability to prescribe an almost arbitrary sparsity pattern for M. Note that a meaningful pattern for M should take into account the sparsity structure of C and of B as well. This new approach also allows one to consider lower or upper triangular matrices for M, C, and B.

As the main advantage of this general formulation, we can add further conditions to a given Frobenius norm minimization. This allows the inclusion of probing constraints in the general computation of a preconditioner. Consider the given minimization problem

$$\min_M \|C_0 M - B_0\|_F^2$$

e.g. with $C_0 = A$ and $B_0 = I$ for approximating A^{-1}, or $C_0 = I$ and $B_0 = A$ for approximating A. Then for any given set of probing vectors collected in the rectangular matrix $e \in \mathbb{R}^{n\times k}$, we can add probing conditions $\min_M \|e^T(C_0M - B_0)\|_2 = \min_M \|g^T M - h^T\|_2$ with $g^T := e^T C_0$ and $h^T := e^T B_0$ in the form

$$\min_M \left\| \begin{pmatrix} C_0 \\ \rho g^T \end{pmatrix} M - \begin{pmatrix} B_0 \\ \rho h^T \end{pmatrix} \right\|_F^2 , \qquad 0 \le \rho \in \mathbb{R} \tag{3.3}$$

with a weight ρ. The number ρ determines whether to emphasize the original norm minimization or to the additional probing condition. The k probing vectors, stored in e, are added row-wise to the corresponding matrices C_0 and B_0 in form of the matrices $g, h \in \mathbb{R}^{n\times k}$. This can be regarded as a generalization of a Frobenius norm approximation trying to improve the preconditioner on a certain subspace given by e.

Likewise, this can be seen as a regularization technique for a general probing approach. In standard probing, we are only able to consider probing vectors and sparsity patterns of M that lead to easy-to-solve linear systems for computing the entries of M from the probing conditions. In this new approach, we can choose any probing vectors e and sparsity patterns of M that also lead to under- or overdetermined linear systems due to the embedding of the probing method into the general matrix approximation setting.

In the given original problem $\min_M \|C_0 M - B_0\|_F$, it is also possible to replace the Frobenius norm by any other norm, e.g. by adding a weight matrix (as considered in [75]) $\min_M \|W(C_0 M - B_0)\|_F$. By choosing the rectangular matrix

$$W := \begin{pmatrix} I \\ \rho e^T \end{pmatrix} \tag{3.4}$$

as weight, we end up with the same generalized MSPAI norm minimization (3.3).

3.3.1 Sparse Approximate Inverses and Probing

As a first application, we use the method described above to add some probing information to a SPAI, i.e. an unfactorized approximation of the inverse A^{-1} of a given matrix $A \in \mathbb{R}^{n\times n}$: We choose $C_0 = A$ in (3.3), and the n-dimensional identity matrix I for B_0. Here, h becomes the given probing vector e, and accordingly $g = e^T A$. With that, we obtain the MSPAI Frobenius norm minimization

$$\min_M \left\| \begin{pmatrix} A \\ \rho e^T A \end{pmatrix} M - \begin{pmatrix} I \\ \rho e^T \end{pmatrix} \right\|_F^2 = \min_M \|W(AM - I)\|_F^2 \tag{3.5}$$

with weight matrix (3.4). The result of this method is an approximation M to the inverse A^{-1}, which satisfies

$$g^T = e^T A \quad \overset{M\approx A^{-1}}{\Longrightarrow} \quad g^T M \approx e^T.$$

In many applications, the matrix A is not given explicitly but only via a sparse approximation $\tilde{A}$. Nevertheless, we assume to be able to compute $g^T = e^T A$ exactly. In this situation, we modify the above minimization to

$$\min_M \left\| \begin{pmatrix} \tilde{A} \\ \rho e^T A \end{pmatrix} M - \begin{pmatrix} I \\ \rho e^T \end{pmatrix} \right\|_F^2 . \tag{3.6}$$

Then probing is used to derive a sparse approximate inverse M for $\tilde{A} \approx A$ with respect to the exact probing conditions $e^T AM = e^T$, where $\tilde{A}$ is an approximation of the original matrix A.

3.3.2 Explicit Approximation and Probing

With our approach, we can not only add constraints to the approximation of the inverse A^{-1} of a given matrix, but also to explicit approximations of A. This can be used if A is (almost) dense in order to get a sparse approximation on A. In contrast to the last section, we set $C_0 = I$ and $B_0 = A$ with probing vector $g = e$. Consequently, we choose $h^T = e^T A$. Altogether, this yields

$$\min_M \left\| \begin{pmatrix} I \\ \rho e^T \end{pmatrix} M - \begin{pmatrix} A \\ \rho e^T A \end{pmatrix} \right\|_F^2 = \min_M \|W(M - A)\|_F^2 \tag{3.7}$$

with the above weight matrix W (3.4). Now, we get a sparse approximation M of A which also tries to reflect the properties of A on certain vectors:

$$h^T = e^T A \approx e^T M.$$

In many cases, A is again only given implicitly or (almost) dense, and we have to use a sparse approximation $\tilde{A}$. This leads to the MSPAI problem

$$\min_M \left\| \begin{pmatrix} I \\ \rho e^T \end{pmatrix} M - \begin{pmatrix} \tilde{A} \\ \rho e^T A \end{pmatrix} \right\|_F^2 . \tag{3.8}$$

Note that this explicit approach only makes sense if linear equations in M can be solved easily, as the inverse of M has to be used for preconditioning.

3.3.3 Explicit Factorized Approximation and Probing

The MSPAI probing approach does not only allow one to improve unfactorized methods, but also preconditioners which are computed in a factorized form such as ILU or incomplete Cholesky. Here, we assume an already computed factored approximation $A \approx LU$, e.g. given by ILU or the Gauss-Seidel method. The aim is to improve the factors with respect to the given probing conditions. Thereby, we keep one factor (L or U) fixed and recompute the other factor. That means, we set in (3.3) $B_0 = A$, $C_0 = L$ and for the probing constraints $g^T = e^T L$, $h^T = e^T A$. In order to get an upper (respectively lower) triangular factor, we restrict the pattern for M to upper (lower) triangular form. Altogether, we solve

$$\min_{\tilde{U}} \left\| \begin{pmatrix} L \\ \rho e^T L \end{pmatrix} \tilde{U} - \begin{pmatrix} A \\ \rho e^T A \end{pmatrix} \right\|_F^2 = \min_{\tilde{U}} \left\| W(L\tilde{U} - A) \right\|_F^2 . \tag{3.9}$$

Having computed this improved version $\tilde{U}$ of U, we consider $A \approx L\tilde{U}$ with $\tilde{U}$ fixed. We obtain the ansatz for the improved lower triangular factor $\tilde{L}$ through transposition:

$$\min_{\tilde{L}} \left\| \begin{pmatrix} \tilde{U}^T \\ \rho e^T \tilde{U}^T \end{pmatrix} \tilde{L}^T - \begin{pmatrix} A^T \\ \rho e^T A^T \end{pmatrix} \right\|_F^2 = \min_{\tilde{L}} \left\| W(\tilde{U}^T \tilde{L}^T - A^T) \right\|_F^2 . \tag{3.10}$$

The probing constraints have to be adapted as well to $g^T = e^T \tilde{U}$ and $h^T = e^T A^T$. The result of these probing steps are the improved factorizations

$$A \approx L\tilde{U} \ , \text{ and } \ A \approx \tilde{L}\tilde{U}.$$

Note that the initial factorization $A \approx LU$ can be produced by any preconditioning technique, which yields factorized approximations of A.

As an important special case, we have to deal with the problem that A may only be given implicitly through an approximation $\tilde{A}$. We assume to be able to compute the exact values for e^TA and e^TA^T. Then we have to modify the minimization to the form

$$\min_{\tilde{U}} \left\| \begin{pmatrix} L \\ \rho e^T L \end{pmatrix} \tilde{U} - \begin{pmatrix} \tilde{A} \\ \rho e^T A \end{pmatrix} \right\|_F^2 , \tag{3.11}$$

or

$$\min_{\tilde{L}} \left\| \begin{pmatrix} \tilde{U}^T \\ \rho e^T \tilde{U}^T \end{pmatrix} \tilde{L}^T - \begin{pmatrix} \tilde{A}^T \\ \rho e^T A^T \end{pmatrix} \right\|_F^2 . \tag{3.12}$$

3.3.4 Approximating a Factorization of A^{-1}

In contrast to the section above, we start with a given sparse factorized approximation of the inverse $A^{-1} \approx UL$ or $UAL \approx I$. U and L may result from some algorithm like AINV or FSPAI. Again, we add the probing conditions and substitute $B = I$, $C = UA$, $g^T = e^TUA$ and $h^T = e^T$ in formula (3.3) from above. This gives the following MSPAI probing problem:

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

To ensure that $\tilde{L}$ has lower triangular sparsity pattern, we restrict the pattern to that form. Again, the result is an enhanced new factor $\tilde{L}$. As in the explicit factorized case before, we apply the same method once again to the transposed problem, in order to obtain an improved $\tilde{U}$:

$$\min_{\tilde{U}} \left\| \begin{pmatrix} L^TA^T \\ \rho e^T L^T A^T \end{pmatrix} \tilde{U}^T - \begin{pmatrix} I \\ \rho e^T \end{pmatrix} \right\|_F^2 = \min_{\tilde{U}} \left\| W(L^TA^T\tilde{U}^T - I) \right\|_F^2 .$$

For an implicitly given matrix A, we again have to use an approximation $\tilde{A}$ and arrive at

$$\min_{\tilde{L}} \left\| \begin{pmatrix} U\tilde{A} \\ \rho e^T UA \end{pmatrix} \tilde{L} - \begin{pmatrix} I \\ \rho e^T \end{pmatrix} \right\|_F^2 .$$

3.3.5 Application to Schur Complements

The MSPAI probing approach is especially interesting for preconditioning Schur complements. Therefore, we consider the Frobenius norm minimization method more deeply. Given a matrix

$$H = \begin{pmatrix} A & B \\ C & D \end{pmatrix} ,$$

we consider preconditioning the Schur complement $S_D = D - CA^{-1}B$. In a first step, we need approximations of S_D which avoid the explicit computation of this matrix. In order to achieve this, we use different methods, e.g. factorized SPAI or SPAI for approximating A^{-1} by some $\tilde{M}$ and computing the sparse approximation $\tilde{S}_D = D - C\tilde{M}B$, or using SPAI with target matrix for a sparse approximation of $A^{-1}B$ by $\min_M \|AM - B\|_F$. This also leads to a sparse approximation for the Schur complement. Based on $\tilde{S}_D$ we employ the probing approach to define a sparse approximation on $\tilde{S}_D$ or $\tilde{S}_D^{-1}$ which is improved with respect to a collection of probing vectors relative to the exact Schur complement.

A second approach can be derived by observing that the lower left block in the inverse of matrix H is the inverse of the Schur complement S_D:

$$H^{-1} = \begin{pmatrix} S_A^{-1} & -A^{-1}BS_D^{-1} \\ -D^{-1}CS_A^{-1} & S_D^{-1} \end{pmatrix} .$$

Therefore, we modify the general probing approach (3.3) to

$$\min_{M_B, M_D} \left\| \begin{pmatrix} A & B \\ C & D \\ 0 & \rho e^T S_D \end{pmatrix} \cdot \begin{pmatrix} M_B \\ M_D \end{pmatrix} - \begin{pmatrix} 0 \\ I \\ \rho e^T \end{pmatrix} \right\|_F^2 . \tag{3.13}$$

Then the computed M_D gives an approximation to S_D^{-1}, because the last columns of H^{-1} are approximated by M_B and M_D.

We formulate another method for computing M_D based on the weight matrix W. By using the equation

$$\begin{pmatrix} \rho u^T & \rho e^T \end{pmatrix} \cdot \begin{pmatrix} A & B \\ C & D \end{pmatrix} \cdot \begin{pmatrix} -A^{-1}BS_D^{-1} \\ S_D^{-1} \end{pmatrix} = -\rho e^T CA^{-1}BS_D^{-1} + \rho e^T DS_D^{-1} = \rho e^T ,$$

with an additional vector u, we again arrive at the probing minimization

$$\min_{M_B, M_D} \left\| \begin{pmatrix} & I \\ \rho u^T & \rho e^T \end{pmatrix} \cdot \left[\begin{pmatrix} A & B \\ C & D \end{pmatrix} \cdot \begin{pmatrix} M_B \\ M_D \end{pmatrix} - \begin{pmatrix} 0 \\ I \end{pmatrix} \right] \right\|_F^2 , \tag{3.14}$$

with M_D as approximate inverse for the Schur complement. Note that for $u^T = -e^T CA^{-1}$ (3.13) and (3.14) are identical. Hence, problem (3.14) is more general, but in the following, we will only consider (3.13).

Box 3.1 on page 56 gives a concise overview of the several MSPAI probing variants above. The employment in numerical examples is presented in Chapter 5, whereas efficient implementation approaches take place in Chapter 4.

3.4 Probing Vectors

The actual choice of the probing vectors is crucial for the quality of the resulting preconditioner. We have to carefully choose the subspaces, on which the preconditioner should coincide with the action of the given matrix. This question is closely related to the origin of the given problem, and further a priori knowledge may be helpful.

3.4.1 Standard Choices of Probing Vectors

As possible choices for the probing vectors, which are also used in the numerical examples in Chapter 5, we present three different variants.

MSPAI Probing Variants

Basic formulation for general matrices $C_0, B_0 \in \mathbb{R}^{n\times n}$, probing vectors $e \in \mathbb{R}^{n\times k}$, and weight $\rho \geq 0$ in order to obtain MSPAI $M \in \mathbb{R}^{n\times n}$:

$$\min_M \left\| \begin{pmatrix} C_0 \\ \rho e^T C_0 \end{pmatrix} M - \begin{pmatrix} B_0 \\ \rho e^T B_0 \end{pmatrix} \right\|_F^2 \quad \Longrightarrow \quad \begin{array}{rcl} C_0 M & \approx & B_0 \\ e^T C_0 M & \approx & e^T B_0 \end{array} ,$$

which is equivalent to $\min_M \|W(C_0M - B_0)\|_F^2$ with weight matrix $W = \begin{pmatrix} I \\ \rho e^T \end{pmatrix}$.

Sparse Approximate Inverse Probing

$$\min_M \left\| \begin{pmatrix} A \\ \rho e^T A \end{pmatrix} M - \begin{pmatrix} I \\ \rho e^T \end{pmatrix} \right\|_F^2 \quad \Longrightarrow \quad \begin{array}{rcl} M & \approx & A^{-1} \\ e^T A M & \approx & e^T \end{array} .$$

If A is not given explicitly, set $C_0 = \tilde{A}$, where $\tilde{A}$ is some sparse approximation to A, but still compute $e^T A$ exactly. For given approximate inverse factorization $UAL \approx I$ or $UL \approx A^{-1}$:

$$\min_{\tilde{L}} \left\| \begin{pmatrix} UA \\ \rho e^T UA \end{pmatrix} \tilde{L} - \begin{pmatrix} I \\ \rho e^T \end{pmatrix} \right\|_F^2 \quad \Longrightarrow \quad \begin{array}{rcl} UA\tilde{L} & \approx & I \\ e^T UA\tilde{L} & \approx & e^T \end{array} .$$

Obtain the corresponding probed upper triangular factor $\tilde{U}$ through transposition.

Explicit Probing

$$\min_M \left\| \begin{pmatrix} I \\ \rho e^T \end{pmatrix} M - \begin{pmatrix} A \\ \rho e^T A \end{pmatrix} \right\|_F^2 \quad \Longrightarrow \quad \begin{array}{rcl} M & \approx & A \\ e^T M & \approx & e^T A \end{array} .$$

Again, for implicitly given A substitute B_0 by some sparse approximation $\tilde{A}$, but provide $e^T A$ exactly. In case of given explicit factorization $LU \approx A$:

$$\min_{\tilde{U}} \left\| \begin{pmatrix} L \\ \rho e^T L \end{pmatrix} \tilde{U} - \begin{pmatrix} A \\ \rho e^T A \end{pmatrix} \right\|_F^2 \quad \Longrightarrow \quad \begin{array}{rcl} L\tilde{U} & \approx & A \\ e^T L\tilde{U} & \approx & e^T A \end{array} .$$

Transposition yields probed lower triangular factor $\tilde{L}$.

Schur Complement Probing

MSPAI probing for block matrix $H = \begin{pmatrix} A & B \\ C & D \end{pmatrix}$:

$$\min_{M_B, M_D} \left\| \begin{pmatrix} A & B \\ C & D \\ 0 & \rho e^T S_D \end{pmatrix} \begin{pmatrix} M_B \\ M_D \end{pmatrix} - \begin{pmatrix} 0 \\ I \\ \rho e^T \end{pmatrix} \right\|_F^2 \quad \Longrightarrow \quad \begin{array}{rcl} M_D & \approx & S_D^{-1} \\ e^T M_D & \approx & e^T \end{array} ,$$

where S_D^{-1} denotes the lower right block of H^{-1}.

Box 3.1: Employment of MSPAI probing for different applications, where A in general denotes the coefficient matrix in $Ax = b$ and I the unity matrix with the respective dimension.

- **KP0**: For a given k we define

$$e_m(j) = 1 \text{ for } j = m, k+m, 2k+m, ..., \text{ and } m = 1, ..., k.$$

Then, we normalize the vectors e_m such that they are of length 1. For $k = 1$, this leads to the probing vector $e = (1, 1, ..., 1)^T/\sqrt{n}$, and for $k = 2$ to $e = (e_1, e_2)/\sqrt{n/2}$ with $e_1 = (1, 0, 1, 0, 1, 0, ...)^T$ and $e_2 = (0, 1, 0, 1, 0, 1, ...)^T$. This choice is motivated by PDE examples, by MILU, and by the usual probing vector approach.

- **KP1**: k vectors from a set of orthogonal basis vectors, e.g.

$$\sqrt{\frac{2}{n+1}}\left(\sin\left(\frac{\pi jm}{(n+1)}\right)\right)_{j=1,...,n} \quad \text{for } m = 1, ..., k. \tag{3.15}$$

This is motivated by the close relation between the Sine Transform and many examples from PDE or image restoration. Similarly, for 2D problems we can define the Kronecker product of these 1D vectors.

- **KP2**: Computing k eigenvector approximations relative to the largest and/or smallest eigenvalues (or singular vectors). This is some kind of black box approach when there is no a priori knowledge at hand. In this case, we have additional costs for computing eigenvector approximations. In most cases, however, very rough and cheap approximations are sufficient. Moreover, too exact approximations would make the preconditioner act on only one eigenvalue. A rough approximation is partly "blurred" and thus, acts not only on the one eigenvalue connected to it, but also on the eigenvalues located closely around. In the following, we choose eigenvectors to the smallest eigenvalues only. This may be a bad choice for instance in image restoration problems, as we have seen in Section 2.1.5. Here, we would need a preconditioner which acts in an optimal way on the large eigenvalues, because the small ones are related to the noise which corrupts the solution.

The weight ρ used in the probing Frobenius norm minimization is usually chosen $\rho > 1$. In cases where it is well known that the standard probing approach works accurately, large values for ρ lead to good preconditioners. So the choice of ρ indicates whether we consider the given problem as a regularized least squares problem related to given probing vectors (large ρ) or whether the minimization is used for a slight modification of a given preconditioner (ρ close to 1).

Sometimes it is impossible to satisfy the probing condition sufficiently. For the unfactorized or factorized approximate inverse approach, the probing vector $e = (1, ..., 1)^T$ leads to a sparse vector $g^T = e^T A$, e.g. for the discretization of the Laplace operator or other elliptic PDE problems. Therefore, for a sparse pattern of M, the vector $g^T M$ will be also sparse and cannot be a good approximation to the given dense vector e. In such cases, the above approach is only efficient for approximating A itself. For A^{-1} one has to choose ρ carefully. Otherwise, a slight improvement in the probing condition by selecting a much larger ρ will spoil the matrix approximation and result in a bad preconditioner. Section 3.5 will provide a deeper insight into this effect.

3.4.2 Graph Based Identification of Probing Vectors

The original probing method by Chan an Mathew [24] computes banded approximations which recover the largest entries in Schur complements without forming them explicitly.

These matrices arose from 1D interfaces in 2D non-overlapping domain decomposition methods. The key to the success of this banded approach lies in the special decay properties of these matrices. The largest values are located on the main diagonal and on the inner band, respectively. Thus, a banded approach clearly makes sense. However, there are many more general cases such as Navier-Stokes or metal deformation problems, where a simple band structure does not cover the largest values, which lie outside the band. For this class of problems, Siefert and Sturler [88] proposed a graph based approach to both determine a meaningful sparsity pattern and compute the adequate probing vectors. This algorithm of *structured probing* for a matrix $A \in \mathbb{R}^{n\times n}$ and an a priori chosen sparsity pattern $P \in \{0,1\}^{n\times n}$ consists of five steps (see [88]):

1. Compute a graph $\mathscr{G}$ derived from P.
2. Color the graph $\mathscr{G}$ to obtain a mapping $\phi : \{1,\dots,n\} \mapsto \{1,\dots,p\}$, where p is the number of colors. The color for vertex i is then given by $\phi(i)$.
3. Generate the matrix of probing vectors $e \in \{0,1\}^{n\times p}$ such that
$$e_{ij} = \begin{cases} 1 & \text{if } \phi(i) = j, \\ 0 & \text{otherwise.} \end{cases}$$
4. Compute $W = Ae$.
5. Read off the entries of the approximation M by
$$m_{ij} = \begin{cases} W_{i,\phi(j)} & \text{if } p_{ij} = 1, \\ 0 & \text{otherwise.} \end{cases}$$

The choice of P requires some knowledge about A, but not too detailed. Helpful information could be the structure of an underlying discretization mesh or just some indicators where large matrix entries lie. Large matrix entries are often related to locality on some graph. For the coloring of the graph related to the structure of P, Siefert and Sturler suggest, on the one hand, considering the adjacency graph and perform a distance-2 coloring or, on the other hand, using the column intersection graph in connection with a distance-1 coloring. In general, the latter graph type entails more computational effort to color it. Hence, one should use the adjacency graph. For the actual coloring, the authors present a few heuristics, e.g. a simple greedy distance-2 coloring algorithm, a balanced variant thereof, and a more elaborate prime divisor distance-2 coloring. The colored graph and the resulting mapping ϕ allow then to read off the entries of $W = Ae$ and put them into the correct positions in M. The numerical results in [88] show a much better recovery of large off-inner-band components than banded probing does. Hence, structured probing yields more accurate approximations and provides valuable alternatives for probing vectors in MSPAI.

3.5 Theoretical Results for MSPAI

We present a theoretical result for MSPAI probing considering the case $\rho \to \infty$ in the explicit factorized formulation. It establishes the basis for more general considerations in terms of a suitable choice of probing vectors and prescribing meaningful sparsity patterns for the preconditioner.

Theorem 3.1 *Let*

$$A = \begin{pmatrix} 2 & -1 & & \\ -1 & 2 & \ddots & \\ & \ddots & \ddots & -1 \\ & & -1 & 2 \end{pmatrix}$$

be a standard three-point stencil discretization of the 1D Laplacian with dimension n, and L the lower triangular part of A. For the resulting bidiagonal upper triangular matrix U from factorized explicit probing (3.9)

$$\lim_{\rho \to \infty} \rho e^T L U = e^T A$$

holds for all probing vectors

$$e \in \mathbb{R}^{n \times 1} : \begin{pmatrix} e_{j-1} \\ e_j \\ e_{j+1} \end{pmatrix} \notin \operatorname{span} \left\{ \begin{pmatrix} 1 \\ 2 \\ 4 \end{pmatrix} \right\} \text{ for } j = 2, \dots, n-1 \text{ and } (e_{n-1}, e_n) \neq (0,0), \tag{3.16}$$

i.e. LU satisfies the probing condition exactly.

Proof We explicitly compute the entries of U column-wise. The minimization problem reads

$$\min_U \left\| \begin{pmatrix} L \\ \rho e^T L \end{pmatrix} U - \begin{pmatrix} A \\ \rho e^T A \end{pmatrix} \right\|_F^2 =$$

$$\min_U \left\| \begin{pmatrix} 2 & & & \\ -1 & 2 & & \\ & \ddots & \ddots & \\ & & -1 & 2 \\ \hline & \rho e^T L & & \end{pmatrix} \begin{pmatrix} u_{11} & u_{12} & & \\ & \ddots & \ddots \end{pmatrix} - \begin{pmatrix} 2 & -1 & & \\ -1 & 2 & \ddots & \\ & \ddots & \ddots & -1 \\ & & -1 & 2 \\ \hline & \rho e^T A & & \end{pmatrix} \right\|_F^2$$

$$=: \min_U \|HU - K\|_F^2 .$$

The first column U_1 of U consists only of u_{11}, so we minimize $\|HU_1 - k_1\|_2^2$ with solution $u_{11} = \frac{h_1^T k_1}{\|h_1\|_2^2} = \frac{h_1^T h_1}{\|h_1\|_2^2} = 1$. Therefore, $e^T L U_1 = e^T a_1$ with a_1 being the first column of A.

For the columns $j = 2, \dots, n-1$, we define

$$\mathscr{N} := \|HU_j - k_j\|_2^2 = \langle HU_j, HU_j \rangle - 2 \langle HU_j, k_j \rangle + \|k_j\|_2^2 . \tag{3.17}$$

Using

$$U_j = \begin{pmatrix} 0 \\ \vdots \\ u_{j-1} \\ u_j \\ \vdots \\ 0 \end{pmatrix} \quad \Longrightarrow \quad LU_j = \begin{pmatrix} 0 \\ \vdots \\ 2u_{j-1} \\ -u_{j-1} + 2u_j \\ -u_j \\ \vdots \\ 0 \end{pmatrix},$$

we obtain

$$\rho e^T L U_j = \rho \left[\underbrace{(2e_{j-1} - e_j)}_{=:s_1} u_{j-1} + \underbrace{(2e_j - e_{j+1})}_{=:s_2} u_j \right] \tag{3.18}$$

and thus

$$\langle HU_j, HU_j \rangle = (5 + \rho^2 s_1^2) u_{j-1}^2 + (5 + \rho^2 s_2^2) u_j^2 + (2\rho^2 s_1 s_2 - 4) u_{j-1} u_j. \tag{3.19}$$

In the next step, we define

$$g_j := \rho \underbrace{(-e_{j-1} + 2e_j - e_{j+1})}_{=:\bar{g}} \qquad \Longrightarrow \qquad k_j = \begin{pmatrix} 0 \\ \vdots \\ -1 \\ 2 \\ -1 \\ \vdots \\ g_j \end{pmatrix}$$

and compute

$$\langle HU_j, k_j \rangle = (\rho^2 \bar{g} s_1 - 4) u_{j-1} + (\rho^2 \bar{g} s_2 + 5) u_j. \tag{3.20}$$

Inserting (3.19) and (3.20) into (3.17) yields

$$\begin{aligned} \mathscr{N} \quad = \quad & (5 + \rho^2 s_1^2) u_{j-1}^2 + (5 + \rho^2 s_2^2) u_j^2 + (2\rho^2 s_1 s_2 - 4) u_{j-1} u_j \\ & + (8 - 2\rho^2 \bar{g} s_1) u_{j-1} + (-10 - 2\rho^2 \bar{g} s_2) u_j + \|k_j\|_2^2 . \end{aligned}$$

In order to obtain the minimizers u_{j-1}, u_j, we consider the Jacobian of $\mathscr{N}$:

$$\Delta \mathscr{N} \overset{!}{=} 0 \qquad \Longrightarrow \qquad \begin{pmatrix} 10 + 2\rho^2 s_1^2 & 2\rho^2 s_1 s_2 - 4 \\ 2\rho^2 s_1 s_2 - 4 & 10 + 2\rho^2 s_2^2 \end{pmatrix} \begin{pmatrix} u_{j-1} \\ u_j \end{pmatrix} = \begin{pmatrix} 2\rho^2 \bar{g} s_1 - 8 \\ 10 + 2\rho^2 \bar{g} s_2 \end{pmatrix}$$

The determinants for the solution are

$$\begin{vmatrix} 10 + 2\rho^2 s_1^2 & 2\rho^2 s_1 s_2 - 4 \\ 2\rho^2 s_1 s_2 - 4 & 10 + 2\rho^2 s_2^2 \end{vmatrix} = 20\rho^2 (s_1^2 + s_2^2) + 16\rho^2 s_1 s_2 + 84, \tag{3.21}$$

$$\begin{vmatrix} 2\rho^2 \bar{g} s_1 - 8 & 2\rho^2 s_1 s_2 - 4 \\ 10 + 2\rho^2 \bar{g} s_2 & 10 + 2\rho^2 s_2^2 \end{vmatrix} = \rho^2 \bar{g} (20 s_1 + 8 s_2) - 16\rho^2 s_2^2 - 20\rho^2 s_1 s_2 - 40,$$

$$\begin{vmatrix} 10 + 2\rho^2 s_1^2 & 2\rho^2 \bar{g} s_1 - 8 \\ 2\rho^2 s_1 s_2 - 4 & 10 + 2\rho^2 \bar{g} s_2 \end{vmatrix} = \rho^2 \bar{g} (8 s_1 + 20 s_2) + 20\rho^2 s_1^2 + 16\rho^2 s_1 s_2 + 68.$$

Hence, we arrive at

$$u_{j-1} = \frac{\rho^2 \bar{g} (5 s_1 + 2 s_2) - 4\rho^2 s_2^2 - 5\rho^2 s_1 s_2 - 10}{5\rho^2 (s_1^2 + s_2^2) + 4\rho^2 s_1 s_2 + 21}, \tag{3.22a}$$

$$u_j = \frac{\rho^2 \bar{g} (2 s_1 + 5 s_2) + 5\rho^2 s_1^2 + 4\rho^2 s_1 s_2 + 17}{5\rho^2 (s_1^2 + s_2^2) + 4\rho^2 s_1 s_2 + 21}. \tag{3.22b}$$

These values only exist if the Jacobian of $\mathscr{N}$ is nonsingular. For the analysis, we consider the determinant (3.21) as a function of s_1 and s_2:

$$f(s_1, s_2) := 20\rho^2 (s_1^2 + s_2^2) + 16\rho^2 s_1 s_2 + 84.$$

The Jacobian of $f(s_1, s_2)$ is

$$\begin{pmatrix} 40\rho^2 & 16\rho^2 \\ 16\rho^2 & 40\rho^2 \end{pmatrix},$$

which is spd and hence nonsingular. Thus, $f(s_1, s_2)$ has a unique minimum at $(s_1, s_2) = (0, 0)$ with $f(0, 0) = 84$. Therefore, the Jacobian of $\mathscr{N}$ never becomes singular. Nevertheless, a problem occurs if $(s_1, s_2) = (0, 0)$, because then, all the terms involving ρ vanish in (3.22). This is the case for (see the Definitions (3.18) of s_1 and s_2)

$$\begin{aligned} s_1 : \qquad & e_j = 2e_{j-1}, \\ s_2 : \qquad & e_{j+1} = 2e_j = 4e_{j-1}, \end{aligned}$$

i.e. if

$$\begin{pmatrix} e_{j-1} \\ e_j \\ e_{j+1} \end{pmatrix} \in \operatorname{span} \left\{ \begin{pmatrix} 1 \\ 2 \\ 4 \end{pmatrix} \right\}.$$

In this case, $e^T L$ is zero at the indices $j-1$ and j. Hence $e^T L U_j = 0$, whereas $e^T a_j = -1$ for probing vectors e of that subspace. Thus, the probing condition cannot be satisfied exactly in the current column j, since it does not take probing information into account at all. The structure of e and L annihilates the action of u_{j-1} and u_j. The following computations are carried out excluding this case.

Equations (3.22) allow us to determine the limits for $\rho \to \infty$:

$$\tilde{u}_{j-1} \quad := \quad \lim_{\rho\to\infty} u_{j-1} = \frac{\bar{g}(5s_1 + 2s_2) - 4s_2^2 - 5s_1 s_2}{5(s_1^2 + s_2^2) + 4s_1 s_2}, \tag{3.23a}$$

$$\tilde{u}_j \quad := \quad \lim_{\rho\to\infty} u_j = \frac{\bar{g}(2s_1 + 5s_2) + 5s_1^2 + 4s_1 s_2}{5(s_1^2 + s_2^2) + 4s_1 s_2}. \tag{3.23b}$$

We observe that $\tilde{U}_j = (0, \ldots, \tilde{u}_{j-1}, \tilde{u}_j, \ldots, 0)^T$ satisfies the probing conditions exactly (for $(s_1, s_2) \neq (0, 0)$):

$$\begin{aligned} e^T L\tilde{U}_j &= s_1 \tilde{u}_{j-1} + s_2 \tilde{u}_j \\ &= \frac{(5(s_1^2 + s_2^2) + 4s_1 s_2)\bar{g}}{5(s_1^2 + s_2^2) + 4s_1 s_2} \\ &= \bar{g} = -e_{j-1} + 2e_j - e_{j+1} = e^T a_j. \end{aligned}$$

For the last column U_n, we get $\bar{g} = -e_{n-1} + 2e_n$ and

$$\begin{pmatrix} s_1 \\ s_2 \end{pmatrix} = \begin{pmatrix} 2 & -1 \\ 0 & 2 \end{pmatrix} \begin{pmatrix} e_{j-1} \\ e_j \end{pmatrix},$$

i.e. $(s_1, s_2) = (0, 0) \iff (e_{j-1}, e_j) = (0, 0)$. Thus, we can use equations (3.23) and conclude $e^T L\tilde{U}_n = e^T a_n$, which completes the proof. $\square$

The proof for Theorem 3.1 also provides deeper insight into the requirements for both the probing vectors and the prescribed sparsity pattern. If they are not concerted, MSPAI probing cannot satisfy the probing conditions in every column. In the proof, s_1 and s_2 must satisfy $(s_1, s_2) \neq (0, 0)$. This is a purely structural requirement, if we believe that involving probing information in every column is useful for the preconditioner. We can even generalize the result:

Theorem 3.2 *Consider the general MSPAI probing ansatz*

$$\min_M \left\| \begin{pmatrix} C_0 \\ \rho e^T C_0 \end{pmatrix} M - \begin{pmatrix} B_0 \\ \rho e^T B_0 \end{pmatrix} \right\|_F^2$$

with k probing vectors collected in the matrix $e \in \mathbb{R}^{n \times k}$. In the computation of the j-th column m_j of M, the error $E_j := \left\| e^T C_0 m_j - e^T B_0(:,j) \right\|_2$ in the probing conditions can only be reduced if and only if $\mathscr{P}(m_j)$ satisfies

$$\mathscr{P}_e \cap \mathscr{P}(m_j) \neq \emptyset, \tag{3.24}$$

where

$$\mathscr{P}_e := \bigcup_{j=1}^{k} \mathscr{P}\left(e(:,j)^T C_0\right). \tag{3.25}$$

Furthermore, let $\mathscr{J}$ denote the indices j such that $\mathscr{P}_e \cap \mathscr{P}(m_j) = \emptyset$ and consequently define $\hat{B} = B_0(:,\mathscr{J})$. Then, the overall error $E := \left\| e^T(C_0 M - B_0) \right\|_F$ in the probing constraints has a lower bound

$$E \geq \left\| \frac{1}{\sqrt{k}} e^T \hat{B} \right\|_F. \tag{3.26}$$

We have $E \geq 0$ if and only if $\mathscr{J} = \emptyset$.

Proof The pattern condition (3.24) follows directly by the structural considerations in the proof of Theorem 3.1. The condition $(s_1, s_2) \neq (0,0)$ is straightforward to extend to the general case discussed here. The union $\mathscr{P}_e$ of the k column patterns in e and the prescribed pattern of m_j must coincide in at least one nonzero position. For the lower error bound (3.26), we define $\hat{M} = (:,\mathscr{J})$, $\mathscr{I}$ as the complementary set of $\mathscr{J}$, and thus $\tilde{M} = M(:,\mathscr{I})$ and $\tilde{B} = B_0(:,\mathscr{I})$. Using the norm inequality $\|x\|_2^2 \geq \frac{1}{k}\|x\|_F^2$ for $x \in \mathbb{R}^k$, we immediately get

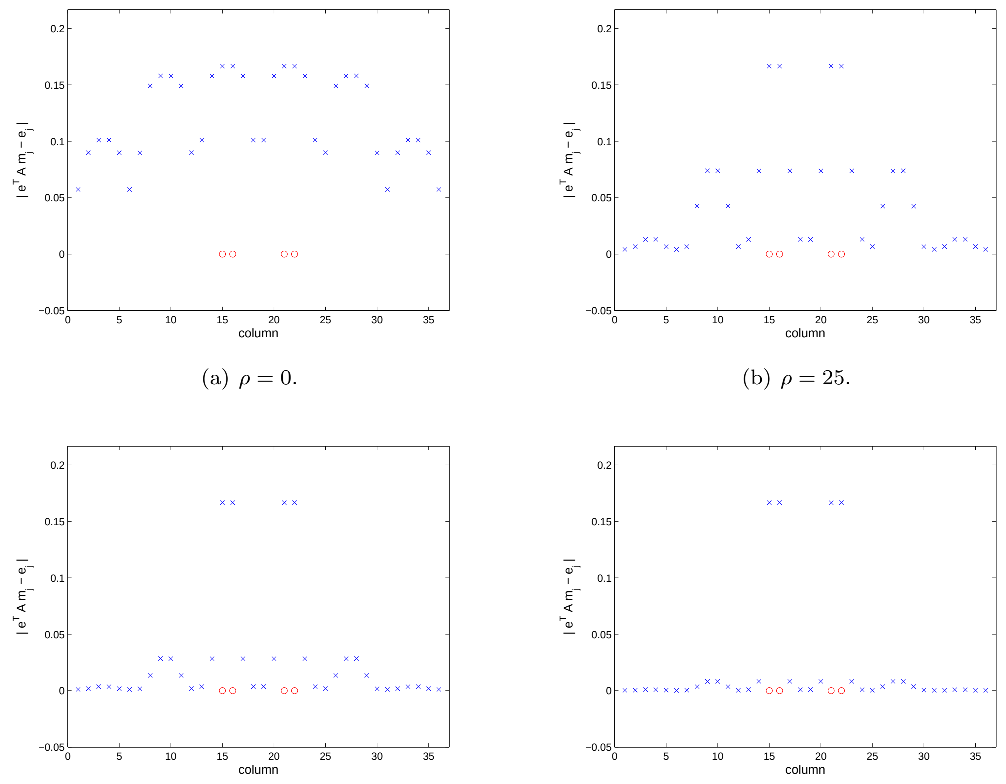
$$\begin{aligned}
E^2 &= \left\| e^T(C_0 M - B_0) \right\|_F^2 = \sum_{j=1}^{n} \left\| e^T(C_0 M(:,j) - B_0(:,j)) \right\|_2^2 \\
&\geq \frac{1}{k} \sum_{j=1}^{n} \left\| e^T(C_0 M(:,j) - B_0(:,j)) \right\|_F^2 = \frac{1}{k} \left\| e^T(C_0 M - B_0) \right\|_F^2 \\
&= \frac{1}{k} \left(\left\| e^T(C_0 \tilde{M} - \tilde{B}) \right\|_F^2 + \left\| e^T(C_0 \hat{M} - \hat{B}) \right\|_F^2 \right) \\
&\geq \left\| \frac{1}{\sqrt{k}} e^T \hat{B} \right\|_F^2.
\end{aligned}$$

□

Theorem 3.2 gives explanations to many effects. For instance, the explicit unfactorized MSPAI probing for the probing vector $e = 1/\sqrt{n}(1,\ldots,1)^T$ is always well-posed as far as satisfying the probing conditions is concerned. $e^T C_0 = e^T I$ is always a full vector, and hence, the intersection $\mathscr{P}_e \cap \mathscr{P}(m_j)$ $(j = 1,\ldots,n)$ never results in the empty set for any arbitrary non-empty pattern of M.

Also, the problems we faced in the proof of Theorem 3.1 with special probing vectors of type (3.16) are caused by the violation of (3.24). E.g. adding a third entry u_{j-2} in the pattern of U_j would yield a simple remedy and probing information of type (3.16) could be fulfilled in every column. Furthermore, the error E has then lower bound 0 according to (3.26).

The situation changes for inverse probing. If $|\mathscr{P}_e|$ is small and the prescribed pattern for M is very sparse, the pattern condition (3.24) is quite likely to be violated. The only ways out are different choices of probing vectors or a more carefully determined $\mathscr{P}(M)$. The following example demonstrates the deterioration in the probing constraints. Note that we only regard the accuracy in probing constraints here, and not how this affects the preconditioning effect of M.



(a) $\rho = 0$. (b) $\rho = 25$.

(c) $\rho = 50$. (d) $\rho = 100$.

Figure 3.1: Componentwise errors $\left|e^T A m_j - e_j\right|$ (blue "x"s) for columns $j = 1, \ldots, 36$. Red circles mark the columns where $\mathscr{P}\left(e^T A\right) \cap \mathscr{P}(m_j) = \emptyset$.

Example 3.1 We apply unfactorized inverse probing (3.7) with $e = \frac{1}{6}(1, \ldots, 1)^T$ to the standard five-point stencil discretization of the 2D Laplacian with dimension $n = 36$, $\mathscr{P}(M) = \mathscr{P}(A)$, and analyze the componentwise error $\left|e^T A m_j - e_j\right|$ $(j = 1, \ldots, 36)$ in the probing constraints. In Figure 3.1, we depict these error components for different probing weights $\rho \in \{0, 25, 50, 100\}$. Since $e^T A$ has 16 zero values, condition (3.24) is violated for the indices $\mathscr{J} = \{15, 16, 21, 22\}$. Hence, the error in these components remains untouched, whereas the error is reduced and nearly annihilated in the other columns. In Figure 3.2, we see that the predicted value for the lower error bound is 0.3333, which is nearly achieved with

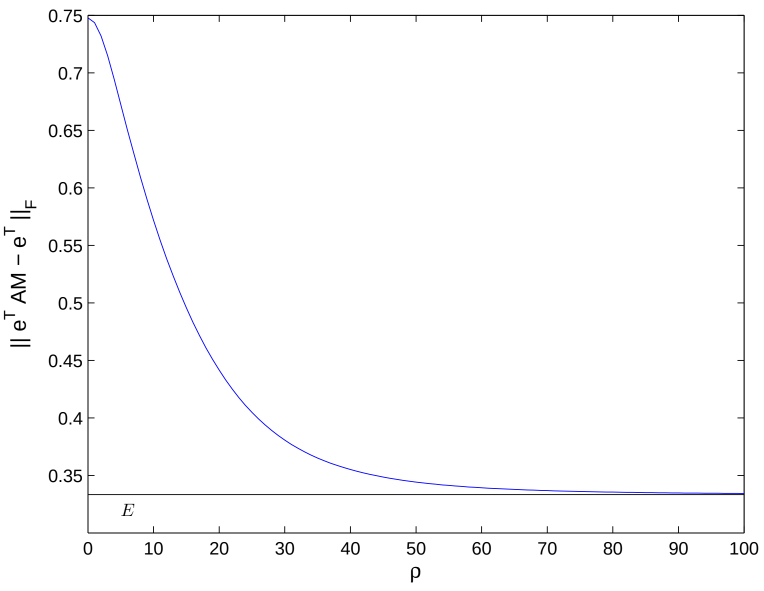


Figure 3.2: The errors in the constraints tend to the predicted lower bound E as ρ gets larger.

the numerical value 0.3355 for $\rho = 100$, i.e. for the indices $j \notin \mathscr{J}$, the probing constraints are satisfied exactly as $\rho \to \infty$.

Remark 3.1 *The pattern conditions here only give requirements for the choice of probing vectors and for the pattern of M if we want to assure that probing conditions are taken into account in every column of M. They do not imply whether this is meaningful for the actual problem, i.e. if the fulfillment of the probing conditions improves the preconditioner and leads to a lower condition number. Especially in the inverse probing case, this can spoil the preconditioner completely. This can be seen by the tridiagonal 1D Laplacian in Theorem 3.1. If additional entries in the pattern of U would be introduced, probing constraints could be satisfied in every column. However, this would also make the "main" approximation $\min_U \|LU - A\|_F$ less accurate.*

3.6 Symmetrization Techniques

For the SPAI algorithm as well as the generalized problem (3.2), the solution matrices M are usually nonsymmetric. Nevertheless, for (positive definite) symmetric matrices A we would like to have a (positive definite) symmetric preconditioner such that it is still possible to use e.g. the preconditioned conjugate gradient method (pcg). Otherwise, one has to rely on iterative methods for nonsymmetric problems, such as BiCGSTAB or GMRES which have higher computational costs. To overcome this problem, we present three symmetrization techniques for preconditioners, e.g. for M of the previous sections. The first approach is based on MSPAI Frobenius norm minimization. In the second method, we combine two iteration steps writing the resulting iteration as one step with a symmetric preconditioner. Based on that approach, we can derive a special type of unfactorized symmetrization for spd matrices. Finally, we also provide techniques for resymmetrizing factorized approximations.

3.6.1 Unfactorized Symmetrization by Frobenius Norm Minimization

First, we consider M derived by the MSPAI minimization relative to symmetric matrices C_0 and B_0, and a set of general probing vectors collected in e. We want to derive a symmetrization of M with a similar sparsity pattern. We start with the trivial symmetrization

$$\bar{M} := M + M^T \ .$$

We can improve this matrix by returning to the initial minimization problem allowing additional changes on the main diagonal entries. Hence we define

$$\hat{M} := \alpha \left(M + M^T \right) + D = \alpha \bar{M} + D \tag{3.27}$$

with a scaling factor $\alpha \in \mathbb{R}$ and a diagonal correction $D \in \mathbb{R}^{n \times n}$. Obviously, $\hat{M}$ is symmetric by construction. We determine α and D in two steps. In order to find an optimal value for α, we insert $\alpha \bar{M}$ into the basic problem (3.3):

$$\min_{\alpha} \left\| \underbrace{\begin{pmatrix} C_0 \\ \rho g^T \end{pmatrix}}_{=:C} \alpha \bar{M} - \underbrace{\begin{pmatrix} B_0 \\ \rho h^T \end{pmatrix}}_{=:B} \right\|_F^2 = \min_{\alpha} \left\| \alpha C \bar{M} - B \right\|_F^2 . \tag{3.28}$$

For the analytic solution, we consider the inner product (denoted as $\langle ., . \rangle_F$), which, for general A and B, is defined by the Frobenius norm:

$$\langle A, B \rangle_F = \|AB\|_F^2 = \operatorname{tr}\left(A^T B \right) . \tag{3.29}$$

In these minimizations, the trace of the product of two matrices can be evaluated by sparse dot products and only a few possibly dense dot products:

$$\operatorname{tr}(AB) = \sum_{j=1}^{N} a_j^T b_j \tag{3.30}$$

with the j-th *row* a_j of A and b_j the j-th *column* of the matrix B.

With formula (3.29), we can solve (3.28) for α, using standard properties of the inner product:

$$\begin{aligned} \left\| \alpha C \bar{M} - B \right\|_F^2 &= \left\langle \alpha C \bar{M} - B, \alpha C \bar{M} - B \right\rangle_F \\ &= \alpha^2 \left\langle C \bar{M}, C \bar{M} \right\rangle_F - 2\alpha \left\langle C \bar{M}, B \right\rangle_F + \langle B, B \rangle_F \ . \end{aligned}$$

The derivative of the last term with respect to α leads to the solution, namely the optimal α which minimizes the Frobenius norm in (3.28):

$$\begin{aligned} & 2\alpha \left\langle C \bar{M}, C \bar{M} \right\rangle_F - 2 \left\langle C \bar{M}, B \right\rangle_F \overset{!}{=} 0 \\ \Longleftrightarrow \qquad & \alpha = \frac{\left\langle C \bar{M}, B \right\rangle_F}{\left\langle C \bar{M}, C \bar{M} \right\rangle_F} = \frac{\operatorname{tr}\left(\left(C \bar{M} \right)^T B \right)}{\|C \bar{M}\|_F^2} . \end{aligned}$$

Note that α is quite cheap to compute, because all occuring matrices are sparse (possibly up to a few full rows in C and B). Furthermore, for spd A in our examples with $C_0 = A$,

$B_0 = I$ or vice versa, α will always be positive. Often (not very surprisingly) it will be near $\frac{1}{2}$.

The second summand of the symmetrized $\hat{M}$ is the diagonal correction D with diagonal entries d_k $(k = 1, \ldots, n)$. The computation of each d_k is similar to the computation of the α above, resp. to the standard SPAI approach. So we present only the ansatz and the result. We insert the whole expression for $\hat{M}$ with known optimal α into (3.2). With the substitute matrices C and B from above, we get:

$$\begin{aligned}\min_D \left\|C\left(\alpha\bar{M} + D\right) - B\right\|_F^2 &= \min_D \left\| CD - \underbrace{\left(B - \alpha C\bar{M}\right)}_{=:F} \right\|_F^2 \\ &= \min_D \sum_{k=1}^{n} \|Cd_k - f_k\|_2^2\end{aligned}$$

By exploiting the representation of the Frobenius norm as a sum of Euclidean norms, each summand represents a least squares problem for exactly one d_k. For every $k \in \{1, \ldots, n\}$ we have a dot product representation, which can be derived with respect to d_k. We obtain

$$d_k = \frac{c_k^T f_k}{\|c_k\|_2^2},$$

with c_k being the k-th column of C and f_k the k-th column of F. $C\bar{M}$ has already been computed when determining α, so the computation of K and subsequently of D does not lead to exceeding computational efforts compared to the whole MSPAI process.

Note that these computations are fast because of the sparsity of the underlying matrices and the low rank of the possibly dense submatrices g and h. But in general, this symmetrization technique does not necessarily yield a positive definite preconditioner.

3.6.2 Symmetrization by Combining two Basic Iteration Steps

A second approach for a symmetrization of M considers two basic iterative steps: one with preconditioner M, the second one with M^T. Here, we consider M as approximation of the inverse of A, so the first step is related to the iteration matrix $I - AM$, the second to $I - AM^T$. The two consecutive steps are described by the iteration matrix

$$\begin{aligned}(I - AM^T) \cdot (I - AM) &= I - AM^T - AM + AM^T AM \\ &= I - A(\bar{M} - M^T AM).\end{aligned}$$

Hence, the resulting symmetric preconditioner is given by $\bar{M} - M^T AM$. To add an additional degree of freedom, we consider the damped iteration with preconditioner αM. This leads to

$$\begin{aligned}(I - \alpha AM^T) \cdot (I - \alpha AM) &= I - \alpha AM^T - \alpha AM + \alpha^2 AM^T AM \\ &= I - \alpha A(\bar{M} - \alpha M^T AM)\end{aligned}$$

and the symmetrized preconditioner

$$\hat{M}_\alpha := \bar{M} - \alpha M^T AM. \tag{3.31}$$

This preconditioner is denser than M, but may lead to a reduction of the error comparable with two steps based on αM. To be sure that the two iteration steps lead to an improved approximate solution, we need $\|I - \alpha AM\| < 1$.

Theorem 3.3 *Let us assume that $\lambda(AM) > 0$ holds for all eigenvalues of AM. Then for*

$$0 \leq \alpha < \frac{2}{\lambda_{\max}(AM)},$$

we get $\|I - \alpha AM\| < 1$ in the spectral norm. The optimal α, which leads to a minimum norm of $I - \alpha AM$, is given by

$$\alpha = \frac{2}{\lambda_{\max}(AM) + \lambda_{\min}(AM)}$$

with

$$\|I - \alpha AM\| \approx \left| \frac{\lambda_{\max}(AM) - \lambda_{\min}(AM)}{\lambda_{\max}(AM) + \lambda_{\min}(AM)} \right| < 1.$$

Proof In order to get the minimal condition number, we have to determine the optimal α such that the maximum values of

$$1 - \alpha\lambda_{\min}(AM) \quad \text{and} \quad 1 - \alpha\lambda_{\max}(AM)$$

are minimized. This is fulfilled for α satisfying

$$1 - \alpha\lambda_{\min}(AM) = -\left(1 - \alpha\lambda_{\max}(AM)\right).$$

Therefore, we compute for the optimal α:

$$\alpha = \frac{2}{\lambda_{\min}(AM) + \lambda_{\max}(AM)} < \frac{2}{\lambda_{\max}(AM)}.$$

Furthermore, for this α holds

$$\|I - \alpha AM\| \leq \frac{\lambda_{\max}(AM) - \lambda_{\min}(AM)}{\lambda_{\max}(AM) + \lambda_{\min}(AM)}.$$

$\square$

Remark 3.2 *To derive this optimal α, one needs approximations for the extreme eigenvalues of AM. In most cases, quite rough approximations are sufficient.*

This symmetrization with $\bar{M}$ and $\hat{M}_\alpha$ does not include an additional probing condition. Fortunately, we can show that this symmetrization process preserves the probing property for slightly modified preconditioners:

Theorem 3.4 *Assume that M is a preconditioner that satisfies the probing condition $e^T AM = e^T$ exactly. For $\tilde{M} := M + M^T - \alpha MAM^T$ with $\alpha < 2$ (e.g. $\alpha = 0$ and $\bar{M}$) the Rayleigh quotient for vector Ae is given by*

$$(e^T A)\tilde{M}(Ae) = (2 - \alpha)(e^T A)A^{-1}(Ae) = (2 - \alpha)(e^T A)M(Ae)\ ,$$

and therefore the range of values of the preconditioned matrix with $\tilde{M}$ is nearly unchanged relative to the probing space.

Proof The following holds:

$$\begin{aligned}
(e^T A)\tilde{M}(Ae) &= (e^T A)(M + M^T - \alpha MAM^T)(Ae) \\
&= e^T Ae + e^T Ae - \alpha e^T Ae = (2-\alpha)e^T Ae \;, \\
(2-\alpha)e^T Ae &= (2-\alpha)(e^T A)A^{-1}(Ae) \;, \\
(2-\alpha)e^T Ae &= (2-\alpha)(e^T AM)Ae \;=\; (2-\alpha)(e^T A)M(Ae) \;.
\end{aligned}$$

□

Remark 3.3 *For $\alpha = 1$, $\tilde{M}$ satisfies the probing condition exactly:*

$$e^T A\tilde{M} = e^T A(M + M^T - \alpha MAM^T) = e^T + e^T AM^T - \alpha e^T AM^T = e^T.$$

3.6.3 SPAI Acceleration

Until now we have only derived symmetric preconditioners, but often for spd A, we need an spd preconditioner. So now we consider only spd matrices A. If $\bar{M}$ is symmetric indefinite, the above methods will not lead to an spd preconditioner. So let us assume that $\bar{M} = M + M^T$ is positive definite. If this is not true, we could replace $\bar{M}$ by $\bar{M} + \beta I$ with small β. Note that for a good approximation M, $\bar{M}$ should be nearly positive definite, and therefore we expect to find such a small $\beta > 0$.

Proposition 3.1 *Let A and $\bar{M}$ be symmetric positive definite. Then $\hat{M}_\alpha = \bar{M} - \alpha M^T AM$ will also be positive definite as long as $\alpha < \lambda_{\min}(\bar{M}, M^T AM)$, the minimum eigenvalue of the generalized positive definite eigenvalue problem $\bar{M} = \lambda M^T AM$.*

Remark 3.4 *Therefore, we have to choose α such that it satisfies the inequalities*

$$\alpha \le \min\left\{\frac{2}{\lambda_{\max}(AM) + \lambda_{\min}(AM)}, \lambda_{\min}(\bar{M}, M^T AM)\right\} =: \alpha_{\text{opt}}.$$

We can derive a deeper analysis by considering the preconditioner $\bar{M} - \alpha\bar{M}A\bar{M}/4$, which is the result of applying the above symmetrizing construction on $\bar{M}/2$.

Theorem 3.5 (SPAI Acceleration) *Let A and $\bar{M}$ be spd. Then, for $\bar{M} - \alpha\bar{M}A\bar{M}/4$, the optimal α is given by*

$$\frac{2}{\lambda_{\max}(A\bar{M}/2) + \lambda_{\min}(A\bar{M}/2)}.$$

With this choice we get a new improved condition number of the preconditioned system. The condition is improved by a factor

$$\frac{\lambda + 2\mu}{4\lambda} \approx \frac{1}{4}.$$

Proof The minimum eigenvalue of the preconditioned system is given by $\frac{2\lambda\mu}{\lambda+\mu}$ and the maximum eigenvalue by $\frac{\lambda(\lambda+2\mu)}{2(\lambda+\mu)}$ with λ and μ the maximum, resp. minimum eigenvalues of $\bar{M}A/2$. □

The acceleration part of the name refers to the improvement of the overall condition number by a factor of $\frac{1}{4}$. Note that we can also sparsify $\bar{M} - \alpha M^T AM$ carefully in order to avoid a preconditioner that would be too dense. Furthermore, we can sometimes save costs in computing $\hat{M}_\alpha$ or $A\hat{M}_\alpha$. In SPAI for example, we have already computed $R = I - AM$ and we may now use this information, e.g. in the form

$$M + M^T - \alpha M^T AM = M + M^T(I - \alpha AM) = M + M^T(1 - \alpha + \alpha R) .$$

In case we want to symmetrize a preconditioner M that is an approximation of A itself, we can apply the same method on M^{-1} and get

$$M^{-1} + M^{-T} - \alpha M^{-T} AM^{-1} \tag{3.32}$$

as a symmetric approximation for A. When M is the upper or lower triangular part of A — the Gauss-Seidel-preconditioner — this symmetrization approach is closely related to SSOR-type preconditioners. Hence, we can find efficient implementations similar to the Eisenstat trick [39].

In general, we can apply this symmetrizing method if AM has only positive eigenvalues, and for spd A we derive an spd preconditioner if $\bar{M}$ is spd. The effectiveness of these symmetrization techniques emerges in the following examples.

Example 3.2 For the case of explicit approximations of A, we choose an $A := LL^T$, where L comes from an IC(0) factorization of the 5 point discretization of the 2D Laplacian, and therefore is of block tridiagonal structure. This example is completely artificial and

Table 3.1: Condition numbers for no preconditioning I, unsymmetrized preconditioner M, $\bar{M}$, $\hat{M}$ and $\hat{M}_\alpha$ employed as explicit approximative preconditioners AM^{-1}.

n	I	M	$\bar{M}$	$\hat{M}$	$\hat{M}_\alpha$
100	12.061	8.288	8.255	8.155	2.604
400	13.849	7.085	7.055	7.017	2.302
1600	14.436	5.759	5.759	5.759	1.984

its only purpose is to demonstrate the result of a symmetrization step. We probe it with $e = 1/\sqrt{n}(1, \ldots, 1)^T$ and $\rho = 20$ with a tridiagonal prescribed sparsity pattern and obtain a preconditioner M. This M is then symmetrized both to $\hat{M}$ using the $\alpha\bar{M} + D$ approach (3.27) and to $\hat{M}_\alpha$ from (3.31). As shown in Table 3.1, this ansatz yields symmetric preconditioners with nearly unchanged condition numbers, which can then be employed in iterative solvers for spd matrices, e.g. the pcg method. Furthermore, in this example $\hat{M}_\alpha$ leads to a condition number improved by nearly a factor $\frac{1}{4}$ — as expected. Note that, in the explicit approximation, we use M^{-1} in the symmetrization via $\hat{M}_\alpha$ (3.32). In all examples, the simple and cheap symmetrization $(M + M^T)/2$ is sufficient to obtain a symmetric preconditioner of the same quality as the original nonsymmetric M. Hence, the computation of optimal α and D in $\hat{M}$ is often unnecessary. For the symmetrization of an approximate

Table 3.2: Condition numbers for no preconditioning I, unsymmetrized preconditioner M, $\bar{M}$, $\hat{M}$ and $\hat{M}_\alpha$ employed as approximative inverse preconditioners AM.

n	I	M	$\bar{M}$	$\hat{M}$	$\hat{M}_\alpha$
100	48.374	8.448	8.459	8.463	2.638
400	178.064	30.706	30.713	30.720	8.194
1600	680.617	117.031	117.035	117.050	29.773

inverse preconditioner, we compute M by SPAI applied to the 2D Laplacian with the static pattern of A^2. The condition numbers of the symmetrized versions are shown in Table 3.2. Note that in this second example, we do not apply probing. We only want to display the symmetrization aspect. Again, the cheap symmetrization $\bar{M}$ is sufficient, and $\hat{M}_\alpha$ leads to a condition number which is reduced by a factor of nearly $\frac{1}{4}$.

3.6.4 Symmetrization for Factorized Approximations

If the given matrix A is symmetric positive definite, we can generate a symmetric factorization based on e.g. incomplete Cholesky or FSPAI, resulting in a triangular matrix L such that $A \approx LL^T$ or $L^T AL \approx I$. Then, the methods of the previous two sections enable us to add probing conditions and replace L^T by another factor $\tilde{U} := \tilde{L}^T$ for fixed L . But then we lose symmetry, because of the two different factors L and $\tilde{U}$ for the preconditioner $L\tilde{U}$. To regain symmetry, we set

$$L_\alpha := L_\alpha(L, \tilde{L}) := L + \alpha(\tilde{L} - L), \qquad \alpha \in [0, 1], \tag{3.33}$$

as a convex combination of both factors. An optimal α can be computed by inserting L_α into the original MSPAI minimization problem

$$\min_\alpha \|W(L_\alpha L_\alpha^T - A)\|_F \ , \tag{3.34}$$

or

$$\min_\alpha \|W(L_\alpha^T AL_\alpha - I)\|_F \tag{3.35}$$

in the approximative inverse case. This leads to a polynomial of degree 4 in α of the form

$$\begin{aligned} \left\|WR + \alpha WH + \alpha^2 WK\right\|_F^2 &= \operatorname{tr}\left(RW^TWR\right) + 2\alpha\operatorname{tr}\left(HW^TWR\right) + \\ &+ \alpha^2\operatorname{tr}\left(HW^TWH + 2KW^TWR\right) + \\ &+ 2\alpha^3\operatorname{tr}\left(KW^TWH\right) + \alpha^4\operatorname{tr}\left(KW^TWK\right) \ . \end{aligned} \tag{3.36}$$

For (3.34) the matrices H, K, and R are given by

$$R := LL^T - A \ , \ \ K := (\tilde{L} - L)(\tilde{L} - L)^T \ , \ \ H := (\tilde{L} - L)L^T + L(\tilde{L} - L)^T \ , \tag{3.37}$$

and for (3.35) by

$$R := L^T AL - I \ , \ \ K := (\tilde{L} - L)^T A(\tilde{L} - L) \ , \ \ H := (\tilde{L} - L)^T AL + L^T A(\tilde{L} - L) \ . \tag{3.38}$$

We compute the minima of these polynomials and choose the solution with minimum norm for α. Therefore we only have to compute the trace of products of sparse matrices. Note that if in L_α both L and $\tilde{L}$ satisfy the probing condition, then also L_α satisfies the probing condition.

The substitute matrix K contains the difference $\tilde{L} - L$ in second order. This difference is supposed to be quite small by construction. So, we can simplify (3.36) by dropping the coefficients which contain K. The result is a polynomial in α of second order:

$$\begin{aligned} \left\|WR + \alpha WH + \alpha^2 WK\right\|_F^2 &\approx \operatorname{tr}\left(RW^TWR\right) + 2\alpha\operatorname{tr}\left(HW^TWR\right) + \\ &\quad +\alpha^2\operatorname{tr}\left(HW^TWH\right) . \end{aligned} \tag{3.39}$$

Equation (3.39) has one unique minimum, which can be computed directly by using the derivative with respect to α and evaluating one trace and one Frobenius norm. Due to the structure of W, the evaluation of the traces is quite cheap. For instance, $\operatorname{tr}\left(HW^TWR\right)$ can be computed exploiting $W^TW = I + \rho^2 ee^T$ and properties of the trace:

$$\operatorname{tr}\left(HW^TWR\right) = \operatorname{tr}\left(HR + \rho^2 Hee^TR\right) = \operatorname{tr}\left(HR\right) + \rho^2 \operatorname{tr}\left((e^TR)(He)\right).$$

We compute both summands using (3.30) without evaluating the matrix-matrix products.

Note that it is not advisable to add an additional diagonal correction in L_α, because this would result in the minimization of a function of degree 4 in the diagonal entries $d_1, ..., d_n$. However, it is possible to add a diagonal correction by neglecting the probing part and choosing the diagonal correction only with respect to the matrix approximation problem in order to generate all ones on the main diagonal positions. So after having computed L_α as above, we can choose a diagonal matrix D by

$$\operatorname{diag}\left(DL_\alpha^TAL_\alpha D\right) = (1, \ldots, 1) \tag{3.40}$$

and replace L_α by $L_D := L_\alpha D$.

For an unknown matrix A, we again have to replace the weight matrix W and consider the minimization problem

$$\min_\alpha \left\| \begin{pmatrix} L_\alpha L_\alpha^T - \tilde{A} \\ \rho e^T L_\alpha L_\alpha^T - e^T A \end{pmatrix} \right\|_F^2 \quad \text{or} \quad \min_\alpha \left\| \begin{pmatrix} L_\alpha^T \tilde{A} L_\alpha - I \\ \rho e^T L_\alpha^T A L_\alpha - e^T \end{pmatrix} \right\|_F^2$$

in order to determine the optimal value for α.

Altogether, let us assume that starting with a factor L, we have generated an approximation $\tilde{U}$ by probing. With fixed $\tilde{U}$ we can compute a new factor $\tilde{L}$ by the transposed minimization. Then, we have the following choices for symmetrizing one or two approximate factors:

- take factors $\tilde{U}$ and $\tilde{U}^T$, e.g. $A \approx \tilde{U}^T\tilde{U}$ or $\tilde{U}A\tilde{U}^T \approx I$,
- take the symmetrization relative to L and $\tilde{U}$, either by exact minimization based on the polynomial of degree 4, denoted by $L_{\alpha,4}(L, \tilde{U}^T)$, or by approximate minimization denoted by $L_{\alpha,2}(L, \tilde{U}^T)$,
- use $\tilde{L}$ and $\tilde{L}^T$ as factorization,
- use $L_{\alpha,m}(\tilde{L}, \tilde{U}^T)$ with exact ($m = 4$) or approximate ($m = 2$) minimization.

Box 3.2 on page 73 summarizes the symmetrization approaches of this chapter. Further numerical examples employing these methods can be found in Chapter 5.

Example 3.3 The MSPAI symmetrization techniques described in the previous section yield factorized preconditioners, where symmetry is regained. The following examples demonstrate that symmetrization does not worsen the condition number of the system. As an example, we take again a 2D Laplacian for different dimensions n. Then we compute an incomplete Cholesky factor for the explicit case and a static FSPAI with the pattern of A for inverse preconditioning. Then we apply probing and use the different symmetrization approaches in order to have a preconditioner which we can use for iterative solvers for symmetric problems. For the explicit approximation, we choose $\rho = 50$ and $e = 1/\sqrt{n}\cdot(1, \ldots, 1)^T$ as probing parameters and combine the incomplete Cholesky factor with the one obtained from

Table 3.3: Condition numbers for no preconditioning I, L from IC(0), $\tilde{U}$ by probing L, $\tilde{U}$ employed together with L, for $L_\alpha = L + \alpha(\tilde{U}^T - L)$ with approximated and exact polynomial as explicit factorized preconditioners.

n	I	L	$L, \tilde{U}$	$L_{\alpha,2}(L, \tilde{U}^T)$	$L_{\alpha,4}(L, \tilde{U}^T)$
100	48.374	5.120	3.462	3.177	3.050
400	178.064	16.593	7.650	8.716	7.899
1600	680.617	61.020	41.922	38.475	39.045

Table 3.4: Condition numbers for no preconditioning I, FSPAI L, probed FSPAI $\tilde{L}$ employed together with L, for $L_\alpha = L + \alpha(\tilde{L} - L)$ as inverse approximative preconditioners.

n	I	L^TAL	$L^TA\tilde{L}$	$L_{\alpha,4}^TAL_{\alpha,4}$
100	48.374	13.827	13.729	13.722
400	178.064	50.223	50.862	50.400
1600	680.617	191.529	194.574	192.433

probing. To do so, we apply convex symmetrization (3.33) using the exact and the approximative polynomials. Table 3.3 shows that the condition numbers are preserved throughout symmetrizing. To save costs, we should use only one step of probing and possibly one additional step of approximate minimization $L_{\alpha,2}$.

For the inverse case with FSPAI, we set $\rho = 7$ and use the eigenvector which corresponds to the smallest eigenvalue as probing vector. Because even quite rough approximations are sufficient here, the computation time for this probing vector is acceptable. The resulting condition numbers are shown in Table 3.4.

Symmetrization Techniques

Unfactorized Symmetrization

By Frobenius Norm Minimization

Compute

$$\hat{M} = \alpha(M + M^T) + D = \alpha\bar{M} + D, \qquad \alpha \in \mathbb{R}, D \text{ diagonal matrix,}$$

via

$$\min_{\alpha} \left\| \alpha C_0 \bar{M} - B_0 \right\|_F^2 \qquad \text{and} \qquad \min_{D} \left\| C_0(\alpha\bar{M} + D) - B_0 \right\|_F^2 .$$

Combination of Two Basic Iteration Steps

The formulation for $M \approx A^{-1}$ with $\bar{M} = M + M^T$ is

$$\hat{M}_\alpha = \bar{M} - \alpha M^T A M \qquad \text{with} \qquad \alpha_{\text{opt}} = \frac{2}{\lambda_{\max}(AM) + \lambda_{\min}(AM)},$$

where $\lambda_{\min}, \lambda_{\max}$ denote approximations to the extreme eigenvalues of AM. For $M \approx A$, apply this method to M^{-1}.

SPAI Acceleration

For $M \approx A^{-1}$ and spd $\bar{M} = M + M^T$ compute

$$\hat{M}_A = \bar{M} - \frac{\alpha}{4}\bar{M}A\bar{M} \quad \text{with} \quad \alpha_{\text{opt}} = \min\left\{ \frac{2}{\lambda_{\max}(AM) + \lambda_{\min}(AM)}, \lambda_{\min}(\bar{M}, M^T A M) \right\}.$$

The condition is improved by a factor $\approx \frac{1}{4}$. For $A \approx M$, see (3.32).

Factorized Symmetrization

Combine the original lower triangular matrix L and probed lower triangular matrix $\tilde{L}$ to

$$L_\alpha = L_\alpha(L, \tilde{L}) = L + \alpha(\tilde{L} - L).$$

For the optimal symmetrization parameter α, minimize either the exact polynomial

$$\alpha^4 \text{tr}\left(KW^TWK\right) + 2\alpha^3 \text{tr}\left(KW^TWH\right) + \alpha^2 \text{tr}\left(HW^TWH + 2KW^TWR\right)$$
$$+ 2\alpha \text{tr}\left(HW^TWR\right) + \text{tr}\left(RW^TWR\right),$$

or its simplified version

$$\alpha^2 \text{tr}\left(HW^TWH\right) + 2\alpha \text{tr}\left(HW^TWR\right) + \text{tr}\left(RW^TWR\right),$$

using the substitute matrices K, W, H, R from (3.37) in the explicit approximation case, and (3.38) in the inverse setting.

Box 3.2: Overview of the symmetrization techniques developed in this chapter.

Chapter 4

Efficient Implementation

The previous chapters were devoted to the theoretical development and presentation of preconditioners which are all based on Frobenius norm minimizations. Now, we will focus on the efficient computation of these preconditioners, both serial and in parallel computing environments. We have already seen that the computation of one column of a SPAI or MSPAI finally reduces to the solution of a least squares (LS) problem. In principle, SPAI and the MSPAI variants lead to the same type. Therefore, we restrict ourselves to the SPAI case.

At first, we address the efficient solution of LS problems, taking into account the special structures which arise in the computation of sparse approximate inverse preconditioners. We will refer to these LS problems as SPAI-type LS problems. It is well-known that QR updates can significantly accelerate the computation of SPAI with many pattern updates. We will combine this approach with certain sparse solution techniques. Furthermore, we will investigate the use of float QR implementations, i.e. the QR decomposition is carried out in single precision arithmetic and then improved by iterative refinement steps. This was joint work with Andreas Roy [82].

The second part of this chapter will present an improvement for computing the SPAI of structured matrices for structured sparsity patterns. By caching, we avoid the redundant computation of identical LS solutions and hence, reduce the overall runtime significantly. The topic of Section 4.3 will be the effect on the runtimes in parallel environments if we provide a maximum sparsity pattern as it was discussed theoretically in Section 2.1.3.

The parallel implementation in C++ is a result of joint work with Matous Sedlacek [86, 87]. Insightful comments and ideas by Tobias Weinzierl enhanced the performance significantly. The comparison of our MSPAI 1.0 implementation and the current standard implementation SPAI 3.2 [91] in terms of functionality and runtimes will conclude this chapter. All parallel tests were performed on the InfiniCluster [66] at the chair for "Rechnertechnik und Rechnerorganisation/Parallelrechnerarchitektur" at Technische Universität München. The InfiniCluster consists of 32 nodes. Each node has four AMD Opteron 850 processors at 2.4 GHz with 8 GB of main memory. For the communication, the nodes are equipped with two Gigabit Ethernet ports and with a MT23108 InfiniBand Host Channel Adapter card.

4.1 Solution of SPAI-type Least Squares Problems

In Chapter 2, we have already seen that the computation of a sparse approximate inverse M of a given matrix $A \in \mathbb{R}^{n \times n}$ using Frobenius norm minimization results in the independent computation of each column m_k ($k = 1, \ldots, n$), see equation (2.3). The actual entries of m_k are then the result of the solution of an LS problem

$$\min_{\hat{m}_k} \left\| \hat{A}\hat{m}_k - \hat{e}_k \right\|_2^2 . \tag{4.1}$$

$\hat{A}$, $\hat{m}_k$, and $\hat{e}_k$ are the reduced forms of the matrix A, the k-th column m_k of M, and the k-th unit vector e_k. Hereby, the prescribed sparsity pattern of m_k defines the set of nonzero entries $\mathscr{J} = \mathscr{P}(m_k)$, and $\mathscr{I}$ is the set of nonzero rows in $A(:, \mathscr{J})$ according to (2.5). Moreover, we have already defined $p = |\mathscr{I}|$ and $q = |\mathscr{J}|$, and therefore $\hat{A} = A(\mathscr{I}, \mathscr{J}) \in \mathbb{R}^{p \times q}$, $\hat{m}_k = m_k(\mathscr{J}) \in \mathbb{R}^q$, and $\hat{e}_k = e_k(\mathscr{I}) \in \mathbb{R}^p$. We can carry over this approach in a straightforward way to the more general case of MSPAI probing. In Section 3.3, we stated the general form of MSPAI probing with f probing vectors collected in $e \in \mathbb{R}^{n \times f}$ as

$$\min_M \left\| \begin{pmatrix} C_0 \\ \rho e^T C_0 \end{pmatrix} M - \begin{pmatrix} B_0 \\ \rho e^T B_0 \end{pmatrix} \right\|_F^2 = \min_M \|CM - B\|_F^2 .$$

With the substitute matrices $C, B \in \mathbb{R}^{(n+f) \times n}$, we obtain the LS problem for one column m_k

$$\min_{m_k} \left\| \hat{C}\hat{m}_k - \hat{b}_k \right\|_2^2 , \qquad k = 1, \ldots, n.$$

Again, $\hat{C}$, $\hat{m}_k$, and $\hat{b}_k$ are the reduced forms of C, m_k, and b_k according to the sparsity pattern prescribed for m_k. Hence, it is sufficient to discuss the solution methods for one case, because all MSPAI variants lead to the same type of minimization problem in the end. For convenience, we choose the classic SPAI formulation (4.1).

Björck [17] characterizes the solutions of a general linear LS problem

$$\min_x \|Ax - b\|_2 , \qquad A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^m, x \in \mathbb{R}^n \tag{4.2}$$

by the following theorem.

Theorem 4.1 (Theorem 1.1.2 in [17]) *Denote the set of all solutions to (4.2) by*

$$\mathscr{S} := \{x \in \mathbb{R}^n : \|Ax - b\|_2 = \min\}.$$

Then $x \in \mathscr{S}$ if and only if the following orthogonality condition holds:

$$A^T(b - Ax) = 0. \tag{4.3}$$

Proof Assume that $\hat{x}$ satisfies $A^T\hat{r} = 0$, where $\hat{r} = b - A\hat{x}$. Then for any $x \in \mathbb{R}^n$ we have $r = b - Ax = \hat{r} + A(\hat{x} - x) \equiv \hat{r} + Ae$. Squaring this we obtain

$$r^T r = (\hat{r} + Ae)^T(\hat{r} + Ae) = \hat{r}^T\hat{r} + \|Ae\|_2^2 ,$$

which is minimized when $x = \hat{x}$. On the other hand suppose $A^T\hat{r} = z \neq 0$, and take $x = \hat{x} + \varepsilon z$. Then $r = \hat{r} - \varepsilon Az$, and

$$r^T r = \hat{r}^T\hat{r} - 2\varepsilon z^T z + \varepsilon^2 (Az)^T Az < \hat{r}^T\hat{r}$$

for sufficiently small ε. Hence $\hat{x}$ is not a least squares solution. $\square$

For our purposes, equation (4.3) implies that we can obtain the LS solution $\hat{m}_k$ through solving the corresponding *normal equation*

$$\hat{A}^T \hat{A} \hat{m}_k = \hat{A}^T \hat{e}_k. \tag{4.4}$$

$\hat{A}^T \hat{A}$ is symmetric positive definite if $\hat{A}$ has full column rank q. This is guaranteed if the original matrix A is nonsingular. Hence we can solve the normal equation using Cholesky decomposition and obtain the unique solution of the LS problem. The drawback of that method is a higher condition number, since $\kappa(\hat{A}^T \hat{A}) = \kappa^2(\hat{A})$. This may cause breakdown in the Cholesky algorithm for ill-conditioned matrices $\hat{A}$ when $1/\kappa(A)$ is of the order of the square root of the machine precision. Then, square roots of negative numbers may occur and the Cholesky factorization fails. Huckle [59] suggests to iteratively solve normal equation (4.4) using the cg method. The condition number can be improved by a cheap Jacobi preconditioning step, and we avoid the explicit evaluation of the product $\hat{A}^T \hat{A}$. Moreover, within the cg method, we perform the matrix-vector products in sparse mode and therefore exploit sparsity structures in $\hat{A}$. The disadvantage of that method emerges if we have to solve an enlarged matrix in a further step. The iterative solution of the LS problem cannot use information from the previous solution process.

If we find a decomposition $\hat{A} = Q \begin{pmatrix} R \\ 0 \end{pmatrix}$ with orthogonal $Q \in \mathbb{R}^{p \times p}$ and upper triangular $R \in \mathbb{R}^{q \times q}$, we can use it to compute the solution of the LS problem. With $(\tilde{e}_1, \tilde{e}_2)^T = Q^T \hat{e}_k$, $\tilde{e}_1 \in \mathbb{R}^q$, $\tilde{e}_2 \in \mathbb{R}^{p-q}$, we get

$$\begin{aligned} \left\| \hat{A}\hat{m}_k - \hat{e}_k \right\|_2^2 &= \left\| Q^T (\hat{A}\hat{m}_k - \hat{e}_k) \right\|_2^2 = \left\| \begin{pmatrix} R\hat{m}_k - \tilde{e}_1 \\ \tilde{e}_2 \end{pmatrix} \right\|_2^2 \\ &= \| R\hat{m}_k - \tilde{e}_1 \|_2^2 + \|\tilde{e}_2\| \geq \|\tilde{e}_2\|_2^2 . \end{aligned}$$

Therefore, the LS solution is given by

$$\hat{m}_k = R^{-1} \tilde{e}_1.$$

One way to derive a QR decomposition is *Gram-Schmidt orthogonalization*, see for instance [17, 41, 46]. Here we obtain a QR factorization of a matrix $\hat{A}$ with orthogonal matrix $Q \in \mathbb{R}^{p \times p}$ and upper triangular matrix $R \in \mathbb{R}^{p \times q}$. We perform the orthogonalization successively column by column, starting with the first one. We compute the next column through subtraction of the orthogonal projections on all the columns we computed before and then we normalize it. Hence, we get an explicit orthonormal matrix Q such that $\hat{A} = QR$. Appending a new column to $\hat{A}$ would only result in performing the orthogonalization step with respect to the already computed columns and is therefore a straightforward task. But this classical Gram-Schmidt method is numerically unstable which leads to a loss of orthogonality in Q. However, we can use the *Modified Gram-Schmidt method*, which performs the subtractions of the orthogonalized columns already when they have been computed. This results in a more reliable algorithm. We have to pay for this increased numerical stability with a higher computational effort. Moreover, if $\hat{A}$ is very ill-conditioned, modified Gram-Schmidt fails as well if we do not use reorthogonalization. This would further increase the computational costs.

This leads us to a third way of solving (4.1), namely the implicit QR decomposition. This class of methods is called implicit because Q is never determined explicitly. We only save a set of vectors which enables us to recover the action of Q on a vector quite cheaply. Note that we omit here other techniques for solving LS problems such as singular value decomposition or QR decomposition using Givens rotations. We restrict the representation to the most promising approach, implicit QR decomposition with Householder reflections.

4.1.1 QR Decomposition with Householder Reflections

Given a vector $v \in \mathbb{R}^n$. Then, according to [35], a matrix $H \in \mathbb{R}^{n \times n}$ of the form

$$H = I - 2\frac{vv^T}{v^T v}$$

is called *Householder reflection* and v is referred to as *Householder vector*. H is symmetric ($H^T = H$), orthogonal ($HH^T = H^T H = I$), and idempotent ($H^2 = I$). The action of H on a vector $y \in \mathbb{R}^m$ is the reflection of y on a plane perpendicular to v. The application to y is straightforward

$$Hy = y - 2\frac{v^T y}{v^T v}v. \tag{4.5}$$

If we define

$$v := y - \alpha e_1, \qquad \alpha := -\operatorname{sgn}(y_1)\, \|y\|_2\,,$$

we can use this construction to derive a reflection of y onto a multiple of the first unit vector e_1. The choice $\alpha = -\operatorname{sgn}(y_1)\, \|y\|_2$ preserves the length of y and avoids numerical cancellation if y is already near the first unit vector.

We can now use Householder reflections to successively transform the matrix $\hat{A}$ of our LS problem (4.1) to upper triangular form. For that purpose, we reflect the subdiagonal part of the first column $a_1 \in \mathbb{R}^p$ onto the first unit vector e_1 of dimension p:

$$A \to A^{(1)} : \qquad A^{(1)} = H_1 A = \begin{pmatrix} \alpha_1 & \\ 0 & \\ \vdots & \underbrace{H_1 a_2, \ldots, H_1 a_q}_{=:T_1} \\ & \\ 0 & \end{pmatrix}.$$

Then, we construct orthogonal matrices

$$H_{k+1} = \begin{pmatrix} I_k & \\ & \tilde{H}_{k+1} \end{pmatrix},$$

where I_k is the k-dimensional unity, and $\tilde{H}_{k+1}$ is the Householder transformation which reflects the first column of the remainder matrix T_k onto e_1 of dimension $p-k$. We perform these successive steps for $k = 2, \ldots, g$ with $g := \min(p-1, q)$ and end up with

$$H_g \cdots H_1 A = \begin{pmatrix} R \\ 0 \end{pmatrix} =: \tilde{R}$$

with upper triangular matrix $R \in \mathbb{R}^{q \times q}$ and

$$Q = H_1 \cdots H_g.$$

Since we can reconstruct the Householder transformations H_k quite easily from the Householder vectors (see (4.5)), we store the latter efficiently in the lower triangular part of $\tilde{R}$. We only need one further vector to store the factors α_k, $k = 1, \ldots, g$. The costs for the QR decomposition without the explicit computation of Q are

$$\#\text{flops} = 2q^2 \left(p - \frac{q}{3}\right).$$

As stated above, we do not need to know Q explicitly but only the action of Q or Q^T on a vector, respectively. For the solution of our SPAI LS problem (4.1), we have to apply Q^T on $\hat{e}_k$. Then, we get the solution $\hat{m}_k$ by a simple backward substitution with R. If we append columns and rows to $\hat{A}$, we do not need to recompute the whole QR decomposition. The QR update for this case will be the topic of Section 4.1.3. First, we will investigate how to exploit the potential sparsity structure of $\hat{A}$.

4.1.2 Sparse Least Squares Problems

The current SPAI standard implementation SPAI 3.2 by Barnard and Grote [10, 91] solves the LS problems (4.1) using LAPACK [76] routines for dense Householder QR decomposition. This means extracting $\hat{A}$ for every column and converting it to the dense LAPACK storage scheme. The routine `dgeqrf` computes the QR factorization, `dormqr` applies Q^T to the right-hand side and finally, `dtrtrs` solves the upper triangular system with backward substitution. The solution $\hat{m}_k$ then overwrites the right-hand side $\hat{e}_k$. Since we want to compute a preconditioner with about the same `nnz` and thus, about the same density as A, the dimension $p \times q$ of $\hat{A}$ does not become too large and the use of dense LAPACK functions is still efficient. However, this algorithm does not take sparsity into account.

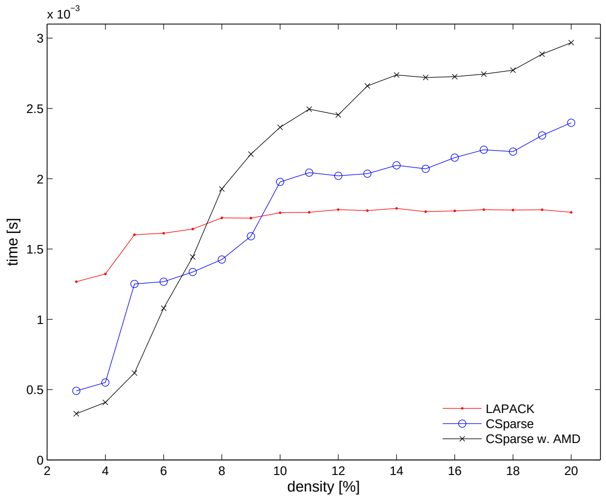


Figure 4.1: Runtime comparison for increasing density in 100×100 random test matrices for implicit QR decomposition using LAPACK, CSparse without preordering, and CSparse with AMD preordering (taken from [82]).

The matrices which we try to precondition with SPAI or MSPAI are usually sparse or the result of a sparsification process. Therefore, we can assume that in certain cases the small matrices $\hat{A}$ related to the reduced LS problem (4.1) are sparse, too.

Among the several implementations available for sparse QR decompositions we choose the rather recent library *CSparse* [29] by Davis [32]. It is designed for solving sparse square systems of equations for real matrices and also provides sparse Cholesky decomposition methods as well as QR decomposition using Householder reflections. One can also preorder the matrices using the approximate minimum degree reordering from Section 1.3.3. Likewise for LU decomposition, AMD preordered QR decomposition leads to a sparser upper triangular factor R (see also [45]). The extension *CXSparse* [31] to complex problems is also available, but we will only use CSparse.

For the sparse QR decomposition, CSparse uses Householder reflections. Like many other direct sparse solvers, it first performs a fill-in reducing column permutation of $\hat{A}^T\hat{A}$, followed by a symbolic QR decomposition in order to determine the sparsity structure of R and of the Householder vectors. With this precise information about the structures and memory usage, CSparse carries out the numerical QR decomposition involving a minimum of fill-in in R. All operations use sparse matrix and vector storage formats and take full advantage of sparsity.

Clearly, there will be a threshold in the density of the matrices $\hat{A}$ from which on the dense LAPACK QR method will be faster again due to the overhead for administrating the sparse data structures of CSparse. In order to get an impression where this threshold lies, we performed several empirical runtime tests with artificially constructed matrices, where we increased the density step by step. We construct matrices with fixed dimension 100×100 with random nonzero diagonal and increase the density by inserting random rows and columns and further diagonals. Then we compute the implicit QR factorization with LAPACK, with CSparse using no preordering, and CSparse with AMD preordering. In Figure 4.1, we can see that LAPACK leads to nearly constant runtimes, since it treats the test matrices all as dense. Moreover, the CSparse variant with the AMD preordering is fastest up to a density of about 7%, and CSparse without preordering has the lowest runtimes for densities ranging from about 7% to about 9.5%. Further runtime tests with different matrix dimensions and sparsity structures revealed that in general it is advisable to use CSparse with approximate minimum degree preordering up to a density of about 7% and CSparse without preordering up to about 12%. For matrices which have more entries relative to their dimension, we recommend using LAPACK QR routines for dense matrices. We also generated sets of test matrices which arise in concrete SPAI computation for several test matrices from Matrix Market [77] to achieve a more realistic picture. For the complete testing results, see [82]. The runtimes we observed there yield the conservative bounds of about 12% and 20%. We chose this rather small problem size, because the matrices $\hat{A}$ arising in the computation of SPAI or MSPAI typically have dimensions in that order of magnitude. Note that we measured these runtimes in a C implementation only reading one test matrix after another from a file and computing the implicit QR decomposition. In a complete SPAI or MSPAI implementation, these values may vary. This test was performed on an Intel Pentium D 2.8 GHz using LAPACK 3.1.1 and CSparse 2.0.6.

For simplicity, we only employ a switch between sparse and dense QR methods in our implementation MSPAI 1.0. A second switch involving also CSparse with AMD preordering will be contained in a future release. Figure 4.2 shows concrete SPAI runtimes using both LAPACK and sparse QR methods. We increased the threshold for the switch between dense and sparse from 5% to 30% and observe a minimum runtime for about 12% for the test matrix orsirr2. In the actual MSPAI implementation, we set the switch to a default value of 15%. Note that this computation using exclusively LAPACK takes 7.28s — considerably slower

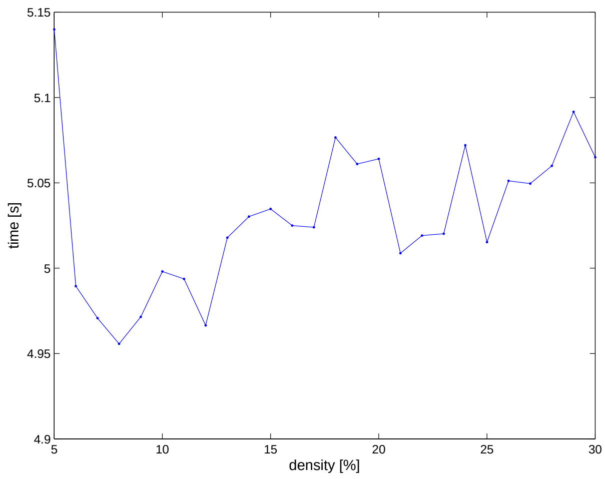


Figure 4.2: Runtime comparison for different switch values between LAPACK and sparse QR methods. SPAI for test matrix orsirr2 with $\Upsilon_{12,7}$, $\varepsilon = 10^{-3}$, start pattern $\mathscr{P}(A)$. Time with LAPACK QR only: 7.28s.

than with involving CSparse. These runtimes were measured on an Intel Core2Duo with 2 2.4 GHz processors using ATLAS 3.6.0.

4.1.3 QR Updates and Sparse QR Updates

In the above review of LS solution methods, we also mentioned the applicability of several methods in terms of updating a given QR decomposition. This is a central task in computing SPAI or MSPAI, since the SPAI-specific pattern updates lead to an enlarged LS problem. Therefore, we will now investigate how these QR updates result in improved runtimes. This section follows closely the presentation from [48, 59].

First, we repeat the notation from Section 2.1.2. Let us assume that we have already computed the LS solution $\hat{m}_k$ using the QR decomposition

$$\hat{A} = Q\tilde{R} \in \mathbb{R}^{p\times q} \text{ with } Q \in \mathbb{R}^{p\times p},\; \tilde{R} = \begin{pmatrix} R \\ 0 \end{pmatrix} \in \mathbb{R}^{p\times q},\; R \in \mathbb{R}^{p\times p}.$$

Hereby holds $p \geq q$ according to the index set of columns $\mathscr{J}$ with $q = |\mathscr{J}|$, and the shadow thereof, the set of indices of nonzero rows $\mathscr{I}$ in $A(:,\mathscr{J})$ with $p = |\mathscr{I}|$. If $\|Am_k - e_k\|_2$ is larger than a user provided ε, SPAI performs a pattern update (Section 2.1.2) in order to increase the accuracy of the approximation. The result is a set of new columns

$$\tilde{\mathscr{J}}, \qquad \tilde{q} := \left|\tilde{\mathscr{J}}\right|.$$

This leads to an increased set of columns $\mathscr{J} \cup \tilde{\mathscr{J}}$ and consequently to an enlarged shadow

$$\mathscr{I} \cup \tilde{\mathscr{I}}, \qquad \tilde{p} := \left|\tilde{\mathscr{I}}\right|.$$

Note that $\tilde{\mathscr{I}}$ can also be the empty set, and $p + \tilde{p} \geq q + \tilde{q}$ holds. Hence, starting with

$$A(\mathscr{I}, \mathscr{J}) = \hat{A} = \begin{pmatrix} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \end{pmatrix},$$

the enlarged submatrix reads

$$A(\mathscr{I} \cup \tilde{\mathscr{I}}, \mathscr{J} \cup \tilde{\mathscr{J}}) =: \bar{A} = \begin{pmatrix} \boxtimes & \boxtimes & \times & \times & \boxtimes & \times \\ \boxtimes & \boxtimes & \times & \times & \boxtimes & \times \\ \otimes & \otimes & 0 & 0 & \otimes & 0 \\ \boxtimes & \boxtimes & \times & \times & \boxtimes & \times \\ \boxtimes & \boxtimes & \times & \times & \boxtimes & \times \\ \otimes & \otimes & 0 & 0 & \otimes & 0 \\ \boxtimes & \boxtimes & \times & \times & \boxtimes & \times \end{pmatrix}.$$

$\boxtimes$ denotes the entries in the new columns and already present rows $A(\mathscr{I}, \tilde{\mathscr{J}})$, whereas $\otimes$ stands for the entries in the new columns and new rows $A(\tilde{\mathscr{I}}, \tilde{\mathscr{J}})$, respectively. New columns and rows can occur in any arbitrary position. But we can find a row permutation P_r and a column permutation P_c such that

$$P_r \bar{A} P_c =: \tilde{A} = \begin{pmatrix} \times & \times & \times & \boxtimes & \boxtimes & \boxtimes \\ \times & \times & \times & \boxtimes & \boxtimes & \boxtimes \\ \times & \times & \times & \boxtimes & \boxtimes & \boxtimes \\ \times & \times & \times & \boxtimes & \boxtimes & \boxtimes \\ \times & \times & \times & \boxtimes & \boxtimes & \boxtimes \\ 0 & 0 & 0 & \otimes & \otimes & \otimes \\ 0 & 0 & 0 & \otimes & \otimes & \otimes \end{pmatrix} = \begin{pmatrix} \hat{A} & A(\mathscr{I}, \tilde{\mathscr{J}}) \\ 0 & A(\tilde{\mathscr{I}}, \tilde{\mathscr{J}}) \end{pmatrix}. \tag{4.6}$$

The lower left zero block in $\tilde{A}$ is SPAI-specific, since $\mathscr{I}$ is the shadow of $\mathscr{J}$, i.e. all zero rows are omitted in $A(:, \mathscr{J})$. Therefore, in new rows $\tilde{\mathscr{I}}$, there must be zero values in the columns $\mathscr{J}$. This allows us to employ a simplified version of the standard methods for QR updates [17, 46]. Furthermore, we only have the case where rows or columns are added. From one pattern update step to the next, no rows or columns can be deleted. Thus, using $\hat{A} = Q\tilde{R}$, we can write

$$\begin{aligned} \tilde{A} &= \begin{pmatrix} Q & \\ & I_{\tilde{p}} \end{pmatrix} \left(\begin{array}{c|c} R & Q_1^T A(\mathscr{I}, \tilde{\mathscr{J}}) \\ \hline 0 & Q_2^T A(\mathscr{I}, \tilde{\mathscr{J}}) \\ 0 & A(\tilde{\mathscr{I}}, \tilde{\mathscr{J}}) \end{array}\right) = \begin{pmatrix} Q & \\ & I_{\tilde{p}} \end{pmatrix} \left(\begin{array}{ccc|ccc} \star & \star & \star & \tilde{\boxtimes} & \tilde{\boxtimes} & \tilde{\boxtimes} \\ & \star & \star & \tilde{\boxtimes} & \tilde{\boxtimes} & \tilde{\boxtimes} \\ & & \star & \tilde{\boxtimes} & \tilde{\boxtimes} & \tilde{\boxtimes} \\ \hline & & & \tilde{\boxtimes} & \tilde{\boxtimes} & \tilde{\boxtimes} \\ & 0 & & \tilde{\boxtimes} & \tilde{\boxtimes} & \tilde{\boxtimes} \\ & & & \otimes & \otimes & \otimes \\ & & & \otimes & \otimes & \otimes \end{array}\right) \\ &= \begin{pmatrix} Q & \\ & I_{\tilde{p}} \end{pmatrix} \left(\begin{array}{c|c} R & B_1 \\ \hline 0 & B_2 \end{array}\right). \end{aligned} \tag{4.7}$$

The subscript of $I_{\tilde{p}}$ denotes the dimension of the unity, $Q1$ and $Q2$ are defined as $Q_1 := Q(1 : q, :)$, $Q_2 := Q(q + 1 : p, :)$, $\star$ symbolize the entries of R, and $\tilde{\boxtimes}$ represent the entries of

$Q^T A(\mathscr{I}, \tilde{\mathscr{J}})$. The upper right block B_1 has dimension $q \times \tilde{q}$. In order to complete the QR decomposition of $\tilde{A}$, we further have to compute the QR decomposition of the lower right block B_2 with dimension $(p + \tilde{p} - q) \times \tilde{q}$. Let therefore denote $B_2 = Q_B R_B$. Then, (4.7) becomes

$$\tilde{A} = \begin{pmatrix} Q & \\ & I_{\tilde{p}} \end{pmatrix} \begin{pmatrix} I_q & \\ & Q_B \end{pmatrix} \begin{pmatrix} R & B_1 \\ 0 & R_B \end{pmatrix}, \tag{4.8}$$

and we can solve the augmented LS problem for the right-hand side $P_r e_k(\mathscr{I} \cup \tilde{\mathscr{I}})$. From the solution $\tilde{m}_k = P_c^T m_k(\mathscr{J} \cup \tilde{\mathscr{J}})$ of the permuted system, we can recover the solution components in the correct order by $P_c \tilde{m}_k$. We summarize this procedure in Algorithm 4.1.

Algorithm 4.1 QR update for SPAI-like LS problems.

```
Require: $\hat{A}$, $Q$, $R$, $\mathscr{J}$, $\mathscr{I}$, $\bar{A} = A(\mathscr{I} \cup \tilde{\mathscr{I}}, \mathscr{J} \cup \tilde{\mathscr{J}})$
1: Find row permutation $P_r$ and column permutation $P_c$, such that $P_r \bar{A} P_c = \tilde{A}$ (4.6)
2: Apply $Q^T$ to $A(\mathscr{I}, \tilde{\mathscr{J}})$
3: $B_1 \leftarrow$ rows $(1 : q)$ from $Q^T A(\mathscr{I}, \tilde{\mathscr{J}})$
4: $B_2 \leftarrow$ rows $(q+1) : p$ from $Q^T A(\mathscr{I}, \tilde{\mathscr{J}})$ and $A(\tilde{\mathscr{I}}, \tilde{\mathscr{J}})$, see (4.7)
5: Compute QR decomposition of $B_2 = Q_B R_B$
6: $\tilde{m}_k \leftarrow$ Solve augmented LS problem via enlarged QR decomposition (4.8) for right-hand
   side $P_r e_k(\mathscr{I} \cup \tilde{\mathscr{I}})$
7: $m_k(\mathscr{J} \cup \tilde{\mathscr{J}}) \leftarrow P_c \tilde{m}_k$
```

Instead of computing the full QR decomposition of $\bar{A}$, we can reduce the computational costs to that of one application of Q^T and the QR decomposition of a smaller submatrix B_2, see lines 2 and 5 in Algorithm 4.1.

Example 4.1 We compare the runtimes for the computation of SPAI for orsirr2 [77] with different update parameters Υ, see Definition 2.1. First, we do not perform QR updates, i.e. every augmented $\bar{A}$ is completely factorized. Then, we measure the times when we apply QR updates using LAPACK routines, therefore not taking into account sparsity structures. Table 4.1 summarizes the result. For the settings $\Upsilon_{8,4}$, $\Upsilon_{12,9}$, and $\Upsilon_{108,1}$ QR updates lead

Table 4.1: Comparison of runtimes for SPAI with (t_0) and without (t_1) using QR updates. The example matrix orsirr2 has dimension $n = 886$ and we start with diagonal pattern and an $\varepsilon = 10^{-5}$, which is low enough such that a maximum number of pattern updates are carried out.

Υ	no QR updates (t_0)	QR updates (t_1)	$\frac{t_0}{t_1}$
$\Upsilon_{8,4}$	0.77	0.61	1.26
$\Upsilon_{12,9}$	10.15	3.10	3.27
$\Upsilon_{108,1}$	83.20	19.51	4.26
$\Upsilon_{1,108}$	0.10	0.13	0.77

to a significantly improved runtime. The first two can be seen as a standard setting for SPAI pattern updates, whereas the third is an extreme one only adding one new entry to the pattern per update step. In that case, the block B_2 consists of only one column. Thus the factorization is clearly cheaper than factorizing the whole matrix $\bar{A}$. The other extreme parameter pair $\Upsilon_{1,108}$ is only shown for completeness. $\Upsilon_{1,108}$ means that we apply only one pattern update step adding 108 new entries. Hence, B_2 is nearly as large as $\hat{A}$, so the overhead of the QR updates outweigh the potentially small savings. However, this setting is not advisable in practice, since we should only add a few new entries per update step. Adding more than one is a heuristic, which performs well up to a “few” new entries. If we add too many, the resulting preconditioner does not improve the condition number sufficiently. For $\Upsilon_{108,1}$, we obtain a condition number of 1.8. Conversely, $\Upsilon_{1,108}$ leads to $\kappa = 163.4$. This test was run on an Intel Core2Duo processor with 2×2.4 GHz using ATLAS 3.6.0 for the QR decompositions. We used our C++ MSPAI implementation.

Our contribution to the QR update approach is the combination with the sparse techniques from Section 4.1.2. For our tests [82, 87], we implemented three variants of QR updates:

- **QRUP1, Dense-Dense:** The initial QR decomposition of $\hat{A}$ is carried out with dense QR LAPACK methods. We also compute the update factorization of B_2 in dense mode.
- **QRUP2, Sparse-Sparse:** Both the factorization of $\hat{A}$ and the factorization of B_2 are computed using the sparse routines from CSparse, see Section 4.1.2 and [29, 32].
- **QRUP3, Hybrid (Sparse-Dense):** When the density of $\hat{A}$ is low enough, we determine its QR factorization by sparse methods, but factorize B_2 with LAPACK routines.

The special structure of B_2 motivates the hybrid variant QRUP3. By equation (4.7), we can see that an orthogonal matrix Q^T is applied to the first $p - q$ rows of B_2. Hence, these first rows are usually dense, i.e. the overall density of B_2 is unlikely to be low enough so that we could benefit from sparse methods. This is also the reason why we omitted the fourth possible combination “Dense-Sparse”. All three methods require elaborate storage schemes for the Householder reflectors and the upper triangular matrices. Especially QRUP3 leads to very complex structures, since it involves both dense and sparse storage types. For the details on the implementation, we refer to [82].

Now, we want to concentrate on the reduction of runtimes when we use QR updates QRUP1–QRUP3 for the computation of SPAI. The example above already demonstrated the effectiveness of QRUP1. In Table 4.2 we can see that the other variants also lead to reduced runtimes. The combination of QR updates and sparse techniques leads to further improvements.

Table 4.2: Comparison of runtimes for SPAI with the various update methods QRUP1–QRUP3, and without using QR updates. Example matrix orsirr2 has dimension $n = 886$ and we start with diagonal pattern and an $\varepsilon = 10^{-5}$, which is low enough such that a maximum number of pattern updates are carried out.

Υ	dense QR	sparse QR	QRUP1	QRUP2	QRUP3
$\Upsilon_{8,4}$	0.77	1.10	0.61	0.61	0.58
$\Upsilon_{12,9}$	10.15	5.98	3.10	3.06	3.26
$\Upsilon_{108,1}$	83.20	49.91	19.51	14.25	19.27
$\Upsilon_{1,108}$	0.10	0.18	0.13	0.16	0.16

4.1.4 Single Precision QR Decomposition

QR updates are a methodical improvement for SPAI computations. A rather technical approach to reduce computation times are single precision routines for the QR factorizations.

The lack of accuracy should then be partially diminished by a few steps of iterative refinement. Theoretically, single precision computations are faster by a factor of two than operations on double precision variables. On special architectures such as the Cell processor [64], even accelerations of factor ten can be achieved.

The idea is to perform the QR decompositions for our LS problems of the form (4.1) in single precision to save time. Then, in double precision, we want to improve the accuracy of the solution using iterative refinement steps. For the LU decomposition, this was already explained in Section 1.1.4. Björck and Golub formulate the equivalent procedure based on normal equations in [19]. We follow the representation in [46] and define $r := \hat{e}_k - \hat{A}\hat{m}_k$. When

$$\begin{pmatrix} I_p & \hat{A} \\ \hat{A}^T & 0 \end{pmatrix} \begin{pmatrix} r \\ \hat{m}_k \end{pmatrix} = \begin{pmatrix} \hat{e}_k \\ 0 \end{pmatrix}, \tag{4.9}$$

then $\left\| \hat{A}\hat{m}_k - \hat{e}_k \right\|_2 = \min$, since

$$\left.\begin{aligned} r + \hat{A}\hat{m}_k = \hat{e}_k \\ \hat{A}^T r = 0 \end{aligned}\right\} \quad \Longrightarrow \quad \hat{A}^T\hat{A}\hat{m}_k = \hat{e}_k.$$

We obtain the j-th refinement step (setting $r^0 = 0$, $m_k^0 = 0$) by

$$\begin{aligned} \begin{pmatrix} f^j \\ g^j \end{pmatrix} &= \begin{pmatrix} \hat{e}_k \\ 0 \end{pmatrix} - \begin{pmatrix} I_p & \hat{A} \\ \hat{A}^T & 0 \end{pmatrix} \begin{pmatrix} r^j \\ \hat{m}_k^j \end{pmatrix}, & (4.10a) \\ \begin{pmatrix} I_p & \hat{A} \\ \hat{A}^T & 0 \end{pmatrix} \begin{pmatrix} p^j \\ z^j \end{pmatrix} &= \begin{pmatrix} f^j \\ g^j \end{pmatrix}, & (4.10b) \\ \begin{pmatrix} r^{j+1} \\ x^{j+1} \end{pmatrix} &= \begin{pmatrix} r^j \\ x^j \end{pmatrix} + \begin{pmatrix} p^j \\ z^j \end{pmatrix}. \end{aligned}$$

Given the QR decomposition $\hat{A} = Q\tilde{R} = Q\begin{pmatrix} R \\ 0 \end{pmatrix}$, we can formulate (4.10b) as follows:

$$\begin{pmatrix} I_q & 0 & R \\ 0 & I_{p-q} & 0 \\ R^T & 0 & 0 \end{pmatrix} \begin{pmatrix} h \\ f_2 \\ z \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ g \end{pmatrix}.$$

f_1, f_2 are the first q and $p - q$ rows of $Q^T f$, respectively. h consists of the rows $(1 : q)$ of $Q^T p$. Hence, we can calculate p and z by solving the triangular systems

$$\begin{aligned} R^T h &= g, \\ Rz &= f_1 - h, \end{aligned}$$

and setting $p = Q\begin{pmatrix} h \\ f_2 \end{pmatrix}$. The advantage of this refinement method is that it does not only update the solution $\hat{m}_k^j$ iteratively, but also the residual r^j. Björck and Golub emphasize that the residuals (4.10a) must be computed in a higher precision than the factorization of $\hat{A}$ if they should lead to improvements.

Example 4.2 For our numerical test of this type of refinement, we compute static SPAI preconditioners M with $\mathscr{P}(M) = \mathscr{P}(A)$ for a few test matrices from [77, 101]. Additionally, we compute static SPAI preconditioners, where we determine the QR decompositions via

Table 4.3: Comparison of iteration counts for double precision SPAI and SPAI computed using single precision QR decomposition and 5 steps of iterative refinement. Both are computed statically without pattern updates and with $\mathscr{P}(A)$, unless stated differently. We use BiCGSTAB as solver up to a relative residual of 10^{-6}. If no right-hand sides were provided by [77, 101], we used random right-hand sides.

Matrix	double precision SPAI	SPAI using iterative refinement QR
orsirr2	90.5	93.5
bcsstk14	259.5	257.5
pores1 ($\mathscr{P}(A^2)$)	5.5	5.5
pores2 ($\mathscr{P}(A^2)$)	328.5	390.5
pores3	127.5	122.5
raefsky1	77.5	77.5
raefsky2	100.5	100.5
raefsky3	68.5	68.5
raefsky4	3	3
raefsky5	2	2
raefsky6	17	17
sherman1	48.5	48.5
sherman2	0.5	0.5
sherman3	139.5	136.5
sherman4	33.5	33.5
sherman5	62.5	62.5
fidap015 ($\mathscr{P}(A^3)$)	83.5	no convergence

single precision LAPACK methods. The resulting LS solutions are then refined with 5 steps of iterative refinement presented above. In Table 4.3, we can see that in many cases, the quality of SPAI computed with iterative refinement in terms of iteration number in BiCGSTAB is identical to the preconditioning effect of SPAI computed completely with double precision QR. However, even in this rather small collection of test problems, there are test matrices where the refinement SPAI leads to a considerably increased iteration count. For fidap015, not even convergence was observed. A meaningful comparison of runtimes would require a pure C implementation and cannot be provided by our MATLAB test.

This example shows that SPAI-type preconditioners which are computed with single precision QR and iterative refinement can yield results of equal quality, but there is no guarantee for that. If this approach behaves differently for every input matrix, the routines in SPAI or MSPAI for the LS solutions lose their black box character. This is the reason why we omitted single precision QR in our current MSPAI implementation. However, Demmel et al. [33, 34] recently presented approaches which allow to compute the residuals in extra-high precision, e.g. in double-double precision. Moreover, they employ several algorithms for estimating condition numbers within their computations and they developed strategies to detect convergence in the iterative refinement steps. They also propose to add special much more elaborate routines for extra-precise iterative refinement to LAPACK. Therefore, we recommend to test these new functions for future versions of MSPAI as soon as they are officially contained in LAPACK.

4.2 Caching Algorithm

Beside efforts in improving the runtime of LS solutions, we also consider optimizations on a higher algorithmic level. The computation of a static MSPAI for a matrix $A \in \mathbb{R}^{n \times n}$ involves QR decompositions of n smaller submatrices $\hat{A}$, one for each column (see Section 2.1.1). In case of dynamic pattern updates, we need to factorize again n submatrices and so on. However, matrices arising from PDE discretizations are typically very structured. For instance, they have a band structure or block structures. In terms of computing an MSPAI for such matrices, we can therefore assume that many $\hat{A}$ are identic. Hence, many QR decompositions are carried out redundantly without reusing the previously computed factors. We define the ratio

$$\sigma := 1 - \frac{\#\text{ pairwise different } \hat{A}}{\text{dimension of } A}.$$

Therefore, σ roughly indicates the ratio of how many QR decompositions could be saved, see Table 4.4. Note that σ does not contain information about the size of the identic $\hat{A}$, i.e. σ does not necessarily give a reliable measure about possible runtime improvements. However,

Table 4.4: Number of pairwise different submatrices $\hat{A}$ for several test matrices from [77] when we compute a SPAI M with a static pattern $\mathscr{P}(M) = \mathscr{P}(A)$. Matrices CFDxxx arise from computational fluid dynamics, see Section 5.2. Matrix laplace100 is a standard five-point stencil discretization of the 2D Laplace equation.

Number	Matrix	dimension	# different $\hat{A}$	σ
1	orsirr1	1030	920	10.7%
2	orsirr2	886	776	12.4%
3	cavity03	317	107	66.2%
4	mahindas	1258	881	30.0%
5	bcsstk25	15439	14752	4.4%
6	bcsstk16	4884	3773	22.7%
7	bcsstm25	15439	5239	66.1%
8	fidap011	16614	11399	93.1%
9	fidap014	3251	3169	2.5%
10	fidapm29	13668	10839	20.7%
11	s2rmq4m1	5489	5123	6.7%
12	CFD8x8x8o	512	145	71.7%
13	CFD8x8x8o24	512	328	35.9%
14	CFD16x16x32o24o210	8192	385	95.3%
15	CFD16x16x16o24	4096	349	91.5%
16	laplace100	10000	24	99.8%

saving all previously computed factorizations would lead to uncontrollable and, even worse, unpredictable memory usage. Therefore, we only save a fixed number of factorizations: we use a cache.

In Algorithm 4.2, we see how caching works. If $\hat{A}$ and the factorization thereof is already in the cache, we use it to compute the current $\hat{m}_k$. When the cache element additionally contains the same right-hand side $\hat{b}_k$, we can directly extract the solution $\hat{m}_k$ from the cache. However, if $\hat{A}$ cannot be found in the cache, we determine its QR decomposition and put it into the cache together with the right-hand side $\hat{b}_k$ and the solution $\hat{m}_k$. In case the

Algorithm 4.2 Caching for MSPAI.

```
Require: $\hat{A}$, $\hat{b}_k$, Cache size $cs$
 1: if $\hat{A} \in$ cache then
 2:    if $\hat{b}_k \in$ cache then
 3:       extract previously computed solution $\hat{m}_k$
 4:    else
 5:       compute $\hat{m}_k$ using cached $\hat{A} = QR$
 6:       put $\hat{m}_k$ into cache
 7:    end if
 8:    increase access count of this $QR$ by 1
 9: else
10:    compute QR decomposition of $\hat{A}$
11:    solve for $\hat{m}_k$
12:    put $QR = \hat{A}$, $\hat{b}_k$, and $\hat{m}_k$ into cache
13:    if cache size $cs$ is exceeded then
14:       replace cache element with lowest access count by $QR = \hat{A}$, $\hat{b}_k$, and $\hat{m}_k$
15:    end if
16: end if
17: return $\hat{m}_k$
```

maximum cache size cs is reached, we replace the cache element with lowest access count, namely the element which is accessed most rarely, by the new QR decomposition, the new right-hand side, and the new LS solution. This ensures a limited memory consumption. Furthermore, we employ the *last recently used principle* (*LRU*) [96]. As a consequence, elements which are required more frequently stay longer in the cache (i.e. have a higher access count), whereas less frequently needed ones (low access count) drop out. Moreover, elements are stored in a sorted way according to their access count in order to be accessed fastest. Hence, the choice of the cache size cs plays an important role. If chosen too small, not even frequently arising $\hat{A}$ are kept in the cache long enough and drop out too early. Conversely, if cs is too large, the times for searching $\hat{A}$ in the cache increase and lead to an overhead. Moreover, the cache contains too many "unimportant" $\hat{A}$, thus wasting memory. More detailed information on our implementation of MSPAI caching such as data structures and hashing via superkeys is given by [87].

As shown in Figure 4.3, MSPAI caching leads to significant runtime improvements in the case of structured matrices. In other cases, where not as many or only smaller $\hat{A}$ can be cached and reused, MSPAI caching does not lead to significantly higher runtimes. Therefore, we can employ it as default method. When the structure of a matrix A allows us to profit from avoiding redundant QR decompositions, MSPAI caching will profit thereof. Conversely, if A is unstructured, our caching approach will not increase the runtimes. According to our tests, which are reported in a more detailed way in [87], we set the default cache size to 60. For unstructured matrices with few redundant $\hat{A}$, this value should be chosen smaller. For highly structured matrices, a higher cache size will lead to even larger savings. Thus, knowledge about the structure of the underlying matrix allows a more meaningful choice of the cache size than the default value and so to optimize the runtimes. When we run MSPAI in a parallel environment, each computing node has its own cache. Therefore, caching does not cause further communication overhead between the nodes.

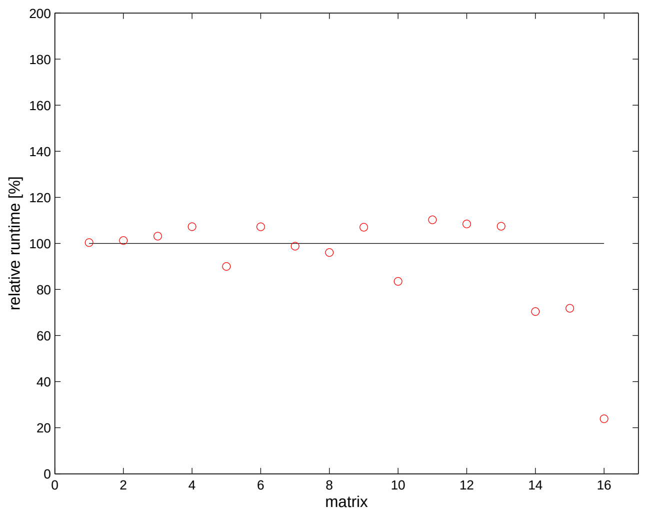


Figure 4.3: Runtime comparison for the test matrices from Table 4.4 with default cache size 60. SPAIs involving pattern updates were computed using different parameter settings.

4.3 Maximum Sparsity Pattern

As a further runtime improvement, we also implemented support for maximum sparsity patterns in our MSPAI 1.0 code which were explained in Section 2.1.3. The user can provide an additional file with the maximum pattern which MSPAI may use for pattern update steps. Since we are more interested here in the runtimes than in the choice of meaningful maximum patterns, we precomputed SPAIs with several pattern update steps and took the pattern of the resulting matrices as maximum patterns for our tests. Thereby, we guarantee that the results are identical in both cases.

First, we test the performance with *no prefetching*, i.e. if one computing node needs a column of A, it only requests this single column. As we can see in Table 4.5, this already leads to considerable runtime reductions. The reason for this effect are the savings in the pattern update steps. On the one hand, less set operations have to be performed and less ρ_j have to be computed (see Section 2.1.2). On the other hand, this also affects the amount of communication between the nodes. Since we have to compute less ρ_j, less columns of A have to be requested and transferred. Hence, network traffic is reduced.

Table 4.5: Runtime comparisons for computing a SPAI of test matrix fidap011 [77] with dimension $n = 16614$, diagonal start pattern, and $\Upsilon_{6,8}$ in a parallel environment with p computing nodes. "MP" indicates the use of a maximum pattern.

p	unrestricted SPAI	MP (no prefetching)	MP (full prefetching)
16	27.85	19.93	18.71
32	16.68	11.59	10.66
64	10.75	6.95	6.38
128	6.45	4.17	3.97

Huckle [61] proposed to use the maximum pattern from which every processor requests the columns it needs in the beginning of the computation such that no further communication would be needed. We refer to this variant as maximum pattern with *full prefetching.* With this strategy we get further rather slight improvements in the runtimes, see again Table 4.5. After the initialization process, there is still some communication required for the pattern update steps.

We introduce a third variant where we only prefetch $2k$ more columns than we actually need, the *k-prefetching.* Hereby, when one node needs column j, it requests columns $(j-k):(j+k)$ within the maximum pattern. This heuristic is motivated by the assumption that if index j is added to the sparsity pattern, then it is quite likely that indices near j will be added in one of the next update steps. Table 4.6 shows increased runtimes for this variant compared to no and full prefetching (Table 4.5).

Table 4.6: Runtime comparisons for computing a SPAI of test matrix fidap011 [77] using a maximum pattern with k-prefetching, $\Upsilon_{6,8}$, and $p = 16$ computing nodes.

k	maximum pattern with k-prefetching
1	20.52
3	20.46
10	20.62
100	20.47
1000	27.15

In the context of maximum sparsity patterns, we made another observation. Since Barnard's SPAI implementation shows a perfect scaling behavior in parallel environments [10], we took over the communication server and the parallel caching strategy from the SPAI 3.2 code. Parallel caching saves several columns which were previously requested such that another request for these columns does not result in internodal communication. Instead, they can just be taken from the local memory. When we switch off this cache, the absolute runtimes increase dramatically but also the impact of the prefetching strategies is much larger, see Table 4.7. So, a large part of the effect of maximum patterns is already anticipated by the parallel caching. Nevertheless, these runtimes strongly encourage the use of a maximum sparsity pattern. Further test data sets can be found in [87].

Table 4.7: Runtime comparisons for computing a SPAI of test matrix fidap011 [77] diagonal start pattern and $\Upsilon_{6,8}$ in a parallel environment with p computing nodes. "MP" indicates the use of a maximum pattern. Parallel caching is now disabled.

p	unrestricted SPAI	MP (no prefetching)	MP (full prefetching)
16	236.57	90.01	88.13
32	166.26	64.21	63.67
64	116.51	43.66	43.57
128	95.38	32.82	32.90

4.4 Comparison of MSPAI 1.0 and SPAI 3.2

In the previous sections, we presented several optimizations to improve the runtime of MSPAI. Moreover, it is especially interesting how our MSPAI 1.0 [82, 86, 87] performs compared to the current standard SPAI implementation SPAI 3.2 [91] by Barnard and Grote

[9, 10]. Of course, we can only compare the unfactorized inverse approximation variant of MSPAI. First, we will point out the functional advantages of MSPAI such as support for problems with complex valued matrices and vectors or the flexibility in terms of starting patterns. In the second part, the focus will be on concrete runtime comparisons.

4.4.1 Features of MSPAI 1.0

The most obvious difference in functionality between MSPAI 1.0 and SPAI 3.2 is the support for all MSPAI probing variants presented in this thesis (Chapter 3, Box 3.1). However, even when we restrict our review to unfactorized inverse approximations, our implementation contains some improvements:

- *Templates* in the C++ programming language [95] allow us to provide support for complex problems $Ax = b$ with $A \in \mathbb{C}^{n\times n}$, $x, b \in \mathbb{C}^n$. Barnard's code does not cover this class of problems.
- SPAI 3.2 is tailored to the pure dynamic pattern update version of SPAI, i.e. it always starts with a diagonal pattern. In contrast to MSPAI 1.0, it is not possible to prescribe a different start pattern or to compute a static SPAI omitting pattern update steps.
- Our implementation returns the residual matrix $R = I - AM$, which can be useful for symmetrization methods such as SPAI acceleration, see Section 3.6.3.
- In MSPAI 1.0, the user can also pass a maximum sparsity pattern, which can lead to an improved parallel behavior and to a faster determination of new indices in the pattern update steps.
- The improvements considered in this chapter such as sparse QR methods, QR updates, and MSPAI caching are further specialties of MSPAI 1.0.

However, in some points, SPAI 3.2 still has a few features which are not yet covered by MSPAI 1.0, but will be contained in future releases. We did not include MATLAB support via mex-files in our code, nor an interface to PETSc [81]. Moreover, we restricted our code to the use of ATLAS (automatically tuned linear algebra software) [3] for solving least squares problems, whereas SPAI 3.2 users can also employ special libraries for their actual hardware such as ACML [1] for AMD processors or the Intel MathKernel Library MKL [67]. This is not a major drawback of MSPAI 1.0, since ATLAS is quite competitive in most cases. Another feature of SPAI 3.2, which we did not imitate, is the block SPAI approach [9]. A graph based algorithm to detect (incomplete) block structures in the original matrix could be useful to determine a meaningful start pattern. This should be equally effective and much cheaper to compute than block SPAI.

As far as parallelization is concerned, both implementations compared here use the message passing interface (MPI) [78]. Barnard showed the communication server principle to be optimal for SPAI-like applications. Parallelization thereof is a quite difficult task, because one cannot estimate in advance how many pattern update steps will be necessary for each column. With the communication server, each node which has completed its task requests further columns from the master node to compute. Hence, the load is distributed as optimal as possible. Therefore, we took over this principle for our implementation. Barnard's second approach to reduce communication between the computing nodes is a parallel cache. A certain number of columns, which were requested before, are stored locally. If they are needed again, they do not have to be transferred over the network but can be taken directly from the local memory. We also took over this parallel caching idea.

4.4.2 Runtimes

To conclude this chapter about the implementation of the algorithms discussed in this thesis, we present direct runtime comparisons between SPAI 3.2 and our MSPAI 1.0 implementation. First, we compare the serial runtimes obtained from including the improvements of this chapter. After that, we will also compare the performance in a parallel computing environment.

The test matrix orsirr2 (dimension $n = 886$) has become one of the standard test matrices for SPAI. Therefore, we will present the serial runtime comparison for this matrix. We used both SPAI 3.2 and our MSPAI 1.0 implementation to compute unfactorized inverse approximations, starting with a diagonal pattern and disabling in both cases the mean value argument for the augmentation in the pattern update steps (see Section 2.1.2). In MSPAI, we enabled QR updates and maximum pattern support. Caching was disabled since it did not show runtime improvements for orsirr2, see Section 4.2. This test was run on an Intel Core2Duo architecture with two 2.4 GHz processors. As we can see in Table 4.8, our implementation outperforms SPAI 3.2 clearly for different pattern update settings Υ. In the standard update scenarios $\Upsilon_{8,4}$ and $\Upsilon_{12,9}$ the runtime is reduced by factors of

Table 4.8: Serial runtime comparisons between SPAI 3.2 and MSPAI 1.0 for matrix orsirr2. Start with diagonal pattern and $\varepsilon = 10^{-5}$.

		MSPAI 1.0			
Υ	SPAI 3.2 (t_0)	QRUP1	QRUP2	QRUP3 (t_1)	$\frac{t_0}{t_1}$
$\Upsilon_{8,4}$	1.56	0.52	0.51	0.46	3.39
$\Upsilon_{12,9}$	9.38	2.95	2.90	2.67	3.51
$\Upsilon_{108,1}$	86.59	17.32	12.50	12.36	7.01
$\Upsilon_{1,108}$	0.01	0.14	0.16	0.16	0.06

more than 3. In the extreme test case $\Upsilon_{108,1}$, we can even observe a reduction by a factor of more than 7. As discussed before in Section 4.1.3, $\Upsilon_{1,108}$ is only shown for completeness. Here, QR updates only lead to an overhead. The resulting preconditioner in this case is by far less effective than in the other pattern update settings.

For the parallel runtime comparison (see Figure 4.4), we consider test matrix fidap014 (dimension $n = 3251$) from Matrix Market [77]. We prescribe a diagonal start pattern and $\varepsilon = 0.01$ for MSPAI 1.0 and SPAI 3.2, as well as $\Upsilon_{16,5}$ as pattern update parameter. The resulting preconditioners are identical. Computations were carried with a doubled number of nodes in every step from $p = 1$ to $p = 128$. No caching, QR updates or maximum sparsity patterns were enabled in MSPAI 1.0 such that the focus was completely on the parallel efficiency. Since we took over the communication server principle and the parallel cache from Barnard's implementation [10], MSPAI 1.0 shows the same good scaling behavior as SPAI 3.2. We omitted the block SPAI approach [9] from SPAI 3.2 in our implementation. Therefore, MSPAI has considerably less computational overhead and therefore clearly lower runtimes.

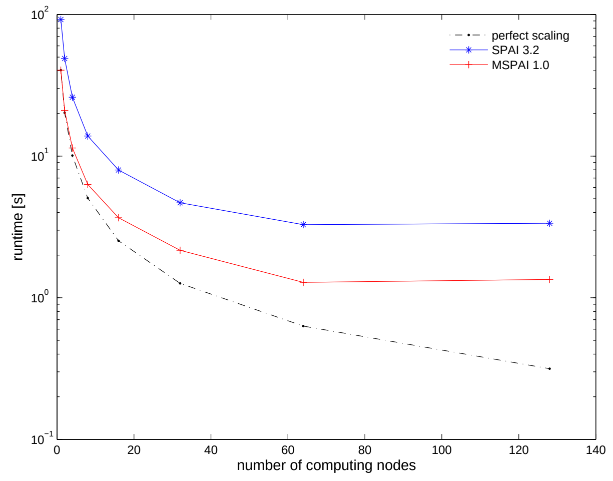


Figure 4.4: Parallel runtime comparison for fidap014 (see Table 4.4). MSPAI 1.0 and SPAI 3.2 in parallel computing environment with nodes $p \in \{1, 2, 4, 8, 16, 32, 64, 128\}$ with $\Upsilon_{16,5}$ pattern updates, $\varepsilon = 0.01$, and diagonal start pattern.

Chapter 5

Numerical Examples

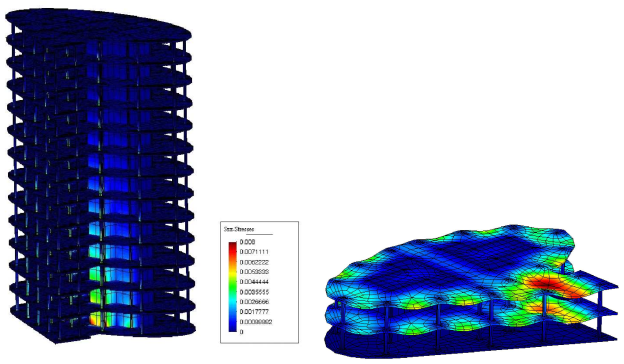
After having developed the new MSPAI probing approach including several symmetrization techniques and having presented an efficient and elaborate implementation in the previous two chapters, we want to focus on numerical tests of the methods presented in this thesis. At the beginning, we will demonstrate the preconditioning of dense matrices with sparsification steps in order to obtain a meaningful sparsity pattern for FSPAI. This first example, which is taken from statics' simulation, was joint work with Ralf-Peter Mundani [79]. The focus of the second section will also be on FSPAI, but now in terms of pattern updates and how they are influenced by approximate minimum degree preordering, see also Section 1.3.3. The example matrices considered here arise in computational fluid dynamics and were provided by Ionel Muntean [21].

Section 5.3 builds the main part of this chapter. We will investigate the effect of MSPAI probing in domain decomposition methods, as well as for Stokes problems. The computation of spectrally equivalent sparse explicit approximations of dense matrices will also be considered. In the next section, we compare classical probing, ILU and MILU preconditioners with MSPAI probing. These results were published in [63].

Finally, we investigate the effect of consecutive probing steps. This was joint work with Stefanie Hörmann [58].

5.1 Example from Statics' Simulation

In order to demonstrate the need to find a good compromise between approximation quality and computation time for a preconditioner, we present here an example from statics' simulation. It is part of the of the UNIQA office tower [100] in Vienna (see Figure 5.1) and it is taken from [79]. The aim is to compute the displacements of the building under its permanent weight. This information is used to determine the stresses and strains in several parts of the building. The discretization is done with a p-version of the Finite Element Method [105], which means that higher accuracy is not achieved by finer meshes, but higher polynomial degrees of the ansatz functions. The geometric structure is divided up recursively by a nested dissection approach [44] and stored in an octree [85]. To solve this system, the Schur complements are computed in the leaves of the octree and passed up to the root node. There, they are assembled in the root matrix. The result of this bottom-up process is a quite dense and ill-conditioned system of linear equations, which we try to precondition and solve here. The solution is then passed down to the leaves again and we get the displacements.



(a) UNIQA tower.

(b) Two floors of the UNIQA tower.

Figure 5.1: Finite Element discretization of the UNIQA office building in Vienna (taken from [79]).

We chose an example representing the discretization of two floors of the UNIQA tower. This leads to a system with about 2500 unknowns and a root matrix A which is symmetric and very ill-conditioned as it has an (estimated) condition number of about 10^{26}. Without preconditioning, no convergence of the cg method could be achieved. Furthermore, with a density of about 50%, it cannot be regarded as sparse anymore. Here, we demonstrate preconditioning with a prescaling step followed by sparsifying the pattern in order to construct a sparse FSPAI preconditioner.

After a symmetric Jacobi prescaling $\tilde{A} := DAD$ with

$$D = \begin{cases} \frac{1}{\sqrt{a_{ij}}} & \text{for} \quad i = j, \\ 0 & \text{for} \quad i \neq j \end{cases},$$

all entries of $\tilde{A}$ lie in $[-1, 1]$, which makes it much easier to identify meaningful thresholds for the sparsification step. With this $\tilde{A}$, the cg method converges after 630 iterations to a relative residual of 10^{-8}. In our MATLAB implementation, this takes 32s on a 1600 MHz Intel Centrino laptop.

Now, we thin-out $\tilde{A}$ using various drop tolerances (`dropTOL`) as threshold, which means dropping entries of $\tilde{A}$, whose absolute values are smaller than `dropTOL`:

$$\tilde{A}_{\text{thin},ij} := \begin{cases} a_{ij} & \text{if} \quad |a_{ij}| > \texttt{dropTOL}, \\ 0 & \text{else.} \end{cases}$$

Then, we compute an FSPAI L with the pattern of $\tilde{A}_{\text{thin}}$ without pattern updates. L is used as a preconditioner for a symmetrically preconditioned cg method. The convergence behavior for different drop tolerances is shown in Figure 5.2. The FSPAIs need significantly fewer iterations to achieve the same accuracy as the Jacobi prescaled matrix. We can also state that for smaller drop tolerances, which means denser $\tilde{A}_{\text{thin}}$ and therefore denser FSPAI, the preconditioning effect is improved. This meets exactly our expectations. In Table 5.1, we present for different `dropTOL`s the `nnz` of $\tilde{A}_{\text{thin}}$, the computation times for FSPAI and the pcg solution process as well as the number of cg iterations. For decreasing `dropTOL`s, we get an increasing number of nz in $\tilde{A}_{\text{thin}}$ and, consequently, higher computation times for FSPAI. On the other hand, the computation time for cg and the iteration number are

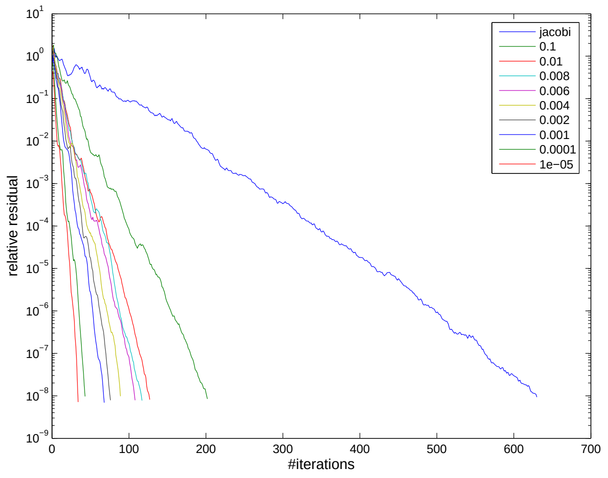


Figure 5.2: Convergence for Jacobi preconditioning and FSPAI for different drop tolerances in the sparsification step.

reduced. The optimal value lies in the middle as a compromise with a total simulation time of ≈ 5.9s.

Table 5.1: Densities and timing results for constructing FSPAI and solving pcg.

`dropTOL`	$\left\|\mathscr{P}\left(\tilde{A}_{\text{thin}}\right)\right\|$	time FSPAI	#iter pcg	time pcg	total time
1.00e-01	14616	0.21	202	10.69	10.90
1.00e-02	47940	0.54	127	6.87	7.41
8.00e-03	53640	0.61	117	6.40	7.01
6.00e-03	61950	0.75	108	5.96	6.71
4.00e-03	76098	0.99	89	4.95	$\boxed{5.94}$
2.00e-03	107436	1.72	76	4.35	6.07
1.00e-03	150472	3.14	68	4.08	7.22
1.00e-04	449380	29.36	43	3.32	32.68
1.00e-05	998722	151.04	34	3.73	154.77

5.2 Effect of AMD on FSPAI Pattern Updates

In Section 2.2.3, we have already observed that an AMD preordering step reduces the `nnz` in an FSPAI with pattern updates significantly. This leads to sparser preconditioners, thus to lower times for matrix-vector products in an iterative solver, and eventually faster computation times for the FSPAI itself.

As stated before, the matrices for this test arise from energy and impulse preserving discretizations of the Navier-Stokes equations on cartesian grids. The simulated geometry is a channel divided into $n_x \cdot n_y \cdot n_z = n$ cells. A Chorin projection [26] leads to symmetric semidefinite matrices of dimension n. Here, we want to present further simulation results, also for settings involving obstacles in the channel. The naming convention is "CFD n_x x

n_y x n_z" for free channel simulation matrices, and "CFD n_x x n_y x n_z o $p_1 l_1$ o $p_2 l_2$" for the obstacle setting. p_i denotes the location and l_i the size of the obstacle. Table 5.2 shows the results for the augmented test set. Again, the AMD preordering step leads to a sparser dynamic FSPAI.

Table 5.2: Comparison of FSPAI for AMD preordered and unpreordered test matrices arising from Navier-Stokes equations also involving obstacles. The first group involves one, the second group involves two obstacles. FSPAI was computed with $\varepsilon = 0.2$, and the pcg algorithm iterated up to a relative residual of 10^{-6} as stopping criterion.

		without preordering		AMD preordering		
Matrix	n	#iter.	$\texttt{nnz}(L)$	#iter.	$\texttt{nnz}(\hat{L})$	$\frac{\texttt{nnz}(\hat{L})}{\texttt{nnz}(L)}$
CFD8x8x8o23	512	35	6903	34	6397	0.93
CFD8x8x8o24	512	40	6357	43	6091	0.96
CFD8x8x8o25	512	41	6133	41	6020	0.98
CFD8x8x8o34	512	34	6901	32	6458	0.94
CFD10x10x10o23	1000	49	15355	49	13651	0.89
CFD10x10x10o24	1000	53	14760	54	13548	0.92
CFD10x10x10o25	1000	48	14480	55	13280	0.92
CFD16x16x16o23	4096	87	75121	83	64774	0.86
CFD16x16x16o24	4096	107	74526	110	64296	0.86
CFD16x16x16o25	4096	89	74246	88	64187	0.86
CFD8x8x16o23o24	1024	52	14945	52	13867	0.93
CFD8x8x16o23o28	1024	67	13509	74	12869	0.95
CFD8x8x16o23o210	1024	70	12989	79	12600	0.97
CFD10x10x20o23o24	2000	69	32647	71	29108	0.89
CFD10x10x20o24o28	2000	83	31080	86	28387	0.91
CFD10x10x20o24o210	2000	86	30520	91	27874	0.91
CFD16x16x32o23o24	8192	139	155869	144	134394	0.86
CFD16x16x32o24o28	8192	161	154302	171	132284	0.86
CFD16x16x32o24o210	8192	175	153742	175	132160	0.86

5.3 Numerical Examples for MSPAI Probing

In the following, we would like to demonstrate our MSPAI probing methods in several applications. All the following numerical examples were computed in MATLAB. If iteration counts are given, we used the pcg method of MATLAB with zero starting vector, random right-hand sides, and a relative residual reduction of 10^{-6} as stopping criterion. KP0, KP1, and KP2 stand for the different types of probing vectors according to Section 3.4.1. k denotes the number of probing vectors used in the computations.

5.3.1 MSPAI Probing for Domain Decomposition Methods

We consider the five-point stencil discretization of the 2D elliptic problem $u_{xx} + \varepsilon u_{yy}$ on a rectangular grid; usually we consider the isotropic problem with $\varepsilon = 1$. After partitioning the matrix to "dissection form", e.g. by a domain decomposition method with m domains, we obtain:

$$\begin{pmatrix} A_1 & & & & F_1 \\ & A_2 & & & F_2 \\ & & \ddots & & \vdots \\ & & & A_m & F_m \\ G_1 & G_2 & \cdots & G_m & A_{m+1} \end{pmatrix} .$$

We refer to this setting as domain decomposition Laplace. To reduce the linear system in A to the smaller problems A_1, ... A_m, we need the Schur complement

$$S = A_{m+1} - F_1 A_1^{-1} G_1 - ... - F_m A_m^{-1} G_m \ .$$

Here, S will in general be a dense matrix. To avoid the explicit computation of S we seek sparse approximations $\tilde{S}$. In a first step, we determine $\tilde{A}_{i,inv}$ $(i = 1, \dots, m)$ approximations for A_i^{-1}, e.g. by SPAI, and then we compute $\tilde{S} = A_{m+1} - F_1 A_{1,inv} G_1 - ... - F_m A_{m,inv} G_m$. In the next step, we try to modify this approximation with respect to some probing vectors in order to improve the condition number of $\tilde{S}^{-1} S$. First we use the explicit approximation from Section 3.3.2.

In Tables 5.3 and 5.4 we compare the resulting explicit preconditioners for different choices of ρ and the probing vectors. k always indicates the number of probing vectors. Again, we denote with KP0, KP1, KP2 the method of probing vector as stated in Section 3.4.1. In a first step, we approximate the matrices A_i^{-1} by sparse approximate inverses M_i with the pattern of A_i. Then, we probe the resulting $\tilde{S}$, keeping the sparsity pattern unchanged. The result is a tridiagonal pattern in the preconditioner. In a second group of examples, we allow more entries in M_i and $\tilde{S}$, which leads to a pentadiagonal pattern. In Table 5.5 we compare

Table 5.3: Condition numbers for domain decomposition Laplace with explicit approximate probing for $k = 3$ and KP0 probing vectors.

m	dim (A)	dim (S)	$\kappa(SM^{-1})$	$\kappa(SM^{-1}_{\rho=0})$	ρ	$\kappa(SM^{-1}_{\rho})$
5	2916	216	83.99	18.2	25	6.34
6	4225	325	118.6	25.7	25	8.83
7	5776	456	159.6	34.6	30	11.4
8	7569	609	206.9	44.8	30	15.8
9	9604	784	260.5	56.4	30	19.3
10	11881	981	320.4	69.4	30	25.4

different iteration numbers for pcg using the symmetrized preconditioner $(M + M^T)/2$. All Tables 5.3–5.5 show that the preconditioner is improved by probing for a wide range of choices for probing vectors and ρ. For these examples the best results were achieved for KP0, $k = 3$, and $\rho \approx 30$. Unfortunately, for the probing with approximate inverses (see Table 5.6), the probing vectors transform into a nearly sparse vector $e^T S$. Hence, also for large ρ the difference $e^T(M - S)$ cannot be reduced significantly, see also Section 3.5. Furthermore, for large ρ the Frobenius norm of the matrix approximation $SM - I$ gets large, and for large ρ the preconditioner gets worse. Nevertheless, we can observe a slight improvement of the condition number also for approximate inverse preconditioners by choosing k, KPx, and especially ρ very carefully. Tables 5.7, 5.8, and 5.9 display the same behavior for the explicit factorized approximations and different symmetrizations. This shows that e.g. the probing vectors KP1 and KP2 are not so efficient for this problem. Nevertheless, probing

Table 5.4: Condition numbers for domain decomposition Laplace with explicit approximate probing for $n = 10$, $m = 8$, different anisotropy, and different sparsity patterns.

sparsity	KP	k	ρ	ε	$\kappa(SM_\rho^{-1})$
tri	1	3	30	1	15.14
tri	2	3	30	1	16.21
tri	0	3	30	1	15.75
tri	0	1	30	1	15.79
tri	0	2	30	1	16.68
tri	0	4	30	1	22.24
tri	0	3	100	1	23.70
tri	0	3	30	0.1	23.60
penta	0	3	25	1	14.08
penta	0	3	30	1	14.80
penta	0	5	30	1	14.00

Table 5.5: Iteration count for domain decomposition Laplace ($\varepsilon = 1$) with explicit approximate probing for KP0, $k = 3$, $\rho = 0$ or $\rho = 30$.

m	2	3	4	5	6	7	8	9	10	11
#it, $\rho = 0$	19	32	39	44	49	56	62	68	74	81
#it, $\rho = 30$	7	10	12	13	13	15	17	19	22	24

Table 5.6: Condition numbers for domain decomposition Laplace with approximate inverse probing, three probing vectors KP0, minimization relative to the weighted Frobenius norm.

m	$\kappa(S)$	$\kappa(SM_{\rho=0})$	ρ	$\kappa(SM_\rho)$
5	83.99	32.90	200	27.07
6	118.6	46.47	200	38.26
7	159.6	62.53	200	51.48
8	206.9	81.04	200	66.76
9	260.5	102.0	250	84.03
10	320.4	125.5	250	103.4

and MSPAI symmetrization lead to improved condition numbers and faster convergence, also in the factorized form.

Table 5.7: Condition numbers for domain decomposition Laplace with explicit approximate factorized probing, one probing vector KP0 and different symmetrizations.

m	S	M_{ILU}	ρ	$L, \tilde{U}$	$\tilde{U}^T, \tilde{U}$	$\tilde{L}, \tilde{U}$	$\tilde{L}, \tilde{L}^T$	$L_{\alpha,4}(\tilde{L}, \tilde{U}^T)$
5	83.99	18.2	22	7.47	5.74	5.95	6.30	5.97
6	118.6	25.7	28	9.84	8.00	8.13	8.39	8.15
7	159.6	34.6	32	13.5	10.5	10.6	10.9	10.7
8	206.9	44.8	36	17.8	13.4	13.6	13.9	13.6
9	260.5	56.4	42	20.5	16.9	17.1	17.3	17.1
10	320.4	69.4	46	25.8	20.7	20.8	21.0	20.8

In investigating the effect of Schur complement probing, we consider the approximation of the Schur complement described by (3.13) in Section 3.3.5. Following Table 5.10, the

Table 5.8: Condition numbers for domain decomposition Laplace with approximate factorized probing for $m = 8$, for different symmetrization variants.

KP	k	ρ	$L, \tilde{U}$	$\tilde{U}^T, \tilde{U}$	$L_{\alpha,4}(L, \tilde{U}^T)$	$\tilde{L}, \tilde{U}$	$\tilde{L}, \tilde{L}^T$	$L_{\alpha,4}(\tilde{L}, \tilde{U}^T)$
0	1	20	29.6	22.2	15.0	21.8	21.4	21.8
0	1	30	21.5	14.9	13.7	15.2	15.5	15.2
0	1	40	15.8	13.6	17.4	13.8	14.0	13.8
0	1	50	13.3	14.5	29.8	14.9	15.5	14.9
0	1	60	13.4	17.7	43.3	17.5	19.2	17.1
0	1	70	13.7	25.7	53.7	23.9	27.6	22.0
0	2	38	16.8	13.5	16.0	13.7	14.1	13.9
0	3	80	32.2	25.2	23.5	22.3	20.0	16.2
1	3	20	20.3	19.5	36.1	17.6	16.9	17.6
1	3	40	14.7	44.4	52.0	25.2	18.1	25.1
2	3	20	17.6	24.3	63.8	20.5	18.1	20.1
2	1	20	28.3	36.6	67.7	32.3	29.0	32.1

Table 5.9: Condition numbers for domain decomposition Laplace with explicit approximate factorized probing with optimal symmetrization.

m	2	3	4	5	6	7	8	9	10
ρ	5	9	12	15	18	21	23	26	29
κ	1.69	2.36	3.71	5.48	7.66	10.3	13.3	16.7	20.5

condition number is strongly deteriorated in the case $\rho = 0$ without including probing. Only the choice KP0 and large $\rho \approx 100$ lead to a condition number reduction. Hence, this approach is only applicable when combined with probing.

Table 5.10: Condition numbers ($\kappa(S_D M_D)$) for Schur complement preconditioning with probing vectors based on (3.13) from Section 3.3.5, $\kappa(S_D) = 12.67$.

ρ	0	1	10	20	50	100	200
KP0, $k = 2$	376.04	364.51	131.61	64.32	20.87	10.43	28.64
KP2, $k = 3$	376.04	359.10	173.54	82.37	24.93	17.42	28.99

In the last example in this domain decomposition setting, we compare the classical interface probing approach based on three vectors with the explicit MSPAI probing to the same subspace KP0, $k = 3$. Both methods are applied to the domain decomposition matrix with $m = 8$ domains. It turns out that the new MSPAI probing gives much better condition numbers than the old approach (see Table 5.11). We would also like to emphasize that classical probing corresponds to the choice of three probing vectors ($k = 3$), whereas our new probing method yields much better results with only one probing vector ($k = 1$). For small m (number of subdomains), both preconditioners lead to similar results. However, for large m, MSPAI probing is significantly superior.

Table 5.11: Condition numbers for domain decomposition Laplace with preconditioned Schur complement S of size 609. The case $m = 8$, $k = 1$, $\kappa(S) = 206.89$ compared with classic tridiagonal KP0-style probing M_p (3 probing vectors) with $\kappa(M_p^{-1}S) = 151.26$.

$kp \mid \rho$	0	5	10	20	50	100
0	44.82	40.45	32.47	21.15	14.56	23.54
1	44.82	39.79	33.40	28.01	36.04	40.48
2	44.82	38.95	33.21	27.88	28.68	22.98

5.3.2 MSPAI Probing for Stokes Problems

We consider the four Stokes examples from IFISS [65] by Elman, Silvester, and Wathen [40]. The stabilized matrices are of the form

$$\begin{pmatrix} A_{st} & B_{st} \\ B_{st}^T & -\beta C \end{pmatrix}$$

with Schur complement $S_s := -\beta C - B_{st} \cdot A_{st}^{-1} B_{st}^T$; we also consider the unstabilized problems

$$\begin{pmatrix} A & B \\ B^T & 0 \end{pmatrix}$$

with Schur complement $S_u = -B \cdot A^{-1} B^T$. In a first step, we again approximate A^{-1} by a

Table 5.12: Iteration count for unstabilized Stokes examples, unfactorized (with KP0, $k = 1$, $\rho = 5$) and factorized (with KP1, $k = 2$, $\rho = 3$). L_D denotes the preconditioner L_α, diagonally transformed as described in (3.40).

	unfactorized			factorized			
Example problem	1	2	3	1	2	3	symmetrization
unprec	39	65	78	39	65	78	
explicit $\rho = 0$	20	34	26	20	34	26	
explicit $\rho = 5$	12	17	19				$\rho = 3$
				18	28	26	$\tilde{L}, \tilde{L}^T$
				18	17	26	$L_\alpha(L, \tilde{L}^T)$
				16	24	15	L_D
inverse $\rho = 0$	25	43	33	28	39	36	
inverse $\rho = 5$	15	24	25				$\rho = 3$
				20	25	31	$\tilde{L}, \tilde{L}^T$
				17	28	17	$L_\alpha(L, \tilde{L}^T)$
				16	27	16	L_D

sparse matrix with the same pattern as A. Then by MSPAI probing with this pattern we get the preconditioner for the Schur complement. In the unfactorized case, we consider the trivial symmetrization via $\bar{M}$. For factorized preconditioners we also apply the symmetrization $L_{\alpha,4}$, possibly with an additional diagonal scaling D to obtain diagonal elements equal to 1 in the preconditioned system (3.40). We apply this procedure to the following example settings:

- **Example 1**: Channel domain with natural boundary on a 16×16 grid, Q1-P0, stabilization parameter $\beta = 1/4$, and uniform pattern. The condition number of S_s is 12.67.

Table 5.13: Iteration count for stabilized Stokes examples, unfactorized (with KP1, $k = 3$, $\rho = 3$) and factorized (with KP1, $k = 3$, $\rho = 3$). Problem 4 with $\rho = 1.5$ in the unfactorized case.

	unfactorized				factorized				
Example problem	1	2	3	4	1	2	3	4	symmetrization
unprec	24	33	74	23	24	33	74	23	
explicit $\rho = 0$	23	34	20	19	23	34	20	19	
explicit ρ	16	22	16	17					
					18	24	19	18	$\tilde{L}, \tilde{L}^T$
					18	24	19	18	$L_\alpha(L, \tilde{L}^T)$
					16	22	19	18	L_D
inverse $\rho = 0$	18	24	17	16	20	27	23	19	
inverse ρ	15	21	21	15					
					18	23	18	15	$\tilde{L}, \tilde{L}^T$
					16	22	19	16	$L_\alpha(L, \tilde{L}^T)$
					16	22	18	16	L_D

- **Example 2**: Flow over backward facing step with natural outflow boundary, Q1-P0, $\beta = 1/4$, $\kappa(S_s) = 78.13$.
- **Example 3**: Lid driven cavity, regularized on stretched 16×16 grid, Q1-P0, $\beta = 1/4$, exponential streamlines with singular S, but with essential condition number $\lambda_{\max}(S_s)/\lambda_2(S_s) = 129.6$.
- **Example 4**: Colliding flow on uniform 16×16 grid, Q1-P0, $\beta = 1/4$, uniform streamlines, with singular S, but essential condition number $\lambda_{\max}(S_s)/\lambda_2(S_s) = 6.92$.

For the stabilized and unstabilized case we display the iteration count for pcg with factorized and unfactorized preconditioner and simple symmetrizations in Tables 5.12 and 5.13. Obviously, the MSPAI probing with different probing vectors and appropriate ρ always gives better results. Following [40], for the above examples one can choose a diagonal preconditioner Q that is given by the underlying Finite Element method. In nearly all examples presented here this preconditioner is only a multiple of the identity matrix and therefore does not lead to a comparable improvement of the iteration count.

5.3.3 MSPAI Probing for Dense Matrices

In many applications we have to deal with dense or nearly dense matrices A. To apply a preconditioner it is helpful to derive a sparse approximation $\tilde{A}$ of A. In a first step we can sparsify the dense matrix by deleting all entries outside an allowed pattern or with small entries. This gives a first approximation $\tilde{A}$. Here, we consider two examples. In a first setting we choose a discretization of the 2D Laplacian with a quite thick bandwidth that should be approximated by a sparser matrix A_p such that $A_p^{-1}A$ has bounded condition. In Figure 5.3 we display the given thick pattern and the sparse pattern that should be used for the approximation in $\tilde{A}$ and A_p. Again, we compute A_p by explicit probing on $\tilde{A}$ and A. In Table 5.14, the resulting condition numbers are displayed. For large ρ, we see that the approximation based on probing is much better than the matrix $\tilde{A}$ that is obtained by reducing A to the allowed pattern by simply deleting nonzero entries.

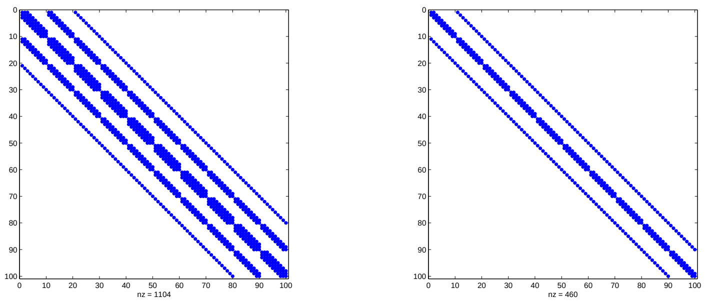


Figure 5.3: Sparsity pattern of original matrix A (left) and sparse approximation A_p (right).

Table 5.14: Condition numbers for sparse 2D Laplacian approximations AA_p^{-1} for different ρ, KP0, $k = 1$.

n	A	$\tilde{A}$, $\rho = 0$	10	50	100	500	1000
100	127.95	799.85	3.15	2.86	2.88	2.90	2.90
225	286.45	372.68	68.40	4.25	3.59	3.50	3.50
400	506.06	1204.35	596.67	41.06	4.72	4.17	4.16
625	786.57	6962.76	20016.42	53.35	9.06	4.84	4.81
900	1127.94	3405.30	4251.85	60.03	14.66	5.54	5.47

In a second example we consider a symmetric dense Toeplitz matrix with first row given by

$$a(1,1) = \frac{\pi}{2}, \quad a(1,2j) = \frac{2}{\pi(2j-1)^2}, \quad j = 1, 2, \ldots \quad .$$

This matrix is related to the generating function (see [80])

$$f(x) = |x - \pi|, \qquad x \in [0, 2\pi].$$

First we replace A by a tridiagonal matrix $\tilde{A}$ by deleting all entries outside the bandwidth. In a second step, we compute the improved tridiagonal approximation A_p by explicit probing. For $n = 1000$ the condition number of A is $1.36{\cdot}10^3$, and the preconditioned system $\tilde{A}^{-1}A$ has condition 150.9. As probing vector we additionally allow the vector $e_\pm := (1, -1, 1, -1, \ldots)$. Table 5.15 shows that probing with $e_\pm$ and KP2 leads to a significantly improved condition number. The choices KP1 and KP0, $k = 1$ fail, while KP0, $k = 2$ again gives a better preconditioner. This is caused by the fact that $e_\pm$ is contained in the probing subspace related to KP0, $k = 2$. Note that this example is dense, not related to a PDE problem, and allows a new probing vector.

Table 5.15: Condition numbers for tridiagonal approximations of the dense Toeplitz matrix to generating function $f(x) = |x - \pi|$ for $\rho = 1000$ and different probing vectors.

$e_\pm$	KP2: $k = 2$	$k = 1$	KP0: $k = 2$	$k = 1$	KP1: $k = 2$	$k = 1$
22.9	18.2	12.7	22.0	113.8	119.2	114.6

5.3.4 Comparison ILU and MILU with MSPAI Probing

In a first example, we compare the direct factorized probing approach with the well-known modified ILU (MILU) and modified incomplete Cholesky (MIC) preconditioners. We will

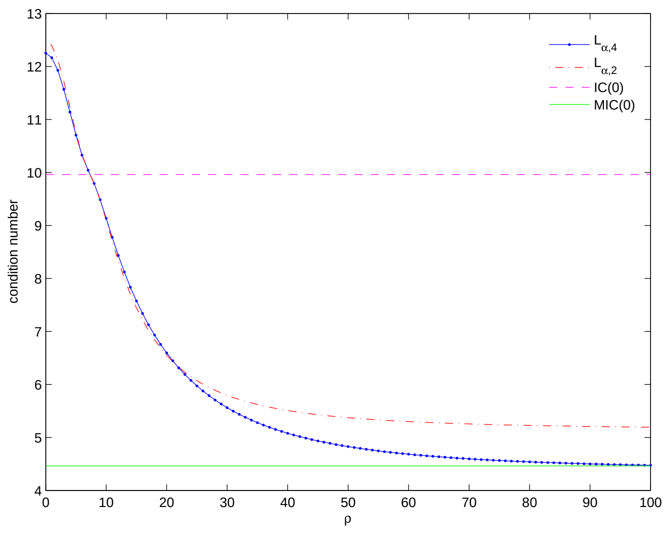


Figure 5.4: Resulting condition numbers for $\rho \in [0, 100]$ with symmetrizations $L_{\alpha,4}$ (solid dotted) and $L_{\alpha,2}$ (dash-dot) compared to modified incomplete Cholesky (solid) and incomplete Cholesky (dashed) preconditioner.

show that in cases where MILU is well behaved the more general MSPAI approach leads to a similar improvement of the condition number. In a second part, we compare the classical probing with MSPAI probing for the domain decomposition example from Section 5.3.1. Starting point of the first example is a 2D Laplacian A with $n = 225$ and a random right-hand side b. The condition number of A is 103.1, and for the solution of the corresponding system, we use a preconditioned conjugate gradient method (pcg). At first, we compute an incomplete Cholesky (IC(0)) preconditioner L with the lower triangular sparsity pattern of A. This reduces the condition number to $\kappa(L^{-1}AL^{-T}) = 9.961$. The solution to a relative residual of TOL$= 10^{-6}$ takes 16 iterations (started with a initial vector of all zeros).

Table 5.16: Iteration count for 2D Laplacian with explicit factorized and symmetrized probing with KP0, $k = 1$, and different ρ.

ρ	0	20	40	60	80	100
$L_{\alpha,4}$	17	13	12	12	12	12
$L_{\alpha,2}$	17	13	13	12	12	12

Table 5.17: Iteration count for 2D Laplacian with explicit factorized and symmetrized probing with KP0, $k = 2$, and different ρ.

ρ	0	20	40	60	80	100
$L_{\alpha,4}$	17	14	13	13	12	12
$L_{\alpha,2}$	17	14	13	13	13	13

The modified incomplete Cholesky preconditioner L_M (MIC(0)), which preserves the row sums of A, leads to a condition number $\kappa(L_M^{-1}AL_M^{-T}) = 4.463$ and 11 iterations until convergence. In order to demonstrate the effect of our probing approach, we apply the direct factorized probing ansatz (3.9) to the incomplete Cholesky factor L for the normalized

Table 5.18: Iteration count for Laplace with explicit factorized and symmetrized probing using one sine basis vector from (3.15) (KP1, $k = 1$), and different ρ.

ρ	0	20	40	60	80	100
$L_{\alpha,4}$	17	14	13	13	14	15
$L_{\alpha,2}$	17	14	13	13	13	13

Table 5.19: Iteration count for Laplace with explicit factorized and symmetrized probing with KP2, $k = 1$, and different ρ.

ρ	0	20	40	60	80	100
$L_{\alpha,4}$	17	14	13	14	17	18
$L_{\alpha,2}$	17	14	13	13	14	14

probing vector $e = \frac{1}{\sqrt{n}}(1, \dots, 1)^T$ and $\rho \in [0, 100]$. The result $\tilde{U}$ is then symmetrized to L_α, where both (3.36) and (3.39) are used. The former solves the resulting polynomial exactly, the latter yields an approximation to the optimal symmetrization parameter α. Figure 5.4 shows the resulting condition numbers $\kappa(L_\alpha^{-1} A L_\alpha^{-T})$. For increasing ρ the condition number tends to the condition number of the MIC, which meets our expectations, since the error of the probing condition $e^T(L_\alpha L_\alpha^T - A)$ decreases, and the MIC satisfies this property exactly. As a consequence, the iteration numbers also decrease as shown in Tables 5.16–5.19. These examples also display that the approximative symmetrization approach leads to reasonable values of α.

Albeit the iteration number is improved only slightly, one advantage over MIC is the inherent parallelism of the underlying Frobenius norm minimization, which is inherited from SPAI. Furthermore, MIC is restricted to a single vector $(1, \dots, 1)^T$, whereas we are able to probe with an arbitrary number of arbitrary probing constraints, see Section 3.4.1.

Note that for the Laplacian the probing approach for deriving an approximate inverse preconditioner cannot be very successful. This is caused by the fact that probing vectors like $e = (1, ..., 1)^T$ will lead to a sparse vector $e^T A$. Hence, we try to approximate a dense vector e^T by a necessarily sparse vector $e^T AM$, which is impossible. Nevertheless, for small ρ a special choice of probing vectors can improve the condition number of the preconditioned system in comparison with standard Frobenius norm approximation with $\rho = 0$. We set $k = 1$ and use a rough approximation to the eigenvector to the smallest eigenvalue as probing vector which can be computed quite fast. Furthermore, we use the symmetrization $\hat{M}_\alpha$. The resulting condition numbers are shown in Table 5.20. For very carefully chosen ρ, we can achieve an improved preconditioner also in this case. Here, Theorem 3.4 may give an explanation for the improved behavior of this symmetrization.

Table 5.20: Iteration count for 2D Laplacian with approximate inverse probing with $kp = 2$, $k = 1$, symmetrization $\hat{M}_\alpha$, and different ρ.

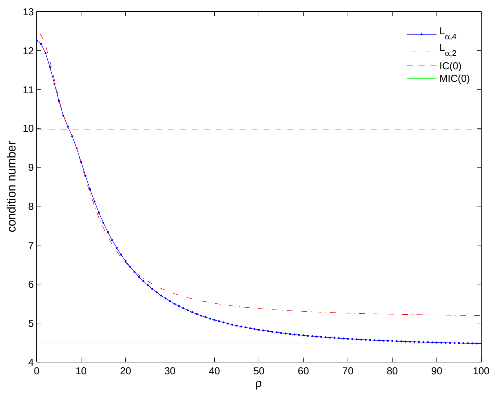
n	100			225			400			625			900		
ρ	0	9	12	0	18	22	0	32	36	0	48	52	0	52	56
#it	14	10	23	20	13	22	26	16	36	32	19	42	37	22	23

5.3.5 Consecutive Probing Steps

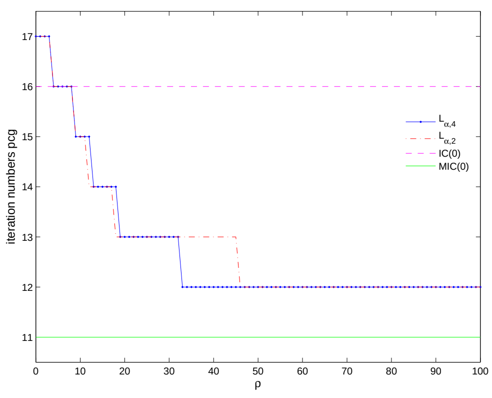
In Section 3.3.3, we already mentioned that in the case of factorized probing, we first improve one of the two given factors L, U by MSPAI probing and then obtain a second probed factor by the transposed formulation (3.10). Furthermore, factorized symmetrization gives us the possibility to combine the two factors to one. Here, we want to investigate whether it makes sense to combine more than two consecutive probing steps. For this example, we consider a standard five-point stencil discretization of the 2D Laplacian of dimension $n = 225$. We compute an ILU(0) factor L and use it as a starting point for probing. After the first application of MSPAI probing with probing vector $e = 1/\sqrt{n}(1, \ldots, 1)^T$, we obtain a matrix $\tilde{U}$. The symmetrized preconditioner $L_\alpha(L, \tilde{U})$ is then employed in pcg to solve the corresponding system. With $\tilde{U}$ as input, we further apply MSPAI probing to obtain a factor $\tilde{L}$ and symmetrize it together with $\tilde{U}$. Finally, a third probing step using $\tilde{L}$ yields $\tilde{\tilde{U}}$.

In Figure 5.5, both the condition numbers and iteration counts for these three consecutive MSPAI probing steps for the weight $\rho \in [0, 100]$ are depicted. Preconditioning with $L_\alpha(L, \tilde{U})$ leads to the same picture as it was observed in Section 5.3.4. After the second probing step, we see that preconditioning with $L_\alpha(\tilde{L}, \tilde{U})$ causes a better condition number for the exact symmetrization, but this does not lead to a lower iteration count. Moreover, the interval in which the weight ρ leads to the lowest iteration count becomes smaller. ρ has to be chosen even more carefully after the third probing step. Inexact symmetrization with minimizing a polynomial of order 2 does not lead to satisfying results anymore. Also with exact symmetrization, the condition number for the preconditioner $L_\alpha(\tilde{L}, \tilde{\tilde{U}})$ deteriorates if ρ is chosen to large. Furthermore, in terms of iteration counts, the interval of optimal ρ has again become smaller.

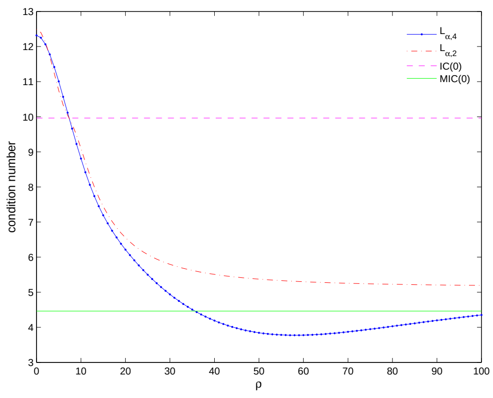
Therefore, for practical applications, we recommend only one or two consecutive probing steps, which can lead to improved convergence behavior, but also forces the weight ρ to be chosen much more carefully.



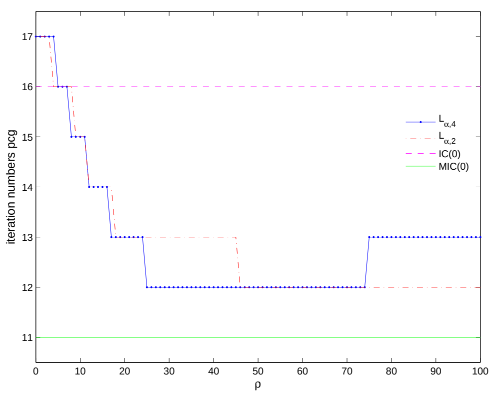
(a) Condition numbers for $L_\alpha(L, \tilde{U})$.



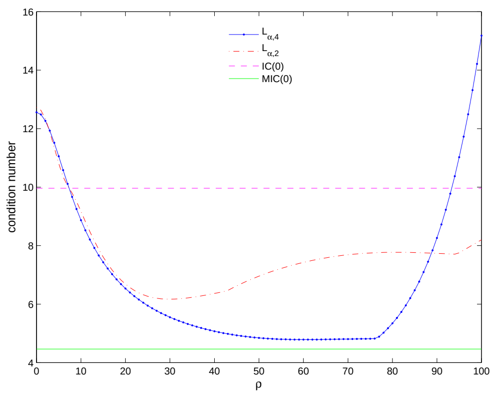
(b) Iteration counts for $L_\alpha(L, \tilde{U})$.



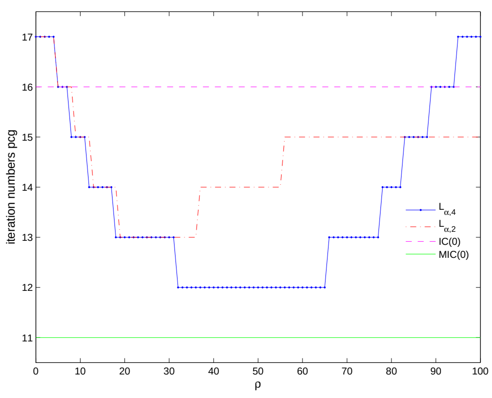
(c) Condition numbers for $L_\alpha(\tilde{U}, \tilde{L})$.



(d) Iteration counts for $L_\alpha(\tilde{U}, \tilde{L})$.



(e) Condition numbers for $L_\alpha(\tilde{L}, \tilde{\tilde{U}})$.



(f) Iteration counts for $L_\alpha(\tilde{L}, \tilde{\tilde{U}})$.

Figure 5.5: Condition numbers and iteration counts for consecutive probing steps.

Conclusions and Future Work

To round up this thesis, we would like to give a summary of the results and point out a few ideas for future work. The fundamentals for the whole topic are given in Chapter 1. In Chapter 2, we elaborated some minor pieces of original research such as SPAI pattern updates for complex systems of linear equations, and a reformulation and extension of a nonsingularity result for SPAI. An investigation on SPAI as a regularization method for image restoration problems showed that SPAI is adequate for this task as well. Furthermore, we presented the effect of a fill-in-reducing graph ordering algorithm on FSPAI pattern updates. The resulting FSPAIs are denser and therefore cheaper to compute and apply. A theoretical result on SPAI and FSPAI for M-matrices proves that the FSPAI of an M-matrix has exclusively non-negative entries. SPAI does not have this property.

The main contribution to ongoing research is the development of the MSPAI probing approach in Chapter 3, which combines the advantages of classical probing, application of modified preconditioners, and Frobenius norm minimizations:

- MSPAI allows us to compute both explicit and inverse approximations in either factorized or unfactorized form.
- Arbitrary factorized preconditioners given such as ILU or AINV, can be improved via MSPAI, since it adds probing information.
- We can improve approximations to Schur complements by adding probing information.
- It is possible to compute sparse spectrally equivalent approximations to dense or almost dense matrices.
- Even matrices that are only known implicitly can be used in MSPAI.
- MSPAI shows great versatility in the choice of the probing subspaces. Whereas classical probing and MILU or MIC are restricted to rather simple probing vectors, we can choose whatever seems to be accurate for the actual problem as probing vectors.
- MSPAI computations are easier to parallelize than for instance the MILU preconditioners. Due to the underlying Frobenius norm minimization, MSPAI is inherently parallel.
- Through various symmetrization techniques, we can regain symmetry and sometimes even positive definiteness in the preconditioner.

Our numerical experiments (Chapter 5) reveal the effectiveness of our method in terms of lower condition numbers, lower iteration counts, and thus faster convergence compared to classical probing, classical SPAI, or the modified preconditioners. However, in the inverse probing case, the weighting factor ρ and the probing vectors are to be chosen carefully such that the preconditioner does not deteriorate. As a first step for understanding this phenomenon, we also presented a purely structural condition for the sparsity patterns of the probing vectors and the prescribed sparsity pattern for the preconditioner. This condition helps to ensure the error reduction in the probing constraints. If we adapt the sparsity patterns such that this structural requirement is fulfilled, the error in the "main" approximation may grow disproportionately and strongly worsen the condition number. The explicit probing variants turned out to be the most stable types of MSPAI.

In Chapter 4, we described the design of an MSPAI implementation which clearly outperforms the current standard SPAI implementation both in serial applications and parallel environments. Furthermore, it has a much larger functionality than SPAI 3.2 such as support for complex coefficient matrices, arbitrarily prescribed start patterns, and maximum sparsity patterns. Moreover, the residual matrix $R = I - AM$ is returned. Runtime comparisons proved the clear raise in efficiency achieved through QR updates, sparse least squares methods, and the employment of maximum sparsity patterns.

Finally, we suggest a few further promising fields of application for MSPAI probing:

- **Image Restoration.** By MSPAI probing, we could construct preconditioners which act stronger on large eigenvalues than for instance SPAI does. Additionally, we can force the preconditioner to neglect the small and noise-amplifying eigenvalues by probing with adequate probing vectors. This would extend our tests of SPAI in image restoration, see Section 2.1.5.
- **Smoothers in Multigrid.** For multigrid [98] methods, it should be possible to compute smoothers with improved smoothing properties, again, by optimizing them on certain parts of the spectrum. SPAI's smoothing properties were already investigated in [22].
- **Preconditioner Updates.** Systems of nonlinear equations solved with Newton- or Broyden-type methods, often lead to a sequence of linear systems of equations, which have to be solved. From step to step, these systems do not differ much. Thus, respective preconditioners are rather similar. For such cases, Tůma [38] proposed not to recompute the preconditioner in every step, but to update the current one. MSPAI probing could help here to determine updates of improved quality. Possibly, fewer preconditioner updates would then be necessary.
- **Interior Point Methods in Optimization.** Interior point methods for linear, nonlinear, and quadratic programming lead to saddle point problems with very ill-conditioned submatrices having special structures. Gondzio et al. [16] therefore investigated the iterative solution of these systems using preconditioning techniques. The special structure of the submatrices favors MSPAI-like approaches in order to get rid of the ill-conditioning.

Bibliography

[1] ACML – AMD Core Math Library. `http://developer.amd.com/acml.jsp`.

[2] P. R. Amestoy, T. A. Davis, and I. S. Duff, *An approximate minimum degree ordering algorithm*, SIAM J. Matrix Anal. Appl., 17 (1996), pp. 886–905.

[3] ATLAS – Automatically Tuned Linear Algebra Software. `http://math-atlas.sourceforge.net/`.

[4] O. Axelsson, *A generalized SSOR method*, BIT, 13 (1978), pp. 443–467.

[5] O. Axelsson, *Iterative Solution Methods*, Cambridge University Press, first ed., 1994.

[6] O. Axelsson and V. Barker, *Finite Element Solution of Boundary Value Problems, Theory and Computation*, Academic Press, Orlando, Florida, 1984.

[7] O. Axelsson and B. Polman, *Block preconditioning and domain decomposition methods*, J. Comp. Appl. Math., 24 (1988), pp. 55–72.

[8] Z. Z. Bai, I. S. Duff, and A. J. Wathen, *A class of incomplete orthogonal factorization methods I*, BIT, 41 (2001), pp. 53–70.

[9] S. Barnard and M. Grote, *A block version of the SPAI preconditioner*, Proc. of the 9th SIAM conference on Parallel Processing for Scientific Computing, held in San Antonio, TX, March 1999., (1999).

[10] S. T. Barnard, L. M. Bernardo, and H. D. Simon, *An MPI implementation of the SPAI preconditioner on the T3E*, International Journal of High Performance Computing Applications, 13 (1999), pp. 107–123.

[11] 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.

[12] 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.

[13] M. Benzi and M. Tůma, *A sparse approximate inverse preconditioner for nonsymmetric linear systems*, SIAM J. Sci. Comput., 19 (1998), pp. 968–994.

[14] ——, *A comparative study of sparse approximate inverse preconditioners*, Appl. Numer. Math., 30 (1999), pp. 305–340.

[15] ——, *Orderings for factorized sparse approximate inverse preconditioners*, SIAM J. Sci. Comput., 21 (2000), pp. 1851–1868.

[16] L. Bergamaschi, J. Gondzio, and G. Zilli, *Preconditioning indefinite systems in interior point methods for optimization*, Computational Optimization and Applications, 28 (2004), pp. 149–171.

[17] Å. Björck, *Numerical Methods for Least Squares Problems*, SIAM, 1996.

[18] Å. Björck and G. Dahlquist, *Numerische Methoden*, Oldenbourg, second ed., 1979.

[19] Å. Björck and G. H. Golub, *Iterative refinement of linear least squares solutions by Householder transformations*, BIT, 7 (1967), pp. 322–337.

[20] M. Bollhöfer, *A robust and efficient ILU that incorporates the growth of the inverse triangular factors*, SIAM J. Sci. Comput., 25 (2003), pp. 86–103.

[21] M. Brenk, H.-J. Bungartz, I. L. Muntean, and T. Neckel, *Simulating large particle movements in drift ratchets using cartesian grids*, in Int. Conf. on Computational Methods for Coupled Problems in Science and Engineering (COUPLED PROBLEMS 2007), M. Papadrakakis, E. Onate, and B. Schrefler, eds., Barcelona, May 2007, International Center for Numerical Methods in Engineering (CIMNE).

[22] O. Bröker, 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.

[23] L. Cesari, *Sulla resoluzioni dei sistemi di equazioni lineari per approssimazioni successivie*, Atti Accad. Naz. Lincei, Rend. Cl. Sci. Fiss. Mat. Nat., 25 (1937), pp. 422–428.

[24] T. F. Chan and T. P. Mathew, *The interface probing technique in domain decomposition*, SIAM J. Mat. Anal. Appl., 13 (1992), pp. 212–238.

[25] K. Chen, *Matrix Preconditioning Techniques and Applications*, Cambrigde Monographs on Applied and Computational Mathematics, first ed., 2005.

[26] A. J. Chorin, *Numerical solution of the Navier-Stokes equations*, Math. Comp., 22 (1968), pp. 745–762.

[27] E. Chow, *A priori sparsity patterns for parallel sparse inverse preconditioners*, SIAM J. Sci. Comput., 21 (2000), pp. 1804–1822.

[28] E. Chow and Y. Saad, *Experimental study of ILU preconditioners for indefinite matrices*, J. Comput. Appl. Math., 86 (1997), pp. 387–414.

[29] CSparse – Concise Sparse Package. `http://www.cise.ufl.edu/research/sparse/CSparse/`.

[30] E. Cuthill and J. McKee, *Reducing the bandwidth of sparse symmetric matrices*, Proc. ACM Nat. Conf., (1969), pp. 157–172.

[31] CXSparse – Extended Concise Sparse Package. `http://www.cise.ufl.edu/research/sparse/CXSparse/`.

[32] T. Davis, *Direct Methods for Sparse Linear Systems*, SIAM, 2006.

[33] J. Demmel, Y. Hida, W. Kahan, X. S. Li, S. Mukherjee, and E. J. Riedy, *Error bounds from extra precise iterative refinement*, ACM Transactions on Mathematical Software (TOMS), 32 (2006), pp. 325–351.

[34] J. Demmel, Y. Hida, X. S. Li, and E. J. Riedy, *Extra-precise iterative refinement for overdetermined least squares problems*, LAPACK Working Notes `http://www.netlib.org/lapack/lawns/downloads/lawn188.pdf`, (2007).

[35] P. Deuflhard and A. Hohmann, *Numerical Analysis in Modern Scientific Computing. An Introduction*, Springer, second ed., 2003.

[36] F. Di Benedetto, C. Estatico, and S. Serra-Capizzano, *Superoptimal preconditioned conjugate gradient iteration for image deblurring*, SIAM J. Sci. Comp., 26 (2005), pp. 1012–1035.

[37] P. F. Dubois, A. Greenbaum, and G. H. Rodrigue, *Approximating the inverse of a matrix for use in iterative algorithms on vector processors*, Computing, 22 (1979), pp. 257–268.

[38] J. Duintjer Tebbens and M. Tůma, *Preconditioner updates for solving sequences of large and sparse nonsymmetric linear systems*, SIAM J. Sci. Comp., 29 (2007), pp. 1918–1941.

[39] S. C. Eisenstat, M. H. Schultz, and A. H. Sherman, *Algorithms and data structures for sparse symmetric Gaussian elimination*, J. Sci. Stat. Comp., 2 (1981), pp. 225–237.

[40] H. C. Elman, D. J. Silvester, and A. J. Wathen, *Finite Elements and Fast Iterative Solvers: with applications in incompressible fluid dynamics*, Oxford University Press, 2005.

[41] G. Fischer, *Lineare Algebra*, Vieweg, twelfth ed., 2000.

[42] D. Flade, *Numerische Methoden der Bildrekonstruktion mit verschiedenen Regularisierungspräkonditionierern*, Diploma thesis, Fakultät für Informatik, Technische Universität München, Dec. 2006.

[43] A. George, *Graph Theory and Sparse Matrix Computation*, Springer, 1993.

[44] J. A. George, *Nested dissection of a regular finite element mesh*, SIAM J. Num. Anal., 10 (1973), pp. 345–363.

[45] J. R. Gilbert, *Predicting structure in sparse matrix computations.*, SIAM J. Matrix Anal. Appl., 15 (1994), pp. 62–79.

[46] G. H. Golub and C. Van Loan, *Matrix Computations*, The John Hopkins University Press, third ed., 1996.

[47] 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.

[48] M. Grote and T. Huckle, *Parallel preconditioning with sparse approximate inverses*, SIAM J. Sci. Comput., 18 (1997), pp. 838–853.

[49] I. Gustafsson, *A class of first order factorization methods*, BIT, 18 (1978), pp. 142–156.

[50] M. Hagemann and O. Schenk, *Weighted matchings for preconditioning symmetric indefinite linear systems*, SIAM J. Sci. Comput., 28 (2006), pp. 403–420.

[51] G. Hämmerlin and K.-H. Hoffmann, *Numerische Mathematik*, Springer, third ed., 1992.

[52] M. Hanke, *Conjugate Gradient Type Methods for Ill-posed Problems*, Longman Scientific & Technical, 1995.

[53] M. Hanke and J. G. Nagy, *Restoration of atmospherically blurred images by symmetric indefinite conjugate gradient techniques*, Inverse Problems, 12 (1996), pp. 157–173.

[54] M. Hanke, J. G. Nagy, and R. Plemmons, *Preconditioned iterative regularization for ill-posed problems*, Numerical Linear Algebra, (1996), pp. 141–163.

[55] M. R. Hestenes and E. Stiefel, *Methods of conjugate gradients for solving linear systems*, J. Res. Nat. Bur. Stand., 49 (1952), pp. 409–436.

[56] N. Higham, *Turing, Wilkinson, and Gaussian elimination.* `http://www.maths.manchester.ac.uk/~higham/talks/twge96.ps.gz`, 1996.

[57] R. M. Holland, G. J. Shaw, and A. J. Wathen, *Sparse approximate inverses and target matrices*, SIAM J. Sci. Comp., 26 (1992), pp. 1000–1011.

[58] S. Hörmann, *Entwicklung und Analyse modifizierter unvollständiger LU-Zerlegungen mittels Probing*, Diploma thesis, Fakultät für Mathematik, Technische Universität München, June 2007.

[59] T. Huckle, *Efficient computation of sparse approximate inverses*, Numer. Linear Algebra, 5 (1998), pp. 57–71.

[60] ——, *Factorized sparse approximate inverses for preconditioning and smoothing*, Selçuk J. Appl. Math., 1 (2000), pp. 63–70.

[61] ——, *Approximate sparsity patterns for the inverse of a matrix and preconditioning*, Appl. Numer. Math., 30 (2003), pp. 291–303.

[62] ——, *Factorized sparse approximate inverses for preconditioning*, The Journal of Supercomputing, 25 (2003), pp. 109–117.

[63] T. Huckle and A. Kallischko, *Frobenius norm minimization and probing for preconditioning*, International Journal of Computer Mathematics, 84 (2007), pp. 1225–1248.

[64] IBM Research Project Cell Processor. `http://www.research.ibm.com/cell/`.

[65] IFISS, *Software package, incompressible flow & iterative solver software version 2.2.* `http://www.maths.manchester.ac.uk/~djs/ifiss/`.

[66] InfiniCluster. `http://www.lrr.in.tum.de/Par/arch/infiniband/ClusterHW/cluster.html`.

[67] Intel MKL – Intel Math Kernel Library. `http://www.intel.com/cd/software/products/asmo-na/eng/307757.htm`.

[68] T. K. Jensen, *Stabilization algorithms for large scale problems*, PhD thesis, 2006.

[69] O. G. Johnson, C. A. Micchelli, and G. Paul, *Polynomial preconditioning for conjugate gradient calculations*, SIAM J. Numer. Anal., 20 (1983), pp. 362–376.

[70] S. K'Ang-Shen, S. Kangshen, and K. Shen, *The Nine Chapters on the Mathematical Art: Companion and Commentary*, Oxford University Press, 1999.

[71] I. Kaporin, *New convergence results and preconditioning strategies for the conjugate gradient method*, Numer. Linear Algebra Appl., 1 (1994), pp. 179–210.

[72] G. Karypis and V. Kumar, *Multilevel k-way hypergraph partitioning*, VLSI Design, 11 (2000), pp. 285–300.

[73] D. Kay, D. Loghin, and A. J. Wathen, *A preconditioner for the steady-state Navier-Stokes equation*, SIAM J. Sci. Comput., 24 (2002), pp. 237–256.

[74] L. Y. Kolotilina, A. A. Nikishin, and A. Y. Yeremin, *Factorized sparse approximate inverse preconditionings IV: Simple approaches to rising efficiency*, Numer. Linear Algebra Appl., 6 (1999), pp. 515–531.

[75] L. Y. Kolotilina and A. Y. Yeremin, *Factorized sparse approximate inverse preconditionings I: Theory*, SIAM J. Mat. Anal. Appl., 14 (1993), pp. 45–58.

[76] LAPACK – Linear Algebra Package. `http://www.netlib.org/lapack/`.

[77] Matrix Market. `http://math.nist.gov/MatrixMarket/`.

[78] MPI – Message Passing Interface. `http://www-unix.mcs.anl.gov/mpi/`.

[79] R.-P. Mundani, *Hierarchische Geometriemodelle zur Einbettung verteilter Simulationsaufgaben*, PhD thesis, Fakultät Informatik, Elektrotechnik und Informationstechnik, Universität Stuttgart, 2005.

[80] M. K. Ng, *Iterative Methods for Toeplitz Systems*, Oxford University Press, 2004.

[81] PETSc – Portable, Extensible Toolkit for Scientific Computation. `http://www-unix.mcs.anl.gov/petsc/petsc-as/`.

[82] A. Roy, *Untersuchung dünnbesetzter QR-Verfahren bei der Berechnung dünnbesetzter approximativer Inverser*, Diploma thesis, Fakultät für Informatik, Technische Universität München, June 2007.

[83] Y. Saad, *Iterative Methods for Sparse Linear Systems*, SIAM, second ed., 2000.

[84] Y. Saad and M. H. Schultz, *GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems*, SIAM J. Sci. Stat. Comput., 7 (1986), pp. 856–869.

[85] H. Samet, *The quadtree and related hierarchical data structures*, ACM Computing Surveys, 16 (1984), pp. 187–260.

[86] M. Sedlacek, *Implementierung und Tests von Hashing- und Caching-Strategien bei Berechnung dünnbesetzter approximativer Inverser (SPAI), sowie einer graphischen Oberfläche für Anwendung von Bildoperatoren*, Systementwicklungsprojekt, Fakultät für Informatik, Technische Universität München, Dec. 2006.

[87] ——, *Effiziente parallele Implementierung des MSPAI Präkonditionierers unter Verwendung von Caching-Strategien und QR-Updates*, Diploma thesis, Fakultät für Informatik, Technische Universität München, Jan. 2008.

[88] C. Siefert and E. Sturler, *Probing methods for saddle-point problems*, Electronic Transactions on Numerical Analysis, 22 (2006), pp. 163–183.

[89] R. D. Skeel, *Scaling for numerical stability in Gaussian elimination*, J. ACM 26, 3 (1979), pp. 494–526.

[90] ——, *Iterative refinement implies numerical stability for Gaussian elimination*, Math. Comp. 35, 151 (1980), pp. 817–832.

[91] SPAI 3.2 – Sparse Approximate Inverses. http://www.computational.unibas.ch/software/spai/.

[92] A. Steger, *Diskrete Strukturen, Band 1: Kombinatorik – Graphentheorie – Algebra*, Springer, first ed., 2001.

[93] J. Stoer, *Numerische Mathematik 1*, Springer, fourth ed., 1983.

[94] J. Stoer and R. Bulirsch, *Einführung in die Numerische Mathematik II*, Springer, second ed., 1978.

[95] B. Stroustrup, *Die C++-Programmiersprache*, Addison-Wesley, fourth ed., 2000.

[96] A. S. Tanenbaum, *Moderne Betriebssysteme*, Pearson Studium, second ed., 2002.

[97] A. Tikhonov, *Solution of incorrectly formulated problems and the regularization method*, Soviet Math. Dokl., 4 (1963), pp. 1035–1038.

[98] U. Trottenberg, C. Oosterlee, and A. Schüller, *Multigrid*, Academic Press, first ed., 2001.

[99] A. Turing, *Rounding-off errors in matrix processes*, Quart. J. Mech. and Applied Math., 1 (1948), pp. 287–308.

[100] UNIQA Tower. http://tower.uniqa.at/.

[101] University of Florida Matrix Collection. http://www.cise.ufl.edu/research/sparse/matrices/.

[102] H. A. van der Vorst, *Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems*, SIAM J. Sci. Comput., 13 (1992), pp. 631–644.

[103] K. Wang, O. Lawlor, and L. V. Kale, *The nonsingularity of sparse approximate inverse preconditioning and its performance based on processor virtualization*, Technical Report, Department of Computer Science, University of Illinois at Urbana-Champaign, (2005).

[104] M. Yannakakis, *Computing the minimum fill-in is NP-complete*, SIAM J. Alg. Disc. Meth., 2 (1981), pp. 77–79.

[105] O. C. Zienkiewicz, *Methode der finiten Elemente*, Carl Hanser Verlag, second ed., 1983.