A Block Version of the Factorized Sparse Approximate Inverse Preconditioner (FSPAI) - Theory and Implementation
Diplomarbeit in Informatik

A Block Version of the Factorized Sparse Approximate Inverse Preconditioner (FSPAI) - Theory and Implementation

Eine Block-Variante des faktorisierten dünnbesetzten approximierenden inversen Präkonditionierers (FSPAI) - Theorie und Implementierung

Supervisor: Univ.-Prof. Dr. Thomas Huckle
Advisor: Dipl.-Inf. Matous Sedlacek
Student: Jürgen Bräckle
Submission date: 07.10.2011
I assure the single handed composition of this diploma thesis only supported by declared resources

<table>
<thead>
<tr>
<th>Ort, Datum</th>
<th>Unterschrift</th>
</tr>
</thead>
</table>
Contents

1 Introduction 4

2 Systems of Linear Equations and Direct Solving Methods 5
  2.1 System of Linear Equations . . . . . . . . . . . . . . . . . . . 5
  2.2 Condition Number . . . . . . . . . . . . . . . . . . . . . . . . . 5
  2.3 Triangular System of Linear Equations . . . . . . . . . . . . . 6
  2.4 Gaussian Elimination . . . . . . . . . . . . . . . . . . . . . . . 6
  2.5 Matrix Factorizations . . . . . . . . . . . . . . . . . . . . . . . 7
    2.5.1 LU Decomposition . . . . . . . . . . . . . . . . . . . . . . 7
    2.5.2 Cholesky Decomposition . . . . . . . . . . . . . . . . . . 8

3 Iterative Solution Methods 9
  3.1 Basic Iterative Methods . . . . . . . . . . . . . . . . . . . . . . 9
  3.2 Krylov Subspace Methods . . . . . . . . . . . . . . . . . . . . . 10
    3.2.1 GMRES . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11
    3.2.2 CG . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11

4 Preconditioner 14
  4.1 Preconditioned Iterative Methods . . . . . . . . . . . . . . . . 14
  4.2 Incomplete LU . . . . . . . . . . . . . . . . . . . . . . . . . . . 15
  4.3 Approximate Inverse Preconditioners . . . . . . . . . . . . . . 16

5 SPAI 17
  5.1 SAI: Preconditioner for a Fixed Sparsity Pattern . . . . . . . . 17
  5.2 SPAI: Updating the Sparsity Pattern . . . . . . . . . . . . . . . 17
  5.3 Theoretical Properties . . . . . . . . . . . . . . . . . . . . . . . 19

6 Block SPAI 21
  6.1 BSAI: Block Preconditioner for a Fixed Sparsity Block Pattern . 21
  6.2 BSPAI: Updating the Sparsity Pattern . . . . . . . . . . . . . . 21

7 Factorized SPAI 24
  7.1 FSAI: Preconditioner for a Fixed Sparsity Pattern . . . . . . . 24
  7.2 FSPAI: Updating the Sparsity Pattern . . . . . . . . . . . . . . 26
  7.3 Theoretical Properties . . . . . . . . . . . . . . . . . . . . . . . 29
  7.4 Updating the Kaporin Functional . . . . . . . . . . . . . . . . . 30
  7.5 Numerical Results . . . . . . . . . . . . . . . . . . . . . . . . . 35

8 Exact FSPAI 39
  8.1 EFSPAI-Level 1 . . . . . . . . . . . . . . . . . . . . . . . . . . . 40
  8.2 EFSPAI-Level 2 . . . . . . . . . . . . . . . . . . . . . . . . . . . 40
  8.3 EFSPAI-Level 3 . . . . . . . . . . . . . . . . . . . . . . . . . . . 40
  8.4 Numerical Results . . . . . . . . . . . . . . . . . . . . . . . . . 44
    8.4.1 EFSPAI-Level 1 and 2 . . . . . . . . . . . . . . . . . . . . 44
    8.4.2 EFSPAI-Level 3 . . . . . . . . . . . . . . . . . . . . . . . . 45
1 Introduction

Large sparse linear systems of equations $Ax = b$ arise from the discretisation of partial differential equations. Therefore they are one of the most important problems in numerics. To solve such systems, iterative methods like GMRES and CG are used. These iterative methods are often divergent or converge very slowly, which can be improved considerably by preconditioning techniques, i.e., we seek a matrix $M$ so that e.g. the right preconditioned system $AMy = b$ and $My = x$ can be solved easier by an iterative method. Standard preconditioners like ILU lead to valuable convergence rates, but are highly sequential.

One possible way to obtain an effective, robust and easy to parallelize preconditioner is the sparse approximate inverse method. The goal is to find a sparse matrix $M$ that approximates $A^{-1}$. For a prescribed non-zero pattern, this can easily be achieved by minimizing $\|AM - I\|_F$. The difficulty here is to find a good sparsity pattern for $M$. With the SPAI algorithm [13] Grote and Huckle developed an effective way to automatically generate such a pattern. Iteratively a new index is added to the current pattern that is estimated to be the most profitable one. Huckle [14] advances this idea for symmetric positive definite (s.p.d.) problems. The FSPAI finds an efficient sparsity pattern for a lower triangular matrix $L$ for the split preconditioned system $L^\top ALy = L^\top b$ and $Ly = x$. Analogue to SPAI we estimate the quality of a new possible index but use it for reducing the Kaporin functional $K(L^\top AL) = (1/n) \text{trace}(L^\top AL)/\det(L^\top AL)^{1/n}$.

First this work is concerned with a detailed derivation of the FSPAI and its theoretical properties. Furthermore we obtain the exact improvement of adding a new index to the current pattern by updating the Kaporin functional, which leads to the Exact FSPAI. Finally we provide the FSPAI for block matrices. There are linear systems with a natural block structure due to their physical origin. The Block FSPAI exploits this additional property and yields a qualitative preconditioner. Standard MATLAB tests approve the quality of these algorithms.
2 Systems of Linear Equations and Direct Solving Methods

Systems of linear equations, or linear systems, arise in many different scientific and engineering fields, and therefore are one of the most important numerical problems.

In this chapter we first discuss the basic theoretical aspects of the existence of solutions. Afterwards we take a closer look at the condition of a linear system. The chapter ends with the main methods for finding the exact solution, named direct methods.

All information here can be found in any introduction book for numerical mathematic. All descriptions are based on [15, 11, 25, 2, 10]. For a more theoretical view, see e.g. [6, 7].

2.1 System of Linear Equations

A system of linear equations is a system

\[
\begin{align*}
A_{11}x_1 + A_{12}x_2 + \ldots + A_{1m}x_m &= b_1 \\
A_{21}x_1 + A_{22}x_2 + \ldots + A_{2m}x_m &= b_2 \\
& \vdots \\
A_{n1}x_1 + A_{n2}x_2 + \ldots + A_{nm}x_m &= b_n
\end{align*}
\]

with \(A_{ij}, b_i \in \mathbb{R}\) all given information, and all \(x_j\) unknowns. This system is typically written in matrix notation

\[
Ax = b
\]  

(2.1)

with \(A \in \mathbb{R}^{n \times m}\) and \(b \in \mathbb{R}^n\). Our goal is to solve these equations, i.e., to find a \(x \in \mathbb{R}^n\) with \(Ax = b\). The question regarding the existence of such a solution is easy to answer.

There exists a solution for \(Ax = b\), if \(\text{rank}(A) = \text{rank}(A|b)\), where \(A|b\) is the augmented matrix obtained by adding the vector \(b\) as a column to \(A\). Especially if we deal with square matrices, there is always a unique solution, if \(A\) is nonsingular, which is then given by \(\hat{x} = A^{-1}b\).

From now on we assume the matrix \(A\) to be nonsingular.

2.2 Condition Number

As the numerical analysis deals with algorithms that approximate the real solution of a problem, it is inevitable to estimate the quality of an algorithmic solution. Even assuming a perfect algorithm, the output can be heavily error-prone due to small input errors.
The **condition number** is a indicator for amplification of the relative input error. In general, for linear systems of equations, the condition number $\kappa(\cdot)$ or $\text{cond}(\cdot)$ of $A$ is given by

$$\kappa(A) := \text{cond}(A) := \|A^{-1}\| \|A\|,$$

where $\|\cdot\|$ is any matrix norm. For s.p.d. systems and the usage of spectral norm, the condition number can be denoted by

$$\kappa(A) = \frac{\lambda_{\text{max}}(A)}{\lambda_{\text{min}}(A)},$$

where $\lambda_{\text{max}}(A)$ and $\lambda_{\text{min}}(A)$ are the maximal and minimal eigenvalues of $A$. In addition, a linear system is called **ill-conditioned**, if the condition number is very large, and well-conditioned, if the condition number is low. As it also effects the convergence behaviour of iterative methods, reducing the condition number is one of the main aspects of preconditioning.

### 2.3 Triangular System of Linear Equations

All direct solution methods are based on reducing $Ax = b$ to a system which is easier to handle, mainly transforming it to systems with triangular structure.

Consider $Lx = b$ with a lower triangular nonsingular matrix $L$:

$$
\begin{pmatrix}
L_{11} & & & \\
L_{21} & L_{22} & & \\
\vdots & \ddots & \ddots & \\
L_{n1} & \cdots & L_{nn} & \\
\end{pmatrix}
\begin{pmatrix}
x_1 \\
x_2 \\
\vdots \\
x_n \\
\end{pmatrix}
=
\begin{pmatrix}
b_1 \\
b_2 \\
\vdots \\
b_n \\
\end{pmatrix}
$$

This can easily be solved top-down by substituting the known solution elements into the next row. This algorithm is called **forward substitution**:

**Algorithm 2.1 Forward Substitution**

for rows $i = 1 \ldots n$

$$x_i = \frac{1}{L_{ii}}(b_i - \sum_{j=1}^{i-1} L_{ij} x_j)$$

end for

The corresponding form for upper triangular matrices is called **back substitution**.

### 2.4 Gaussian Elimination

As previously announced we will present methods to reduce linear systems to triangular ones. The most common method is the **Gaussian elimination**. It transforms the whole system $Ax = b$ using elementary row operations to an equivalent system $Ux = b'$ with an upper triangular $U$ and a new right side $b'$. The elementary row operations that don’t change the solution, are:
• multiplying all elements of a row $i$ and $b_i$ with a scalar $c \in \mathbb{R}_{\neq 0}$
• adding a complete row including the $b$ element to another row
• permuting rows of $A$ and $b$.

For each column $j$ the Gaussian elimination uses these operations to generate zeros below the diagonal.

Algorithm 2.2 Gaussian Elimination without pivoting

\begin{algorithm}
for $j = 1 \ldots n$ do
  for $i = j + 1 \ldots n$ do
    $L_{ij} = A_{ij} / A_{jj}$
    $b_i = b_i - L_{ij} b_j$
    for $k = j + 1 \ldots n$ do
      $A_{ik} = A_{ik} - L_{ij} A_{jk}$
    end for
  end for
end for
\end{algorithm}

This procedure results in a system with upper triangular matrix, which can be solved by back substitution. Notice that in order to prevent divisions by zero elements and to increase the numerical stability, pivoting is included, but will not be discussed here.

The computational costs for this algorithm are

\[ \frac{2}{3} n^3 + O(n^2). \] \hspace{1cm} (2.2)

2.5 Matrix Factorizations

A matrix factorization is a decomposition of the matrix $A$ into a product of matrices such that the linear system is easier to solve. They are especially useful when we have to find a solution of more than one linear system of equations with different right sides but the same matrix. Then we only have to compute the expensive decomposition once.

2.5.1 LU Decomposition

The **LU decomposition** factorizes a matrix $A$ into

\[ A = LU \]

with a lower triangular matrix $L$ and an upper triangular matrix $U$. To make this decomposition unique, we additionally set the diagonal elements of $L$ to one. Note that the following algorithm has similar operations as algorithm 2.2, except the order is a little bit changed. The Gaussian intermediate system $Ux = b'$ can be found here, as the upper triangular matrices are the same, and $b' = L^{-1} b$. As in Part 2.4, we just mention the necessity of pivoting via permutation matrices.
Therefore, we can use the Gaussian elimination algorithm to compute the LU decomposition, only skipping the b-updates. The entries of $L$ are the Gaussian elimination factors $L_{ij}$. The operational costs are of the same order as for the Gaussian elimination (2.2). This leads to the following algorithm, where the $L$ and $R$ entries overwrite $A$. Notice that we don’t have to store the diagonal elements of $L$ as they are set to one a priori.

Algorithm 2.3 LU Decomposition

```
for $j = 1 \ldots n$ do
  for $i = j + 1 \ldots n$ do
    $A_{ij} = A_{ij}/A_{jj}$
    for $k = j + 1 \ldots n$ do
      $A_{ik} = A_{ik} - A_{ij}A_{jk}$
    end for
  end for
end for
```

Now a linear system of equations $b = Ax = LUx$ can be solved using a forward $Ly = b$ and a back substitution $Ux = y$.

2.5.2 Cholesky Decomposition

If the matrix $A$ is additionally s.p.d., it can be decomposed into Cholesky factors

$$A = LL^\top$$

with lower triangular $L$ having positive diagonal entries. The following implementation computes $L$:

Algorithm 2.4 Cholesky Decomposition

```
for $j = 1 \ldots n$ do
  $L_{jj} = \left( A_{jj} - \sum_{k=1}^{j-1} L_{jk}^2 \right)^{\frac{1}{2}}$
  for $i = j + 1 \ldots n$ do
    $L_{ij} = (A_{ij} - \sum_{k=1}^{j-1} L_{ik}L_{jk})/L_{jj}$
  end for
end for
```

As mentioned in A.8.3, this paper uses the factorization

$$A = L^\top L,$$

which can be implemented analogous.

Again linear systems can be solved using forward and back substitution. The computational costs are halved compared to the LU decomposition with $\frac{1}{3}n^3 + O(n^2)$. 8
3 Iterative Solution Methods

Often we are dealing with large and sparse systems of linear equations. A matrix is called sparse if it only has a low number of nonzero entries, typically $O(n)$. Therefore it is more efficient to store only these entries so that the matrix vector multiplication can be performed very effectively in $O(n)$ instead of $O(n^2)$ of time. Such systems mainly occur by discretizations of partial differential equations (PDE). Applying direct methods to large sparse linear systems would be too storage and time consuming with operational costs of $O(n^3)$. Furthermore, intermediate results of direct methods are typically dense, i.e., they have many nonzero entries. Hence much faster methods are of potential interest that exploit the sparsity, in fact use the matrix $A$ only for the matrix vector multiplication. This can be achieved by iterative methods.

An iterative method starts with an initial guess $x^{(0)} \in \mathbb{R}^n$, which is updated in each iteration only using matrix vector multiplications. This method produces a sequence of vectors $x^{(k)}$ that hopefully converges to the exact solution $\hat{x} = A^{-1}b$. This procedure is interrupted, if the current $x^{(k)}$ satisfies certain quality conditions like, for example, a small residual norm.

Three quality characteristics for iterative methods have to be considered:

1. the iterative method should be convergent,
2. the limes should be the solution $\hat{x}$, i.e., $\lim_{k \to \infty} x^{(k)} = \hat{x}$, and
3. the iterative method should converge fast.

This chapter introduces basic iterative methods and the well known Krylov space methods GMRES and CG. We focus on the quality criteria above, as we later want to find preconditioners to improve these methods. The basic iterative part is based on the corresponding chapter in [15], the Krylov part on [26, 11]. [2] also gives an insight into iterative solvers.

3.1 Basic Iterative Methods

The most basic iterative method is the Richardson Iteration. We obtain the iteration function by transforming the linear system:

$$ b = Ax = (A - I)x + x \iff x = b + (I - A)x $$

In each iteration we update the current solution via $x^{(k+1)} = b + (I - A)x^{(k)}$. Now we analyse the convergence of the sequence $x^{(k)}$.

**Theorem 3.1.** If the Richardson Iteration converges, the limes is the solution of the linear system $\hat{x} = A^{-1}b$.

**Proof.** The proof uses fixed point iteration knowledge. Consider the iteration function $\Phi(x) = b + (I - A)x$. As $\Phi$ is continuous, we conclude:

$$ \hat{x} = \lim_{k \to \infty} x^{(k+1)} = \lim_{k \to \infty} \Phi(x^{(k)}) = \Phi(\lim_{k \to \infty} x^{(k)}) = \Phi(\hat{x}) $$

The limes is a fixed point of $\Phi$ and $\hat{x} = A^{-1}b$ is its only one. □
Theorem 3.2. If \( ||I - A||_2 < 1 \), the Richardson Iteration converges. Hence \( ||I - A||_2 \) is also an indicator for the convergence speed.

Proof. Let \( \hat{x} \) be the solution of the linear system, i.e., \( Ax = b \). Let \( x^{(k)} \) be the sequence produced by the Richardson Iteration. We can express the distance between the \( k \)th sequence element and \( x^{(0)} \) depending on the initial distance:

\[
\hat{x} - x^{(k+1)} = \hat{x} - x^{(k)} - (b - Ax^{(k)}) = \hat{x} - x^{(k)} - A\hat{x} + Ax^{(k)}
\]

\[
= (\hat{x} - x^{(k)}) - A(\hat{x} - x^{(k)}) = (I - A)(\hat{x} - x^{(k)})
\]

\[
= (I - A)^k(\hat{x} - x^{(0)})
\]

\[
\Rightarrow ||\hat{x} - x^{(k+1)}||_2 \leq ||I - A||^k_2||\hat{x} - x^{(0)}||_2
\]

Note that Theorem 3.2 can be proved using the Frobenius norm. An even more accurate convergence statement uses the spectral radius \( \rho \).

Theorem 3.3. If and only if \( \rho(A) < 1 \), the Richardson Iteration converges for any \( x^{(0)} \).

Proof. For proof see [26]

Further basic iterative methods are, for example, the Jacobi- or Gauss-Seidel iterations. They can be introduced via splittings \( A = C - R \), but here we introduce them as the Richardson Iteration applied on a preconditioned system. See Section 4 on this.

3.2 Krylov Subspace Methods

A Krylov subspace method generates a sequence \( x^{(k)} \) with

\[ x^{(k)} \in x^{(0)} + K_m(A, r^{(0)}) \]

and

\[ K_m(A, r^{(0)}) = \text{span}(r^{(0)}, Ar^{(0)}, A^2r^{(0)}, \ldots, A^{m-1}r^{(0)}) \]

being the \textbf{Krylov subspace} and

\[ r^{(i)} = b - Ax^{(i)} \]

the \( i \)th \textbf{residual}. \( x^{(k)} \) is chosen to be the best approximation of the linear system solution \( \hat{x} = A^{-1}b \) for all vectors in \( x^{(0)} + K_m(A, r^{(0)}) \) in some sense. After at most \( n \) iterations, the Krylov subspace method finds \( \hat{x} \):

Theorem 3.4.

\[ \exists m \leq n \text{ mit } \hat{x} = A^{-1}b \in x^{(0)} + K_m(A, r^{(0)}) \]
Proof. Let \( m \) be the largest number with \( K_m(A, r^{(0)}) \) linearly independent and \( K_{m+1}(A, r^{(0)}) \) linearly dependent. Then \( A^m r^{(0)} \in K_m(A, r^{(0)}) \), and therefore

\[
AK_m(A, r^{(0)}) \subset K_m(A, r^{(0)}).
\]

As \( A \) is nonsingular, both sides are equal and we can conclude:

\[
AK_m(A, r^{(0)}) = K_m(A, r^{(0)}) \iff A^{-1} K_m(A, r^{(0)}) = K_m(A, r^{(0)})
\]

\[
\Rightarrow A^{-1} r^{(0)} \in K_m(A, r^{(0)})
\]

\[
\Rightarrow \hat{x} = A^{-1} b = A^{-1}(r^{(0)} + A x^{(0)}) = x^{(0)} + A^{-1} r^{(0)} \in x^{(0)} + K_m(A, r^{(0)}).
\]

\[\square\]

As we cannot expect the method to find the exact solution due to rounding errors, we stop the iteration if a norm of the current residual is low enough. We present two important Krylov subspace methods.

- **GMRES** chooses \( x^{(k)} \) to be the minimum of \( \|b - Az\|_2 \) for all \( z \in x^{(0)} + K_m(A, r^{(0)}) \).
- **CG** chooses \( x^{(k)} \) to be the minimum of \( \|\hat{x} - z\|_A \) for all \( z \in x^{(0)} + K_m(A, r^{(0)}) \) for a system with s.p.d. matrix \( A \).

Further Krylov subspace methods can be found in [26], [2] and [11].

### 3.2.1 GMRES

The **generalized minimum residual method (GMRES)** finds \( x^{(m)} \) that minimizes \( \|b - Az\|_2 \) for all \( z \in x^{(0)} + K_m(A, r^{(0)}) \). First it builds an orthonormal basis \( \{v_1, \ldots, v_{m-1}\} \) of \( K_m(A, r^{(0)}) \). To this basis it is easy to find the representation of the optimal vector

\[
x^{(m)} = x^{(0)} + \sum_{j=1}^{m+1} y_j v_j \text{ with } y = \arg\min_{y^r} \|\beta e_1 - \tilde{H}_m y^r\|_2.
\]

(3.1)

(3.1) can be solved using using least square techniques. Both \( \beta \) and \( \tilde{H}_m = (H_{ij})_{1 \leq i, j \leq m+1} \) are also constructed by the Algorithm 3.1.

A detailed view on GMRES is given in [26].

### 3.2.2 CG

We consider a linear system, where \( A \) is s.p.d.. Then the solution \( \hat{x} \) can be considered as the minimum of

\[
F(z) = \frac{1}{2} z^\top A z - b^\top z.
\]

Hence every minimization method on \( F \) can be used as a method for solving \( A x = b \), like for example the steepest descent method [15].
Algorithm 3.1 GMRES without stopping criterion

\[ r^{(0)} = b - Ax^{(0)}, \beta = ||r^{(0)}||_2, v_1 = r^{(0)}/\beta \]

for \( j = 1 \ldots m \) do
\[ w_j = Av_j \]
for \( i = 1 \ldots j \) do
\[ H_{ij} = w_j^T v_i \]
\[ w_j = w_j - H_{ij} v_i \]
end for
\[ H_{j+1,j} = ||w_j||_2 \]
\[ v_{j+1} = w_j/H_{j+1,j} \]
end for
\[ y = \arg\min_{y'} ||\beta e_1 - \tilde{H}_m y'||_2 \]
\[ x^{(m)} = x^{(0)} + \sum_{j=1}^{m} y_j v_j \]

Here we present the Conjugate Gradient (CG) method. In each iteration, the CG sets \( x^{(k)} \) to be the minimum of \( F(z) \) for all \( x^{(0)} + K_m(A, r^{(0)}) \), or equivalent the minimum of \( ||\hat{x} - z||_A \). This minimum is obtained by constructing the \( x^{(k)} \) dependent on another basis \( \{p^{(0)}, \ldots, p^{(m-1)}\} \) of the Krylov subspace \( K_m(A, r^{(0)}) \), which is conjugate orthogonal with respect to \( A \), i.e., \( p^{(i)} A p^{(j)} = 0, \forall i \neq j \). This results in the following algorithm:

Algorithm 3.2 CG without stopping criterion

\[ p^{(0)} = r^{(0)} = b - Ax^{(0)} \]

for \( k = 0 \ldots n - 1 \) do
\[ \alpha^{(k)} = \frac{r^{(k)T} r^{(k)}}{p^{(k)T} A p^{(k)}} \]
\[ x^{(k+1)} = x^{(k)} + \alpha^{(k)} p^{(k)} \]
\[ r^{(k+1)} = r^{(k)} - \alpha^{(k)} A p^{(k)} \]
\[ \beta^{(k)} = \frac{r^{(k+1)T} r^{(k+1)}}{r^{(k)T} r^{(k)}} \]
\[ p^{(k+1)} = r^{(k+1)} + \beta^{(k)} p^{(k)} \]
end for

For more detailed considerations and proofs, see [11, 26]. As we later want to find a preconditioner for the CG, we are interested in convergence propositions. We need properties for the preconditioner that assure a certain quality regarding the iteration rate. The main results are:

Theorem 3.5. The distance between the solution and the \( k \)th sequence element can be estimated using the condition number \( \kappa(A) \):

\[ ||\hat{x} - x_k||_A \leq 2 \left( \frac{\sqrt{\kappa(A)} - 1}{\sqrt{\kappa(A)} + 1} \right)^k ||\hat{x} - x^{(0)}||_A. \]
We receive a more accurate result using the Kaporin functional.

**Theorem 3.6.**

\[ \| r^{(k)} \|_2^2 \leq \left( K(A) \frac{1}{k} - 1 \right)^k \| r^{(0)} \|_2^2 \]  \hspace{1cm} (3.2)

*with the Kaporin functional* \[ K(A) = \frac{1}{n} \frac{\text{trace}(A)}{\text{det}(A)^{\frac{1}{n}}}. \]  \hspace{1cm} (3.3)

We have superlinear convergence, if \( k \) is large enough that \( K(A) \frac{1}{k} < 2 \). The Kaporin functional was introduced by Kaporin in [17]. It takes all eigenvalues of \( A \) into account, as \( \text{trace}(A) \) is the sum and \( \text{det}(A) \) is the product of the eigenvalues. By contrast, the condition number only depends on the extreme eigenvalues. Furthermore, the Kaporin functional satisfies \( K(A) \geq 1 \) and \( K(A) = 1 \) if and only if \( A \) is a multiply of the identity matrix. This Kaporin functional is our approach to derive a sparse approximate inverse preconditioner for s.p.d. systems in Chapter 7. A summary of properties can be also found in [2] under the name K-condition number.
4 Preconditioner

In many numerical applications, the solution of large sparse linear systems $Ax = b$ is the most time consuming part. This necessity of improving the convergence behaviour of iterative methods yields many different preconditioning techniques. Saad [26] describes preconditioners as a “modification of a linear system which makes it easier to solve by a given iterative method”. Here, a preconditioner is a matrix $M$ that transforms $Ax = b$ to an equivalent system, while there exist three possibilities:

- left preconditioning $M Ax = Mb$
- right preconditioning $AM y = b$ and $x = My$
- split preconditioning $M_R A M_L y = M_R b$ and $x = M_L y$ with a factorization $M = M_L M_R$.

These systems have not to be computed explicitly as preconditioners can be integrated into standard iterative solvers. The first part shows a few preconditioned methods. In these methods, the preconditioner is only used for the evaluation of $z = Mr$. Besides a fast computation of $z = Mr$, a few properties have been established for good preconditioners:

- $MA$ approximates $I$
- $M$ reduces condition number, i.e., $\kappa(MA) \ll \kappa(A)$
- eigenvalues of $MA$ cluster around 1.

We present two important preconditioning techniques: ILU and Approximate Inverses. Further information in basic preconditioning is given in [25]. Overviews of preconditioning techniques can be found in [26, 2, 5].

4.1 Preconditioned Iterative Methods

Classical iterative methods are splitting methods. They can be interpreted as applying the Richardson iteration on a left-preconditioned system. The Jacobi algorithm is a splitting method using the preconditioner $M = D^{-1}$ with $D$ the diagonal matrix with entries $D_{ii} = A_{ii}$. This is equivalent to Algorithm 4.1.

\begin{algorithm}
\textbf{Algorithm 4.1} Jacobi algorithm
\begin{algorithmic}
\For{$k = 1 \ldots$ until convergence}
\For{$i = 1 \ldots n$}
\State $x_i^{(k+1)} = \frac{1}{A_{ii}} (b_i - \sum_{j \neq i} A_{ij} x_j^{(k)})$
\EndFor
\EndFor
\end{algorithmic}
\end{algorithm}

An improvement in terms of approximating $I$ is the Gauss-Seidel algorithm. Here our preconditioner is the inverse of the lower left part of the
matrix $A$ including the diagonal entries. This can be written exactly like the Jacobi algorithm but with overwriting the entries of the current solution instead of creating a new sequence vector $x^{(k)}$.

**Algorithm 4.2** Gauss-Seidel algorithm

\[
\begin{aligned}
&\text{for } k = 1 \ldots \text{ until convergence} \text{ do} \\
&\quad \text{for } i = 1 \ldots n \text{ do} \\
&\quad \quad x_i = \frac{1}{A_{ii}} (b_i - \sum_{j \neq i} A_{ij} x_j) \\
&\quad \text{end for} \\
&\text{end for}
\end{aligned}
\]

A preconditioner can also be integrated into the CG method, i.e., we obtain the **preconditioned Conjugate Gradient algorithm (PCG)** according to Algorithm 4.3.

**Algorithm 4.3** PCG

\[
\begin{aligned}
&\quad r^{(0)} = b - Ax^{(0)}, p^{(0)} = z^{(0)} = M^{-1}r^{(0)} \\
&\text{for } k = 0 \ldots n - 1 \text{ do} \\
&\quad \alpha^{(k)} = \frac{r^{(k)}^T z^{(k)}}{r^{(k)}^T p^{(k)}} \\
&\quad x^{(k+1)} = x^{(k)} + \alpha^{(k)} p^{(k)} \\
&\quad r^{(k+1)} = r^{(k)} - \alpha^{(k)} Ap^{(k)} \\
&\quad z^{(k+1)} = M^{-1} r^{(k+1)} \\
&\quad \beta^{(k)} = \frac{r^{(k+1)}^T z^{(k+1)}}{r^{(k)}^T z^{(k)}} \\
&\quad p^{(k+1)} = z^{(k+1)} + \beta^{(k)} p^{(k)} \\
&\text{end for}
\end{aligned}
\]

### 4.2 Incomplete LU

Using the preconditioner $M = A^{-1}$ or $M = (LU)^{-1}$ with $A = LU$ the LU decomposition would be perfect in the sense of iterative method convergence. Unfortunately computing this factorization would be too time and space consuming. The idea of the **incomplete LU Decomposition (ILU)** is to find a factorization

\[
M^{-1} = LU
\]

with $L$ being a lower and $U$ an upper triangular matrix, respectively, that approximates $A$ under some structure conditions. The simplest ILU version is the **ILU(0)**, the ILU with no fill-in. It computes the elements $L_{ij}$ and $U_{ij}$ only if the corresponding $A_{ij}$ is not equal to zero. In Algorithm 4.4, $A$ will be overwritten by the LU decomposition.

This concept can be easily improved by allowing some fill-in. A possible implementation is the **ILU(p)**, where we set all entries that would be filled,
Algorithm 4.4 ILU(0)

\begin{algorithm}
\begin{algorithmic}
\FOR {$j = 1 \ldots n$}
    \FOR {$i = j + 1 \ldots n$ and if $A_{ij} \neq 0$}
        \STATE $A_{ij} = A_{ij} / A_{jj}$
        \FOR {$k = j + 1 \ldots n$ and if $A_{ik} \neq 0$}
            \STATE $A_{ik} = A_{ik} - A_{ij} A_{jk}$
        \ENDFOR
    \ENDFOR
\ENDFOR
\end{algorithmic}
\end{algorithm}

if we apply ILU(0) $p$ times on $A$. This often leads to an effective preconditioner. Note that, with the same idea we can construct an incomplete Cholesky decomposition. For more information on ILU, see [26] or [25].

4.3 Approximate Inverse Preconditioners

The goal of approximate inverse is to find a matrix $M \approx A^{-1}$ for a right preconditioned system $AMy = b$. It was originally proposed by P.O. Frederickson [24] as SAI. The basic technique is to solve

$$\min_M ||AM - I||_F^2$$

(4.1)

as the Frobenius norm is easy to compute. There are two possible approaches to achieve this goal. The first one applies directly on (4.1). The second way is to split (4.1) into $n$ independent minimization problems:

$$\min_M ||AM - I||_F^2 = \sum_{k=1}^{n} \min_{M_k} ||AM_k - e_k||_2^2$$

(4.2)

with matrix columns $M_k$ and unit vectors $e_k$. Both considerations can be solved using iterative minimization algorithms, where the solution is prevented to become too dense. The advantage of (4.2) is the inherent parallelism of the splitting.

A further idea to expose (4.2) is to minimize it for a prescribed sparsity pattern. The SPAI algorithm is a way to handle this and finds a preconditioner without providing a priori knowledge on the pattern. In Chapter 5 we take a closer look at the SPAI idea introduced by Grote and Huckle [13].

Summaries of preconditioners following the approximate inverse concept can be found in [8, 28, 5].
5 SPAI

The first preconditioner we are going to have a closer look at is the sparse approximate inverse (SPAI), introduced by Grote and Huckle [13]. SPAI computes an easy to parallelize and effective preconditioner for an arbitrary linear system of equations based on the approximate inverses technique. Our goal is to find a preconditioner $M$ which minimizes (4.1) under certain sparsity conditions. Barnard, Bernardo, and Simon demonstrated the quasi-optimal scaling behaviour of SPAI in [3].

This method is divided into two parts. First we find the minimum of (4.2) independently for each column $M_k$ for a given sparsity column structure. The main problem is to find a good sparsity pattern for $M_k$. Hence, secondly, SPAI updates the structure iteratively by adding entries that give reason to expect the biggest reduction of (4.2).

5.1 SAI: Preconditioner for a Fixed Sparsity Pattern

Consider a fixed sparsity structure $J$ for the preconditioner $M$. The key of the high parallel potential is the separation into $n$ independent minimization problems for each $M_k$. Therefore SAI solves $n$ least square problems

$$\min_{M_k} \| AM_k - e_k \|_2$$

for each column pattern $J_k$. Because $A$ and $M_k$ are sparse, this system is very small, and it can be implemented using QR decomposition [11]. For a more detailed consideration, see [13].

5.2 SPAI: Updating the Sparsity Pattern

Let $M_k$ be the solution of the minimization problem (5.1) for a given sparsity pattern $J_k$. Now we seek for the optimal new index $j \notin J_k$ so that (5.1) gets as small as possible for the new pattern $J_k \cup \{j\}$. We get a simple to compute upper bound by a one dimensional minimization in $M_{jk}$, keeping all other entries fixed:

$$\min_{M_{jk}} \| A(M_k + M_{jk}e_j) - e_k \|_2 = \min_{M_{jk}} \| r + M_{jk}A_j \|_2$$

with the residual $r = AM_k - e_k$.

**Theorem 5.1.** The solution of (5.2) is

$$M_{jk} = -\frac{r^\top A_j}{\| A_j \|_2^2}$$

(5.3)
Proof. We set the derivative equal to zero:
\[
\frac{\partial}{\partial M_{jk}} ||r_{new}||_2^2 = \frac{\partial}{\partial M_{jk}} ||r + M_{jk}A_j||_2^2 = 2(r + M_{jk}A_j)^\top A_j
\]
\[
= 2(r^\top A_j + M_{jk}||A_j||_2^2) = 0
\]
\[
\Leftrightarrow M_{jk} = -\frac{r^\top A_j}{||A_j||_2^2}
\]
As \( A \) is nonsingular, the second derivative is always strictly positive:
\[
\frac{\partial^2}{\partial^2 M_{jk}} ||r_{new}||_2^2 = 2||A_j||_2^2.
\]
Therefore, \( M_{jk} = -\frac{r^\top A_j}{||A_j||_2^2} \) minimizes the new residual norm.

For each candidate \( j \) the reduction of the norm of the new residual is our quality criterion:

**Theorem 5.2.** Adding the component \( j \) according to (5.3) to the preconditioner reduces the residual norm by
\[
\tau_j := \frac{(r^\top A_j)^2}{||A_j||_2^2}
\]

Proof.
\[
||r_{new}||_2^2 = ||r + M_{jk}A_j||_2^2 = (r + M_{jk}A_j)^\top (r + M_{jk}A_j)
\]
\[
= r^\top r + 2M_{jk}r^\top A_j + M_{jk}^2 A_j^\top A_j
\]
\[
= ||r||_2^2 - 2\frac{(r^\top A_j)^2}{||A_j||_2^2} + \frac{(r^\top A_j)^2}{||A_j||_2^2}||A_j||_2^2
\]
\[
= ||r||_2^2 - \frac{(r^\top A_j)^2}{||A_j||_2^2}
\]
\[
\tau_j \text{ is a lower bound for the reduction capability of index } j. \text{ Hence, it is possible that all } \tau_j \text{ to zero, while there is still room for improvement. The following theorem helps us out.}

**Theorem 5.3.** If the residual \( r \neq \vec{0} \), there is at least one index \( j \) with \( \tau_j > 0 \).

Proof. Proof by contradiction: Assume \( r \neq \vec{0} \) and \( \tau_j = 0 \ \forall j = 1 \ldots n \). Then we can conclude
\[
\forall j : \tau_j = \frac{(r^\top A_j)^2}{||A_j||_2^2} = 0
\]
\[
\Leftrightarrow \forall j : A_j^\top r = 0
\]
\[
\Leftrightarrow A^\top r = \vec{0}
\]
\[
\Leftrightarrow r = \vec{0}, \text{ as } A \text{ is nonsingular, which is a contradiction.}
Obviously, $\forall j \in \mathcal{J}_k : \tau_j = 0$, as we cannot improve through changing the entries with indices in $\mathcal{J}_k$.

$\tau_j$ only has to be computed for a small index set $\hat{\mathcal{J}}$, as both $r$ and $A$ are assumed to be very sparse, and many $\tau_j$ are known to be zero a priori. If we implement the SPAI without any stopping criterion, it produces the exact inverse of $A$. As this mostly will lead to a full matrix and we only want to add the most profitable indices, we stop if the current residual is smaller than a tolerance $\epsilon$ or the number of nonzeros in the column $M_k$ reaches the limit $p_{max}$. This leads to Algorithm 5.1. Note that it is also possible to add more than one index.

**Algorithm 5.1 SPAI**

**Input:** $\mathcal{J}$

for $k = 1 \ldots n$
do

$M_k = \operatorname{argmin}_{M_k} ||AM'_k - e_k||_2$ with $\mathcal{J}_k$

$r = AM_k - e_k$

while $||r||_2 > \epsilon$ and $\text{nnz}(M_k) < p_{max}$ do

$\hat{\mathcal{J}}$ set of potential candidates

for $j \in \hat{\mathcal{J}}$ do

$\tau_j = \frac{(r^\top A_j)^2}{||A_j||_2^2}$

end for

$\mathcal{J}_k = \mathcal{J}_k \cup \{l\}$ with largest $\tau_l$

$M_k = \operatorname{argmin}_{M'_k} ||AM'_k - e_k||_2$ with $\mathcal{J}_k$

$r = AM_k - e_k$

end while

end for

5.3 Theoretical Properties

This is a summary of the most important theoretical properties. For proofs see [13]. They all confirm the quality of the SPAI preconditioner, as they show approximation of the inverse, clustering of the eigenvalues around one and bounding of the condition number.

Let $\epsilon$ be the tolerance number for $||r||_2$ in SPAI, $p_{max}$ the maximal number of nonzeros of all residuals $(AM_k - e_k)$. Then

**Theorem 5.4.**

$$||AM - I||_F \leq \sqrt{n} \epsilon$$

$$||AM - I||_2 \leq \sqrt{n} \epsilon$$

$$||AM - I||_1 \leq \sqrt{p_{max}} \epsilon$$

**Corollary 5.5.** If $\sqrt{p_{max}} \epsilon < 1$, then $M$ is nonsingular.
**Theorem 5.6.** All eigenvalues of $AM$ lie inside a circle of radius $\sqrt{p_{\text{max}} \epsilon}$ around one. If $\sqrt{p_{\text{max}} \epsilon} < 1$, then

$$\frac{\lambda_{\text{max}}}{\lambda_{\text{min}}} \leq \frac{1 + \sqrt{p_{\text{max}} \epsilon}}{1 - \sqrt{p_{\text{max}} \epsilon}}$$

with $\lambda_{\text{max}}$ and $\lambda_{\text{min}}$ being the maximal and minimal eigenvalues of $AM$.

**Theorem 5.7.** Set $\delta := \sqrt{n \epsilon (1 + \sqrt{n \epsilon})}$. If $\delta < 1$, then the condition number of $AM$ is bounded by

$$\kappa(AM) \leq \sqrt{\frac{1 + \delta}{1 - \delta}}.$$
6 Block SPAI

Linear systems with a certain block structure arise naturally from the underlying physical problem or from reordering techniques. A sparse block matrix can be partitioned, so that there is only a low number of nonzero blocks. This can be exploited and leads to fast computable and efficient preconditioners. Many preconditioning techniques were extended to take advantage of additional block characteristics, as well as the SPAI algorithm (see Chapter 5). The Block SPAI (BSPAII) was proposed by Barnard and Grote [4] which has been proven to be a robust and fast preconditioner. Furthermore, derived from SPAI, it is easy to parallelize. Similar to SPAI, the block columns of the preconditioner \( \mathbf{M} \) can be computed independently from each other. First we find the optimal preconditioner column for a given sparsity pattern. Then we augment the sparsity pattern by adding a block that is expected to have the best improvement potential. All statements and proofs have the same structure as the corresponding SPAI parts.

6.1 BSAI: Block Preconditioner for a Fixed Sparsity Block Pattern

We want to find the preconditioner \( \mathbf{M} \) that minimizes \( \| \mathbf{A}\mathbf{M} - \mathbf{I} \|_F^2 \) for a fixed sparsity block pattern \( \mathcal{J} \). For reasons of simplification, we assume a constant blocksize \( b \). We denote the number of block columns by \( n_b = \frac{n}{b} \) and the current sparsity block pattern of block column \( k \) by \( \mathcal{J}_k \). Let \( M_{jk} \) be the \( j \)th block in the \( k \)th block column \( \mathbf{M}_k \). Note the analogue notation for \( \mathbf{A} \). As in SPAI, we can split the residual norm into \( n_b \) independent minimization problems:

\[
\min_{\mathbf{M}} \| \mathbf{A}\mathbf{M} - \mathbf{I} \|_F^2 = \sum_{k=1}^{n_b} \min_{M_k} \| \mathbf{A}\mathbf{M}_k - \mathbf{E}_k \|_F^2 \quad (6.1)
\]

Because of the sparsity, this is equivalent to a very small least squares problem and can be solved for the pattern \( \mathcal{J}_k \) with QR decomposition.

6.2 BSPAI: Updating the Sparsity Pattern

Consider \( \mathbf{M}_k \) as the solution of the \( k \)th subproblem in (6.1) for the current block sparsity pattern \( \mathcal{J}_k \). Again we want to find a selection criterion to update the sparsity pattern. This is achieved by keeping all entries \( \mathbf{M}_k(\mathcal{J}_k) \) fixed and only minimize according to the new possible block entry \( M_{jk} \):

\[
\min_{M_{jk}} \| \mathbf{A}(\mathbf{M}_k + \mathbf{I}_j M_{jk}) - \mathbf{E}_k \|_F = \min_{M_{jk}} \| \mathbf{R} + A_j M_{jk} \|_F \quad (6.2)
\]

with the residual \( \mathbf{R} = \mathbf{A}\mathbf{M}_k - \mathbf{E}_k \).
Theorem 6.1. The solution of (6.2) is

\[ M_{jk} = -(A_j^T A_j)^{-1} A_j^T R \]  

(6.3)

Proof. First we expand the new residual:

\[ ||R_{new}||_F^2 = ||R + A_j M_{jk}||_F^2 \]

\[ = \text{trace}((R + A_j M_{jk})^\top (R + A_j M_{jk})) \]

\[ = \text{trace}(R^\top R) + 2 \text{trace}(M_{jk} A_j^T R) + \text{trace}(M_{jk} A_j^T A_j M_{jk}) \]  

(6.4)

Derivating this with respect to \( M_{jk} \) using (A.5), (A.6) and setting it equal to zero, we obtain

\[ \frac{\partial}{\partial M_{jk}} ||R_{new}||_F^2 = 2 A_j^T R + 2 A_j^T A_j M_{jk} + (A_j^T A_j)^\top M_{jk} \]

\[ = 2 A_j^T R + 2 A_j^T A_j M_{jk} \]

\[ \Leftrightarrow M_{jk} = -(A_j^T A_j)^{-1} A_j^T R \]

The definiteness of the second derivative can be obtained by transforming the first derivative into an equivalent matrix-vector form. Let \( \vec{m} \in \mathbb{R}^{b^2} \) be the vector containing the stacked columns of \( M_{jk} \). Then

\[ \frac{\partial}{\partial M_{jk}} (2 A_j^T R + 2 A_j^T A_j M_{jk}) = \frac{d}{dM_{jk}} (2 A_j^T A_j M_{jk}) \]

which is equivalent to

\[ \frac{\partial}{\partial \vec{m}} 2 \begin{pmatrix} A_j^T A_j & \cdots & A_j^T A_j \\ A_j^T A_j & \cdots & A_j^T A_j \\ \vdots & \vdots & \vdots \\ A_j^T A_j & \cdots & A_j^T A_j \end{pmatrix} \vec{m} = 2 \begin{pmatrix} A_j^T A_j & \cdots \\ \vdots & \vdots \\ A_j^T A_j & \cdots \end{pmatrix} \]

(6.5)

(6.5) is s.p.d., as \( A_j^T A_j \) is s.p.d. Hence, \( M_{jk} = -(A_j^T A_j)^{-1} A_j^T R \) minimizes \( ||R_{new}||_F^2 \).

For each candidate \( j \) the reduction of the norm of the new residual is our quality criterion:

Theorem 6.2. Adding the component \( j \) according to (6.3) to the preconditioner reduces the residual norm by

\[ \tau_j := \text{trace}(R^\top A_j (A_j^T A_j)^{-1} A_j^T R) \]  

(6.6)
Proof. We set (6.3) into the expended residual norm (6.4):

\[ ||R_{new}||_F^2 = \text{trace}(R^\top R) + 2 \text{trace}(M_j^T A_j^T R) + \text{trace}(M_j^T A_j^T A_j M_j) \]
\[ = ||R||_F^2 + 2 \text{trace}\left[\left( - (A_j^T A_j)^{-1} A_j^T R \right)^\top A_j^T R \right] \]
\[ + \text{trace}\left[\left( - (A_j^T A_j)^{-1} A_j^T R \right)^\top A_j^T A_j \left( - (A_j^T A_j)^{-1} A_j^T R \right) \right] \]
\[ = ||R||_F^2 - 2 \text{trace}\left( R^\top A_j (A_j^T A_j)^{-1} A_j^T R \right) \]
\[ + \text{trace}\left( R^\top A_j (A_j^T A_j)^{-1} A_j^T R \right) \]
\[ = ||R||_F^2 - \text{trace}\left( R^\top A_j (A_j^T A_j)^{-1} A_j^T R \right) \]

With this \( \tau_j \), a BSPAI algorithm can be implemented according to Algorithm 5.1. Notice that the tolerance \( \epsilon \) has to be adjusted to \( \sqrt{b \epsilon} \) to obtain a preconditioner of comparable quality like SPAI.
7 Factorized SPAI

Let us consider the linear systems of equations with s.p.d. $A$. We want to find a preconditioner based on the approximate inverse technique that has a sparsity structure which leads to a good approximation of the inverse of $A$. Applying SPAI on this system mostly leads to a preconditioner without the characteristic positive definiteness, and prevents the application of the efficient PCG.

To keep these properties we consider the split preconditioned system $L^TAL = L^Tb$ and $Ly = x$ where $L^TAL$ should approximate the identity matrix. Kolotilina and Yeremin [19, 29, 18] introduced the FSAI which computes the approximate inverse $L$ to the Cholesky factor $L_A$ of $A$ by minimizing $||L_AL - I||_F$.

When applying diagonal scaling explicit knowledge of $L_A$ is not required. For a given sparsity structure the minimizer $L$ can be determined efficiently in parallel. Note that, Yeremin and Nikishin [30] extended the FSAI to unsymmetric problems. Lazarov and Sigmund [21] applied the FSAI to indefinite symmetric systems.

Huckle proposed the Factorized Sparse Approximate Inverse (FSPAIA) [14] and choose an equivalent way by minimizing the Kaprin functional

$$K(L^TAL) = \frac{1}{n} \text{trace}(L^TAL) - \frac{1}{n} \det(L^TAL)^{1/n}. \tag{7.1}$$

(3.2) motivates the Kaprin functional as a good measure for the convergence of the CG (Section 3.2.2), which usually is the preferred iterative method for solving s.p.d. systems. Huckle was able to compute the minimum of (7.1) for a given sparsity pattern and augment this pattern iteratively using a cheap selection criterion, similar to SPAI in Chapter 5. Again the columns of $L$ are independent and therefore this leads to a high parallelizable algorithm. Recently, there was published a parallel C++ implementation of FSPAIA that shows a high parallel scalability, see [27].

In this chapter we present a detailed derivation of the FSPAIA, theoretical properties and a way to obtain the exact Kaprin reduction value of a new index by updating $L$ in (7.1). These cannot be found in any publication.

7.1 FSAI: Preconditioner for a Fixed Sparsity Pattern

Similar to the SAI approach (Section 5.1) we first present a static version of the FSPAIA, i.e., we want to minimize (7.1) for a given sparsity structure. We denote the submatrix $[A_{ij}]_{i \in \mathcal{I}, j \in \mathcal{J}}$ by $A(\mathcal{I}, \mathcal{J})$ for two index sets $\mathcal{I}$ and $\mathcal{J}$, the $k$th column of $A$ by $A_k$. The same holds for $L$. The current sparsity pattern of $L_k$ is denoted by $\tilde{\mathcal{J}}_k$. As $L$ is a nonsingular lower triangular matrix, $\tilde{\mathcal{J}}_k \subset \{k, \ldots, n\}$ always contains $k$. We set $\tilde{\mathcal{J}}_k = \mathcal{J}_k - \{k\}$. To simplify the FSPAIA notations, we define the following Schur complement:

**Definition 7.1.** Consider the index set $\mathcal{I}_k = \{k, \ldots, n\} - \tilde{\mathcal{J}}_k$. Then the Schur complement $A(\mathcal{I}_k)$

$$S := A(\mathcal{I}_k, \tilde{\mathcal{J}}_k) - A(\tilde{\mathcal{J}}_k, \mathcal{I}_k)^T A(\tilde{\mathcal{J}}_k, \tilde{\mathcal{J}}_k)^{-1} A(\tilde{\mathcal{J}}_k, \mathcal{I}_k) \tag{7.2}$$
with entries

\[ S_{ij} = A_{ij} - A_i(\tilde{J}_k)^\top A(\tilde{J}_k, \tilde{J}_k)^{-1} A_j(\tilde{J}_k) \]  

(7.3)

is symmetric positive definite. Especially the diagonal elements \( S_{kk} \) are positive.

**Theorem 7.2.** Let \( J_k \) be fixed sparsity column pattern. The Kaporin functional (7.1) is minimized by \( L \) with

\[ L_{kk} = \frac{1}{\sqrt{S_{kk}}} = \frac{1}{\sqrt{A_{kk} - A_k(\tilde{J}_k)^\top A(\tilde{J}_k, \tilde{J}_k)^{-1} A_k(\tilde{J}_k)}} \]  

(7.4)

\[ L_k(\tilde{J}_k) = -L_{kk}A_k(\tilde{J}_k, \tilde{J}_k)^{-1} A_k(\tilde{J}_k) \]  

(7.5)

for each column \( k \).

**Proof.** To obtain the minimum of (7.1), we set the derivative of (7.1) equal to zero. As the denominator only depends on the diagonal elements \( L_{kk} \), we expand (7.1) to

\[ K(L^\top AL) = \frac{1}{n} \frac{\text{trace}(L^\top AL)}{\det(L^\top AL)\frac{1}{n}} = \frac{1}{n} \sum_{k=1}^n \frac{[L_k^\top A L_k]}{\det(A)^{\frac{n}{2}} \prod_{k=1}^n L_{kk}^{\frac{1}{2}}} \]

\[ = \sum_{k=1}^n \left[ L_{kk}^2 A_{kk} + 2L_{kk}L_k(\tilde{J}_k)^\top A_k(\tilde{J}_k) + L_k(\tilde{J}_k)^\top A(\tilde{J}_k, \tilde{J}_k) L_k(\tilde{J}_k) \right] \]

\[ = \frac{n \det(A)^{\frac{n}{2}} \prod_{k=1}^n L_{kk}^{\frac{1}{2}}}{n \det(A)^{\frac{n}{2}} \prod_{k=1}^n L_{kk}^{\frac{1}{2}}}. \]

(7.6)

We start derivating only with respect to \( L_k(\tilde{J}_k) \), setting this equal to zero and solving it by \( L_k(\tilde{J}_k) \). Inserting the solution into (7.6) results in a function only dependent on \( L_{kk} \) and can be derivated with respect to \( L_{kk} \) to obtain the minimum. The derivative of (7.6) with respect to \( L_k(\tilde{J}_k) \) leads to (7.5):

\[ \frac{\partial K(L^\top AL)}{\partial L_k(\tilde{J}_k)} = \frac{1}{n \det(L^\top AL)^{\frac{1}{2}}} \left( 2L_{kk}A_k(\tilde{J}_k) + 2A(\tilde{J}_k, \tilde{J}_k)L_k(\tilde{J}_k) \right)^\top = 0 \]

\[ \Leftrightarrow L_k(\tilde{J}_k) = -L_{kk}A_k(\tilde{J}_k, \tilde{J}_k)^{-1} A_k(\tilde{J}_k). \]

Inserting this into the expanded form (7.6), we obtain

\[ \frac{1}{n \det(A)^{\frac{n}{2}} \prod_{k=1}^n L_{kk}^{\frac{1}{2}}} \sum_{k=1}^n \left[ L_{kk}^2 A_{kk} + 2L_{kk}(-L_{kk}A_k(\tilde{J}_k, \tilde{J}_k)^{-1} A_k(\tilde{J}_k))^\top A_k(\tilde{J}_k) + \right. \]

\[ + \left. (\tilde{L}_{kk}A_k(\tilde{J}_k, \tilde{J}_k)^{-1} A_k(\tilde{J}_k))^\top A(\tilde{J}_k, \tilde{J}_k) L_k(-L_{kk}A_k(\tilde{J}_k, \tilde{J}_k)^{-1} A_k(\tilde{J}_k)) \right] \]

\[ = \sum_{k=1}^n \left[ L_{kk}^2 A_{kk} - A_k(\tilde{J}_k)^\top A(\tilde{J}_k, \tilde{J}_k)^{-1} A_k(\tilde{J}_k) \right] \]

\[ = \frac{\sum_{k=1}^n L_{kk}^2 S_{kk}}{n \det(A)^{\frac{n}{2}} \prod_{k=1}^n L_{kk}^{\frac{1}{2}}}. \]

(7.7)
We find (7.4) by deriving (7.7) with renamed indices with respect to $L_{kk}$:

\[
\Rightarrow \frac{\partial}{\partial L_{kk}} \left( \frac{\sum_{j=1}^{n} [L_{jj}^2S_{jj}]}{n \det(A)^{\frac{1}{2}} \prod_{j=1}^{n} L_{jj}^{\frac{1}{2}}} \right) = 0
\]

\[
\Leftrightarrow \frac{\partial}{\partial L_{kk}} \left( \frac{\sum_{j=1}^{n} [L_{jj}^2S_{jj}]}{L_{kk}^{\frac{1}{2}}} \right) = \frac{\partial}{\partial L_{kk}} \left( L_{kk}^{-\frac{3}{2}}S_{kk}^{\frac{1}{2}} + L_{kk}^{-\frac{3}{2}} \sum_{j \neq k} L_{jj}^2S_{jj} \right)
\]

\[
= (2 - \frac{2}{n})L_{kk}^{-\frac{1}{2}}S_{kk}^{\frac{1}{2}} - \frac{2}{n} L_{kk}^{-\frac{3}{2}} \sum_{j \neq k} L_{jj}^2S_{jj} \not= 0
\]

\[
\Leftrightarrow (2 - \frac{2}{n})L_{kk}^2S_{kk}^{\frac{1}{2}} - \frac{2}{n} \sum_{j \neq k} L_{jj}^2S_{jj} = 2L_{kk}^2S_{kk}^{\frac{1}{2}} - \frac{2}{n} \sum_{j=1}^{n} L_{jj}^2S_{jj} \not= 0
\]

\[
\Leftrightarrow L_{kk}^2S_{kk} = \frac{1}{n} \sum_{j=1}^{n} L_{jj}^2S_{jj}
\]

(7.8)

For all columns $k$, the right-hand side of (7.8) is the same. Hence there exists a $c > 0$ with $L_{kk}^2S_{kk} = c$ for all $k$. It is obvious that any choice of $c$ satisfies (7.8). Therefore, we choose $c := 1$ as this determination corresponds to a normalisation of the column $L_k$, see (7.14). This leads us to (7.4):

\[
L_{kk}^2S_{kk} = 1 \Leftrightarrow L_{kk} = \frac{1}{\sqrt{S_{kk}}}
\]

It is possible to show that the second derivative is positive definite. Therefore, this is a minimum. \qed

Notice that all columns of $L$ are independent and therefore can be computed in parallel.

### 7.2 FSPAI: Updating the Sparsity Pattern

Similar to the SPAI idea in Section 5, we want to update the sparsity pattern automatically to improve on a current pattern. We are interested in approximating the improvement factor of an entry $L_{jk}$. Consider $L_k$ as the FSAI solutions according to Section 7.2 for the sparsity pattern $J_k$. We keep all entries $L_k(J_k)$ fixed and minimize (7.1) according to the new possible entry $L_{jk}$. This results in an upper bound for the exact improvement, as we only modify $L_{jk}$ instead of all entries.

**Theorem 7.3.** By adding a new index $j$ to the current column pattern $J_k$, the new Kaporin functional $K_{new}$ is bounded by

\[
K_{new} \leq K_{old} \left( 1 - \frac{1}{n} \tau_{jk} \right)
\]

(7.9)
with
\[
\tau_{jk} := \frac{(A_j(J_k)^\top L_k(J_k))^2}{A_{jj}}. \quad (7.10)
\]

**Proof.** First, we have to find the minimizer of the new Kaporin functional with fixed \(L_k(J_k)\). As \(k\) is already in \(J_k\), the Kaporin denominator \(\prod_{k=1}^n L_{kk}^2\) is independent of \(L_{jk}\). Hence, we only have to derive the Kaporin numerator with respect to \(L_{jk}\):

\[
\frac{\partial}{\partial L_{jk}} \left[ \text{trace}(\left(L + e_j L_{jk} e_k^\top\right)^\top A(L + e_j L_{jk} e_k^\top)) \right]
= \frac{\partial}{\partial L_{jk}} \left[ \text{trace}(L^\top A L) + 2 L_{jk} \text{trace}(L^\top A e_j e_k^\top) + L_{jk}^2 A_{jj} \text{trace}(e_k e_k^\top) \right]
= \frac{\partial}{\partial L_{jk}} \left[ \text{trace}(L^\top A L) + 2 L_{jk} \text{trace}((L e_k)^\top A e_j) + L_{jk}^2 A_{jj} \right]
= 2 A_j^\top L_k + 2 L_{jk} A_{jj} \overset{!}{=} 0
\implies L_{jk} = -\frac{A_j^\top L_k}{A_{jj}}. \quad (7.12)
\]

As the second derivative \(\frac{\partial^2}{\partial^2 L_{jk}} = 2 A_{jj}\) is positive, this is a minimum.

Inserting (7.12) into (7.11) leads us to the minimum, which is an upper bound of \(K_{\text{new}}\) due to the way we constructed it. We also use the fact that the denominator \(\text{trace}(L^\top A L)\) is equal to \(n\) for any FSAI solution, see (7.15).

\[
K_{\text{new}} \leq \frac{\left[ \text{trace}(L^\top A L) + 2 L_{jk} A_j^\top L_k + L_{jk}^2 A_{jj} \right]}{n \det(L^\top A L)^{\frac{1}{2}}}
= \frac{n + 2(-\frac{A_j^\top L_k}{A_{jj}}) A_j^\top L_k + (-\frac{A_j^\top L_k}{A_{jj}})^2 A_{jj}}{n \det(L^\top A L)^{\frac{1}{2}}}
= \frac{1}{\det(L^\top A L)^{\frac{1}{2}}} \left[ 1 - \frac{1}{n} \left(\frac{A_j^\top L_k}{A_{jj}}\right)^2 \right] = K_{\text{old}} \left(1 - \frac{1}{n} \tau_{jk}\right)
\]

These \(\tau_{jk}\) will be our indication for the quality of a new entry \(L_{jk}\). Large \(\tau_{jk}\) indicate a large reduction of the Kaporin functional by adding the index \(j\).
Remarks:

- For all $\tau_{jk}$ we only have to solve one linear system $y_k = A(\tilde{J}_k, \tilde{J}_k)^{-1}A_k(\tilde{J}_k)$ instead of one for each index $j$.

- If all $\tau_{jk}$ are small enough we can expect a certain quality of the current preconditioner. Therefore, we will stop updating the pattern if all $\tau_{jk}$ are smaller than a tolerance $\epsilon$. This is the meaningful comparison, as $L_k$ is normalized (7.14) and $\tau_{jk}$ is independent of $n$ (7.25).

- To prevent $L$ from becoming too dense, we implement an upper bound $p_{\text{max}}$ for the size of the index set $J_k$.

- As $A$ and $L_k$ are sparse, we only have to compute $\tau_{jk}$ for a small set of indices $\mathcal{J} = \{j > k; A_j(\tilde{J}_k) \neq 0 \land j \notin \tilde{J}_k\}$.

- FSPAI can be initialized with any lower triangular pattern which includes the diagonal elements.

- Huckle suggests in [14] adding more than one index during each iteration. With this approach the quality of the preconditioner suffers slightly, but the computation of an FSPAI speeds up.

- The FSPAI remains unaffected by applying diagonal scalings $DAD$ with a diagonal matrix $D$. $L_k$ changes to $D^{-1}L_k$, but $\tau_{jk}$ remains unchanged.

Summarizing all steps we can formulate the following Algorithm 7.1 based on diagonal start pattern.

**Algorithm 7.1 FSPAI using diagonal start pattern**

```plaintext
for $k = 1 \ldots n$ do
    $\tilde{J}_k = \{k\}$, $L_{kk} = \frac{1}{\sqrt{A_{kk}}}$
    while $|\tilde{J}_k| + 1 < p_{\text{max}}$ do
        $\mathcal{J}$ set of potential candidates
        for $j \in \mathcal{J}$ do
            $\tau_{jk} = \frac{(A_j(\tilde{J}_k)^\top L_k(\tilde{J}_k))^2}{A_{jj}}$
        end for
        if $\max_j \tau_{jk} < \epsilon$ then
            break
        end if
        $\tilde{J}_k = \tilde{J}_k \cup \{l\}$ with largest $\tau_{lk}$
        $y_k = A(\tilde{J}_k, \tilde{J}_k)^{-1}A_k(\tilde{J}_k)$
        $L_{kk} = 1/(A_{kk} - A_k(\tilde{J}_k)^\top y_k)^{1/2}$
        $L_k(\tilde{J}_k) = -L_{kk}y_k$
    end while
end for
```

28
7.3 Theoretical Properties

In the following part we introduce theoretical properties of FSPAI. For example, we can anticipate that there is always an index with \( \tau_{jk} > 0 \), if the preconditioner is not yet perfect, similar to 5.3. We start with properties helping us to handle the FSPAI method. Let \( L_k \) be the FSAI solution of Theorem 7.2 for a column pattern \( J_k \).

**Lemma 7.4.** The FSAI solution \( L_k \) satisfies

\[
A(J_k,J_k)L_k(J_k) = \begin{pmatrix} S_{kk} & L_{kk} \\ \vec{0} & 0 \end{pmatrix}.
\] (7.13)

**Proof.**

\[
A(J_k,J_k)L_k(J_k) = \begin{pmatrix} A_{kk} & A_k(J_k)^\top \\ A_k(J_k) & A(J_k,J_k) \end{pmatrix} \begin{pmatrix} L_{kk} \\ L_k(J_k) \end{pmatrix}
\]

\[
= \begin{pmatrix} A_{kk}L_{kk} - A_k(J_k)^\top A(J_k,J_k)^{-1}A_k(J_k)L_{kk} \\ A_k(J_k)L_{kk} - A(J_k,J_k)A_k(J_k)A_k(J_k)L_{kk} \end{pmatrix}
\]

\[
= \begin{pmatrix} L_{kk} \\ L_k(J_k)L_{kk} - A_k(J_k)L_{kk} \end{pmatrix}
\]

\[
= \begin{pmatrix} S_{kk} & L_{kk} \\ \vec{0} & 0 \end{pmatrix} \in \mathbb{R}^{J_k}
\]

**Theorem 7.5.** A column \( L_k \) of a FSAI solution (Theorem 7.2) is normalized with respect to the weighted Euclidean norm, i.e.,

\[
\|L_k\|_A = \sqrt{L_k^\top AL_k} = 1.
\] (7.14)

**Proof.**

\[
L_k^\top AL_k = L_k(J_k)^\top A(J_k,J_k)L_k(J_k) = L_k(J_k)^\top S_{kk}L_{kk}^{(7.4)} = 1
\]

Hence, we obtain \( \|L_k\|_A = \sqrt{L_k^\top AL_k} = 1. \)

**Corollary 7.6.** For any FSAI solution (Theorem 7.2), the numerator of the Kaporin functional (7.1) satisfies

\[
\frac{1}{n} \text{tr}(L^\top AL) = 1.
\] (7.15)
Proof.

\[ \frac{1}{n} \text{trace}(L^T AL) = \frac{1}{n} \sum_{k=1}^{n} L_k^T A L_k = \frac{1}{n} \sum_{k=1}^{n} 1 = 1 \]

\[ \square \]

**Theorem 7.7.** If all \( \tau_{jk} \) are equal to zero, the FSPAI solution is optimal regarding the minimization of \( K \) (7.1), i.e.

\[ \forall k : \tau_{jk} = 0 \forall j \in \{k+1, \ldots, n\} - \tilde{J}_k \text{ it follows } L^T AL = I. \quad (7.16) \]

To be more precise, if all \( \tau_{jk} \) are equal to zero for one column \( k \), \( L_k \) is the \( k \)th column of the inverse of the Cholesky factor of \( A \).

**Proof.** Let \( \mathcal{I}_k = \{k + 1, \ldots, n\} \). The diagonal elements of \( L^T AL \) are equal to one according to (7.14). Without loss of generality, we assume \( i > k \) for the non-diagonal element \( L_i^T A L_k \). Therefore, \( J_i \subseteq \mathcal{I}_k \).

To proof \( L_i^T A L_k = 0 \), we need the \( \tau \) assumption.

\[ \forall j \in \mathcal{I}_k - \tilde{J}_k : \tau_{jk} = 0 \Leftrightarrow A_j^T L_k = 0 \]

Together with (7.13) we conclude \( A(\mathcal{I}_k,.)L_k = \vec{0} \), and with \( J_i \subseteq \mathcal{I}_k \)

\[ L_i^T A L_k = L_i(\mathcal{I}_k)^T A(\mathcal{I}_k,.)L_k = 0 \]

Therefore all non-diagonal elements are equal to zero. \( \square \)

**Corollary 7.8.** If the sparsity pattern \( J \) is full, the FSAI solution \( L \) is optimal regarding the minimization of \( K \) (7.1), i.e., it satisfies \( L^T AL = I \).

**Proof.** This follows immediately from (7.16) and \( \{k + 1, \ldots, n\} - \tilde{J}_k = \emptyset \). \( \square \)

**Corollary 7.9.** Without any stopping criterion, the FSPAI algorithm would find the optimal \( L \) regarding the minimization of \( K \) (7.1), i.e., \( L^T AL = I \).

**Proof.** This algorithm always adds a new index to the sparsity pattern, if there is any \( \tau_{jk} > 0 \), i.e., it only stops if all \( \tau_{jk} \) are equal to zero. Now we can apply (7.16). \( \square \)

### 7.4 Updating the Kaporin Functional

We are interested in properties that assure a certain preconditioner quality. Here for example we are looking for some boundary for the Kaporin functional (7.1) in the case \( \forall j : \tau_{jk} < \epsilon \). Unfortunately that does not apply for the Euclidean norm. In the following we show an example with arbitrary small \( \tau \) and arbitrary large Kaporin value: Consider the matrix \( A = L_A^T L_A \) with

\[ L_A = \begin{pmatrix}
    \epsilon & 0 & 0 \\
    \frac{\epsilon}{\sqrt{\epsilon}} & \frac{\sqrt{\epsilon}}{\sqrt{1 - \epsilon^2}} & 0 \\
    \frac{\sqrt{\epsilon}}{\sqrt{1 - \epsilon^2}} & 0 & 1
\end{pmatrix} \quad (7.17) \]
and $\mathcal{J}_1 = \{1, 3\}, \mathcal{J}_2 = \{2, 3\}, \mathcal{J}_3 = \{3\}$. Then we get
\[
\tau_{21} = \frac{\epsilon^2}{\epsilon^2 + \epsilon} < \epsilon \xrightarrow{\epsilon \to 0} 0 \text{ and } K = \left(1 + \frac{1}{\epsilon}\right)^\frac{3}{2} \xrightarrow{\epsilon \to 0} \infty.
\]

In such cases, the FSPAII algorithm would break with an arbitrary bad solution regarding the minimization of the Kaporin functional.

Hence we are interested in finding the exact Kaporin reduction value for adding an index $j$. We are looking for a property that causes the weak reduction approximation of $\tau_{jk}$. If we want to compare the Kaporin functional (7.1) value before and after the update process, we only have to consider the modification of the diagonal element $L_k k$. According to (7.15), the numerator of $K$ remains constant, and the denominator only depends on the diagonal parts.

The first step towards our Kaporin update is the update of the Cholesky factors of $A(\tilde{J}_k, \tilde{J}_k)$. In each iteration we solve a linear system with the matrix $A(\tilde{J}_k, \tilde{J}_k)$. As $A$ is s.p.d. the Cholesky factorization becomes a possible factorization method. After adding a new index $j$ to $\tilde{J}_k$, we don’t need to compute the Cholesky factors of $A(\tilde{J}_k \cup \{j\}, \tilde{J}_k \cup \{j\})$ from scratch. We can update the decomposition according to (7.18).

**Lemma 7.10 (Cholesky Update).** Let us assume the decomposition $A(\tilde{J}_k, \tilde{J}_k) = P^\top L^\top LP$ is known with permutation matrix $P$ and lower triangular $L$. Adding a new index $j$ to $\tilde{J}_k$ we can update the decomposition via
\[
A(\tilde{J}_k \cup \{j\}, \tilde{J}_k \cup \{j\}) = \left[\begin{array}{cc} 1 & 0 \\ 0 & P \end{array}\right] \tilde{P}^\top \tilde{L}^\top \tilde{L} \left[\begin{array}{cc} 1 & 0 \\ 0 & P \end{array}\right] \tilde{P} \tag{7.18}
\]
with $\tilde{P}$ being the permutation matrix that permutes the index $j$ to the first position, e.g., $PA_j(\tilde{J}_k \cup \{j\}) = \left(\begin{array}{c} A_{jj} \\ A_j(\tilde{J}_k) \end{array}\right)$, and $\tilde{L} = \left(\begin{array}{c} L_1 \\ \sqrt{S_{jj}} \end{array}\right)$ with
\[
\tilde{L}_3 = L, \\
\tilde{L}_2 = L_{3}^{-\top} PA_j(\tilde{J}_k), \text{ and } \\
\tilde{L}_1 = \sqrt{S_{jj}} = \sqrt{A_{jj} - \tilde{L}_2^\top \tilde{L}_2}.
\]

**Proof.**
\[
A(\tilde{J}_k \cup \{j\}, \tilde{J}_k \cup \{j\}) = \tilde{P}^\top \left(\begin{array}{cc} A_{jj} & A_j(\tilde{J}_k)^\top \\ A_j(\tilde{J}_k) & A(\tilde{J}_k, \tilde{J}_k) \end{array}\right) \tilde{P}
\]
\[
= \tilde{P}^\top \left(\begin{array}{cc} A_{jj} & A_j(\tilde{J}_k)^\top \\ A_j(\tilde{J}_k) & P^\top L^\top LP \end{array}\right) \tilde{P}
\]
\[
= \tilde{P}^\top \left(\begin{array}{cc} 1 & 0 \\ 0 & P^\top \end{array}\right) \left(\begin{array}{cc} A_{jj} & A_j(\tilde{J}_k)^\top P^\top \\ PA_j(\tilde{J}_k) & L^\top L \end{array}\right) \left(\begin{array}{cc} 1 & 0 \\ 0 & P \end{array}\right) \tilde{P}
\]
\[
= \left[\begin{array}{cc} 1 & 0 \\ 0 & P \end{array}\right] \tilde{P}^\top \left(\begin{array}{cc} A_{jj} & A_j(\tilde{J}_k)^\top P^\top \\ PA_j(\tilde{J}_k) & L^\top L \end{array}\right) \left[\begin{array}{cc} 1 & 0 \\ 0 & P \end{array}\right] \tilde{P}.
\]

31
We finish the decomposition by using the ansatz matrix \( \tilde{L} = \begin{pmatrix} \tilde{L}_1 & 0 \\ \tilde{L}_2 & \tilde{L}_3 \end{pmatrix} \).

We obtain

\[
\begin{pmatrix}
A_{jj} & A_j(\tilde{J}_k)^\top P^\top \\
PA_j(\tilde{J}_k) & \tilde{L}_j^\top \tilde{L}_j
\end{pmatrix} = \begin{pmatrix} 1 \\ \tilde{L}_j^\top \tilde{L}_j \end{pmatrix} = \begin{pmatrix} \tilde{L}_1^2 + \tilde{L}_2^\top \tilde{L}_2 & \tilde{L}_2^\top \tilde{L}_3 \\ \tilde{L}_3^\top \tilde{L}_3 \end{pmatrix}
\]

which can be easily solved:

\[
\begin{align*}
&\Rightarrow \tilde{L}_j^\top \tilde{L}_j = \tilde{L}_3^\top \tilde{L}_3 \\
&\Rightarrow PA_j(\tilde{J}_k) = \tilde{L}_3^\top \tilde{L}_2 \\
&\Rightarrow A_{jj} = \tilde{L}_1^2 + \tilde{L}_2^\top \tilde{L}_2
\end{align*}
\]

Next we want to update the diagonal entry \( L_{kk} = \frac{1}{\sqrt{S_{kk}}} \). As we need all entries \( S_{ij} \) of \( S \) in Section 8, we update the whole Schur complement defined in (7.2):

**Theorem 7.11** (Schur complement update). Adding a new index \( j \) to \( \tilde{J}_k \) modifies the Schur complement (7.2) in the following way: consider \( \mathcal{I}_k = \{k, \ldots, n\} - \tilde{J}_k \) and the updated sets \( \mathcal{I}'_k = \mathcal{I}_k - \{j\} \) and \( \mathcal{J}'_k = \mathcal{J}_k \cup \{j\} \). Then the new Schur complement satisfies

\[
S_{\text{new}} := A(\mathcal{I}'_k, \mathcal{I}'_k) - A(\mathcal{J}'_k, \mathcal{I}'_k)^\top A(\mathcal{J}'_k, \mathcal{J}'_k)^{-1} A(\mathcal{J}'_k, \mathcal{I}'_k) = S(\mathcal{I}_k, \mathcal{I}_k) - S(\mathcal{I}_k, \mathcal{J}_k)^\top S_{jj}^{-1} S(\mathcal{J}_k, \mathcal{I}_k).
\]

(7.19)

**Proof.** Assume that we know the Cholesky decomposition \( A(\mathcal{J}_k, \mathcal{J}_k) = \tilde{L}_j^\top \tilde{L}_j \). To update \( S \) we use (7.18):

\[
S_{\text{new}} = A(\mathcal{I}_k, \mathcal{I}_k) - A(\mathcal{J}_k, \mathcal{I}_k)^\top A(\mathcal{J}_k, \mathcal{J}_k)^{-1} A(\mathcal{J}_k, \mathcal{I}_k)
\]

\[
= A(\mathcal{I}_k, \mathcal{I}_k) - A(\mathcal{J}_k, \mathcal{I}_k)^\top \left( \hat{P}^\top \tilde{L}_j^\top \hat{L} \hat{P} \right)^{-1} A(\mathcal{J}_k, \mathcal{I}_k)
\]

\[
= A(\mathcal{I}_k, \mathcal{I}_k) - A(\mathcal{J}_k, \mathcal{I}_k)^\top \hat{P}^\top \tilde{L}_j^\top \hat{L}^{-1} \left( \hat{L}^\top \hat{P} A(\mathcal{J}_k, \mathcal{I}_k) \right)
\]

\[
= Y = \begin{pmatrix} Y_1 \\ Y_2 \end{pmatrix}
\]

We solve the system \( \tilde{L}_j^\top Y = \hat{P} A(\mathcal{J}_k, \mathcal{I}_k) = \begin{pmatrix} A(j, \mathcal{I}_k) \\ A(j, \mathcal{J}_k) \end{pmatrix} \) to obtain \( Y \):

32
\[
\begin{pmatrix}
\sqrt{S_{jj}} & A_j(\tilde{J}_k)^\top \hat{L}^{-1} \\
0 & \hat{L}^\top
\end{pmatrix}
\begin{pmatrix}
Y_1 \\
Y_2
\end{pmatrix}
= \begin{pmatrix}
A(j, I'_k) \\
A(\tilde{J}_k, I'_k)
\end{pmatrix}
\]
\[
\Rightarrow Y_2 = \hat{L}^{-\top} A(\tilde{J}_k, I'_k)
\]
\[
Y_1 = \frac{1}{\sqrt{S_{jj}}} \left( A(j, I'_k) - A_j(\tilde{J}_k)^\top \hat{L}^{-1} \hat{L}^{-\top}(\tilde{J}_k, I'_k) \right) = \frac{1}{\sqrt{S_{jj}}} S(j, I'_k)
\]
(7.21)

Inserting (7.21) into (7.20) leads to the following update formula:

\[
S_{\text{new}} = A(I'_k, I'_k) - Y^\top Y
= A(I'_k, I'_k) - A(\tilde{J}_k, I'_k)^\top \hat{L}^{-1} \hat{L}^{-\top} A(\tilde{J}_k, I'_k) - S(j, I'_k)^\top \frac{1}{S_{jj}} S(j, I'_k)
= S(I'_k, I'_k) - S(j, I'_k)^\top S(j, j)^{-1} S(j, I'_k)
\]

This Schur complement is updated by a Schur complement. Considering the addition of a set of indices instead of one index \(j\), the analogue form can be derived. Now, it is possible to state the following Theorem.

**Theorem 7.12 (Kaporin update).** Consider adding an index \(j\) to the current sparsity pattern \(J_k\). Then the Kaporin functional (7.1) is reduced by the factor

\[
K_{\text{new}} = K_{\text{old}} (1 - \mu_{jk})^{\frac{1}{2}} 
\]
(7.22)

with

\[
\mu_{jk} := \frac{S_{jk}^2}{S_{jj} S_{kk}} 
\]
(7.23)

**Proof.** The Kaporin functional of a FSAI solution \(L\) (7.2) using (7.15) can be written:

\[
K_{\text{old}} = \frac{1}{\det(L^\top A L)^{\frac{1}{2}}} = \frac{1}{\det(A)^{\frac{1}{2}} \prod_{j=1}^{n} L_{jj}^{\frac{2}{3}}} = \frac{S_{kk}^{\frac{1}{2}}}{\det(A)^{\frac{1}{2}} \prod_{j \neq k} L_{jj}^{\frac{2}{3}}}
\]

With updating \(S_{kk}\) from (7.19) via

\[
(S_{kk})_{\text{new}} = S_{kk} - S_{jk}^2 S_{jj}^{-1} S_{jk} = S_{kk} \left( 1 - \frac{S_{jk}^2}{S_{jj} S_{kk}} \right)
\]
équation (7.22) follows immediately. \(\square\)

33
This result shows the exact Kapoorin reduction for augmenting the sparsity pattern. Hence $\mu_{jk}$ (7.23) is a better indication than $\tau_{jk}$ (7.10) for the quality of adding an index $j$. This leads us to the Exact FSPAI (EFSPAI) algorithm in Chapter 8. But first we want to compare $\mu_{jk}$ and $\tau_{jk}$.

For that purpose we need $\tau_{jk}$ in a similar representation:

**Lemma 7.13.** The $\tau_{jk}$ defined in (7.10) can be written as

$$\tau_{jk} = \frac{S_{jk}^2}{A_{jj}S_{kk}}$$  \hspace{1cm} (7.24)

**Proof.**

$$\tau_{jk} = \frac{(A_j(\tilde{J}_k)^\top L_k(\tilde{J}_k))^2}{A_{jj}}$$

$$= \frac{1}{A_{jj}} \left[ \left( A_{kj} \right)^\top \left( -L_{kk}A(\tilde{J}_k, \tilde{J}_k)^{-1}A_k(\tilde{J}_k) \right) \right]^2$$

$$= \frac{L_{kk}^2}{A_{jj}} \left[ A_{jk} - A_j(\tilde{J}_k)^\top A(\tilde{J}_k, \tilde{J}_k)^{-1}A_k(\tilde{J}_k) \right]^2$$

$$= \frac{S_{jk}^2}{A_{jj}S_{kk}}$$

**Theorem 7.14.** $\tau_{jk}$ (7.24) and $\mu_{jk}$ (7.23) satisfy

$$0 \leq \tau_{jk} \leq \mu_{jk} < 1.$$  \hspace{1cm} (7.25)

$\tau_{jk}$ is a less accurate approximation of $\mu_{jk}$, if $\frac{A_{jj}}{A_j(\tilde{J}_k)^\top A(\tilde{J}_k, \tilde{J}_k)^{-1}A_j(\tilde{J}_k)}$ is close to one.

**Proof.** We have the representations $\tau_{jk} = \frac{S_{jk}^2}{A_{jj}S_{kk}}$ and $\mu_{jk} = \frac{S_{jk}^2}{S_{jj}S_{kk}}$.

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

$$S_{jj} = A_{jj} - A_j(\tilde{J}_k)^\top A(\tilde{J}_k, \tilde{J}_k)^{-1}A_j(\tilde{J}_k) \leq A_{jj}$$

using the positive definiteness of $A(\tilde{J}_k, \tilde{J}_k)^{-1}$.

The third inequality is concluded from applying (B.9) on the s.p.d. matrix $S$ (7.2). For the matrix example in (7.17), the factor $\frac{A_{jj}}{A_j(\tilde{J}_k)^\top A(\tilde{J}_k, \tilde{J}_k)^{-1}A_j(\tilde{J}_k)}$ is $\frac{1}{1-\epsilon^2-\epsilon}$, and close to one for small $\epsilon$.

The previous theorem directly yields an estimation of the Kapoorin functional.
Corollary 7.15.

\[ K_{new} = K_{old} (1 - \mu_{jk})^\frac{1}{n} \leq K_{old} (1 - \tau_{jk})^\frac{1}{n} \leq K_{old} \left( 1 - \frac{1}{n} \tau_{jk} \right) \tag{7.26} \]

Proof. The first inequality is trivial with (7.25).

The right-hand side is the estimation concluded from the derivation of \( \tau_{jk} \) (7.9). We get the last inequality applying Bernoulli’s inequality (B.2) on \( -\frac{1}{n} \tau_{jk} \):

\[ \left( 1 - \frac{1}{n} \tau_{jk} \right)^n \geq 1 - \tau_{jk} \Leftrightarrow 1 - \frac{1}{n} \tau_{jk} \geq (1 - \tau_{jk})^\frac{1}{n} \]

\[ \square \]

7.5 Numerical Results

This section examines the qualities of FSPAI in a few numerical tests. All tests in this thesis were done in MATLAB on a machine with Intel Pentium D 2.8 GHz, x86_64 GNU/Linux, Linux 2.6.32 – 33 – generic. All times measured were determined as the mean value of several tests to reduce the variance. The s.p.d. matrices used can be found in the Matrix Market [1] or in the Tim Davis matrix collection [9].

bcdstk14 structural problem, Roof of the Omni Coliseum, Atlanta, \( n = 1806 \), \( nnz = 63454 \), cond = 1.3e+10

nos3 structural problem, Finite element approximation to biharmonic operator on a rectangular plate, \( n = 960 \), \( nnz = 15844 \), cond = 7.3e+4

nos7 2D/3D problem, Poisson’s equation in unit cube, \( n = 729 \), \( nnz = 4617 \), cond = 4.0e+9

lund_a structural problem, Lund eigenvalue problem, \( n = 147 \), \( nnz = 2449 \), cond = 9.0e+5

msc01440 Test matrix from msc/nastran cylf8.out2, \( n = 1440 \), \( nnz = 44998 \), cond = 7.0e+6

The chosen iterative method is the PCG with starting vector \( x^{(0)} = 0 \). We stop, if the relative residual is below 1e-8. We apply it to the preconditioned linear systems and the unit vector \( e_1 \) as the right-hand side. There are no time measurements of PCG, as the MATLAB matrix-vector product is too fast compared to the operations in FSPAI, that the FSPAI solution just with diagonal pattern would always be the best preconditioner, which is not the case considering the C++ implementation [27].

As one main interest in the FSPAI implementation is the comparison to EFSPAI in the next chapter, we implement FSPAI only adding one index in each
update and starting with diagonal pattern. Therefore we can update the set of potential new candidates \( \mathcal{J} = \{ j > k; A_{ij} \neq 0 \land j \notin \mathcal{J}_k \} \) very efficiently.

\[
\mathcal{J}_{\text{new}} = \left( \mathcal{J} \cup \{ j > k; A_{ij} \neq 0 \land j \notin \mathcal{J}_k \} \right) - \{ l \}
\] (7.27)

In MATLAB this can be written:

```matlab
Jhat = setdiff(union(Jhat,setdiff(find(A(k+1:n,l))'+k,Jktilde)),l);
```
with \( l \) the index previously added.

<table>
<thead>
<tr>
<th>matrix</th>
<th>( \epsilon )</th>
<th>time</th>
<th>nnz</th>
<th>cond</th>
<th>Kap</th>
<th>iter</th>
<th>var</th>
</tr>
</thead>
<tbody>
<tr>
<td>bcsstk14</td>
<td>0.5</td>
<td>2.44</td>
<td>1870</td>
<td>5.383e+3</td>
<td>1.341</td>
<td>266</td>
<td>0.367</td>
</tr>
<tr>
<td></td>
<td>0.4</td>
<td>2.53</td>
<td>1902</td>
<td>5.277e+3</td>
<td>1.327</td>
<td>259</td>
<td>0.347</td>
</tr>
<tr>
<td></td>
<td>0.3</td>
<td>2.75</td>
<td>1988</td>
<td>7.017e+3</td>
<td>1.301</td>
<td>243</td>
<td>0.325</td>
</tr>
<tr>
<td></td>
<td>0.2</td>
<td>3.52</td>
<td>2243</td>
<td>5.943e+3</td>
<td>1.251</td>
<td>211</td>
<td>0.288</td>
</tr>
<tr>
<td></td>
<td>0.1</td>
<td>5.31</td>
<td>2874</td>
<td>4.402e+3</td>
<td>1.183</td>
<td>158</td>
<td>0.255</td>
</tr>
<tr>
<td></td>
<td>0.05</td>
<td>8.12</td>
<td>3763</td>
<td>3.772e+3</td>
<td>1.138</td>
<td>140</td>
<td>0.175</td>
</tr>
<tr>
<td>msc01440</td>
<td>0.5</td>
<td>1.66</td>
<td>1440</td>
<td>3.838e+4</td>
<td>Inf</td>
<td>991</td>
<td>0.711</td>
</tr>
<tr>
<td></td>
<td>0.4</td>
<td>1.75</td>
<td>1477</td>
<td>3.863e+4</td>
<td>Inf</td>
<td>983</td>
<td>0.689</td>
</tr>
<tr>
<td></td>
<td>0.3</td>
<td>1.88</td>
<td>1531</td>
<td>3.840e+4</td>
<td>Inf</td>
<td>977</td>
<td>0.657</td>
</tr>
<tr>
<td></td>
<td>0.2</td>
<td>3.15</td>
<td>1995</td>
<td>3.875e+4</td>
<td>Inf</td>
<td>889</td>
<td>0.523</td>
</tr>
<tr>
<td></td>
<td>0.1</td>
<td>6.86</td>
<td>3541</td>
<td>1.798e+4</td>
<td>1.458</td>
<td>535</td>
<td>0.348</td>
</tr>
<tr>
<td></td>
<td>0.05</td>
<td>8.32</td>
<td>4265</td>
<td>1.780e+4</td>
<td>1.399</td>
<td>494</td>
<td>0.280</td>
</tr>
</tbody>
</table>

Table 1: FSPAI with different \( \epsilon, p_{\text{max}} = 5 \)

Tables 1 and 2 show, representatively for all tested matrices, the qualities of the FSPAI applied on bcsstk14 and msc01440. nnz is here the number of non-zero entries in the FSPAI solution \( L \), cond the condition number of the preconditioned matrix \( L^\top AL \), Kap the Kaporin function value of \( L^\top AL \), iter the number of iterations of PCG and var the variance of the eigenvalues of \( L^\top AL \) around the expected value 1 (see (7.15)). The entries with value Inf occur when the determinant is estimated to zero.

Table 1 shows FSPAI tests for a constant maximum number of entries per column \( p_{\text{max}} \) and a variable tolerance \( \epsilon \). Analogous in Table 2 for a constant \( \epsilon \) and variable \( p_{\text{max}} \). For both matrices, CG doesn’t converge. You can see that FSPAI with only a few entries can lead to an effective preconditioner. Generally all quality criteria for preconditioners reduce with lower \( \epsilon \) respectively \( p_{\text{max}} \). The eigenvalues are clustered around 1, which can be seen in Figure 1. Notice in Table 1 the condition number rises once with lowering \( \epsilon \). The Kaporin functional is a better convergence indication than the condition number for the CG.
Table 2: FSPAII with different $p_{max}$, $\epsilon = 0.01$

For a small number of indices the FSPAII is almost perfect. For example for the matrix lund_a, we found the perfect pattern for $p_{max} = 3$. For each column $k$ we choose the pattern $\tilde{J}_k$ that minimizes the Kaporin functional for all $\tilde{J}_k \subset \{k + 1, \ldots, n\}$ with $|\tilde{J}_k| = 2$. This solution is only differs from the FSPAII solution with $\epsilon = 0$ and $p_{max} = 3$ in one index.

Figure 2 shows that the pattern of the lower left part of $A$ can be a good choice for a starting pattern for FSPAII. FSPAII with $\epsilon = 0.1$ and $p_{max}$ applied on msc01440 only chooses indices corresponding to nonzero entries of $A$. 

<table>
<thead>
<tr>
<th>matrix</th>
<th>$p_{max}$</th>
<th>time</th>
<th>nnz</th>
<th>cond</th>
<th>Kap</th>
<th>iter</th>
<th>var</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>0.09</td>
<td>1806</td>
<td>9.916e+3</td>
<td>1.510</td>
<td>457</td>
<td>0.430</td>
</tr>
<tr>
<td>bcsstk14</td>
<td>2</td>
<td>2.37</td>
<td>2581</td>
<td>4.558e+3</td>
<td>1.220</td>
<td>189</td>
<td>0.256</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>4.57</td>
<td>2807</td>
<td>4.479e+3</td>
<td>1.190</td>
<td>171</td>
<td>0.232</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>5.26</td>
<td>2861</td>
<td>4.436e+3</td>
<td>1.184</td>
<td>167</td>
<td>0.227</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>5.44</td>
<td>2874</td>
<td>4.402e+3</td>
<td>1.183</td>
<td>158</td>
<td>0.225</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.07</td>
<td>1440</td>
<td>3.840e+4</td>
<td>Inf</td>
<td>991</td>
<td>0.711</td>
</tr>
<tr>
<td>msc01440</td>
<td>2</td>
<td>1.73</td>
<td>2292</td>
<td>4.947e+4</td>
<td>Inf</td>
<td>861</td>
<td>0.540</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>3.91</td>
<td>2940</td>
<td>4.568e+4</td>
<td>1.567</td>
<td>775</td>
<td>0.431</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>5.78</td>
<td>3277</td>
<td>4.129e+4</td>
<td>1.507</td>
<td>717</td>
<td>0.383</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>6.87</td>
<td>3541</td>
<td>1.702e+4</td>
<td>1.458</td>
<td>535</td>
<td>0.348</td>
</tr>
</tbody>
</table>
Figure 1: bcsstk14: eigenvalue histograms of $L^TAL$ with FSPAI

Figure 2: msc01440: sparsity structure
8 Exact FSPAI

In the last chapter,

\[ \mu_{jk} = \frac{S_{jk}^2}{S_{jj}S_{kk}} \quad (8.1) \]

was found in (7.23) to be the exact improvement value for augmenting the current index set \( J_k \) by \( j \). The idea to choose this criterion for the pattern update instead of \( \tau_{jk} \) is obvious. We can expect a sparsity pattern to be chosen that leads to a better preconditioner, but at the cost of more computation to be done. We call this algorithm Exact Factorized Sparse Approximate Inverse (EFSPAI).

**Algorithm 8.1 EFSPAI**

\[
\text{for } k = 1 \ldots n \text{ do}
\]

\[
\tilde{J}_k = \emptyset
\]

\[
\text{while } |\tilde{J}_k| + 1 < p_{\text{max}} \text{ do}
\]

\[
\text{for } j \in \tilde{J} \text{ do}
\]

\[
\mu_{jk} = \frac{S_{jk}^2}{S_{jj}S_{kk}}
\]

\[
\text{end for}
\]

\[
\text{if } \max_j \mu_{jk} < \epsilon \text{ then}
\]

\[
\text{break}
\]

\[
\text{end if}
\]

\[
\tilde{J}_k = \tilde{J}_k \cup \{l\} \text{ with largest } \mu_{lk}
\]

\[
\text{end while}
\]

\[
y_k = A(\tilde{J}_k, J_k)^{-1} A_k(\tilde{J}_k)
\]

\[
L_{kk} = \frac{1}{A_{kk} - A_k(\tilde{J}_k)^{\top} y_k}^{1/2}
\]

\[
L_k(\tilde{J}_k) = -L_{kk} y_k
\]

\[
\text{end for}
\]

We can reduce the number of calculated \( \mu \)'s to the same index set

\[ \tilde{J} = \{ j > k; A_j(\tilde{J}_k) \neq 0 \land j \notin \tilde{J}_k \} \]

similar to Section 7.2, as \( \tau_{jk} = 0 \Leftrightarrow \mu_{jk} = 0 \) holds. We stop the construction of the current column, if all \( \mu_{jk} \) are lower than a given tolerance \( \epsilon \), or if the number of entries reaches the give limit \( p_{\text{max}} \).

Considering the \( S_{jj} \) in the denominator in (8.1), we would have to solve a linear system \( A(\tilde{J}_k, \tilde{J}_k)^{-1} A_j(\tilde{J}_k) \) for each possible candidate \( j \). This would be too time-consuming. The update idea (7.19) helps us finding an efficient way to compute \( \mu_{jk} \). For the previously added index \( l \), the Schur complement can be updated via

\[ S_{\text{new}} = S(I_k, I_k) - S(l, I_k)^{\top} \frac{1}{S_{ll}} S(l, I_k) \]

\[ (8.2) \]
with \( I_k := \{k, \ldots, n\} - \tilde{J}_k - \{l\} \). If we start with diagonal pattern, i.e., \( \tilde{J}_k = \emptyset \), we can initialize \( S \) with \( A \). This can be seen in the definition of \( S \) (7.2).

Furthermore, \( S_{ij} \) only needs to be updated if \( i, j \in \tilde{J} \cup \{k\} \), because we only update \( S(\mathcal{I}_k, \mathcal{I}_k) \) and \( S_{ij} \) is equal to \( A_{ij} \) for \( i, j \) with \( A_i(\tilde{J}_k) = \vec{0} \) or \( A_i(\tilde{J}_k) = \vec{0} \). Therefore, we implement EFSPAI only with starting diagonal pattern and adding only one index per update.

Note, that realizations of Approximate Inverse methods using exact pattern update have been considered by Gould and Scott [12] for the general case in connection with SpAI. Huckle [14] found the best next index using multivariate minimization for FSPAI. We present three possible implementations of EFSPAI that exert the update idea in different ways.

### 8.1 EFSPAI-Level 1

For \( \mu_{jk} (8.1) \), \( S_{jk} \) and \( S_{kk} \) can be computed similarly to the FSPAI implementation. We only want to update \( S_{jj} \). Let \( s \) be the array containing all \( S_{jj} \) for the sparsity pattern \( \tilde{J}_k \). Consider \( l \) as the index previously added to \( \tilde{J}_k \). Using (8.2), we can update \( S_{jj} \) by

\[
S_{jj} = S_{jj} - \frac{S^2_{jl}}{S_{ll}}. \tag{8.3}
\]

For all \( j \) and \( S_{jl} = A_{jl} - A_j(\tilde{J}_k)^\top A(\tilde{J}_k, \tilde{J}_k)^{-1} A_l(\tilde{J}_k) \), we only have to solve one linear system \( y_l = A(\tilde{J}_k, \tilde{J}_k)^{-1} A_l(\tilde{J}_k) \).

We can initialize the array \( s \) with the diagonal elements of \( A \), i.e., \( \text{diag}(A) \), and update only the array entries \( \tilde{J} \). This yields Algorithm 8.2.

### 8.2 EFSPAI-Level 2

Now we enhance the updating idea of level 1. In addition to \( S_{jj} \) we also update \( S_{jk} \) using (8.2)

\[
S_{jk} = S_{jk} - \frac{S_{jl} S_{lk}}{S_{ll}}. \tag{8.4}
\]

We define two arrays \( s \) and \( s_k \) with \( s[j] = S_{jj} \) and \( s_k[j] = S_{jk} \), initialized with \( s = \text{diag}(A) \) and \( s_k = A_k \). This yields Algorithm 8.3.

### 8.3 EFSPAI-Level 3

In EFSPAI-level 2, the only entry in \( S \) we have to compute without any update is \( S_{jl} \). If we want to omit the solution of any linear system, we have to update the entire matrix \( S \) (8.2). Although many entries have to be considered, it can be written very efficiently using vector products. This yields the EFSPAI-level 3 Algorithm 8.4.
Algorithm 8.2 EFSPAI-level 1

for $k = 1 \ldots n$ do
  $s = \text{diag}(A)$, $\mathcal{J}_k = \emptyset$, initialize $\hat{\mathcal{J}}$
  while true do
    $y_k = A(\hat{\mathcal{J}}_k, \mathcal{J}_k)^{-1} A_k(\mathcal{J}_k)$
    for $j \in \hat{\mathcal{J}}$ do
      $S_{jk} = A_{jk} - A_j(\hat{\mathcal{J}}_k)^\top y_k$
      $\mu_{jk} = \frac{s_{jk}^2}{\|s_j\|_2}$
    end for
    if $\max_j \mu_{jk} < \epsilon$ then
      break
    end if
    $l = \text{index with largest } \mu_{lk}$
    $(\hat{\mathcal{J}}_k)_{\text{old}} = \hat{\mathcal{J}}_k$
    $\hat{\mathcal{J}}_k = \hat{\mathcal{J}}_k \cup \{l\}$
    if $|\hat{\mathcal{J}}_k| + 1 \geq p_{\text{max}}$ then
      break
    end if
    update $\hat{\mathcal{J}}$
    $y_l = A((\hat{\mathcal{J}}_k)_{\text{old}}, (\hat{\mathcal{J}}_k)_{\text{old}})^{-1} A_l((\hat{\mathcal{J}}_k)_{\text{old}})$
    for $j \in \hat{\mathcal{J}} \cup \{k\}$ do
      $S_{jl} = A_{jl} - A_j((\hat{\mathcal{J}}_k)_{\text{old}})^\top y_l$
      $s[j] = s[j] - \frac{s_{jl}^2}{\pi[l]}$
    end for
  end while
  $y_k = A(\mathcal{J}_k, \hat{\mathcal{J}}_k)^{-1} A_k(\mathcal{J}_k)$
  $L_{kk} = 1/(A_{kk} - A_k(\mathcal{J}_k)^\top y_k)^{1/2}$
  $L_k(\mathcal{J}_k) = -L_{kk} y_k$
end for
Algorithm 8.3 EFSPAI-level 2

for $k = 1 \ldots n$ do
    $s = \text{diag}(A)$, $s_k = A_k$, $\hat{J}_k = \emptyset$, initialize $\hat{J}$
while true do
    for $j \in \hat{J}$ do
        $\mu_{jk} = \frac{s_k[j]^2}{s[j]s[k]}$
    end for
    if $\max_j \mu_{jk} < \epsilon$ then
        break
    end if
    $l =$ index with largest $\mu_{lk}$
    $(\hat{J}_k)_{\text{old}} = \hat{J}_k$
    $\hat{J}_k = \hat{J}_k \cup \{l\}$
    if $|\hat{J}_k| + 1 \geq p_{\text{max}}$ then
        break
    end if
    update $\hat{J}$
    $y_l = A((\hat{J}_k)_{\text{old}}, (\hat{J}_k)_{\text{old}})^{-1} A_l((\hat{J}_k)_{\text{old}})$
    for $j \in \hat{J} \cup \{k\}$ do
        $S_{jl} = A_{jl} - A_j((\hat{J}_k)_{\text{old}})^\top y_l$
        $s[j] = s[j] - \frac{S_{jl}}{s_l}$
        $s_k[j] = s_k[j] - \frac{S_{jl}s_k[l]}{s[l]}$
    end for
end while
$y_k = A(\hat{J}_k, \hat{J}_k)^{-1} A_k(\hat{J}_k)$
$L_{kk} = 1 / (A_{kk} - A_k(\hat{J}_k)^\top y_k)^{1/2}$
$L_k(\hat{J}_k) = -L_{kk} y_k$
end for
Algorithm 8.4 EFSPAI-level 3

for \( k = 1 \ldots n \) do
    \( S = A, \hat{J}_k = \emptyset, \) initialize \( \hat{J} \)
    while true do
        for \( j \in \hat{J} \) do
            \( \mu_{jk} = \frac{s_{kj}^2}{s_{jj} S_{kk}} \)
        end for
        if \( \max_j \mu_{jk} < \epsilon \) then
            break
        end if
        \( l = \) index with largest \( \mu_{lk} \)
        \( \hat{J}_k = \hat{J}_k \cup \{l\} \)
        if \( |\hat{J}_k| + 1 \geq p_{\text{max}} \) then
            break
        end if
        update \( \hat{J} \)
        \( \hat{J}' = \hat{J} \cup \{k\} \)
        \( S(\hat{J}', \hat{J}') = S(\hat{J}', \hat{J}') - S_i(\hat{J}') \frac{1}{s_{ii}} S_i(\hat{J}')^\top \)
    end while
    \( y_k = A(\hat{J}_k, \hat{J}_k)^{-1} A_k(\hat{J}_k) \)
    \( L_{kk} = 1/(A_{kk} - A_k(\hat{J}_k)^\top y_k)^{1/2} \)
    \( L_k(\hat{J}_k) = -L_{kk} y_k \)
end for
8.4 Numerical Results

In this section we compare FSPAI and the EFSPAI versions regarding the computation time and the quality of the preconditioner. All tests were done in MATLAB in the same environment, with matrices and notations introduced in Part 7.5.

8.4.1 EFSPAI-Level 1 and 2

EFSPAI-level 1 (EFSPAI(1)) and EFSPAI-level 2 (EFSPAI(2)) were implemented based on Algorithms 8.2 and 8.3 with updating the set of possible new entries \( \hat{J} \), like in Part 7.5.

<table>
<thead>
<tr>
<th>matrix</th>
<th>( \epsilon )</th>
<th>FSPAI</th>
<th>EFSPAI(1)</th>
<th>EFSPAI(2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>bcsstk14</td>
<td>0.5</td>
<td>2.44</td>
<td>3.27</td>
<td>2.13</td>
</tr>
<tr>
<td></td>
<td>0.4</td>
<td>2.53</td>
<td>3.31</td>
<td>2.23</td>
</tr>
<tr>
<td></td>
<td>0.3</td>
<td>2.75</td>
<td>3.72</td>
<td>2.63</td>
</tr>
<tr>
<td></td>
<td>0.2</td>
<td>3.52</td>
<td>5.08</td>
<td>3.93</td>
</tr>
<tr>
<td></td>
<td>0.1</td>
<td>5.31</td>
<td>8.48</td>
<td>7.30</td>
</tr>
<tr>
<td></td>
<td>0.05</td>
<td>8.12</td>
<td>13.42</td>
<td>12.06</td>
</tr>
<tr>
<td>lund_a</td>
<td>0.5</td>
<td>0.09</td>
<td>0.10</td>
<td>0.06</td>
</tr>
<tr>
<td></td>
<td>0.4</td>
<td>0.09</td>
<td>0.10</td>
<td>0.06</td>
</tr>
<tr>
<td></td>
<td>0.3</td>
<td>0.09</td>
<td>0.10</td>
<td>0.06</td>
</tr>
<tr>
<td></td>
<td>0.2</td>
<td>0.10</td>
<td>0.10</td>
<td>0.07</td>
</tr>
<tr>
<td></td>
<td>0.1</td>
<td>0.19</td>
<td>0.22</td>
<td>0.19</td>
</tr>
<tr>
<td></td>
<td>0.05</td>
<td>0.51</td>
<td>0.72</td>
<td>0.68</td>
</tr>
<tr>
<td>msc01440</td>
<td>0.5</td>
<td>1.66</td>
<td>2.03</td>
<td>1.29</td>
</tr>
<tr>
<td></td>
<td>0.4</td>
<td>1.75</td>
<td>2.19</td>
<td>1.43</td>
</tr>
<tr>
<td></td>
<td>0.3</td>
<td>1.88</td>
<td>2.38</td>
<td>1.63</td>
</tr>
<tr>
<td></td>
<td>0.2</td>
<td>3.15</td>
<td>4.63</td>
<td>3.85</td>
</tr>
<tr>
<td></td>
<td>0.1</td>
<td>6.86</td>
<td>11.16</td>
<td>10.35</td>
</tr>
<tr>
<td></td>
<td>0.05</td>
<td>8.32</td>
<td>13.12</td>
<td>12.30</td>
</tr>
</tbody>
</table>

Table 3: Time measurements with different \( \epsilon \), \( p_{max} = 5 \)

In Tables 3 and 4 we have time measurements of the FSPAI, EFSPAI(1) and EFSPAI(2) computation for variable \( \epsilon \) respectively variable pattern limit \( p_{max} \). Obviously FSPAI is the fastest algorithm. We observe that the EFSPAI(1) computational time is longer by a factor of 1.3 to 1.6, whereby this factor en-
larges with denser pattern. The EFSPAI(1) time is at most two times longer than the FSPAI time, which can be seen by comparing the algorithms 7.1 and 8.2. The additional update part has exactly the same structure as the FSPAI $\tau$ computation.

EFSPAI(2) is faster than EFSPAI(1). Notice the nearly constant difference between these two algorithms, for example 1.1 to 1.3 seconds for bcsstk14. EFSPAI(2) mainly differs from EFSPAI(1) by updating $\forall j: s_k[j] = s_k[j] - \frac{S_{jl}s_k[l]}{s[l]}$ instead of $\forall j: S_{jk} = A_{jk} - A_{j}(\tilde{J}_k)^\top y_k$ in the $\tau$ computation. This update is not necessary, if we stop due to one of the stopping criteria. Hence EFSPAI(2) has one computation less than EFSPAI(1) for each column.

### 8.4.2 EFSPAI-Level 3

If we want to implement EFSPAI-level 3 (EFSPAI(3)) similar to EFSPAI(1), it would be too slow. The $S$ updates are one of the most time consuming parts, and for EFSPAI(3), we have $O(|\tilde{J}|^2)$ updates instead of $O(|\tilde{J}|)$. But the update of the Schur complement (8.2) can be efficiently implemented using MATLAB matrix operations. Consider $l$ as the previously added index.

\begin{verbatim}
1 Y = S(l, :) ; Y(l) = [] ;
2 S(l, :) = [] ; S(:, l) = [] ;
3 S = S - 1/S(l, l) * Y' * Y ;
\end{verbatim}

First we delete the $l$th row and column of the current Schur complement $S$, and then we update it. As we don’t use $\tilde{J}$, we also want to implement the

<table>
<thead>
<tr>
<th>matrix</th>
<th>$p_{max}$</th>
<th>EFSPAI</th>
<th>EFSPAI(1)</th>
<th>EFSPAI(2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>bcsstk14</td>
<td>2</td>
<td>2.37</td>
<td>3.11</td>
<td>2.06</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>4.57</td>
<td>6.74</td>
<td>5.49</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>5.26</td>
<td>8.08</td>
<td>6.84</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>5.44</td>
<td>8.48</td>
<td>7.24</td>
</tr>
<tr>
<td>lund_a</td>
<td>2</td>
<td>0.10</td>
<td>0.10</td>
<td>0.07</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.16</td>
<td>0.19</td>
<td>0.16</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>0.18</td>
<td>0.22</td>
<td>0.19</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>0.18</td>
<td>0.23</td>
<td>0.19</td>
</tr>
<tr>
<td>msc01440</td>
<td>2</td>
<td>1.73</td>
<td>2.11</td>
<td>1.37</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>3.91</td>
<td>5.70</td>
<td>4.77</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>5.78</td>
<td>8.95</td>
<td>8.06</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>6.87</td>
<td>11.15</td>
<td>10.27</td>
</tr>
</tbody>
</table>

Table 4: Time measurements with different $p_{max}$, $\epsilon = 0.1$
\( \mu_{jk} \) computation in vectorized form. Consider \( \mu_k \) as the vector containing the \( \mu_{jk} \). Then the computation of \( \mu_k \) using the matrix \( S \) from above can be done in MATLAB like:

```matlab
\mu_k = S(:,1)^2 ./ \text{diag}(S) / S(1,1);
```

Note that we initialize \( S = A(k:n,k:n) \), and therefore \( S(:,1) \) contains the \( k \)th column of \( S \). The maximum of \( \mu_k \) is our next new index.

When we want to compare this EFSPAI(3) with FSPAI, we have to find an analogue MATLAB implementation taking advantage of matrix operations. This cannot be done perfectly as we still need \( \tilde{J}_k \) in FSPAI for computing \( \tau_{jk} \).

Consider \( I := \{k+1,\ldots,n\} \setminus \tilde{J}_k \). A possible implementation is:

```matlab
1 yk = A(\tilde{J}_k,\tilde{J}_k) \setminus A(\tilde{J}_k,k); 
2 Skk = A(k,k) - A(\tilde{J}_k,k)'*yk; 
3 \tau_k = (A(I,k) - A(\tilde{J}_k,I)'*yk)^2 ./ \text{diag}(A(I,I)) / Skk;
```

Now we compare these two algorithms FSPAI and EFSPAI(3). In Table 5 the computational times of FSPAI and EFSPAI(3) with different \( \epsilon \) are presented, and we can observe that EFSPAI(3) is a little bit faster. But as this implementation is not representative for a C++ implementation, we do not consider ESPAI(3) for any following conclusion.

<table>
<thead>
<tr>
<th>matrix</th>
<th>( \epsilon )</th>
<th>FSPAI</th>
<th>EFSPAI(3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>bcsstk14</td>
<td>0.5</td>
<td>2.51</td>
<td>1.97</td>
</tr>
<tr>
<td></td>
<td>0.4</td>
<td>2.54</td>
<td>2.01</td>
</tr>
<tr>
<td></td>
<td>0.3</td>
<td>2.65</td>
<td>2.13</td>
</tr>
<tr>
<td></td>
<td>0.2</td>
<td>2.99</td>
<td>2.52</td>
</tr>
<tr>
<td></td>
<td>0.1</td>
<td>3.72</td>
<td>3.43</td>
</tr>
<tr>
<td></td>
<td>0.05</td>
<td>4.85</td>
<td>4.68</td>
</tr>
<tr>
<td>msc01440</td>
<td>0.5</td>
<td>1.37</td>
<td>1.00</td>
</tr>
<tr>
<td></td>
<td>0.4</td>
<td>1.41</td>
<td>1.05</td>
</tr>
<tr>
<td></td>
<td>0.3</td>
<td>1.47</td>
<td>1.11</td>
</tr>
<tr>
<td></td>
<td>0.2</td>
<td>1.94</td>
<td>1.58</td>
</tr>
<tr>
<td></td>
<td>0.1</td>
<td>3.31</td>
<td>2.98</td>
</tr>
<tr>
<td></td>
<td>0.05</td>
<td>3.90</td>
<td>3.43</td>
</tr>
</tbody>
</table>

Table 5: Time measurements with different \( \epsilon, p_{\text{max}} = 5 \), between EFSPAI(3) and an analogue implemented FSPAI.
8.4.3 Convergence Results

Now we want to see if this additional amount of time pays off. We solved the preconditioned systems analogue to the tests in Part 7.5 with PCG. The Tables 6 and 7 present the comparison of the FSPAI and EFSPAI solution regarding the number of non-zero entries and the number of PCG iterations. The iterations show that there is almost no difference in quality. More precisely, both solutions are almost equal, which can be seen in Figure 3. The few differences in the structure mainly occur because EFSPAI stops later for some columns, as $\mu_{jk} \geq \tau_{jk}$, not because of different indices chosen.

It is possible to construct matrices where the FSPAI fails, while the EFSPAI works much better, for example the matrix in (7.17). But for most matrices, $\tau_{jk}$ is a convincing indication of the quality of index $j$.

Notice that the EFSPAI update idea may be unstable. In all tests, the three EFSPAI versions found the same pattern. But using the updated $S_{kk}$ for computing $L_{kk} = \frac{1}{\sqrt{S_{kk}}}$ yields an error-prone solution. During one update step $S_{jk} = S_{jk} - \frac{S_{jl}S_{lk}}{S_{ll}}$, we lose significant information, if $S_{jk}$ and $\frac{S_{jl}S_{lk}}{S_{ll}}$ are nearly equal. Therefore, we need an implementation of the Schur complement both fast to compute and numerical stable.

<table>
<thead>
<tr>
<th>matrix</th>
<th>$p_{max}$</th>
<th>mnz FSPAI</th>
<th>mnz EFSPAI</th>
<th>iter FSPAI</th>
<th>iter EFSPAI</th>
</tr>
</thead>
<tbody>
<tr>
<td>bcsstk14</td>
<td>2</td>
<td>2581</td>
<td>2581</td>
<td>189</td>
<td>189</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>2807</td>
<td>2823</td>
<td>171</td>
<td>171</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>2861</td>
<td>2893</td>
<td>167</td>
<td>165</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>2874</td>
<td>2913</td>
<td>158</td>
<td>164</td>
</tr>
<tr>
<td>lund_a</td>
<td>2</td>
<td>198</td>
<td>198</td>
<td>86</td>
<td>86</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>211</td>
<td>212</td>
<td>84</td>
<td>84</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>211</td>
<td>212</td>
<td>84</td>
<td>84</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>211</td>
<td>212</td>
<td>84</td>
<td>84</td>
</tr>
<tr>
<td>msc01440</td>
<td>2</td>
<td>2292</td>
<td>2292</td>
<td>861</td>
<td>857</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>2940</td>
<td>2940</td>
<td>775</td>
<td>775</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>3277</td>
<td>3341</td>
<td>717</td>
<td>713</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>3541</td>
<td>3623</td>
<td>535</td>
<td>527</td>
</tr>
</tbody>
</table>

Table 6: Convergence results with different $p_{max}$, $\epsilon = 0.1$
<table>
<thead>
<tr>
<th>matrix</th>
<th>$\epsilon$</th>
<th>FSPA1</th>
<th>EFSPA1</th>
<th>FSPA1</th>
<th>EFSPA1</th>
</tr>
</thead>
<tbody>
<tr>
<td>bcsstk14</td>
<td>0.5</td>
<td>1870</td>
<td>1870</td>
<td>266</td>
<td>266</td>
</tr>
<tr>
<td></td>
<td>0.4</td>
<td>1902</td>
<td>1902</td>
<td>259</td>
<td>259</td>
</tr>
<tr>
<td></td>
<td>0.3</td>
<td>1988</td>
<td>1988</td>
<td>243</td>
<td>243</td>
</tr>
<tr>
<td></td>
<td>0.2</td>
<td>2243</td>
<td>2250</td>
<td>211</td>
<td>214</td>
</tr>
<tr>
<td></td>
<td>0.1</td>
<td>2874</td>
<td>2913</td>
<td>158</td>
<td>164</td>
</tr>
<tr>
<td></td>
<td>0.05</td>
<td>3763</td>
<td>3867</td>
<td>140</td>
<td>137</td>
</tr>
<tr>
<td>hund_a</td>
<td>0.5</td>
<td>147</td>
<td>147</td>
<td>98</td>
<td>98</td>
</tr>
<tr>
<td></td>
<td>0.4</td>
<td>147</td>
<td>147</td>
<td>98</td>
<td>98</td>
</tr>
<tr>
<td></td>
<td>0.3</td>
<td>147</td>
<td>147</td>
<td>98</td>
<td>98</td>
</tr>
<tr>
<td></td>
<td>0.2</td>
<td>151</td>
<td>151</td>
<td>98</td>
<td>98</td>
</tr>
<tr>
<td></td>
<td>0.1</td>
<td>211</td>
<td>212</td>
<td>84</td>
<td>84</td>
</tr>
<tr>
<td></td>
<td>0.05</td>
<td>427</td>
<td>438</td>
<td>56</td>
<td>55</td>
</tr>
<tr>
<td>msc01440</td>
<td>0.5</td>
<td>1440</td>
<td>1440</td>
<td>991</td>
<td>991</td>
</tr>
<tr>
<td></td>
<td>0.4</td>
<td>1477</td>
<td>1477</td>
<td>983</td>
<td>984</td>
</tr>
<tr>
<td></td>
<td>0.3</td>
<td>1531</td>
<td>1531</td>
<td>977</td>
<td>977</td>
</tr>
<tr>
<td></td>
<td>0.2</td>
<td>1995</td>
<td>2015</td>
<td>889</td>
<td>855</td>
</tr>
<tr>
<td></td>
<td>0.1</td>
<td>3541</td>
<td>3623</td>
<td>535</td>
<td>527</td>
</tr>
<tr>
<td></td>
<td>0.05</td>
<td>4265</td>
<td>4315</td>
<td>494</td>
<td>493</td>
</tr>
</tbody>
</table>

Table 7: Convergence results with different $\epsilon$, $p_{max} = 5$

Figure 3: msc01440: sparsity structure with $\epsilon = 0.1$, $p_{max} = 5$
9 Block FSPAI

Similar to the BSP AI we try to introduce a block version for FSPAI. The **Block FSPAI (BFSPAI)** also starts with minimizing the Kaporin functional (3.2)

\[
K(L^\top AL) = \frac{1}{n} \text{trace}(L^\top AL) \frac{1}{\det(L^\top AL)^{\frac{1}{n}}}
\]  

(9.1)

for a prescribed sparsity pattern. In its second step the pattern is iteratively extended by new profitable indices. In contrast to FSPAI, this all operates on whole blocks and block columns. With block notations, all statements and proofs can be written nearly identical to the FSPAI analogons. Therefore we present the BSP AI with the same structure as the FSPAI, including all theoretical properties and update formulas. We always reference the statements to the corresponding FSPAI ones.

A block version for the FSAI has been published before by Janna, Ferronato and Gambolati [16], combined with the ILU.

### 9.1 BFSAI: Block Preconditioner for a Fixed Sparsity Block Pattern

First we present a static version of the BFSPAI, i.e., we minimize (9.1) for a given block index structure. For reasons of simplification we assume a constant block size \( b \). We denote the number of block columns by \( n_b = \frac{n}{b} \), the block in the \( j \)th row and \( k \)th column by \( A_{jk} \) and the \( k \)th block column by \( A_k \). The block submatrix \( [A_{ij}]_{i,j \in I,J} \) is denoted by \( A(I,J) \) for two block index sets \( I \) and \( J \). The same holds for the matrix \( L \). The current block sparsity pattern of \( L_k \) is denoted by \( J_k \). As \( L \) is a nonsingular lower triangular matrix, \( J_k \subset \{k,\ldots,n\} \) always contains \( k \). We set \( \tilde{J}_k = J_k - \{k\} \). The block entries of \( S(7.2) \) are

\[
S_{jk} = A_{jk} - A_j(\tilde{J}_k)^\top A(\tilde{J}_k,\tilde{J}_k)^{-1} A_k(\tilde{J}_k)
\]  

(9.2)

with s.p.d diagonal blocks \( S_{kk} \). We define the function \( \text{chol}(A) \) as the Cholesky factor of \( A \), i.e., \( A = \text{chol}(A)^\top \text{chol}(A) \).

**Theorem 9.1.** Let \( \tilde{J}_k \) be the fixed sparsity block column pattern. The Kaporin functional (9.1) is minimized by \( L \) with

\[
L_{kk} = \text{chol}(S_{kk})^{-1} = \text{chol}(A_{kk} - A_k(\tilde{J}_k)^\top A(\tilde{J}_k,\tilde{J}_k)^{-1} A_k(\tilde{J}_k))^{-1}
\]  

(9.3)

\[
L_k(\tilde{J}_k) = - \text{chol}(A(\tilde{J}_k,\tilde{J}_k)^{-1} A_k(\tilde{J}_k))L_{kk}
\]  

(9.4)

for each block column \( k \). Note that this is the block correspondence of Theorem 7.2.

**Proof.** To obtain the minimum of (9.1), we set the derivative of (9.1) equal to zero. As the denominator only depends on the diagonal blocks \( L_{kk} \), we expand
(9.5) to

\[
K(L^\top AL) = \frac{1}{n} \frac{\text{trace}(L^\top AL)}{\det(L^\top AL)^\frac{2}{n}} = \frac{1}{n} \sum_{k=1}^{n_b} \frac{\text{trace}(L_k^\top AL_k)}{\det(A)^\frac{2}{n} \prod_{k=1}^{n_b} \det(L_{kk})^\frac{1}{2}}
\]

\[
= \frac{1}{n \det(A)^\frac{2}{n} \prod_{k=1}^{n_b} \det(L_{kk})^\frac{1}{2}} \sum_{k=1}^{n_b} \left[ \text{trace}(L_{kk}^\top A_{kk} L_{kk}) + 2 \text{trace}(-A_k(\tilde{J}_k, \tilde{J}_k) L_{kk}) + \text{trace}(-A_k(\tilde{J}_k, \tilde{J}_k)^\top A_k(\tilde{J}_k, \tilde{J}_k)) \right]
\]

We start derivating only with respect to \( L_k(\tilde{J}_k) \), setting this equal to zero and solving it by \( L_k(\tilde{J}_k) \). Inserting the solution into (9.5) results in a function only dependent on \( L_{kk} \), which can be derivated with respect to \( L_{kk} \) to obtain the minimum.

The derivative of (9.5) with respect to \( L_k(\tilde{J}_k) \) using (A.5) and (A.6) leads to (9.4).

\[
\frac{\partial K(L^\top AL)}{\partial L_k(\tilde{J}_k)} = \frac{1}{n \det(A)^\frac{2}{n} \prod_{k=1}^{n_b} \det(L_{kk})^\frac{1}{2}} \left( 2A_k(\tilde{J}_k)L_{kk} + 2A(\tilde{J}_k, \tilde{J}_k) L_k(\tilde{J}_k) \right) = 0
\]

\[
\Leftrightarrow L_k(\tilde{J}_k) = -A(\tilde{J}_k, \tilde{J}_k)^{-1} A_k(\tilde{J}_k) L_{kk}
\]

Inserting this into the expanded form (9.5), we obtain

\[
\sum_{k=1}^{n_b} \frac{\text{trace}(L_{kk}^\top A_{kk} L_{kk})}{n \det(A)^\frac{2}{n} \prod_{k=1}^{n_b} \det(L_{kk})^\frac{1}{2}} = \sum_{k=1}^{n_b} \frac{\text{trace}(L_{kk}^\top (A_{kk} - A_k(\tilde{J}_k) A(\tilde{J}_k, \tilde{J}_k)^{-1} A_k(\tilde{J}_k)) L_{kk})}{n \det(A)^\frac{2}{n} \prod_{k=1}^{n_b} \det(L_{kk})^\frac{1}{2}}
\]

We find (9.3) by derivating (9.6) using (A.6) and (A.7) with renamed indices
with respect to $L_{kk}$:

$$0 \overset{!}{=} \frac{\partial}{\partial L_{kk}} \left( \sum_{j=1}^{n_{b}} \text{trace} \left( L_{jj}^\top S_{jj} L_{jj} \right) \frac{1}{n \text{det}(A)^{2/n} \prod_{j=1}^{n_{b}} \text{det}(L_{jj})^{2/n}} \right)$$

$$\Leftrightarrow 0 \overset{!}{=} \frac{1}{n \text{det}(L_{kk})^{2/n}} \left( \sum_{j=1}^{n_{b}} \text{trace} \left( L_{jj}^\top S_{jj} L_{jj} \right) \right)$$

$$\Leftrightarrow \text{det}(L_{kk})^{2/n} \left( 2S_{kk} L_{kk} \right) - \frac{1}{n} \sum_{j=1}^{n_{b}} \text{trace} \left( L_{jj}^\top S_{jj} L_{jj} \right) \left( L_{kk}^{-\top} \right)$$

$$\Leftrightarrow L_{kk}^\top S_{kk} L_{kk} = \frac{1}{n} \sum_{j=1}^{n_{b}} \text{trace} \left( L_{jj}^\top S_{jj} L_{jj} \right) (I) \quad (9.7)$$

For all columns $k$, the right-hand side of condition (9.7) is the same. Hence there exists a $c > 0$ with $L_{kk}^\top S_{kk} L_{kk} = cI$ for all $k$. It is obvious that any choice of $c$ satisfies (9.7). Therefore, we choose $c := 1$, as with this determination, the FSAI results in the same solution using the corresponding sparsity pattern. Note that this block column $L_k$ is not normalized (see (9.13)), which has to be considered in the stopping criterion. This leads us to (9.3):

$$\Leftrightarrow L_{kk}^\top S_{kk} L_{kk} = I$$

As we are interested in a Cholesky factor $L$, we select the only minimizer with lower triangular structure and positive diagonal elements, i.e.

$$L_{kk} = \text{chol}(S_{kk})^{-1}.$$  

It is obvious that the FSAI results in the same solution as the BSAI using the corresponding sparsity pattern. Therefore this is a minimum.

Notice that all block columns of $L$ are independent and therefore can be computed in parallel.

### 9.2 BFSAI: Updating the Sparsity Pattern

Similar to FSPAi in Section 7, we want to update the sparsity pattern automatically to improve on a current pattern. We are interested in approximating the improvement factor of an entry $L_{jk}$. Consider $L_k$ as the BFSAI solution according to Section 9.1 for the sparsity pattern $J_k$. We keep all entries $L_k(J_k)$ fixed and minimize (9.1) according to the new possible block $L_{jk}$. This results in an upper bound for the exact improvement, as we only modify $L_{jk}$ instead of all entries.
Theorem 9.2. By adding a new index \( j \) to the current block column pattern \( J_k \), the new Kaporin functional \( K_{\text{new}} \) is bounded by

\[
K_{\text{new}} \leq K_{\text{old}} \left( 1 - \frac{1}{n} \tau_{jk} \right)
\]  

(9.8)

with

\[
\tau_{jk} := \text{trace}(L_k^\top A_j A_j^{-1} A_j^\top L_k) = \|A_j^\top L_k\|_2^2.
\]  

(9.9)

Note that this is the block correspondence of 7.9.

Proof. We have to find the minimizer of the new Kaporin functional with fixed \( L_k(J_k) \). As \( k \) is already in \( J_k \), the Kaporin denominator \( n \det(A) \prod_{k=1}^n \det(L_{kk}) \frac{1}{n} \) is independent of \( L_{jk} \). Hence, we only have to derivate the Kaporin numerator with respect to \( L_{jk} \). Consider \( E_k \) as the \( k \)th unit block vector, i.e., the \( k \)th block of \( E_k \) is the identity matrix \( I_{b \times b} \). With A.5 and A.6, we get

\[
\frac{\partial}{\partial L_{jk}} \left[ \text{trace}\left((L + E_j L_{jk} E_k^\top)^\top A(L + E_j L_{jk} E_k^\top)\right)\right]
= \frac{\partial}{\partial L_{jk}} \left[ \text{trace}(L^\top A L) + 2 \text{trace}(L^\top A_j L_{jk} E_k^\top) + \text{trace}(E_k L_{jk}^\top A_j E_j L_{jk} E_k^\top)\right]
= \frac{\partial}{\partial L_{jk}} \left[ \text{trace}(L^\top A L) + 2 \text{trace}(L_{jk} E_k^\top A_j E_j L_{jk})\right]
= \frac{\partial}{\partial L_{jk}} \left[ \text{trace}(L^\top A L) + 2 \text{trace}(L_{jk} A_j L_{jk}) + \text{trace}(L_{jk} A_j L_{jk})\right]
= 2A_j^\top L_k + 2A_j L_{jk} - 1 = 0
\Rightarrow L_{jk} = -A_j^{-1} A_j^\top L_k.
\]  

(9.10)

(9.11)

As the second derivative is positive definite with the argumentation done in the proof for (6.3), this is a minimum.

Inserting (9.11) into (9.10) leads us to the minimum, which is an upper bound of \( K_{\text{new}} \) due to the way we constructed it. We also use the fact that the denominator \( \text{trace}(L^\top A L) \) is equal to \( n \) for any FSAI solution, see (9.14).

\[
K_{\text{new}} \leq \frac{\left[ \text{trace}(L^\top A L) + 2 \text{trace}(L_k^\top A_j L_{jk}) + \text{trace}(L_{jk}^\top A_j L_{jk})\right]}{n \det(L^\top A L)^{\frac{1}{n}}}
= \frac{\left[ n + 2 \text{trace}(L_k^\top A_j (-A_j^{-1} A_j^\top L_k)) + \text{trace}((-A_j^{-1} A_j^\top L_k)^\top A_j (-A_j^{-1} A_j^\top L_k))\right]}{n \det(L^\top A L)^{\frac{1}{n}}}
= \frac{\left[ n - \text{trace}(L_k^\top A_j A_j^{-1} A_j^\top L_k)\right]}{n \det(L^\top A L)^{\frac{1}{n}}}
= \frac{1}{\det(L^\top A L)^{\frac{1}{n}}} \left[ 1 - \frac{1}{n} \text{trace}(L_k^\top A_j A_j^{-1} A_j^\top L_k)\right]
= K_{\text{old}} \left( 1 - \frac{1}{n} \tau_{jk} \right)
\]

52
These $\tau_{jk}$ will be our indicator for the quality of a new entry $L_{jk}$. Large $\tau_{jk}$ indicate a large reduction of the Kaporin functional by adding the index $j$. Because $A_j$ and $L_k$ are very sparse, we only have to compute $\tau_{jk}$ for a low number of indices $j$.

Remarks:

- If all $\tau_{jk}$ are small enough we can expect a certain quality of the current preconditioner. Therefore, we will updating the pattern if all $\tau_{jk}$ are smaller than a tolerance $\hat{\epsilon}$. We choose $\hat{\epsilon} = b\epsilon$, as this is the comparison with a normalized block column $L_k$ (see (9.13)) and $\tau_{jk}$ is always lower than $b$ (see (9.24)).
- To prevent $L$ from becoming too dense, we implement an upper bound $p_{max}$ for the size of the index set $J_k$.
- As $A$ and $L_k$ are sparse, we only have to compute $\tau_{jk}$ for a small set of indices, $J = \{j > k; A_j(J_k) \neq 0 \land j \notin \hat{J}_k\}$.
- BFSPAII can be initialized with any lower triangular pattern which includes the diagonal elements.
- Similar to FSPAII it is possible to add more than one index in each update.
- For the $\tau_{jk}$, the Cholesky factors of $A_{jj} = \hat{L}_j^T \hat{L}_j$ can be pre-computed in parallel. Then the $\tau$ computation modifies to $\tau_{jk} = \|\hat{L}_j^T A_j^T L_k\|^2_F$.

Summarizing all steps we can formulate the following Algorithm 9.1 based on diagonal start block pattern.

9.3 Theoretical Properties

In the following part we introduce theoretical properties of BFSPAII similar to FSPAII. Let $L$ be the BFSPAII solution of Theorem 9.1 for a block column pattern $J_k$.

Lemma 9.3. The BFSPAII solution $L_k$ satisfies

$$A(J_k, J_k) L_k(J_k) = \begin{pmatrix} S_{kk} & L_{kk} \\ 0 & 0 \end{pmatrix}$$

Equation (9.12)

Note that this is the block correspondence of (7.13).
Algorithm 9.1 BFSPAI using diagonal start block pattern

for $k = 1 \ldots n_b$ do
  $J_k = \{k\}$, $L_{kk} = \text{chol}(S_{kk})^{-1}$
  while $|\tilde{J}_k| + 1 < p_{\text{max}}$ do
    $\tilde{J}$ set of potential candidates
    for $j \in \tilde{J}$ do
      $\tau_{jk} = ||A_j^\top L_k||^2_{j,j}$
    end for
    if $\max_j \tau_{jk} < \epsilon b$ then
      break
    end if
    $J_k = J_k \cup \{l\}$ with largest $\tau_{lk}$
    $y_k = A(\tilde{J}_k, \tilde{J}_k)^{-1} A_k(\tilde{J}_k)$
    $L_{kk} = \text{chol}(A_{kk} - A_k(\tilde{J}_k)^\top y_k)^{-1}$
    $L_k(\tilde{J}_k) = -y_k L_{kk}$
  end while
end for

Proof.

\[
A(\tilde{J}_k, \tilde{J}_k) L_k(\tilde{J}_k) = \begin{pmatrix}
  A_{kk} & A_k(\tilde{J}_k)^\top \\
  A_k(\tilde{J}_k) & A(\tilde{J}_k, \tilde{J}_k)
\end{pmatrix}
\begin{pmatrix}
  L_{kk} \\
  L_k(\tilde{J}_k)
\end{pmatrix}
= \begin{pmatrix}
  A_{kk} & A_k(\tilde{J}_k)^\top \\
  A_k(\tilde{J}_k) & A(\tilde{J}_k, \tilde{J}_k)
\end{pmatrix}
\begin{pmatrix}
  L_{kk} \\
  -A(\tilde{J}_k, \tilde{J}_k)^{-1} A_k(\tilde{J}_k) L_{kk}
\end{pmatrix}
= \begin{pmatrix}
  A_{kk} L_{kk} - A_k(\tilde{J}_k)^\top A(\tilde{J}_k, \tilde{J}_k)^{-1} A_k(\tilde{J}_k) L_{kk} \\
  A_k(\tilde{J}_k) L_{kk} - A(\tilde{J}_k, \tilde{J}_k) A(\tilde{J}_k, \tilde{J}_k)^{-1} A_k(\tilde{J}_k) L_{kk}
\end{pmatrix}
= \begin{pmatrix}
  A_{kk} - A_k(\tilde{J}_k)^\top A(\tilde{J}_k, \tilde{J}_k)^{-1} A_k(\tilde{J}_k) L_{kk} \\
  A_k(\tilde{J}_k) L_{kk} - A_k(\tilde{J}_k) L_{kk}
\end{pmatrix}
= \begin{pmatrix}
  S_{kk} L_{kk} \\
  0
\end{pmatrix}
\]

\[\Box\]

Theorem 9.4. The weighted Frobenius norm of the BFSAI solution $L_k$ (Section 9.1) is

\[||L_k||_A = \sqrt{\text{trace}(L_k^\top A L_k)} = \sqrt{b}.\]  \hspace{1cm} (9.13)

Note that this is the block correspondence of (7.14).
Proof.

\[
L_k^\top AL_k = L_k(J_k)^\top A(J_k,J_k)L_k(J_k) \overset{(9.12)}{=} L_k(J_k)^\top \begin{pmatrix} S_{kk}L_{kk} \\ 0 \end{pmatrix}
\]

\[
= L_{kk}S_{kk}L_{kk} \overset{(9.3)}{=} I_{b\times b}
\]

Hence, we obtain \( ||L_k||_A = \sqrt{\text{trace}(L_k^\top AL_k)} = \sqrt{b} \) \( \square \)

**Corollary 9.5.** For any BFSAI solution 9.1, the numerator of the Kaporin functional (9.1) satisfies

\[
\frac{1}{n} \text{trace}(L^\top AL) = 1.
\]

(9.14)

Note that this is the block correspondence of (7.15).

Proof.

\[
\frac{1}{n} \text{trace}(L^\top AL) = \frac{1}{n} \sum_{k=1}^{m_b} \text{trace}(L_k^\top AL_k) \overset{(9.13)}{=} \frac{1}{n} \sum_{k=1}^{m_b} b = \frac{n_b b}{n} = 1
\]

\( \square \)

**Theorem 9.6.** If all \( \tau_{jk} \) are equal to zero, the BFSAI solution is optimal regarding the minimization of \( K \) (9.1), i.e.,

\[
\forall k : \tau_{jk} = 0 \quad \forall j \in \{k+1, \ldots, n_b\} - \tilde{J}_k \text{ it follows } L^\top AL = I.
\]

(9.15)

To be more precise, if all \( \tau_{jk} \) are equal to zero for one block column \( k \), \( L_k \) is the \( k \)th block column of the inverse of the Cholesky factor of \( A \). Note that this is the block correspondence of (7.16).

Proof. Let \( I_k = \{k+1, \ldots, n_b\} \). The diagonal blocks of \( L^\top AL \) are Identity matrices according to the proof of (9.13). Without loss of generality, we assume \( i > k \) for the non-diagonal blocks \( L_i^\top AL_k \). Therefore \( J_i \subset \tilde{I}_k \).

To proof \( L_i^\top AL_k = 0 \), we need the \( \tau \) assumption.

\[
\forall j \in I - \tilde{J}_k : \tau_{jk} = ||A_j^\top L_k||_{A_j^\top A_j}^2 = 0 \iff A_j^\top L_k = 0
\]

Together with (9.12) we conclude \( A(I,.)L_k = 0 \), and with \( J_i \subset \tilde{I} \)

\[
L_i^\top AL_k = L_i(I)^\top A(I,.)L_k = 0
\]

Therefore all non-diagonal blocks are zero blocks. \( \square \)

**Corollary 9.7.** If the sparsity pattern \( J \) is full, the BFSAI solution \( L \) is optimal regarding the minimization of \( K \) (9.1), i.e., it satisfies \( L^\top AL = I \). Note that this is the block correspondence of Corollary 7.8.

55
Proof. This follows immediately from Theorem 9.6 and \( \{k+1,\ldots,n_k\} - \tilde{J}_k = \emptyset \).

\[ \]

**Corollary 9.8.** Without any stopping criterion, the FSPA1 algorithm would find the optimal \( L \) regarding the minimization of \( K (9.1) \), i.e., \( L^\top A L \). Note that this is the block correspondence of Corollary 7.9.

**Proof.** This algorithm always adds a new index to the sparsity pattern, if there is any \( \tau_{jk} > 0 \), i.e., it only stops if all \( \tau_{jk} \) are equal to zero. Now we can apply Theorem 9.6.

9.4 Upating the Kaporin Functional

The aim of this part is to find the exact Kaporin reduction value for adding a index \( j \). If we want to compare the Kaporin functional (9.1) value before and after the update process, we only have to consider the modification of the diagonal blocks \( L_{kk} \). According to (9.14), the numerator of \( K \) remains constant, and the denominator only depends on the diagonal blocks. The first step towards our Kaporin update is the update of the Cholesky factors of \( A(\tilde{J}_k, \tilde{J}_k) \). In each iteration we solve a linear system with the matrix \( A(\tilde{J}_k, \tilde{J}_k) \). As \( A \) is s.p.d. the Cholesky factorization becomes a possible factorization method. After adding a new index \( j \) to \( \tilde{J}_k \), we don’t need to compute the Cholesky factors of \( A(\tilde{J}_k \cup \{j\}, \tilde{J}_k \cup \{j\}) \) from scratch. We can update the decomposition according to (9.16).

**Lemma 9.9 (Cholesky Block Update).** Let us assume the decomposition \( A(\tilde{J}_k, \tilde{J}_k) = P^\top L^\top L P \) is known with permutation matrix \( P \) and lower triangular \( L \). Adding a new index \( j \) to \( \tilde{J}_k \) we can update the decomposition via

\[
A(\tilde{J}_k \cup \{j\}, \tilde{J}_k \cup \{j\}) = \begin{bmatrix} I & 0 \\ 0 & P \end{bmatrix} L^\top L \begin{bmatrix} I & 0 \\ 0 & P \end{bmatrix} \] (9.16)

with \( \hat{P} \) being the permutation matrix that permutes the block index \( j \) to the first position, e.g., \( \hat{P} A_j(\tilde{J}_k \cup \{j\}) = \begin{pmatrix} A_{jj} \\ A_j(\tilde{J}_k) \end{pmatrix} \), and \( \hat{L} = \begin{pmatrix} \tilde{L}_1 & 0 \\ \tilde{L}_2 & \tilde{L}_3 \end{pmatrix} \) with

\[
\hat{L}_3 = \hat{L}, \\
\hat{L}_2 = \hat{L}^{-\top} PA_j(\tilde{J}_k), \text{ and} \\
\hat{L}_1 = \text{chol}(S_{jj}) = \text{chol}(A_{jj} - \tilde{L}_2^\top \tilde{L}_2).
\]

Note that this is the block correspondence of (7.18).
We obtain
\[ J = \text{(Schur complement update)} \]
update the whole Schur complement defined in (7.2):

**Proof.**

\[
A(\tilde{J}_k \cup \{j\}, \tilde{J}_k \cup \{j\}) = \tilde{P}^T \begin{pmatrix} A_{jj} & A_j(\tilde{J}_k)^\top \\ A_j(\tilde{J}_k) & A(\tilde{J}_k, \tilde{J}_k) \end{pmatrix} \tilde{P} \\
= \tilde{P}^T \begin{pmatrix} I & 0 \\ 0 & P \end{pmatrix} \begin{pmatrix} A_{jj} & A_j(\tilde{J}_k)^\top P^T \\ PA_j(\tilde{J}_k) & L^\top L \end{pmatrix} \begin{pmatrix} I & 0 \\ 0 & P \end{pmatrix} \tilde{P} \\
= \left[ \begin{pmatrix} I & 0 \\ 0 & P \end{pmatrix} \tilde{P} \right]^\top \begin{pmatrix} A_{jj} & A_j(\tilde{J}_k)^\top P^T \\ PA_j(\tilde{J}_k) & L^\top L \end{pmatrix} \left[ \begin{pmatrix} I & 0 \\ 0 & P \end{pmatrix} \tilde{P} \right].
\]

We finish the decomposition by using the ansatz matrix \( \tilde{L} = \begin{pmatrix} \tilde{L}_1 & 0 \\ \tilde{L}_2 & \tilde{L}_3 \end{pmatrix} \).

We obtain

\[
\begin{pmatrix} A_{jj} & A_j(\tilde{J}_k)^\top P^T \\ PA_j(\tilde{J}_k) & L^\top L \end{pmatrix} = \tilde{L}^\top \tilde{L} = \begin{pmatrix} \tilde{L}_1^\top \tilde{L}_1 + \tilde{L}_2^\top \tilde{L}_2 & \tilde{L}_2^\top \tilde{L}_3 \\ \tilde{L}_2^\top \tilde{L}_3 & \tilde{L}_3^\top \tilde{L}_3 \end{pmatrix}
\]

which can be easily solved:

\[
\Rightarrow \tilde{L}^\top \tilde{L} = \begin{pmatrix} \tilde{L}_1 & \tilde{L}_2 \\ \tilde{L}_2 & \tilde{L}_3 \end{pmatrix} \Rightarrow \tilde{L}_3 = \tilde{L} \\
\Rightarrow PA_j(\tilde{J}_k) = \tilde{L}_3^\top \tilde{L}_2 \Rightarrow \tilde{L}_2 = \tilde{L}_3^\top PA_j(\tilde{J}_k) \\
\Rightarrow A_{jj} = \tilde{L}_1^\top \tilde{L}_1 + \tilde{L}_2^\top \tilde{L}_2 \Rightarrow \tilde{L}_1 = \text{chol}(A_{jj} - \tilde{L}_2^\top \tilde{L}_2).
\]

\( \blacksquare \)

Next we want to update the diagonal block \( L_{kk} \). As \( L_{kk} = \text{chol}(S_{kk})^{-1} \), we update the whole Schur complement defined in (7.2):

**Theorem 9.10 (Schur complement update).** Adding a new block index \( j \) to \( \tilde{J}_k \) modifies the Schur complement (7.2) in the following way: consider \( I_k = \{k, \ldots, n_k\} - \tilde{J}_k \) and the updated sets \( I'_k = I_k - \{j\} \) and \( \tilde{J}'_k = \tilde{J}_k \cup \{j\} \). Then the new Schur complement satisfies

\[
S_{\text{new}} := A(I'_k, I'_k) - A(\tilde{J}'_k, I'_k)^\top A(\tilde{J}'_k, \tilde{J}'_k)^{-1}A(\tilde{J}'_k, I'_k) \\
= S(I_k, I_k) - S(j, I_k)^\top S(j, j)^{-1}S(j, I_k).
\]

(9.17)

Note that this is the block correspondence of (7.19).
Proof. Assume that we know the Cholesky decomposition \( A(\tilde{J}_k, \tilde{J}_k) = \hat{L}^\top \hat{L} \).

To update \( S \) we use (9.16):

\[
S_{\text{new}} = A(I_k, I_k) - A(\tilde{J}_k, I_k) \hat{L}^\top (\tilde{J}_k, I_k)^{-1} A(\tilde{J}_k, I_k) \\
= A(I_k, I_k) - A(\tilde{J}_k, I_k)^\top \left( \hat{P}^\top \hat{L}^\top \hat{L} \hat{P} \right)^{-1} A(\tilde{J}_k, I_k) \\
= A(I_k, I_k) - A(\tilde{J}_k, I_k)^\top \hat{P}^\top \hat{L}^{-1} \hat{L}^\top \hat{P} A(\tilde{J}_k, I_k) \\
\]

We solve the system \( \hat{L}^\top Y = \hat{P} A(\tilde{J}_k, I_k) = \begin{pmatrix} A(j, I_k) \\ A(\tilde{J}_k, I_k) \end{pmatrix} \) to obtain \( Y \):

\[
\begin{pmatrix} \text{chol}(S_{jj})^\top & A_j(\tilde{J}_k)^\top \hat{L}^{-1} \\ 0 & \hat{L}^\top \end{pmatrix} \begin{pmatrix} Y_1 \\ Y_2 \end{pmatrix} = \begin{pmatrix} A(j, I_k) \\ A(\tilde{J}_k, I_k) \end{pmatrix} \\
\Rightarrow Y_2 = \hat{L}^{-1} A(\tilde{J}_k, I_k) \\
Y_1 = \text{chol}(S_{jj})^{-\top} \left( A(j, I_k) - A_j(\tilde{J}_k)^\top \hat{L}^{-1} \hat{L}^\top (\tilde{J}_k, I_k) \right) \\
= \text{chol}(S_{jj})^{-\top} S(j, I_k) \\
\]

Inserting (9.21) into (9.20) leads to the following update formula:

\[
S_{\text{new}} = A(I_k, I_k) - Y^\top Y \\
= A(I_k, I_k) - A(\tilde{J}_k, I_k)^\top \hat{L}^{-1} \hat{L}^\top A(\tilde{J}_k, I_k) - \\
- S(j, I_k)^\top \text{chol}(S_{jj})^{-1} \text{chol}(S_{jj})^{-\top} S(j, I_k) \\
= S(I_k, I_k) - S(j, I_k)^\top S_{jj}^{-1} S(j, I_k) \\
\]

This Schur complement is updated by a Schur complement. Considering the addition of a set of block indices instead of one block index \( j \), the analogue form can be derived. Now, it is possible to state the following Theorem.

**Theorem 9.11** (Kaporin update). Consider adding a block index \( j \) to the current sparsity pattern \( \tilde{J}_k \). Then the Kaporin functional (9.1) is reduced by the factor

\[
K_{\text{new}} = K_{\text{old}} \det \left( I - S_{kk}^{-1} S_{jk}^\top S_{jj}^{-1} S_{jk} \right)^{\frac{1}{2}} \\
\]

Note that this is the block correspondence of (7.22).
Proof. The Kaporin functional of a BFSAI solution $L$ (9.1) using (9.14) can be written:

$$K = \frac{1}{\det(L^\top AL)^{\frac{1}{n}}} = \frac{1}{\det(A)^{\frac{1}{n}} \prod_{j=1}^{n} \det(L_{jj})^{\frac{1}{n}}}$$

$$= \frac{(\det(L_{kk}^{-1}) \det(L_{kk}^{-\top}))^{\frac{1}{n}}}{\det(A)^{\frac{1}{n}} \prod_{j\neq k} \det(L_{jj})^{\frac{1}{n}}} = \frac{\det(S_{kk})^{\frac{1}{n}}}{\det(A)^{\frac{1}{n}} \prod_{j\neq k} \det(L_{jj})^{\frac{1}{n}}}$$

With updating $S_{kk}$ from (9.17) via

$$(S_{kk})_{\text{new}} = S_{kk} - S_{jk}^{-1}S_{jj}^{-1}S_{jk} = S_{kk} (I - S_{kk}^{-1}S_{jk}^{-1}S_{jj}^{-1}S_{jk}) ,$$

equation (9.22) follows immediately.

Now we want to compare this exact Kaporin reduction for augmenting the sparsity pattern with $\tau_{jk}$.

For that purpose we need $\tau_{jk}$ in a similar representation:

**Lemma 9.12.** The $\tau_{jk}$ defined in (9.9) can be written as

$$\tau_{jk} = \text{trace}(S_{kk}^{-1}S_{jk}^{-1}S_{jj}) \quad (9.23)$$

Note that this is the block correspondence of (7.24).

**Proof.** $\tau_{jk}$ was defined as $\text{trace}(L_k^\top A_j A_{jj}^{-1}A_j^\top L_k)$.

$$A_j^\top L_k = A_j (\tilde{J}_k)\top L_k (\tilde{J}_k)$$

$$= \left( \begin{array}{c} A_{kJ} \\ A_j (\tilde{J}_k) \end{array} \right) \top \left( \begin{array}{c} L_{kk} \\ -A (\tilde{J}_k, \tilde{J}_k)^{-1}A_k (\tilde{J}_k)L_{kk} \end{array} \right)$$

$$= \left( A_{jk} - A_j (\tilde{J}_k)\top A (\tilde{J}_k, \tilde{J}_k)^{-1}A_k (\tilde{J}_k) \right) L_{kk}$$

$$= S_{jk} \text{ chol}(S_{kk})^{-1}$$

Now we can rewrite:

$$\tau_{jk} = \text{trace}(L_k^\top A_j A_{jj}^{-1}A_j^\top L_k)$$

$$= \text{trace}(A_j^\top L_k L_k^\top A_j A_{jj}^{-1})$$

$$= \text{trace}(S_{jk} \text{ chol}(S_{kk})^{-1} \text{ chol}(S_{kk})^{-\top} S_{jj}^{-1} S_{jj}^{-1})$$

$$= \text{trace}(S_{jk} S_{kk}^{-1} S_{jj}^{-1} A_{jj}^{-1})$$

$$= \text{trace}(S_{kk}^{-1} S_{jj}^{-1} S_{jk})$$

Now we can estimate $\tau_{jk}$. 

\[59\]
Theorem 9.13. $\tau_{jk}$ (9.23) satisfies
\[ 0 \leq \tau_{jk} < b. \tag{9.24} \]

Note that this is the block correspondence of (7.25).

Proof. The first inequality is trivial with the $\tau_{jk}$ definition in (9.9).

The second one uses (B.6):
\[
\tau_{jk} = \text{trace}(S^{-1}_{kk} S^{-1}_{jk} S^{-1}_{jj} S_{jk}) \\
= b - \text{trace}(I_b \times b - S^{-1}_{kk} S^{-1}_{jk} S_{jj} S_{jk}) \\
= b - \text{trace}(S^{-1}_{kk} (S_{kk} - S^{-1}_{jk} S_{jj} S_{jk})) \\
< b
\]

Furthermore (9.23) yields an estimation of the Kaporin functional.

\[ K_{\text{new}} \leq K_{\text{old}} \left( 1 - \frac{1}{b} \tau_{jk} \right)^{\frac{1}{n}} \leq K_{\text{old}} \left( 1 - \frac{1}{n} \tau_{jk} \right) \tag{9.25} \]

Note that this is the block correspondence of (7.26).

Proof. For the first inequality, we only need the inequality (B.8).
\[
\det (I - S^{-1}_{kk} S^{-1}_{jk} S^{-1}_{jj} S_{jk}) \\
= \det (S^{-1}_{kk} (S_{kk} - S_{jk} S^{-1}_{jj} S_{jk})) \\
\leq \left( \frac{1}{b} \text{trace} (S^{-1}_{kk} (S_{kk} - S^{-1}_{jk} S_{jj} S_{jk})) \right)^b \\
= \left( \frac{1}{b} (\text{trace}(I_b \times b) - \text{trace}(S^{-1}_{kk} S^{-1}_{jk} S_{jj} S_{jk})) \right)^b \\
= \left( 1 - \frac{1}{b} \tau_{jk} \right)^b
\]
leads to
\[ K_{\text{new}} = K_{\text{old}} \det (I - S^{-1}_{kk} S^{-1}_{jk} S^{-1}_{jj} S_{jk})^{\frac{1}{b}} \leq K_{\text{old}} \left( 1 - \frac{1}{b} \tau_{jk} \right)^{\frac{1}{n}}. \]

The right-hand side in (9.25) is the estimation concluded from the derivation of $\tau_{jk}$ (9.8). We get the last inequality using Bernoulli’s inequality (B.2) on $-\frac{1}{n} \tau_{jk}$:
\[
\left( 1 - \frac{1}{n} \tau_{jk} \right)^{nb} \geq 1 - \frac{1}{b} \tau_{jk} \\
\iff 1 - \frac{1}{n} \tau_{jk} \geq \left( 1 - \frac{1}{b} \tau_{jk} \right)^{\frac{1}{n}}
\]

\qed
9.5 Numerical Results

In this Part we want to compare FSPAI and BFSPAI. All tests were done in the same environment as Part 7.5. For a demonstration of the advantages of BFSPAI over FSPAI, we need matrices with distinct block structures, i.e., all nonzero entries should be apportioned among a few blocks. We find these block matrices by determining the density of the nonzero blocks for a block size $b$. The following matrices all have a density of over 80%, and can be found in the Tim Davis matrix collection [9]. The block structure can be seen in Figure 4.

- **nos3** structural problem, Finite element approximation to biharmonic operator on a rectangular plate, $n = 960$, $nnz = 15844$, $\text{cond} = 7.3e+4$, $b = 2$
- **bcstk11** structural problem, symmetric stiffness matrix, ore car, $n = 1473$, $nnz = 34241$, $\text{cond} = 5.8e+6$, $b = 3$
- **bcstk14** structural problem, Roof of the Omni Coliseum, Atlanta, $n = 1806$, $nnz = 63454$, $\text{cond} = 1.3e+10$, $b = 6$
- **bcstk16** structural problem, stiffness matrix - corp. of engineers dam, $n = 4884$, $nnz = 290378$, $\text{cond} = 7.0e+9$, $b = 3$
- **bcstk24** structural problem, symmetric stiffness matrix, ore car, $n = 3562$, $nnz = 159910$, $\text{cond} = 7.2e+10$, $b = 2$
- **nasa2910** duplicate structural problem, structure from nasa langley, $n = 2910$, $nnz = 174296$, $\text{cond} = 1.3e+6$, $b = 5$

We implement BFSPAI in MATLAB in the following way: Because we want to compute on block structures, we first find the block pattern matrix $P_A$ of $A$ with $P_A(i,j) \neq 0 \Leftrightarrow$ block $A_{ij} \neq 0$:
\begin{verbatim}
[i, j] = ind2sub(size(A), find(A));
i = ceil(i/b); j = ceil(j/b);
P_A = sparse(i, j, ones(size(i)), n, n);
\end{verbatim}

ind2sub() transforms the linear indices from \texttt{find(A)} into matrix indices. These can be divided by \( b \) to get the corresponding block indices. Then the entire BFSPAI code can be implemented analogue to FSPAI. A closer consideration is given on the \( \tau_{jk} \) computation. We precompute the Cholesky factors of \( A^{-1}_{jj} = \hat{L}^\top \hat{L} \). Then \( \tau_{jk} = \text{trace}(L_k^\top A_j A^{-1}_{jj} A_j^\top L_k) = \| \hat{L}^\top A_j^\top L_k \|_F^2 \) can be implemented as \( \| Y \|_F^2 = \sum_{i,j} Y(i,j)^2 \).

Now we study the BFSPAI performance. Table 8 offers a comparison between FSPAI and BFSPAI for different matrices and variable \( \epsilon \). The limit for the pattern size \( p_{\text{max}} \) is not considered. We observe that BFSPAI is notably faster. An exception is constituted by nos3, where the advantage in computation speed begins with smaller \( \epsilon \). For a fixed \( \epsilon \), FSPAI and BFSPAI provide approximately efficient preconditioners, i.e., the PCG iteration numbers are nearly identical. This approves the stopping criterion \( \tau_{jk} < b \epsilon \). We get this quality at the cost of a denser preconditioner. We can see, the preconditioner generated by BFSPAI has approximately \( b \)-times more entries than the FSPAI preconditioner. The larger \( b \), the more inefficient entries are part of the added blocks. In return BFSPAI is faster by a factor of approximately \( \frac{1}{b} \). This behaviour can be also observed in the Figures 5 to 10. There the computational times of the algorithms are displayed to produce a certain quality of the preconditioner. The BFSPAI graph is always to the left of the FSPAI one, i.e., a preconditioner with a certain iteration rate can be produced in less time by BFSPAI. For matrices with larger block size, the BFSPAI graph is steeper and the block concept is more effective. Also notice the knee in the kurve in Figure 5. This is one of the rare examples where the Kaporin value gets smaller but the iteration rate increases.

Note that the density of the preconditioner affects the PCG computational time, if the number of nonzeros of \( L \) is in the order of \( \text{nnz}(A) \). Hence, it is possible that BFSPAI leads to a slower convergence speed, as it produces a much denser preconditioner. There are further tests needed on C++ environment.
Figure 5: nos3: expenditure of time needed for a certain iteration rate

Figure 6: bcsstk11: expenditure of time needed for a certain iteration rate
Figure 7: bcsstk14: expenditure of time needed for a certain iteration rate

Figure 8: bcsstk16: expenditure of time needed for a certain iteration rate
Figure 9: bcsstk24: expenditure of time needed for a certain iteration rate

Figure 10: nasa2910: expenditure of time needed for a certain iteration rate
<table>
<thead>
<tr>
<th>matrix</th>
<th>$\epsilon$</th>
<th>time FSPAI</th>
<th>time BFSPAI</th>
<th>nnz FSPAI</th>
<th>nnz BFSPAI</th>
<th>iter FSPAI</th>
<th>iter BFSPAI</th>
</tr>
</thead>
<tbody>
<tr>
<td>nos3</td>
<td>0.2</td>
<td>0.68</td>
<td>0.57</td>
<td>1008</td>
<td>1422</td>
<td>212</td>
<td>226</td>
</tr>
<tr>
<td></td>
<td>0.1</td>
<td>1.31</td>
<td>0.62</td>
<td>1418</td>
<td>1600</td>
<td>153</td>
<td>215</td>
</tr>
<tr>
<td></td>
<td>0.04</td>
<td>1.50</td>
<td>1.43</td>
<td>1550</td>
<td>3939</td>
<td>146</td>
<td>133</td>
</tr>
<tr>
<td>bcsstk11</td>
<td>0.2</td>
<td>2.18</td>
<td>0.71</td>
<td>1983</td>
<td>3130</td>
<td>850</td>
<td>837</td>
</tr>
<tr>
<td></td>
<td>0.1</td>
<td>2.92</td>
<td>0.91</td>
<td>2317</td>
<td>4115</td>
<td>752</td>
<td>762</td>
</tr>
<tr>
<td></td>
<td>0.04</td>
<td>5.17</td>
<td>2.37</td>
<td>3379</td>
<td>11248</td>
<td>626</td>
<td>453</td>
</tr>
<tr>
<td>bcsstk14</td>
<td>0.2</td>
<td>3.64</td>
<td>0.53</td>
<td>2243</td>
<td>6124</td>
<td>211</td>
<td>224</td>
</tr>
<tr>
<td></td>
<td>0.1</td>
<td>5.63</td>
<td>0.57</td>
<td>2875</td>
<td>6772</td>
<td>158</td>
<td>216</td>
</tr>
<tr>
<td></td>
<td>0.04</td>
<td>10.38</td>
<td>1.65</td>
<td>4262</td>
<td>23989</td>
<td>126</td>
<td>146</td>
</tr>
<tr>
<td>bcsstk16</td>
<td>0.2</td>
<td>12.21</td>
<td>3.59</td>
<td>4955</td>
<td>9709</td>
<td>189</td>
<td>194</td>
</tr>
<tr>
<td></td>
<td>0.1</td>
<td>13.74</td>
<td>3.60</td>
<td>5342</td>
<td>9776</td>
<td>165</td>
<td>192</td>
</tr>
<tr>
<td></td>
<td>0.04</td>
<td>22.15</td>
<td>4.73</td>
<td>7244</td>
<td>13920</td>
<td>140</td>
<td>162</td>
</tr>
<tr>
<td>bcsstk24</td>
<td>0.2</td>
<td>10.75</td>
<td>5.00</td>
<td>5291</td>
<td>8003</td>
<td>1432</td>
<td>1415</td>
</tr>
<tr>
<td></td>
<td>0.1</td>
<td>14.78</td>
<td>6.23</td>
<td>6569</td>
<td>9939</td>
<td>1174</td>
<td>1181</td>
</tr>
<tr>
<td></td>
<td>0.04</td>
<td>26.39</td>
<td>12.04</td>
<td>9772</td>
<td>18011</td>
<td>902</td>
<td>854</td>
</tr>
<tr>
<td>nasa2910</td>
<td>0.2</td>
<td>11.41</td>
<td>1.83</td>
<td>4261</td>
<td>14680</td>
<td>822</td>
<td>856</td>
</tr>
<tr>
<td></td>
<td>0.1</td>
<td>19.94</td>
<td>2.80</td>
<td>6197</td>
<td>24015</td>
<td>680</td>
<td>587</td>
</tr>
<tr>
<td></td>
<td>0.04</td>
<td>34.82</td>
<td>4.88</td>
<td>9271</td>
<td>41995</td>
<td>523</td>
<td>424</td>
</tr>
</tbody>
</table>

Table 8: FSPAI vs. BFSPAI with different $\epsilon$
10 Conclusion and Outlook

First we had a look at the FSPAI, a realization of the sparse approximate inverse technique on a s.p.d. system. FSPAI automatically captures a sparsity pattern for the preconditioner. We presented a detailed derivation of the FSPAI algorithm. Numerical results showed, that FSPAI produces an efficient preconditioner and can improve the convergence rate of PCG with a low number of entries. Note that FSPAI has its main strength in its inherent parallelism, which was not shown here. Furthermore, we found theoretical properties that helped us to derive the EFSPAI, which updates the pattern analogue to FSPAI, but chooses the optimal next index. We were able to show that the EFSPAI has larger computational costs, but is almost no improvement compared to FSPAI. This further confirms the quality of FSPAI. It is possible to implement EFSPAI with similar computational times as FSPAI. But to substantiate this statement we have to do more tests in different languages and environments. Lastly we developed a block version of the FSPAI. The BFSPAI leads to similar convergence rates for the iterative method but with shorter computation time. In return, it generates a denser preconditioner. We can conclude that BFSPAI effectively exploits a certain block structure and reduces the computation time compared to FSPAI. In addition, it keeps its parallelism.

The next step to improve BFSPAI is the determining of the block structure by applying graph algorithms. Then the BFSPAI has to be extended to handle resulting variable block sizes. Furthermore FSPAI can be modified to be applicable on indefinite symmetric linear systems. For FSAI this has been done before by Lazarov and Sigmund [21].
A Linear Algebra and Matrix Basics

This chapter covers the basic linear algebra and matrix concepts used in this thesis. It mainly contains proofless information with several examples. For further information in linear algebra, see for example [6]. See [7] for an introduction in matrix methods. An overview of matrix concepts needed in numerics is given in [15, 11]. Linear algebra basics more specialized on linear systems can be found in [26, 2]. For simplification, the following definitions are restricted to real numbers, but can be easily generalized to complex numbers.

A.1 Matrix

A matrix $A$ is a construct with $n$ rows, $m$ columns containing entries, in mathematical written $A \in \mathbb{R}^{n \times m}$. A few matrix notations:

- $A_{ij}$ is the entry in row $i$ and column $j$.
- $A(\mathcal{I}, \mathcal{J})$ is the submatrix $[A_{ij}]_{i \in \mathcal{I}, j \in \mathcal{J}}$ for two index sets $\mathcal{I}$ and $\mathcal{J}$.
- $A(., k)$ or $A_k$ is the $k$th column vector of $A$.
- $A(k, .)$ is the $k$th row vector of $A$.

Some special matrices that should be mentioned:

- a matrix with the same number of rows and columns is denoted as square matrix
- the identity matrix $I \in \mathbb{R}^{n \times n}$ is a square matrix containing only diagonal entries all of value 1.
- the zero matrix is a matrix with only entries equal to 0, i.e., $0_{n \times m}$.
- a matrix is called triangular, if all entries on one side of the diagonal are equal to 0. A lower/upper triangular matrix only has non-zero entries below/above the diagonal.
- a matrix $P \in \mathbb{R}^{n \times n}$ is called permutation matrix, if all entries are 0 or 1, and if in every row and column is exactly one nonzero.
- a matrix $v$ with just one column is called vector, i.e., $v \in \mathbb{R}^n$.
- the zero vector is a vector with only entries equal to 0, i.e., $\vec{0}$.

A.2 Matrix Operations

Now we list the main functions used to operate with matrices.
A.2.1 Transpose Matrix

The transpose $A^\top$ of a matrix $A \in \mathbb{R}^{n \times m}$ is a $m \times n$ matrix constructed by switching the rows and the columns, i.e. $(A^\top)_{ij} := A_{ji}$. A matrix $A$ with $A^\top = A$ is called symmetric.

A.2.2 Addition of Matrices

Two matrices $A$ and $B$ with the same number of rows and columns can be added in the following way: $(A + B)_{ij} = A_{ij} + B_{ij}$. The transpose is a linear function with respect to the matrix sum, i.e., $(A + B)^\top = A^\top + B^\top$.

A.2.3 Multiplication of Matrices

Two matrices $A$ and $B$ can be multiplied, if the left matrix $A$ has the same number of columns as the right matrix $B$ has rows, i.e., $A \in \mathbb{R}^{n \times m}$ and $B \in \mathbb{R}^{m \times p}$. The product is then defined by $[AB]_{ij} = \sum_{k=1}^{m} A_{ik}B_{kj}$. The matrix multiplication is associative and distributive over the matrix addition. The neutral element of the matrix multiplication is the identity matrix $I$. The transposed matrix product can be easily established by simply reversing the matrix order: $(AB)^\top = B^\top A^\top$. The inner vector product is a special case of the matrix multiplication. The inner product of two vectors $u, v \in \mathbb{R}^n$ is defined by $u^\top v = \sum_{k=1}^{n} u_k v_k$.

A.3 Trace of a Matrix

The trace of a square matrix $A \in \mathbb{R}^{n \times n}$ is the sum of the diagonal entries, i.e., $\text{trace}(A) = \sum_{k=1}^{n} A_{kk}$. A few properties of the matrix trace with $A, B \in \mathbb{R}^{n \times n}$ and $c \in \mathbb{R}$ are:

$$
\text{trace}(A + B) = \text{trace}(A) + \text{trace}(B)
$$

$$
\text{trace}(cA) = c \text{trace}(A)
$$

$$
\text{trace}(A^\top) = \text{trace}(A)
$$

$$
\text{trace}(AB) = \text{trace}(BA)
$$

A.4 Determinant of a Matrix

The determinant is an important function in linear algebra, and there are two possible ways to define it. Here we choose the Leibnitz formula as we are only interested in matrices. Let $A$ be a square matrix $A \in \mathbb{R}^{n \times n}$. Then

$$
\det(A) = \sum_{\pi \in S_n} \text{sgn}(\pi)A_{1,\pi(1)} \cdots A_{n,\pi(n)}
$$

with $S_n$ being the symmetric group, the set of permutations $\pi$ from $\{1, \ldots, n\}$ to $\{1, \ldots, n\}$, and $\text{sgn}(\cdot)$ the signum function of a permutation $\pi$. For example
for an $2 \times 2$ matrix, the determinant is
\[
\det \begin{pmatrix} a & b \\ c & d \end{pmatrix} = ad - bc.
\]

An often useful presentation for theoretical purposes are the cofactor expansions

by row $i$: \[\det(A) = \sum_{j=1}^{n} A_{ij} C_{ij}\] (A.1)
or by column $j$: \[\det(A) = \sum_{i=1}^{n} A_{ij} C_{ij},\]

with the cofactors \( C_{ij} = (-1)^{i+j} \det(M_{ij}) \), where \( M_{ij} \) is the \((n-1) \times (n-1)\) part of \( A \) after deleting the $i$th row and the $j$th column. Some further determinant properties:

Let \( A, B \) and \( L \) \((n \times n)\) be square matrices, \( L \) a triangular matrix and \( c \) a real scalar. Then the following equations hold:

\[
\det(L) = \prod_{k=1}^{n} L_{kk}
\]
\[
\det(A^\top) = \det(A)
\]
\[
\det(cA) = c^n \det(A)
\]
\[
\det(AB) = \det(A) \det(B)
\]
\[
\det(A^{-1}) = \det(A)^{-1}
\]

A.5 Rank of a Matrix

For the existence of solutions of linear systems of equations, we need the rank concept. Considering the columns of a matrix \( A \in \mathbb{R}^{n \times m} \) as vectors, the column rank is the dimension of the subspace spanned by these vectors. The span of the vectors \( v_i \), denoted as \( \text{span}(v_1, \ldots, v_n) \), is the linear subspace \( \{ x = \alpha_1 v_1 + \ldots + \alpha_n v_n, \alpha_i \in \mathbb{R} \} \). The dimension of this space is the maximal number of linear independent vectors \( v_i \). Corresponding to this, we can define the row rank by the row vectors of \( A \). The main rank Theorem (A.2) enables us to use only one notation:

\[
\text{row rank} = \text{column rank} \text{ for any matrix } A \in \mathbb{R}^{n \times m} \quad \text{(A.2)}
\]

We denote this definite rank by \( \text{rank}(A) \). \( A \) has full rank, if it has the maximal possible rank, i.e., \( \text{rank}(A) = \min(n,m) \).

A.6 Inverse of a Matrix

Let \( A \) be a \( n \times n \) matrix. The inverse of \( A \) is a matrix \( C \in \mathbb{R}^{n \times n} \) with

\[ AC = CA = I. \]
As the matrix $C$ is unique, we denote it by $A^{-1}$. If a square matrix has an inverse, it is called invertible or nonsingular. The following equivalence relation gives us possibilities to identify nonsingular matrices:

$$A \text{ is nonsingular } \iff \text{rank}(A) = n \iff \det(A) \neq 0.$$  

Furthermore, a few properties will help us to work with inverse matrices. Consider $A$, $B$, and $P$ as $n \times n$ matrices, $P$ as a permutation matrix and $c$ as a scalar. Let $C$ be the cofactor matrix whose entries are the cofactors $C_{ij}$. Then they satisfy:

$$(AB)^{-1} = B^{-1}A^{-1}$$

$$(A^{-1})^\top = (A^\top)^{-1} =: A^{-\top}$$

$$(cA)^{-1} = \frac{1}{c}A^{-1}$$

$$P^{-1} = P^\top$$

$$A^{-1} = \frac{1}{\det(A)} C^\top \quad \text{(Cramer's rule)} \quad (A.3)$$

Hence, we can conclude that the properties symmetric and triangular remain preserved after inverting the matrix.

### A.7 Eigenvalues

Eigenvalues play an important role for estimating the quality of the solution and the solvability of linear systems. A scalar $\lambda \in \mathbb{R}$ is called eigenvalue of the square matrix $A \in \mathbb{R}^{n \times n}$, if there exists a vector $x \in \mathbb{R}^n$, $x \neq \vec{0}$ with

$$Ax = \lambda x.$$  

Then $x$ is an eigenvector of $A$. The set of all eigenvalues of $A$ is called spectrum, denoted by $S(A)$. The spectral radius $\rho(A)$ is the maximum absolute eigenvalue of $A$, i.e., $\rho(A) = \max_{\lambda \in S(A)} |\lambda|$.

A few useful properties are:

$$\text{trace}(A) = \sum_{\lambda \in S(A)} \lambda$$

$$\det(A) = \prod_{\lambda \in S(A)} \lambda$$

### A.8 Positive Definite Matrix

Let $A$ be a square $n \times n$ matrix in the following.
A.8.1 Definitions

A matrix $A$ is said to be

- **positive definite**, if $x^\top Ax > 0, \forall x \in \mathbb{R}^n, x \neq \vec{0}$
- **positive semidefinite**, if $x^\top Ax \geq 0, \forall x \in \mathbb{R}^n$
- **symmetric positive definite (s.p.d.)**, if $A$ is symmetric and positive definite

A.8.2 Properties

Let $A$ be positive definite and $\mathcal{J}$ a subset of $\{1, \ldots, n\}$. Then

- $A$ is nonsingular and $A^{-1}$ is positive definite,
- all eigenvalues of $A$ are positive,
- all eigenvalues of $A$ are non-negative, if $A$ is positive semidefinite,
- $B^\top AB$ is positive definite with an nonsingular matrix $B$

Proof. Consider an arbitrary vector $x \in \mathbb{R}^n$. Then

$$x^\top (B^\top AB) x = (Bx)^\top ABx > 0 \text{ with } Bx \neq \vec{0}$$

, and

- $A(\mathcal{J}, \mathcal{J})$ is positive definite.

Proof. Consider an arbitrary vector $x \in \mathbb{R}^{|\mathcal{J}|}, x \neq \vec{0}$. Set the vector $y \in \mathbb{R}^n$ via $y(j) = x$ and $y(j) = 0$ for all $j \notin \mathcal{J}$. Then

$$x^\top A(\mathcal{J}, \mathcal{J}) x = y^\top Ay > 0.$$

Inequalities for positive definite matrices can be found in Appendix B.

A.8.3 Cholesky Decomposition

Let $A$ be s.p.d. Then there is one and only one lower triangular matrix $L$ with positive diagonal elements with

$$A = LL^\top,$$

which is called the **Cholesky decomposition**.

In this paper, without loss of generality, the term Cholesky decomposition is mainly used for the similar decomposition $A = L^\top L$, $L$ being the lower triangular with positive diagonal elements.
A.8.4 Schur Complement

Let $A$ be positive definite. Consider the partitioning of the set $\{1, \ldots, n\}$ into subsets $\mathcal{J} = \{k, \ldots, n\}$ and $\mathcal{I} = \{1, \ldots, n\} - \mathcal{J}$. Then the Schur complement

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

(A.4)
is positive definite. This can be even more generalized. Consider an arbitrary subset $\mathcal{J} \subset \{1, \ldots, n\}$ and $\mathcal{I} = \{1, \ldots, n\} - \mathcal{J}$. Then $S = A(\mathcal{I}, \mathcal{I}) - A(\mathcal{I}, \mathcal{J}) A(\mathcal{J}, \mathcal{J})^{-1} A(\mathcal{J}, \mathcal{I})$ is positive definite.

Proof. Let $P$ be the permutation matrix that permutes the subset $\mathcal{J}$ to $\{k, \ldots, n\}$. Then we can apply (A.4) on the positive definite matrix $P^T A P$.

A.9 Norms

Whenever vectors and solutions have to be comparable or errors have to be measured, normalisation and norms become necessary. This part gives a brief introduction to vector- and matrixnorms.

A.9.1 Vector norm

A vector norm is a function $\|\cdot\| : \mathbb{R}^n \to \mathbb{R}_{\geq 0}$ satisfying the three conditions:

- $\|x\| = 0 \iff x = \vec{0}$,
- $\|cx\| = |c| \|x\| \ \forall x \in \mathbb{R}^n, \forall c \in \mathbb{R}$, and
- $\|x + y\| \leq \|x\| + \|y\| \ \forall x, y \in \mathbb{R}^n$.

A few commonly used vector norms:

$$\|x\|_1 = \sum_{k=1}^{n} |x_k| \quad \text{(absolute sum norm)}$$

$$\|x\|_2 = \left(\sum_{k=1}^{n} x_k^2\right)^{\frac{1}{2}} = \sqrt{x^\top x} \quad \text{(Euclidean norm)}$$

$$\|x\|_\infty = \max_k |x_k| \quad \text{(maximum norm)}$$

$$\|x\|_W = \sqrt{x^\top Wx}, W \text{ s.p.d matrix} \quad \text{(weighted Euclidean norm)}$$

Let $W = L^\top L$ be the Cholesky decomposition of the s.p.d. matrix $W$, $x \in \mathbb{R}^n$. Then the weighted vector norm can be written in a Euclidean norm form:

$$\|x\|_W = \sqrt{x^\top Wx} = \sqrt{(Lx)^\top (Lx)} = \|Lx\|_2$$

Therefore the weighted Euclidean norm is a vector norm.
A.9.2 Matrixnorm

A matrixnorm is a function $||·|| : \mathbb{R}^{n \times n} \rightarrow \mathbb{R}_{\geq 0}$ satisfying the three conditions:

- $||A|| = 0 \Leftrightarrow A = 0_{n \times n}$,
- $||cA|| = |c| ||A|| \forall A \in \mathbb{R}^{n \times n}, \forall c \in \mathbb{R}$, and
- $||A + B|| \leq ||A|| + ||B|| \forall A, B \in \mathbb{R}^n$.

A few examples for vector norms:

$||A||_1 = \max_j \sum_i |A_{ij}|$ (maximal column sum)

$||A||_2 = \sqrt{\rho(A^\top A)}$ (spectral norm)

$||A||_\infty = \max_i \sum_j |A_{ij}|$ (maximal row sum)

$||A||_F = \sqrt{\text{trace}(A^\top A)} = \sqrt{\sum_i \sum_j A_{ij}^2}$ (Frobenius norm)

$||A||_W = \sqrt{\text{trace}(A^\top WA)}$ (weighted Frobenius norm)

All the examples above are multiplicative, i.e., $||AB|| \leq ||A||||B||$. Furthermore the spectral norm and the Frobenius norm are compatible with the Euclidean vector norm, i.e., $||Ax||_2 \leq ||A||_2||x||_2$ and $||Ax||_2 \leq ||A||_F||x||_2$.

A.10 Matrix Calculus

For all the block algorithm techniques matrix derivatives are required. Here we list all matrix derivatives with respect to a matrix $X$ used in this paper. Following the Matrix Cookbook [23], we define the matrix derivative using the partial derivative

$$
\left(\frac{\partial F(X)}{\partial X}\right)_{ij} = \frac{\partial F(X)}{\partial X_{ij}}
$$

where $F$ is a function. We conclude the following derivatives:

$$
\frac{\partial}{\partial X} \text{trace}(X^\top A) = \text{trace}(A^\top X) = A \quad \text{(A.5)}
$$

$$
\frac{\partial}{\partial X} \text{trace}(X^\top AX) = AX + A^\top X \quad \text{(A.6)}
$$

$$
\frac{\partial}{\partial X} \det(X) = \det(X) (X^{-1})^\top \quad \text{(A.7)}
$$
Proof. of (A.5). For the partial derivative, we expand the trace representation:
\[
\frac{\partial}{\partial X_{ij}} \text{trace}(X^\top A) = \frac{\partial}{\partial X_{ij}} \sum_k (X(:, k)^\top A(:, k)) = \frac{\partial}{\partial X_{ij}} \sum_k \sum_l (X_{lk} A_{lk}) = A_{ij}
\]

Proof. of (A.6). For the partial derivative, we expand the trace representation:
\[
\frac{\partial}{\partial X_{ij}} \text{trace}(X^\top AX) = \frac{\partial}{\partial X_{ij}} \sum_k (X(:, k)^\top AX(:, k)) = \frac{\partial}{\partial X_{ij}} X(:, j)^\top AX(:, j)
\]
\[
= \frac{\partial}{\partial X_{ij}} \sum_l \sum_p X_{lj} A_{lp} X_{pj} = 2A_{ii} X_{ij} + \sum_{p \neq i} A_{ip} X_{li} + \sum_{l \neq i} A_{li} X_{lj} = A(i, .) X(:, j) + A(:, i)^\top X(:, j)
\]

Proof. of (A.7). For the partial derivative \(\frac{\partial \det(X)}{\partial X_{ik}}\), we use the Cofactor expansion of \(\det(X)\) (see (A.1)) expanding the \(i\)th row:
\[
\det(X) = \sum_j X_{ij} C_{ij}
\]

Then we can calculate the partial derivative:
\[
\frac{\partial \det(X)}{\partial X_{ik}} = \frac{\partial}{\partial X_{ik}} \sum_j X_{ij} C_{ij} = C_{ik}
\]

With Cramer’s rule (A.3)
\[
X^{-1} = \frac{1}{\det(X)} C^\top,
\]
we get the simple equation
\[
\frac{d}{dX} \det(X) = C = \det(X) (X^{-1})^\top.
\]
B Inequalities

This chapter is a summary of all inequalities used in this thesis. It includes standard analysis inequalities, that can be found in any book on analysis (e.g. [20, 22]) and inequalities for positive definite matrices.

**Theorem B.1** (Inequality of arithmetic and geometric means). Let \( x_1, \ldots, x_n \) be nonnegative numbers. Then

\[
\sqrt[n]{\prod_{k=1}^{n} x_k} \leq \frac{1}{n} \sum_{k=1}^{n} x_k \tag{B.1}
\]

**Theorem B.2** (Bernoulli's inequality). Consider \( x \in \mathbb{R} \) with \( x \geq -1 \) and \( n \in \mathbb{N} \). Then

\[
(1 + x)^n \geq 1 + nx \tag{B.2}
\]

### B.1 Inequalities for Positive Definite Matrices

If \( A \) is s.p.d. and \( B \) is a positive semidefinite \( n \times n \) matrix, then the following inequalities hold:

- \( \text{trace}(A) > 0, \text{trace}(B) \geq 0 \) \hfill (B.3)
- \( \det(A) > 0, \det(B) \geq 0 \) \hfill (B.4)
- \( \text{trace}(AB) \geq 0 \) \hfill (B.5)
- \( \text{trace}(AB) > 0, \text{if } B \text{ is s.p.d.} \) \hfill (B.6)
- \( \sqrt[n]{\det(B)} \leq \frac{1}{n} \text{trace}(B) \) \hfill (B.7)
- \( \det(AB) \leq \left( \frac{1}{n} \text{trace}(AB) \right)^n \) \hfill (B.8)
- \( \frac{a_{ij}^2}{a_{ii}a_{jj}} < 1 \) \hfill (B.9)

*Proof.* of (B.3) and (B.4). All eigenvalues of \( A \) are positive. As \( \text{trace}(A) \) is the sum and \( \det(A) \) is the product of the eigenvalues, we see (B.3) and (B.4). The analogue conclusion can be done with \( B \). □

*Proof.* of (B.5). Let \( A = L^\top L \) be the Cholesky factorization of \( A \). Then

\[
\text{trace}(AB) = \text{trace}(L^\top LB) = \text{trace}(LBL^\top) \stackrel{(B.3)}{\geq} 0.
\]

Analogues, we can conclude (B.6). □

*Proof.* of (B.7)

Consider the trace and the determinant as the sum and the product of the eigenvalues of \( B \). Using (B.1) on the eigenvalues leads to (B.7). □
Proof. of (B.8). Let \( A = L^\top L \) be the Cholesky factorization of \( A \). Then

\[
det(AB) = det(L^\top LB) = det(LBL^\top) \\
\leq \left( \frac{1}{n} \text{trace}(LBL^\top) \right)^n = \left( \frac{1}{n} \text{trace}(L^\top LB) \right)^n \\
= \left( \frac{1}{n} \text{trace}(AB) \right)^n
\]

\( \square \)

Proof. of (B.9). Consider the s.p.d. submatrix

\[
A' = \begin{pmatrix} a_{ii} & a_{ij} \\ a_{ij} & a_{jj} \end{pmatrix}
\]

Rearranging the positive determinant results in the inequality (B.9):

\[
det(A') = a_{ii}a_{jj} - a_{ij}^2 > 0 \iff \frac{a_{ij}^2}{a_{ii}a_{jj}} < 1
\]

\( \square \)
References


