Sparse Approximate Inverse Methods

Bachelor thesis in Applied Mathematics
July 2013
Student : K. van Geffen
Primary Supervisor : dr. B. Carpentieri
Secondary Supervisor : prof. dr. H. Waalkens
Abstract
In undergraduates numerical mathematics courses I was strongly warned that inverting a matrix for computational purposes is generally very inefficient. Not only do we have to do more computation than factorization, but we also lose sparsity in the matrix. However, doing a search in the literature, I found that for many computational settings the inverse, although dense, may contain many small entries that can be dropped. As a result we approximate the inverse by a sparse matrix. Techniques for constructing such a sparse approximate inverse can be effectively used in many applications of numerical analysis, e.g. for preconditioning of linear systems and for smoothing multigrid methods. I describe some of the most popular algorithms and through some theory, numerical experiments and examples of applications, I show that sparse approximate inverse methods can be competitive, sometimes even superior, to standard factorization methods based on incomplete LU decomposition.

Keywords: Sparse approximate inverse, Preconditioning, Krylov subspace methods, Iterative methods
Acknowledgements

I would like to gratefully thank my primary supervisor dr. B. Carpentieri for the supervision during this work, the research of this thesis would not have been possible without his expertise on the subject. I would also like to thank my primary supervisor for providing me with an abundance of the needed material and for his extensive feedback on the work during the process. Thanks to his guidance the writing of this thesis has been a great learning experience.
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>Software libraries and packages</td>
<td>34</td>
</tr>
<tr>
<td>5.1.1</td>
<td>PETSc</td>
<td>35</td>
</tr>
<tr>
<td>5.1.2</td>
<td>SPAI</td>
<td>35</td>
</tr>
<tr>
<td>5.1.3</td>
<td>ParaSails</td>
<td>36</td>
</tr>
<tr>
<td>5.1.4</td>
<td>HSL MI12</td>
<td>37</td>
</tr>
<tr>
<td>6</td>
<td>Numerical experiments</td>
<td>39</td>
</tr>
<tr>
<td>6.1</td>
<td>Comparative study on SPAI and ILUT preconditioners</td>
<td>39</td>
</tr>
<tr>
<td>6.2</td>
<td>Electromagnetic Scattering - the Satellite Problem</td>
<td>43</td>
</tr>
<tr>
<td>6.2.1</td>
<td>Spectral deflation</td>
<td>45</td>
</tr>
<tr>
<td>7</td>
<td>Conclusion</td>
<td>49</td>
</tr>
<tr>
<td></td>
<td>References</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>Appendices</td>
<td>51</td>
</tr>
<tr>
<td>A</td>
<td>ILUT algorithm</td>
<td>51</td>
</tr>
<tr>
<td>B</td>
<td>MATLAB code for GMRES(m)</td>
<td>51</td>
</tr>
</tbody>
</table>
1 Introduction

In my thesis I review state-of-the-art techniques for computing sparse approximate inverses. That is, we consider methods for constructing a sparse approximation to the inverse of some matrix. These techniques can be effectively used in many applications of numerical analysis, e.g. for preconditioning of linear systems and for smoothing multigrid methods. For the purpose of this thesis we are mainly interested in linear systems, the problems we address to are of the form

\[ Ax = b, \]  

(1.1)

where \( A \) is an \( n \times n \) large and sparse matrix with real entries and \( b \) is a given, real, right-hand side. Sparse linear systems are ubiquitous in computational science. They arise, for instance, in the numerical solution of partial differential equations, of inverse problems and in optimization. Direct methods are in general very robust, but they are not feasible for large systems; they are expensive in terms of both work and memory. Modern iterative methods, namely Krylov subspace methods, can solve the bottleneck of memory, but it is now well established that convergence might be slow when the coefficient matrix \( A \) is highly nonsymmetric and/or indefinite. With the assistance of a preconditioner, we could try to transform system (1.1) into an equivalent system which is more amenable to an iterative solver.

Well-known standard preconditioning techniques compute a nonsingular approximation \( M \) to the matrix \( A \) and the equivalent system might for example be \( M^{-1}Ax = M^{-1}b \). Usually the construction of \( M \) is based on an incomplete LU decomposition. In contrast, we could construct a preconditioner \( M \) that directly approximates the inverse matrix, i.e. \( M \approx A^{-1} \). In this case the equivalent system might be given by \( MAx = Mb \). From numerical experiments it is observed that the inverse of sparse matrix is typically dense, but a lot of the entries are of small magnitude. This justifies the motivation to approximate the inverse by a sparse matrix. In the last decade, a significant amount of research has been devoted to develop such preconditioning techniques.

There are several advantages of sparse approximate inverse techniques. The most obvious one is that the preconditioning operation reduces to forming one (or more) matrix-vector products, whereas for standard techniques we require a linear solve. Hence, approximate inverse methods are generally less prone to instabilities. Another issue we address to in this thesis is that sparse approximate inverses are suitable for a parallel environment; matrix-vector products are nowadays already efficiently implemented in parallel, but in many cases the constructing phase of the sparse approximate inverse preconditioner is implemented in parallel as well. The standard techniques, however, are highly sequential. Other issues we address to in this thesis are: pattern selection, the use of factorized or non-factorized preconditioners and numerical software packages.

The structure of the thesis is as follows. In Section 2 we give a theoretical background on Krylov subspace methods. In Section 3 we discuss the concept of preconditioning and we review some well-known standard preconditioning techniques. The most popular sparse approximate inverse techniques are described in Section 4. Next, in Section 5, we will see what software packages are provided for some of the described algorithms. In Section 6 we conduct some numerical experiments and we will see an application of sparse approximate inverse in a practical implementation. Finally, in Section 7 we draw some conclusions arising from the work.
2 Krylov Methods

In the field of iterative techniques for solving linear systems, Krylov methods are nowadays among the most important. Many well known algorithms, such as the Generalized Minimum Residual (GMRES), Conjugate Gradient (CG), Full Orthogonalization Method (FOM), Lanczos Method, Biconjugate Gradient (Bi-CG) and Biconjugate Stabilized (Bi-CGSTAB) are based on projection techniques on Krylov subspaces [15]. Depending on the properties of the system, one method tends to be more efficient than the other. Later on we give an overview of these methods and tell for what kind of systems each particular method is suitable. But first we will give a theoretical background on Krylov methods and show that Krylov methods can be introduced as a generalization of Richardson iterations. Recall that the standard Richardson iteration is based on the matrix splitting \( A = I - (I - A) \). Similar techniques that are based on matrix splitting are the Jacobi and Gauss-Seidel method. As it turns out, Krylov methods are a generalization of these type of methods as well.

2.1 Theoretical background on Krylov Methods

In this theoretical background we do not concern ourselves with technical details that come into play when we consider numerical applications, nor do we go into detail on the concept of preconditioning yet. For now we will just focus on the idea behind the Krylov methods. This means that the following is based on exact arithmetic.

Now, as in all iterative methods, we start with an initial guess \( x^{(0)} \). For the purpose of this thesis we assume that there is no information available on the solution \( x \) to the system \( Ax = b \) and hence we will always choose \( x^{(0)} = 0 \). Krylov methods search in the \( k \)th iteration for an approximated solution \( x^{(k)} \) in the Krylov subspace generated by the vector \( b \), which is defined by

\[
K_k(A, b) := \text{span}\{b, Ab, \ldots, A^{k-1}b\}. \tag{2.1}
\]

The subspace is usually simply referred to by \( K_k \). At first glance it is not clear why it makes sense to search for a solution in a Krylov subspace and it is certainly not right away clear why a Krylov method has the opportunity to converge fast. Ilse C.F. Ipsen and Carl D. Meyer tried to answer these questions in an article on Krylov methods [12].

In this theoretical background on Krylov subspace methods, we will follow their reasoning starting with the nonsingular case. After that we consider the singular case and give some restrictions to \( b \) that need to hold in order for a Krylov solution \( x \) to exist. We will always make a distinction between regular solutions and Krylov solutions. We already know that a regular solution exists if and only if \( b \) is in the range of \( A \), i.e. \( b \in R(A) \). We will see that for singular matrices we have to meet slightly different restrictions for the existence of Krylov solutions.

\[\text{In other literature you may find that instead of the vector } b, \text{ the Krylov subspace is generated by the vector } r^{(0)} := b - Ax^{(0)}. \text{ This is a generalization for methods that start with a nonzero vector.}\]
2.1.1 The nonsingular case

When $A$ is nonsingular we know that the unique solution is given by $x = A^{-1}b$. We wish to find the minimal integer $k$ such that $x \in \mathcal{K}_k$. The following definition will help us to find this integer $k$.

**Definition 2.1** The minimal polynomial $q(t)$ of $A$ is the unique monic polynomial of least degree such that $q(A) = 0$.

As a consequence of the Cayley-Hamilton theory, the degree of $q(t)$ cannot exceed $n$. Using the fact that every matrix $A$ is similar to a Jordan matrix $J$, i.e. $A = XJX^{-1}$, we can construct the minimal polynomial $q(t)$ from $J$. This is because $0 = q(A) = q(XJX^{-1}) = Xq(J)X^{-1}$ and hence $q(J) = 0$. Let $A$ have $d$ distinct eigenvalues $\lambda_1, \ldots, \lambda_d$, then to each $\lambda_j$ we can assign an index $m_j$ which is the size of the largest Jordan block associated to $\lambda_j$. This allows us to determine the minimal polynomial of $A$:

$$m := \sum_{j=1}^{d} m_j \quad \text{and} \quad q(t) = \prod_{j=1}^{d} (t - \lambda_j)^{m_j}$$

where $q(t)$ is of degree $m$. Notice that this equation holds for singular matrices as well. However, only when $A$ is nonsingular we can use this polynomial to express $A^{-1}$ in terms of a finite sum of powers from $A$. First we write $q(t)$ as follows:

$$q(t) = \sum_{j=0}^{m} \alpha_j t^j$$

where $\alpha_0 = \prod_{j=1}^{d} (-\lambda_j)^{m_j}$. To express the inverse of $A$ in terms of powers of $A$, we proceed as follows:

$$0 = q(A) = \alpha_0 I + \alpha_1 A + \ldots + \alpha_m A^m \quad \text{and hence} \quad A^{-1} = -\frac{1}{\alpha_0} \sum_{j=0}^{m-1} \alpha_{j+1} A^j.$$  \hspace{1cm} (2.3)

In case $A$ is singular we notice $\alpha_0 = 0$ and the inverse of eqn (2.3) is not well defined. For nonsingular matrices we notice $x = A^{-1}b$ and hence $x \in \mathcal{K}_m$. We formulate the result in the following theorem.

**Theorem 2.2** Let $q(t)$ be the minimal polynomial of degree $m$ to the nonsingular matrix $A$, then $\mathcal{K}_m$ is the smallest Krylov subspace that contains the exact solution $x = A^{-1}b$.

The minimal polynomial gives us some insight on why it makes sense to look for a solution in a Krylov subspace. Also, when $A$ has a low degree minimal polynomial, we might expect fast convergence. Next we will see what the Krylov method computes to singular systems.

2.1.2 The singular case

As said before, every matrix is similar to a Jordan matrix. For this reason we confine this discussion to Jordan matrices, because with a little extra work one could generalize the idea using a similarity transformation. The Jordan matrix is unique up to a permutation of the
diagonal blocks of which it exists. Moreover, Jordan blocks from zero eigenvalues are nilpotent; a \( k \times k \) Jordan block from a zero eigenvalue is a nilpotent matrix of index \( k \). So we can even confine this discussion a little more to Jordan matrices \( A_J \) of the form:

\[
A_J = \begin{pmatrix} C & 0 \\ 0 & N \end{pmatrix}
\]  

(2.4)

where \( C \) is a nonsingular Jordan matrix and \( N \) is a singular nilpotent matrix of index \( i \) (meaning that \( i \) is the smallest positive integer such that \( N^i = 0 \), of course \( i \) is equal to the index of the zero eigenvalue of \( A \)). To determine whether a Krylov solution exist, we need the following lemma which is stated without proof (see [12]).

**Lemma 2.3** For the nilpotent system \( Nx = c \), a Krylov solution for a nonzero right-hand never exists, even when a regular solution does exist. ■

With this in mind, we try to find a Krylov solution to the system \( A_J x = b \) where \( A_J \) is of the form (2.4). First partition both \( x \) and \( b \) as follows

\[
x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \quad \text{and} \quad b = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}
\]

such that \( Ax = b \) implies \( Cx_1 = b_1 \) and \( Nx_2 = b_2 \). The second is a nilpotent system and we know that \( b_2 = 0 \) must hold in order for a Krylov solution \( x_2 \) to exist, i.e. \( b = (b_1, 0)^T \). Also notice:

\[
A^i = \begin{pmatrix} C & 0 \\ 0 & N \end{pmatrix}^i = \begin{pmatrix} C^i & 0 \\ 0 & N^i \end{pmatrix} = \begin{pmatrix} C^i & 0 \\ 0 & 0 \end{pmatrix}
\]

From this we conclude that for singular Jordan matrices, a Krylov solution can only exist if \( b \in \mathbb{R}(A^i) \). To see that this is also a sufficient condition we first consider the matrix \( C \). This matrix is nonsingular and hence we know that there exist a minimal polynomial \( q(t) \) of degree \( m - i \) and a corresponding polynomial \( p(t) \) of degree \( m - i - 1 \) such that \( p(C) = C^{-1} \) (\( m \) can not exceed \( n \)). Let \( b = (b_1, 0)^T \in \mathbb{R}(A^i) \), the solution satisfies:

\[
x = \begin{pmatrix} C^{-1} b_1 \\ 0 \end{pmatrix} = \begin{pmatrix} p(C) & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} b_1 \\ 0 \end{pmatrix} = \begin{pmatrix} p(C) & 0 \\ 0 & p(N) \end{pmatrix} \begin{pmatrix} b_1 \\ 0 \end{pmatrix} = p(A)b \in \mathcal{K}_{m-i}(A, b).
\]

Ipsen and Meyer proved in their article on Krylov methods that the Krylov solution can be found with a certain pseudo inverse, namely the Drazin inverse\(^2\). They also proved that this Krylov solution is unique. In stead of following the proof, we give the conclusive theorem that tells us something about the existence and uniqueness of a Krylov solution for singular systems.

**Theorem 2.4** For the singular system \( Ax = b \), a Krylov solution exists and is unique if and only if \( b \in \mathbb{R}(A^i) \) where \( i \) is the index of the zero eigenvalue of \( A \). Let \( m - i \) be the degree of the minimal polynomial of \( A \), then for the Krylov solution it holds that \( x = A^Pb \in \mathcal{K}_{m-i}(a, b) \) where \( A^P \) is the Drazin inverse of \( A \). If \( b \notin \mathbb{R}(A^i) \) then the system does not have a Krylov solution in \( \mathcal{K}_n(A, b) \). ■

In conclusion, whenever a linear system is solved with a Krylov subspace solver, either for nonsingular or singular systems, the solution that is found (if it exists) is unique and does not depend on the choice of \( x^{(0)} \).

\(^2\)For nonsingular matrices, the Drazin inverse generalizes to the ordinary inverse. This shows that there is a gradual transition from the singular to the nonsingular case.
2.1.3 Some remarks on Krylov methods

Recall the definition of the Krylov subspace from eqn (2.1). From the power method we know that as $k$ increases, the vector $A^{k-1}b$ becomes more and more part of the eigenspace that corresponds to dominant eigenvalue of $A$. Convergence to this eigenspace is of the order $O(|\lambda_2/\lambda_1|)$ where $\lambda_1$ and $\lambda_2$ are the dominant and second dominant eigenvalue. In general we might expect that, for $k$ sufficiently large, the set \{ $b, Ab, \ldots, A^{k-1}b$ \} becomes more and more linear dependent. From a numerical point of view this is not favorable and for that reason the Krylov space is always computed in a way such that the space is spanned by an orthogonal set. This is done by an algorithm called the Arnoldi method, see subsection 2.2.

As there exist an iteration step $m$ at which the exact solution is found, Krylov methods could be interpreted as direct methods. However, in practical implementations, one would like to terminate the process of iterating far before the $m$th step. Also, in general one is satisfied with an approximated solution as long as the approximation holds within reasonable bounds. Since we terminate the process after some $k << m$ steps, we will actually interpret Krylov subspace solvers as iterative techniques.

2.2 Arnoldi Method

We start of with a Krylov subspace $\mathcal{K}_i(A, b)$ at the $i$th iteration step and we assume that we know an orthogonal basis \{ $v_1, \ldots, v_i$ \} for it. In the next iteration step we would like to extend the Krylov subspace with another vector, but now we are only interested in extending the orthogonal basis. In order to do this we compute the vector $Av_i$ and orthogonalize it with respect to the vectors $v_1, \ldots, v_i$. The orthogonalized vector $v_{i+1}$ is computed in the following way:

$$h_{i+1,i}v_{i+1} = Av_i - \sum_{j=1}^{i} h_{j,i}v_j$$  \tag{2.5}

where $h_{i+1,i}$ is a constant chosen such that the vector $v_{i+1}$ is normalized. With the constants $h_{j,i} = (v_j, Av_i)$ it is ensured that $(v_j, v_{i+1}) = 0$ for $j = 1, \ldots, i$ and hence \{ $v_1, \ldots, v_{i+1}$ \} is an extended orthogonal basis for $\mathcal{K}_{i+1}$. The process is started with choosing $v_1 = b/\|b\|_2$.

Intuitively speaking, we can say that $v_1$ is the normalized part of $A^{i-1}b$ that is linear independent of span\{ $b, Ab, \ldots, A^{i-2}b$ \}. From the theory on Krylov methods we also know that the process terminates at step $m$, where $m$ is the degree of the minimal polynomial. So when we follow the Arnoldi method we can state:

$$\mathcal{K}_i(A, b) = \text{span} \{ v_1, \ldots, v_i \}, \quad \forall i \leq m$$  \tag{2.6}

For $i = m$ the Krylov subspace becomes invariant under $A$. Recall the definition on invariance.

**Definition 2.5** A set $\mathcal{V}$ is invariant under $A$ if for every $v \in \mathcal{V}$ it holds that $Av \in \mathcal{V}$. \hfill \blacksquare

There is a very compact form of notation for the Arnoldi method. When we define the $n \times i$ matrix $V_i$ which consist of the column vectors $v_1, \ldots, v_i$ and the $(i + 1) \times i$ upper Hessenberg matrix $H_{i+1,i}$ where the elements $h_{k,l}$ are defined by the Arnoldi method, then from eqn (2.5) it is clear that the following holds:

$$AV_i = V_{i+1}H_{i+1,i}$$  \tag{2.7}

This form of notation will be very helpful in understanding certain Krylov methods.
2.3 Generalization on Richardson iterations

We will now consider iterative solvers that are based on matrix splitting. It will become clear that these solvers indirectly search for an approximated solution in some Krylov subspace of increasing dimension in each iteration step. As an example we first consider the Richardson iteration, based on the standard splitting

\[ A = I - (I - A) \]

The corresponding iteration is given by:

\[ x^{(i)} = b + (I - A)x^{(i-1)} + r^{(i-1)} \quad \text{where} \quad r^{(i-1)} := b - Ax^{(i-1)} \quad (2.8) \]

and an iterative scheme for the residual is given by:

\[ r^{(i)} = (I - A)r^{(i-1)} = (I - A)^i r^{(0)} \quad (2.9) \]

In general, matrix splitting is of the form \( A = N - P \) where \( N \) is considered to be some approximation of \( A \) for which a linear system of the form \( Ny = d \) is easily solvable and requires low costs on operations and memory. This idea is also on the basis of preconditioning and we will return to this subject later.

For the Richardson iteration we have chosen \( N = I \) and we hope that the polynomial \( P_i(A) = (I - A)^i \) converges to zero reasonably fast. Of course in most cases, \( I \) will be a poor approximation of \( A \). Alternatives are the Jacobi and Gauss-Seidel methods, where one chooses \( N \) to be the diagonal part and the lower triangular part of \( A \) respectively. Whenever \( N \) is chosen to be unequal to identity, we can view \( N \) as an preconditioner on the system \( Ax = b \). Defining \( B := N^{-1}A \) and \( c := N^{-1}b \) the system to be solved becomes \( Bx = c \) and the splitting on \( A \) with \( N \neq I \) becomes equivalent with the standard splitting on \( B \) to the transformed system. Hence, if the Richardson Iteration generalizes to Krylov subspace solver, all iterative solvers based on matrix splitting do. See e.g. [9, Chapter 7].

We again assume \( x^{(0)} = 0 \) and hence \( r^{(0)} = b \). Using equations (2.8) and (2.9) we obtain:

\[ x^{(i)} = b + r^{(1)} + \ldots + r^{(i-1)} = \sum_{i=0}^{i-1} (I - A)^i b \]

from which we conclude

\[ x^{(i+1)} \in \text{span}\{b, Ab, ..., A^{i-1}b\} = K_{i-1}(A, b) \]

As it seems, Richardson iteration approximates solutions to the linear system indirectly from a Krylov subspace of increasing dimension. In Krylov subspace methods we compute the best approximation to the exact solution that is contained in the Krylov subspace of increasing dimension. Hence, Krylov methods are more efficient than algorithms that are based on matrix splitting.
2.4 Krylov subspace methods

In Krylov subspace methods we are looking for a solution $x^{(k)} \in K_k$ that is in some sense the best approximation to the solution $x$. Finding a solution in a subspace is done by projection and there are different types of projection that we could use. In all projections we are dealing with some sort of minimization problem. Depending on the problem, one type of minimization may be numerical more favorable then the other. We consider four types of projections [9, Chapter 8]:

1. The Ritz-Galerkin approach requires that
   \[ b - Ax^{(k)} \perp K_k(A, b) \]  
   \[ (2.10) \]
2. The minimum residual approach requires:
   \[ \|b - Ax^{(k)}\|_2 \text{ to be minimal over } K_k(A, b) \]  
   \[ (2.11) \]
3. The Petrov-Galerkin approach requires that:
   \[ b - Ax^{(k)} \text{ is orthogonal to some other suitable } k\text{-dimensional subspace} \]  
   \[ (2.12) \]
4. The minimum error approach requires:
   \[ \|x - x^{(k)}\|_2 \text{ to be minimal over } A^T K_k(A^T, b) \]  
   \[ (2.13) \]

We will treat briefly some (aforementioned) popular methods. Methods as CG, the Lanczos method and FOM rely on the Ritz-Galerkin approach. The minimum residual approach has lead to methods as GMRES and MINRES. The Petrov-Galerkin approach gives rise to methods like Bi-CG and BI-CGSTAB. And at last, a method that relies on the minimum error approach is for example GMERR.

2.4.1 Ritz-Galerkin projection

We begin with a formulation of the the solution to the best approximation in a general sense: since $x^{(k)} \in K_k = R(V_k)$, see eqn (2.7), we know that there exist $y \in \mathbb{R}^k$ such that:

\[ x^{(k)} = V_k y. \]  
\[ (2.14) \]

When we have constructed a basis \{v_1, \ldots, v_k\} the best approximation is determined by finding $y$. This will hold in general for Krylov methods, but the solution we may find depends on the type of projection we choose. We will begin with the Ritz-Galerkin approach, recall that for this approach we require that $b - Ax^{(k)} \perp K_k(A, b)$. The most common method based on this approach is the Full Orthogonalization Method (FOM).

Full Orthogonalization Method

In FOM we do not require $A$ to possess any special properties and we can just apply the Ritz-Galerkin projection. Hence, we find that $y$ must satisfy the following condition:

\[ V_k^T (b - Ax^{(k)}) = V_k^T (b - AV_k y) = 0. \]
From Arnoldi Method we know that \( \mathbf{b} = \| \mathbf{b} \|_2 \mathbf{v}_1 \) and hence \( V_k^T \mathbf{b} = \| \mathbf{b} \|_2 \mathbf{e}_1 \), where \( \mathbf{e}_1 \) is the first canonical unit vector in \( \mathbb{R}^k \). From Arnoldi Method we also know that \( V_k^T \mathbf{AV}_k = V_k^T V_{k+1} H_{k+1,k} \). If we combine this results with the Ritz-Galerkin projection, we find that \( y \) is the solution of the following linear system:

\[ H_{k,k} y = \| \mathbf{b} \|_2 \mathbf{e}_1 \]

This system can be quite efficiently solved by performing \( k - 1 \) Givens rotations on the diagonal- and subdiagonal entries of \( H_{k,k} \), after which we need to solve a lower triangular system. In essence we are performing an efficient \( QR \) factorization on the Hessenberg matrix.

As it turns out this method becomes significantly more efficient when \( A \) is symmetric, it is known as the Lanczos method. If \( A \) is symmetric positive definite (SPD) there are some additional numerical benefits which lead to the even more efficient Conjugate Gradient method (CG).

**Lanczos method**

To see why a symmetric matrix leads to a more efficient algorithm, consider the matrix \( V_k^T \mathbf{AV}_k \). As we know it equals an upper Hessenberg matrix with entries we know from Arnoldi’s method, moreover we see that it is symmetric and we conclude that the Hessenberg matrix is tri-diagonal.

The corresponding notation is as follows: \( V_k^T \mathbf{AV}_k = T_{k,k} \). And now the best approximation is found by solving the tri-diagonal system:

\[ T_{k,k} y = \| \mathbf{b} \|_2 \mathbf{e}_1 \]

Applying Arnoldi’s method simplifies as well, since in the next iteration step \( \mathbf{Av}_k \) has to be orthogonalized with respect to only the previous two vectors \( \mathbf{v}_k \) and \( \mathbf{v}_{k-1} \). In practical implementations there are algorithms that prevent the necessity to store all the vectors \( \{\mathbf{v}_1, \ldots, \mathbf{v}_k\} \), therefore the cost in terms of computation and memory are reduced tremendously, especially when \( n \) is very large.

**Conjugate Gradient method**

There are different ways in which one can interpret the CG method. When we follow the above reasoning, the main benefit of the Conjugate gradient method is that we can solve the tri-diagonal even more efficiently by performing an LU factorization on \( T_{k,k} \). By symmetric positive definiteness we know that this factorization exist and the tri-diagonal form makes it possible to do this in an way that is low in computational cost. Another way of interpreting the CG method is that we try to minimize the \( A \)-norm of the error \( e^{(k)} \) in each step. We do this by making the error \( A \)-orthogonal to \( \{\mathbf{v}_1, \ldots, \mathbf{v}_k\} \), by using the \( A \)-norm we prevent that we actual have to compute any errors, instead the residuals are used in an elegant way.

### 2.4.2 The minimum residual approach

In the minimum residual approach we want to minimize \( \| \mathbf{b} - A x^{(k)} \|_2 = \| \mathbf{b} - \mathbf{AV}_k y \|_2 \) on the Krylov subspace \( \mathcal{K}_k \). Consider the following:

\[
\| \mathbf{b} - \mathbf{AV}_k y \|_2 = \| \mathbf{b} - \mathbf{V}_{k+1} H_{k+1,k} y \|_2 = \| \mathbf{V}_{k+1} (\| \mathbf{b} \|_2 \mathbf{e}_1 - H_{k+1,k} y) \|_2
\]

But as \( \mathbf{V}_{k+1} \) is orthogonal, the norm isn’t influenced by it. Hence we are interested in minimizing \( \| \| \mathbf{b} \|_2 \mathbf{e}_1 - H_{k+1,k} y \|_2 \) over the space \( \mathcal{K}_k \). This is done by determining the least squares solution of

\[ H_{k+1,k} y = \| \mathbf{b} \|_2 \mathbf{e}_1 \]
in the minimum norm sense. For general matrices this approach leads to the GMRES algorithm. For symmetric matrices, \( H_{k+1,k} \) reduces again to a tri-diagonal matrix. Similar to the Lanczos method, this results to a more efficient algorithm known by MINRES.

### 2.4.3 Petrov-Galerkin projection

For nonsymmetric matrices we notice that the Ritz-Galerkin approach becomes too expensive in both memory and computational cost. To overcome this problem, we would like to mimic, in some way, the 3-term recurrence that we encounter for symmetric matrices. We can do this with a Petrov-Galerking approach \((2.12)\): require that \( b - Ax^{(k)} \) is orthogonal to some other suitable \( k \)-dimensional subspace. This is what is done in the Bi-Lanczos and Bi-CG method.

To find the suitable subspace, we have to perform Arnoldi’s method in a slightly different way. The idea is that when we have constructed in some way \( V_i \), the suitable basis should satisfy

\[
W_i^T V_i = D_i := [d_{ii}],
\]

some diagonal matrix with diagonal entries \( d_{ii} := (w_i, v_i) \), and besides that it should satisfy \( W_i^T v_{i+1} = 0 \), i.e. \( W_i \) and \( V_i \) form a bi-orthogonal basis. Then it follows that:

\[
W_i^T AV_i = D_i H_{i,i}
\]

and we would like to choose \( W_i \) such that \( H_{i,i} \) is tri-diagonal. This can be realized by defining the bi-orthogonal basis sets \( \{v_1, \ldots, v_i\} \) and \( \{w_1, \ldots, w_i\} \) in the following manner:

\[
h_{i+1,i}v_{i+1} = Av_i - \sum_{j=1}^{i} h_{j,i}v_j
\]

\[
h_{i+1,i}w_{i+1} = A^T w_i - \sum_{j=1}^{i} h_{j,i}w_j
\]

where \( h_{i+1,i} \) is chosen such that \( v_{i+1} \) is normalized. Notice we generate \( w_i \) with the transpose of \( A \). With the constants \( h_{j,i} = (w_j, Av_i)/d_{j,j} \) for \( j = 1, \ldots, i \) it is ensured that the sets are bi-orthogonal, i.e. \((w_j, v_i) = 0 \) for \( i \neq j \). In matrix notation this gives:

\[
W_i^T AV_i = D_i H_{i,i}
\]

\[
V_i^T A^T W_i = D_i H_{i,i}
\]

Hence \( D_i H_{i,i} \) is symmetric and therefore tri-diagonal, notation: \( D_i T_{i,i} \). This leads to the required 3-term recurrence. The process is started by taking \( v_1 = b/\|b\|_2 \) and choosing some \( w_1 \neq 0 \) such that \((w_1, v_1) \neq 0 \).

As in the Lanczos method, the following equality holds \( AV_k = V_{k+1}T_{k+1,k} \). In the \( k^{th} \) iteration step, the Bi-Lanczos method performs the projection similar to the Lanczos method:

\[
W_k^T (b - Ax^{(k)}) = W_k^T (b - AV_k y)
\]

\[
= W_k^T b - D_k T_{k,k} y = 0
\]

Now consider the following: \( D^{-1}W_k^T b = D^{-1}W_k^T v_1 \|b\|_2 = \|b\|_2 e_1 \). We conclude that, similar as to the Lanczos method, we end up with a tri-diagonal system for \( y \) of the same form:

\[
T_{k,k} y = \|b\|_2 e_1
\]

This method is known as the Bi-Lanczos method. For symmetric matrices this leads to even shorter recurrences such as the Bi-CG and Bi-CGSTAB algorithms.
2.4.4 The minimum error approach

The minimum error approach was formulated as follows. We require:

$$\|x - x^{(k)}\|_2 \quad \text{to be minimal over} \quad A^T K_k(A^T, b).$$

With this approach we are, for some reasons that are beyond the scope of this thesis, able to minimize the forward error. Methods in this class are SYMMLQ and GMERR.
3 Preconditioning

Preconditioning is a technique that is commonly used to accelerate convergence. In practice many iterative methods, including Krylov subspace methods, converge very slow for unpreconditioned systems. By preconditioning the system we try to modify the system in such a way that an iterative method converges significantly faster. In this section we first treat the concept of preconditioning in general. Then we will treat some well-known preconditioning techniques that are based on incomplete factorizations.

3.1 The concept of preconditioning

In order to understand the concept of preconditioning we need to know what properties of the system causes the bad convergence behavior. Next we will see how these properties can be improved by a preconditioning technique. The eigenvalues of the matrix play an important role in this discussion. Recall the definition on the spectrum of a matrix.

Definition 3.1 The set of all the eigenvalues of $A$ is called the spectrum of $A$ and is denoted by $\sigma(A)$.

Another definition that will be useful in our discussion on preconditioning is the condition number of a matrix.

Definition 3.2 The condition number of a matrix $A$ is the quantity

$$K(A) = \|A\|\|A^{-1}\|$$

where $\|\cdot\|$ is any induced matrix norm.

In general the condition number $K(A)$ depends on the choice of the norm and it is not defined if $A$ is singular. The quantity plays an important role in studying the stability properties of linear systems.

Now recall from section 2 eqn (2.1) that for Krylov subspace methods $x^{(k)} \in K_k$. Hence for the residuals it follows that $r^{(k)} = b - Ax^{(k)} \in K_k$ as well. We could also write this as:

$$r^{(k+1)} = P_k(A)r^{(0)}$$

(3.1)

where $P_k$ is a polynomial with degree $k$ that satisfies $P_k(0) = 1$. Notice that in this thesis we will always assume $r^{(0)} = b$ as we assumed $x^{(0)} = 0$. In methods such as GMRES and MINRES we aim to minimize the residuals at each iteration, hence for these methods we have:

$$\|r^{(k)}\|_2 = \min_{P_k} \|P_k(A)r^{(0)}\|_2$$

(3.2)

where we minimize over all polynomials $P_k$ with degree $k$ or less with $P_k(0) = 1$. In a similar way we could derive that for the same class of polynomials the CG method satisfies

$$\|e^{(k)}\|_A = \min_{P_k} \|P_k(A)e^{(0)}\|_2$$

(3.3)

where $\|e^{(k)}\|_A := (e^{(k)}, Ae^{(k)})$ is a well defined $A$-norm for SPD matrices. For that reason we will first consider the case where $A$ is SPD. In this case there exist an orthogonal matrix $Q$ and
a diagonal matrix $D$ such that $A = QDQ^T$. Notice we could write $\|e^{(k)}\|_A = \|A^{1/2}e^{(k)}\|_2$. Then it follows from equations (3.2) and (3.3) that:

$$
\|r^{(k+1)}\|_2 = \min_{P_k}\|QP_k(D)Q^T r^{(0)}\|_2 \leq \min_{P_k}\|P_k(D)\| \cdot \|r^{(0)}\|_2
$$

and

$$
\|e^{(k+1)}\|_A = \min_{P_k}\|A^{1/2}P_k(A)e^{(0)}\|_2 = \min_{P_k}\|QP_k(D)Q^T A^{1/2}e^{(0)}\|_2
\leq \min_{P_k}\|P_k(D)\| \cdot \|e^{(0)}\|_A
$$

where $\|\cdot\|$ denotes some appropriate matrix norm. The inequality follows because:

$$
\min_{P_k}\|QP_k(D)Q^T w\|_2 \leq \|QP_k(D)Q^T w\|_2 \leq \|\hat{P}_k(D)\| \cdot \|w\|_2
$$

where we define $\hat{P}_k$ to be the polynomial that minimizes $\|P_k(D)\|$. We conclude:

$$
\|r^{(k+1)}\|_2/\|r^{(0)}\|_2 \leq \left( \min_{P_k} \max_{i=1,\ldots,n} |P_k(\lambda_i)| \right) \quad \text{for MINRES} \tag{3.4}
$$

and

$$
\|e^{(k+1)}\|_A/\|e^{(0)}\|_A \leq \left( \min_{P_k} \max_{i=1,\ldots,n} |P_k(\lambda_i)| \right) \quad \text{for CG.} \tag{3.5}
$$

It is not clear whether the bounds are sharp, since the polynomial that minimizes $\|QP_k(D)Q^{-1}w\|_2$ might not be the same as the one that minimizes $\|P_k(D)\|$. Nonetheless the upper bound provides us with qualitative information; a small upper bound corresponds to the case of fast convergence. Hence, the smaller the values of the minimizing polynomial $P_k$ (with $P_k(0) = 1$) on the set $\sigma(A)$ are, the faster convergence we may expect. For SPD matrices we can simplify the bounds of equations (3.4) and (3.5). It can be shown that [10]:

$$
\left( \min_{P_k} \max_{i=1,\ldots,n} |P_k(\lambda_i)| \right) \leq 2 \left( \frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} \right)^k \quad \kappa := \lambda_{\min}/\lambda_{\max} \tag{3.6}
$$

and hence the bounds do not depend on the entire spectrum of $A$, but only on ratio of the largest to smallest eigenvalue of $A$. However, for nonsymmetric matrices we could derive for the GMRES method a similar bound as in eqns (3.4). Assume for simplicity that $A$ has a complete set of eigenvalue, i.e. there exist a non singular matrix $V$ and a diagonal matrix $D$ such that $A = VDV^{-1}$. Then the bound is given by:

$$
\|r^{(k+1)}\|_2/\|r^{(0)}\|_2 \leq K(V) \left( \min_{P_k} \max_{i=1,\ldots,n} |P_k(\lambda_i)| \right) \quad \text{for GMRES} \tag{3.7}
$$

where $K(V) = \|V\|\|V^{-1}\|$ is the condition number. In this case we expect the bound to depend on the entire spectrum of $A$. We can understand this by analyzing the following scenarios:

- As mentioned before we require $k \ll n$ so that we think of $P_k$ as a low degree polynomial. Consequently, when the eigenvalues are widely spread in the complex plane, the polynomial cannot be small at a large number of such points.

- Similarly, when a large number of eigenvalues are located around the origin, then a polynomial of low degree cannot be 1 at the origin and have small values at a large number of such points located around the origin.
• In contrast, eigenvalues clustered around a single point \( c \) away from the origin are favorable. Take for example the polynomial \( P_k(z) = (1 - z/c)^k \), hence \( P_k(0) = 1 \) and the polynomial has small values for \( z \in \sigma(A) \). Note that we do not claim that this is the minimizing polynomial. Matrices with clustered eigenvalues will typically show good convergence behavior.

It appears that a clustered spectrum is in general good for convergence. Also for SPD matrices, the bound of equations (3.6) is reduced when the eigenvalues are clustered. Hence, it is of no surprise that the main goal of preconditioning is to transform the system with a preconditioning matrix (or preconditioner) \( M \) such that the eigenvalues of the preconditioned system are clustered.

We can transform the system by applying the preconditioner to the original system. We will treat three different types of preconditioning:

1. **Left-preconditioning.** Apply the iterative method to:
   \[
   M^{-1}Ax = M^{-1}b \tag{3.8}
   \]

2. **Right-preconditioning.** Apply the iterative method to:
   \[
   AM^{-1}u = b, \quad x := M^{-1}u \tag{3.9}
   \]

3. **Two-sided preconditioning.** In many applications the preconditioner is constructed in factored form \( M = M_1M_2 \). In this case we can apply apply the iterative method to:
   \[
   M_1^{-1}AM_2^{-1}u = M_1^{-1}b, \quad x := M_2^{-1}u \tag{3.10}
   \]

Whenever we precondition a system, we are in essence solving a different system. For example, when we are left-preconditioning a system we are solving a system \( M^{-1}Ax = M^{-1}b \). From the Arnoldi method we know that in the first step we have to compute the vector \( v_1 = M^{-1}b/\|M^{-1}b\|_2 \), hence we need to compute the vector \( e = M^{-1}b \). We do not want to determine the inverse of \( M \) explicitly as this is likely to be too expensive. Instead we determine \( c \) by solving the system \( Mc = b \) indirectly. From eqn (2.5) we know that in the following steps we have to solve systems of the form \( M^{-1}Au = v \). This is done in two steps, first we apply the matrix-vector product \( u_1 = Au \) and next we solve \( Mu = u_1 \) indirectly. Here the subscript on \( u_1 \) is to emphasize that it is a dummy variable.

Hence, in order to be useful for numerical applications we require that the system \( Mx = b \) is much easier solved than the original system \( Ax = b \). Note that we will almost never determine \( M^{-1} \) explicitly. This is with the exception of preconditioning techniques where \( M \) is a sparse approximate inverse preconditioner, i.e. the preconditioner is a sparse matrix such that \( M \approx A^{-1} \). In this case left-preconditioning is of the form \( MA = Mb \) and now in each step we have to perform an additional matrix-vector product in stead of the additional linear solve we had to perform before. Matrix-vector products are, especially for sparse matrices, in general much cheaper to perform than linear solves. The construction of sparse approximate inverses is however less straightforward and we will treat this in the next section more extensively.

Another issue of preconditioning is that \( M \) should be chosen such that the preconditioned system is better conditioned; the condition number is directly related to the stability of a linear system. Recall the definition on the forward error.
Definition 3.3 Let $x$ be such that $Ax = b$ and let $x^*$ be an approximated solution to the system corresponding to $A$ and $b$. The forward error is the quantity $\delta x$ such that $x = x^* - \delta x$.

With a posteriori analysis we aim to find the nearby system of $Ax = b$ for which $x^*$ is the exact solution, i.e. we aim to find the minimal perturbations $\delta A$ and $\delta b$ such that:

$$(A + \delta A)(x + \delta x) = b + \delta b.$$ 

This is known as the backward error. Now assume for simplicity that $\delta A = 0$, hence $A(x + \delta x) = b - r$ where we defined the residual $r := b - Ax^* = A(x - x^*) = -A\delta x$. With a posteriori analysis it can be shown that the forward error is related to residual in the following way [14, Section 3.1]:

$$\frac{\|\delta x\|_2}{\|x\|_2} \leq K(A) \frac{\|r\|_2}{\|b\|_2} \tag{3.11}$$

where the condition number has the property $K(A) \geq 1$. We say that the system is ill-conditioned when the condition number is relatively large. In general, a good preconditioner $M$ will improve the condition number. That means $M$ is chosen such that $K(M^{-1}A) \ll K(A)$. In this case we expect from eqn (3.11) a more accurate approximated solution to the linear system.

3.1.1 Discussion on preconditioning

The concept of preconditioning is in theory quite straightforward, but in practical implementations there are a lot of technical details we have to take in account. In this discussion on preconditioning we discuss some of the major issues.

When we construct a preconditioner $M$ we have to make up a balance. A well chosen preconditioner will decrease the amount of iterations steps needed, but as a rule a good preconditioner is likely to be expensive to construct. Also, as we will see later, a very good preconditioner may contain more entries in comparison to other cheap preconditioners. In this case we also have to take in account the additional work needed per iteration step.

In general the amount of information we have from the spectrum of $A$ and that of the preconditioned system is limited. Therefore we can not state a priori much about the convergence behavior of the preconditioned system. For example, the spectrum of an indefinite systems may have eigenvalues on both sides of the complex plane. When we precondition such a system it may happen that the preconditioned matrix has eigenvalues close to zero. It can be shown that this is bad for convergence. Hence, improving a preconditioner can potentially be bad for convergence. This is merely to clarify that the construction of a preconditioner is not an exact science.

We discussed that convergence of Krylov methods depends for an important part on the spectrum of $A$, or on the spectrum of the preconditioned matrix when a preconditioner is applied. We have mentioned three types of preconditioning. Notice the following:

$$\det(M^{-1}A - \lambda I) = \det(M^{-1}(A - \lambda M)) = \det(A - \lambda M)\det(M^{-1}) = \det(AM^{-1} - \lambda I).$$
In a similar fashion we can show the following for preconditioners in factored form $M = M_1 M_2$:

$$
\det(M^{-1} A - \lambda I) = \det(M_2^{-1} M_1^{-1} A - \lambda I) \\
= \det(M_2^{-1} (M_1^{-1} A - \lambda M_2)) \\
= \det(M_1^{-1} A - \lambda M_2) \det(M_2^{-1}) \\
= \det(M_1^{-1} A M_2^{-1} - \lambda I).
$$

We conclude that spectrum of the preconditioned matrices does not depend on the type of implementation. However

$$M^{-1} A v = \lambda v \not\Rightarrow AM^{-1} v = \lambda v$$

and hence the set of eigenvectors does depend on the choice of implementation. As convergence also depends on the degree in which the initial residual $b - A x^{(0)}$ is in the dominant eigenvector direction, the convergence behavior will depend on the chosen implementation.

There are other issues we have to keep in mind when we apply left-, right- or two-sided preconditioning.

- When we are applying left-preconditioning this will have effect on the projection of an approximated solution on the Krylov subspace. For example, in the preconditioned GMRES algorithm we are minimizing the residual

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

which may differ significantly from the actual residual $(b - A x^{(k)})$ when $M$ is ill-conditioned, see eqn (3.11).

- When we are applying right-preconditioning the stopping criteria may be based on

$$\|u - u^{(k)}\|_2$$

which may differ significantly from the actual error $\|x - x^{(k)}\|_2 = \|M^{-1}(u - u^{(k)})\|_2$ when $M$ is ill-conditioned. A benefit from these type of preconditioning is that we do not transform the right-hand side of the the system, see eqn (3.9).

For applications in a general sense we distinguish two types of preconditioners. The first type is the problem-specific preconditioner for which construction is based on very specific information of the problem, such as its geometry and physical properties. Mostly these type of problems are related to applications involving partial differential equations (PDEs). For a narrow class of problems one may obtain good preconditioners in this way, however, this approach requires a deep understanding of the problem. For a broad range of problems this problem specific information is very difficult to exploit or might not even be available. In this case a preconditioner can be based purely on the information that is contained in the coefficient matrix $A$. These are called algebraic or general-purpose preconditioners. Preconditioner of these types are generally not as efficient as the problem-specific types, however, the achieved convergence can certainly be reasonable good.

Finally, one of the most important issues in the development of preconditioning techniques nowadays is the need for parallel processing. Krylov subspace methods are already efficiently implemented in parallel on high-performance computers. However, preconditioning is currently the main stumbling block in achieving high performance for large, sparse linear systems [2]. There is still a need for preconditioners which are inherently parallelizable both in construction as in implementation.
The characteristics of a good preconditioner are summarized:

- The preconditioner clusters the eigenvalues of $A$ to a point away from the origin.
- The preconditioner is (relatively) cheap to construct.
- The preconditioner does not demand much memory.
- Linear solves with the preconditioner are cheap to perform.
- The preconditioner improves the condition number of $A$.
- The preconditioner is parallelizable, both in construction as in implementation.

### 3.2 Preconditioning techniques

The most straightforward approach in preconditioning techniques is obtained by considering the exact solution by a direct method. A well-known direct method is the LU-factorization. The factors are constructed by the Gaussian elimination algorithm and the system to be solved is $LUx = b$ where $L$ and $U^T$ are lower triangular matrices. A preconditioning technique that is derived from this method is the incomplete LU-factorization (ILU). The method constructs a preconditioner

$$M = \tilde{L}\tilde{U}$$

from incomplete factors and tries to capture the biggest entries from $L$ and $U$.

#### 3.2.1 Incomplete LU-factorization

In an ILU factorization we force the factors to have some sparsity pattern. This is done by performing Gaussian Elimination and dropping elements from nondiagonal positions in a suitable way. The factorization is as follows:

$$A = \tilde{L}\tilde{U} - R$$

where $R$ is a residual matrix. In general ILU-factorizations we define a zero pattern $P$ such that

$$P \subset \{(i,j) \mid i \neq j; 1 \leq i, j, \leq n\}$$

and drop elements during Gaussian Elimination that are part of the zero pattern. Suppose we have predetermined the zero pattern $P$. In this case we perform a Static Pattern ILU-factorization. This is implemented in the following algorithm \cite{15}:

There are some remarks on the above algorithm:

- The process may terminate due to a zero pivot.
- In practice the for-loops are run more efficiently by exploiting the zero pattern $P$. When $P$ has some known structure we can implement this in the second and third for-loop such that we only evaluate these loops at a small amount of nonzero entries. In this way the algorithm can still be feasible for large $n$. 

16
Algorithm 3.1 General Static Pattern ILU

for each \((i, j) \in P\) do
  set \(a_{ij} = 0\)
end for

for \(k = 1, \ldots, n - 1\) do
  for \(i = k + 1, \ldots, n\) and if \((i, k) \notin P\) do
    \(a_{ik} = a_{ik}/a_{kk}\)
  end for
  for \(j = k + 1, \ldots, n\) and for \((i, j) \notin P\) do
    \(a_{ij} = a_{ij} - a_{ik} * a_{kj}\)
  end for
end for

\[\hat{\text{Since the diagonal elements of } \tilde{L} \text{ are all equal to one, we don’t need to store them. Hence we could overwrite } A \text{ during the process to store } \tilde{L} \text{ in the strict lower triangular part and } \tilde{U} \text{ in the upper triangular part. This is done with a variant of Gaussian elimination, called the IKJ variant. This is implemented in the algorithm 3.2. Although we could overwrite the matrix } A \text{ to store both factors } \tilde{L} \text{ and } \tilde{U}, \text{ this is in general not how the algorithm is implemented. Every iterative Krylov method requires to perform matrix-vector products with } A, \text{ see e.g. section 2.2 on the Arnoldi method. In practical implementations we need to allocate additional memory for storing the preconditioner.}}\]

Algorithm 3.2 General Static Pattern ILU, IKJ variant

for each \((i, j) \in P\) do
  set \(a_{ij} = 0\)
end for

for \(i = 2, \ldots, n\) do
  for \(k = 1, \ldots, i - 1\) and if \((i, k) \notin P\) do
    \(a_{ik} = a_{ik}/a_{kk}\)
  end for
  for \(j = k + 1, \ldots, n\) and for \((i, j) \notin P\) do
    \(a_{ij} = a_{ij} - a_{ik} * a_{kj}\)
  end for
end for

For general ILU-factorizations it can be shown that the following theorem holds, we will state it without proof.

Theorem 3.4 Algorithm 3.2 produces factors \(\tilde{L}\) and \(\tilde{U}\) such that

\[A = \tilde{L}\tilde{U} - R\]

and the entries of \(R\) are such that

\[r_{ij} = 0 \quad \text{when } (i, j) \notin P\]

and the elements \(-r_{ij}\) for \((i, j) \in P\) are determined by algorithm 3.2.
The nonzero entries $-r_{ij}$ for $(i, j) \in P$ are called fill-in elements. We conclude that for a preconditioner $M = \tilde{L}\tilde{U}$ we have $m_{ij} = a_{ij}$ for $(i, j) \notin P$. Hence, depending on the choice of the zero pattern, $M$ might be a good approximation of $A$. Matrix-vector products with ILU-preconditioner of the form $v = M^{-1}u$ are computed in two steps:

$$u = Mv = \tilde{L}\tilde{U}v$$

$$\implies \tilde{L}u_1 = u$$

$$\implies \tilde{U}v = u_1$$

where $u_1$ is a dummy variable. In practical implementations we overwrite $u_1$ when we are solving for $v$.

### 3.2.2 Zero Fill-In ILU (ILU(0))

For a complete LU factorization of a matrix $A$ the factors $L$ and $U$ are typically less sparse than the original matrix $A$. In ILU(0) we allow no fill-in, i.e. we take as a zero pattern $P$ the zero pattern of $A$:

$$P_A = \{ (i, j) | a_{ij} = 0, i \neq j \}$$

(3.15)

ILU(0) is implemented with this $P$ in algorithm 3.2. This technique ensures us that sparsity properties are maintained in the preconditioner.

### 3.2.3 ILU(p)

It is possible that ILU(0) produces a preconditioner that is not a good approximation to $A$. In ILU(p) we allow more fill-in in the factors $L_p$ and $U_p$ in order to produce a more accurate approximation of $A$. We will show how ILU(1) is performed and how this process generalizes to performing ILU(p) for $p > 1$. To this end we consider an example with a 5-point matrix that is derived from the finite difference discretization of Poisson’s equation on the unit square. Hence we consider the problem:

$$\left( \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} \right) u(x, y) = f(x, y)$$

(3.16)

for some function $f$ and $(x, y) \in [0, 1] \times [0, 1]$. We discretize the problem by applying a 5-point operator on a regular $n$-by-$n$ mesh on the unit square. In this way we obtain a block tridiagonal matrix.

In MATLAB we construct with the command $A = \text{gallery(’Poisson’, 4)}$ the matrix that corresponds to the considered problem. Next we compute the ILU(0) factorization with the command $[\tilde{L}_0, \tilde{U}_0] = \text{ilu}(A, \text{setup})$ where we have set $\text{setup.type} = \text{’nofill’}$. In figure 1 we represented the nonzero structure of the matrices $A$, $\tilde{L}_0$, $\tilde{U}_0$ and the product $\tilde{L}_0\tilde{U}_0$. Notice that there is indeed some additional fill-in in the product of the incomplete factors.

We can use the nonzero structure of the product $\tilde{L}_0\tilde{U}_0$ to allow more fill-in for some new to construct factors $L_1$ and $U_1$. This is what is done in ILU(1). We have performed ILU(1) on the matrix $A$ with algorithm 3.2 where we defined the zero pattern by the complement of the nonzero structure of $\tilde{L}_0\tilde{U}_0$. In figure 2 we represent sparsity patterns of the factors $L_1$, $U_1$ and the incomplete product $L_1U_1$. Notice that the additional fill-in has increased. However, with

---

3The nonzero structure, or sparsity pattern, of a matrix is a set of the row and column indices corresponding to the nonzero entries of the matrix.
Figure 1: Four matrices involved in ILU(0): $A$, $L_0$, $U_0$ and $L_0U_0$

the commands $\text{normest}(A-L_0*U_0)$ and $\text{normest}(A-L_1*U_1)$, where $L_1$ and $U_1$ correspond to the incomplete factors obtained by ILU(1), it is shown that the incomplete factorization obtained by ILU(1) is a much better approximation to $A$.

If we define the ILU(0) factorization by $[L_0, U_0] = \text{ilu}(A, 0)$, then $[L_1, U_1] = \text{ilu}(A, 1)$ is the ILU(1) factorization that uses the pattern of $L_0U_0$. Thus in general $[L_p, U_p] = \text{ilu}(A, p)$ is the ILU($p$) factorization that uses the pattern of the previously determined product $L_{p-1}U_{p-1}$. As $p$ increases this process will become too expensive. Therefore, in practice the same process is approximated by using clever algorithms where we initially associate each entry $a_{ij}$ with a level of fill, denoted by $lev_{ij}$, such that:

$$lev_{ij} = \begin{cases} 0 & \text{if } a_{ij} \neq 0 \text{ or } i = j \\ \infty & \text{otherwise} \end{cases}$$

During the process of Gaussian elimination we update the level of fill whenever an element is modified. This is done in the following way:

$$lev_{ij} = \min\{lev_{ij}, lev_{ik} + lev_{kj} + 1\}$$

For diagonal dominant matrices this approach ensures that a high level of fill is associated with a small entry. Evaluated entries in algorithm 3.2 are dropped whenever the level of fill is bigger than $p$. 

19
3.2.4 ILUT

For matrices that are not diagonally dominant it is possible that in the factors there are many small entries with a low level of fill. The preconditioner is not significantly improved by these small entries, hence the ILU($p$) factorization is not very efficient in this case. In an alternative ILU factorization we drop these entries based on their numerical value, this method is known as ILUT. In algorithm A.1 see appendix A, we give an outline of the ILUT algorithm. In this algorithm the $i^{th}$ row of a matrix $A$ is denoted by the MATLAB notation $a_{i*}$.

In the ILUT algorithm we include to algorithm 3.2 a set of rules for dropping small elements based on their numerical values.

- We define a drop tolerance $\tau$ and for each $i$ in the first for-loop, line 1, we define $\tau_i = \tau \cdot \|a_{i*}\|_2$. Next we drop elements that are smaller than $\tau_i$. By defining $\tau_i$ for each $i$ we take in account that a matrix might be badly scaled. This is what is done in line 5.

- In line 10 we again drop elements smaller than the relative tolerance $\tau_i$. Besides that we restrict the use of memory storage by allowing a maximal amount of fill. For both line 11 and 12 we keep only (at most) the $p$ largest elements.

By this dropping rules we reduce both the computational cost and the memory usage. The benefit of ILUT in comparison to ILU($p$) is that we determine the level of fill in the factors based on there numerical values rather than on the structure of $A$. 

Figure 2: Four matrices involved in ILU(1): $L_0U_0$, $L_1$, $U_1$ and $L_1U_1$
4 Sparse Approximate Inverse Preconditioners

A different type of preconditioning is established by constructing a direct approximation of the inverse. With the preconditioner $M \approx A^{-1}$ the preconditioning step reduces to performing one or more (sparse) matrix-vector products. In subsection 3.1.1 we discussed that in recent developments there is a need for efficient general-purpose, parallel preconditioners. The standard incomplete factorization techniques we described in the previous section are in general highly sequential; this becomes clear from algorithm 3.2 where in each step of the process information from the previous steps is needed to proceed. Similarly, linear solves of the form $z = M^{-1}y$ are highly sequential for incomplete factorization techniques. Hence, these standard techniques are certainly not inherently parallelizable. As we will see later, sparse approximate inverse techniques are indeed inherently parallelizable. The need for parallel processing has been the main driving force in the development of these techniques.

There is yet another reason why approximated inverse techniques may be favored over standard incomplete factorization techniques; ILU preconditioner are known to have a high failure rate for indefinite systems. In an article by Chow and Saad on ILU preconditioners for indefinite systems, it is explained that failures may be caused by breakdowns due to zero pivots, inaccuracy and instability of triangular solves [8]. We treat this issues in more detail and make clear that approximated inverse techniques do not suffer from these problems.

Incomplete LU factorizations may suffer from the same problems as complete LU factorizations do. Small pivots are common for indefinite systems and as a consequence entries in the factors can grow uncontrollably. The factorization becomes unstable and the factorization will be inaccurate, which means that $LU$ will not be a good approximation to $A$. Besides that, small pivots lead to unstable triangular solves; linear systems corresponding to $L$ and $U$ might be ill-conditioned. Moreover, this problem might even occur without the presence of small pivots. To fully understand the consequences to this particular case, consider a preconditioner $M = \tilde{L}\tilde{U}$ from incomplete factorization. Assume $M$ is an accurate preconditioner, that is $M \approx A$. By applying two-sided preconditioning we find from equation (3.13) the following equality:

$$\tilde{L}^{-1}A\tilde{U}^{-1} = I + \tilde{L}^{-1}R\tilde{U}^{-1}$$

where the term $R$ is negligible. In order for the system to be efficiently preconditioned, the term $\tilde{L}^{-1}R\tilde{U}^{-1}$ should be negligible as well. However, it is possible that the incomplete factors are unstable even when $M$ is an accurate preconditioners. This means that the last term is not necessarily small. Approximated inverse preconditioners do not suffer from this problem as the preconditioning step consist of a matrix-vector product. In these techniques we may expect $M$ to be a satisfactory preconditioner as long as it is a good approximation to $A^{-1}$.

In sparse approximate inverse techniques we assume that we are able to approximate the inverse of a sparse matrix with another sparse matrix $M$. This is not necessarily true since the inverse of sparse matrices is generally dense. However, for many sparse matrices it turns out that a lot of entries are very small. We will see an example of a matrix with this properties in subsection 6.1 on numerical experiments with an sparse approximate invers.
There are different sparse approximate inverse techniques and they can be grouped in three categories. The different techniques are based on:

- Frobenius norm minimization.
- Factorized sparse approximate inverses.
- Approximate inverses of ILU-factorizations.

First we will give the motivation to approximate the inverse by a sparse matrix and then will treat the three categories of sparse approximate inverse techniques separately.

4.1 Motivation to sparse approximate inverse techniques

Many linear systems arise from partial differential equations of physical problems. In this field of research it was observed that, very often, the inverse contains many small entries. We would like to be able to predict where the smallest entries are located such that we could drop these entries in an approximation of the inverse. We have already seen that for standard techniques, based on incomplete LU decomposition, pattern selection can be a big issue in the construction of the preconditioner. As we will see later, the same is true for sparse approximate inverse techniques. However, once the sparsity pattern is determined, the construction of a sparse approximate inverse is in many cases fairly straightforward. In this subsection we will also describe how a sparsity pattern can be selected a priori.

Now consider again the example of the Poisson equation from eqn (3.16), this time on some 2D domain $\Omega$. Let $P = (x, y) \in \Omega$, then the exact solution on $\Omega$ is given by:

$$u(P) = \int_\Omega G(P, Q)f(Q)dQ$$

where $Q = (\xi, \eta)$ is a variable point used for integration and $G$ is the Green’s function. For the Poisson equation the Green’s function is given by:

$$G(x, y, \xi, \eta) = \frac{1}{2\pi} \ln \frac{1}{\sqrt{(x - \xi)^2 + (y - \eta)^2}}.$$  

We notice that there is a rapid decay of the Green’s function when the distance between $P$ and $Q$ increases. Also, if we discretize the problem, we obtain a linear system of the form $Ax = b$. The matrix $A$ is a discrete approximation to the continuous operator of the Poisson Equation and we observe from the exact solution that $A^{-1}$ should be a good, discrete, approximation to the Green’s function. The inverse will be typically dense, however, from the rapid decay of the Green’s function we may expect that the inverse contains many small entries which can be dropped. Hence, the inverse can be approximated by a sparse matrix.

We can generalize the idea for other problems arising from partial differential equations. In general, the Green’s function is a mathematical description of the decreasing influence between two points if the distance between them increases. Hence, a lot of entries in the inverse of the discrete Green’s function will be small. Unfortunately, for higher-dimensional problems the decay of the Green’s function has a more complex structure and the distance between two nodes is not necessarily a good measure to characterize the influence between them. As a consequence, it might be difficult to predict a priori sparsity patterns and we need different strategies to obtain an effective sparsity pattern.
Another motivation to approximate the inverse by a sparse matrix arises from matrices that are said to be diagonally dominant. Recall the following definition.

**Definition 4.1** The matrix \( A = [a_{ij}] \) is said to be strictly diagonally dominant by row if

\[
|a_{ii}| > \sum_{j \neq i} |a_{ij}| \quad \text{for all } i
\]

For diagonally dominant matrices the following theorem holds:

**Theorem 4.2** If \( A \) is a strictly diagonally dominant matrix by row, then the matrix is nonsingular and the explicit inverse \( A^{-1} \) is strictly diagonally dominant by column.

In optimal problems, \( A \) is diagonally dominant and has a banded structure. For these matrices we have exponential decay in the inverse \( A^{-1} = [a_{ij}] \):

\[
|a_{ij}| \leq C \gamma^{i-j}
\]

for some constant \( C \) and \( \gamma < 1 \). Hence, entries far away from the banded structure are small and may be dropped. For many problems from practical applications, the matrix \( A \) might only be diagonally dominant by row to some degree, i.e. a few rows might not be diagonally dominant. Still, we might expect the inverse to be diagonally dominant by column to some degree and the inverse can still be approximated by a sparse matrix.

In conclusion, we observe that for many systems arising from practical implementations the inverse can be approximated by a sparse matrix. However, for more complex problems it might be hard to predict where the smallest entries are located. In the next subsection we treat a strategy to determine a priori an effective sparsity pattern for the inverse of problems in general.

### 4.1.1 A priori pattern selection

In this section we describe a priori pattern selection. The pattern selection may be based on the specific structure, e.g. a band structure, and characteristics of the matrix. However, in general the nonzero structure of powers of \( A \) will provide us with a good choice for the a priori pattern. To understand this, consider the second power \( A^2 \); we denote the \( i \)th row of the second power by \( a_{i*}^2 \). For this row it holds that

\[
a_{i*}^2 = a_{i*} A = a_{i*} \begin{pmatrix} a_{1*} \\ \vdots \\ a_{n*} \end{pmatrix} = \sum_{j = 1}^{n} a_{ij} \cdot a_{j*}
\]

and hence the \( i \)th row is obtained by the merging of rows of \( A \). Note that for sparse matrices fewer \((< n)\) rows will be merged. In numerical applications, the cancellation of elements is rare due to finite precision and hence we expect additional fill-in in each row of the second power. If in general we denote the \( i \)th row of \( A^k \) by \( a_{i*}^k \), then it holds that

\[
a_{i*}^k = a_{i*} A^{k-1} = a_{i*} \begin{pmatrix} a_{1*}^{k-1} \\ \vdots \\ a_{n*}^{k-1} \end{pmatrix} = \sum_{j = 1}^{n} a_{ij} \cdot a_{j*}^{k-1}
\]
and hence \( a_l \) is obtained by the merging of rows of \( A^{l-1} \). If we assume that there is now cancellation of elements, then for general matrices the fill-in in powers of \( A \) will increase, i.e. the nonzero structure of \( A^{l-1} \) is in general contained in the nonzero structure of \( A^{l} \). By equation (2.3) we know that for nonsingular matrices \( A^{-1} \) can be written as a linear combinations of powers of \( A \). For this reason we expect the nonzero structure of \( A^{l} \) to be a good a priori pattern selection for some \( l < m \), where \( m \) is the degree of the minimal polynomial of \( A \).

The process of computing higher powers of \( A \) becomes increasingly expensive. When we increase \( l \) by one we have to compute an additional sparse matrix-matrix product \( AA^{l} \). Also, this product becomes increasingly more expensive because the fill-in in \( A^{l} \) increases as well. In practice this means that we choose small values for \( l \). An additional argument for choosing small values is that this produces a very sparse preconditioner. Since the preconditioning step consist of a matrix-vector product with \( M \), this reduces the computing time of a single iteration step. Although the process will require more iterations because the sparser preconditioner is a less accurate approximated inverse, the total computing time for solving \( MA = Mb \) may be reduced in this way. Notice that we have to find a balance since both too small as too large values for \( l \) will increase the total computing time.

The cost of constructing a good pattern is reduced by sparsifying the matrix and compute powers of the sparsified matrix \( \tilde{A} = [\tilde{a}_{ij}] \). This is realized by thresholding \( A \) while transforming it to a binary matrix. With a given threshold \( \tau \) this is done in the following way:

\[
\tilde{a}_{ij} = \begin{cases} 
1 & \text{if } i = j \text{ or } |(D^{-1/2} A D^{-1/2})_{ij}| > \tau \\
0 & \text{otherwise}
\end{cases} 
\]  
(4.3)

where \( D = [d_{ii}] \) prescribed by:

\[
d_{ii} = \begin{cases} 
|a_{ii}| & \text{if } |a_{ii}| > 0 \\
1 & \text{otherwise}
\end{cases} 
\]  
(4.4)

The matrix \( D \) is constructed to account for badly scaled matrices. The use of the a priori sparsity pattern is shown by an example of Chow [6]. Consider figure 3(a); we present a density plot of the discrete Green’s function of a partial differential equation at a point near the center of a square domain. The discrete Green’s function is directly related to the inverse. In figure 3(b) we present an approximation to the discrete function when we use the sparsity pattern of the original matrix \( A \) that arises from the discretization of the problem. In a similar way, an approximated inverse is obtained by using the pattern of the sparsified matrix \( \tilde{A} \), figure 3(c). Notice that in this case we only included the largest entries in the pattern. Now we could try to improve the approximate inverse by using powers of the sparsified matrix; the nonzero structure of \( \tilde{A}^L \) for some integer \( L \), is called the level \( L-1 \) pattern. With the level 1 pattern we obtain the approximation to the inverse of figure 3(d). Notice that we have captured the largest entries of the discrete Green’s function and that the approximation has indeed improved.
4.2 Frobenius norm minimization

Methods that are based on Frobenius norm minimization determine a preconditioner $M$ with some known sparse structure. Mostly the sparse structure is determined iteratively in order to find an efficient preconditioner. The process of determining a sparse structure is treated later. Recall the following definition on the Frobenius norm.

**Definition 4.3** The Frobenius norm of a $m \times n$ matrix is defined as

$$\|A\|_F = \left( \sum_{j=1}^{m} \sum_{i=1}^{n} a_{ij}^2 \right)^{1/2}$$

Let $S$ be the set of all sparse matrices with some known structure. We require $M$ to be the best approximation to $A^{-1}$ in Frobenius norm amongst all the matrices in $S$. Now suppose we want to find a right-sided preconditioner (or right approximated inverse). Then $M$ is the solution to the following minimization problem:

$$\min_{M \in S} \|I - AM\|_F$$

(4.5)

For left-preconditioning we observe that $\|I - MA\|_F = \|I - A^T M^T\|_F$. Hence for this case an equivalent minimization problem is formulated by finding a right approximate inverse for $A^T$. Now notice:

$$\|I - AM\|_F^2 = \sum_{j=1}^{n} \|e_j - Am_j\|^2$$

(4.6)
where \( m_j \) is the \( j \)-th column of \( M \). Hence the minimization problem (4.5) is equivalent to solving \( n \) independent linear least square problems. Since they are independent, the construction of these type of preconditioners is inherently parallelizable. By exploiting sparsity properties of \( A \) and \( M \) the costs of solving each least square problem can be reduced significantly; Let \( G \) be the nonzero pattern of \( M \), i.e. the complementary set \( G^c := \{ (i,j) | m_{ij} = 0 \} \).

is the zero pattern of \( M \). This set contains the row and column indices of the preconditioner that are necessarily zero. Now we define for each \( j \) the following subset:

\[
J := \{ i | (i,j) \in G \}.
\] (4.7)

This set contains the indices of the columns of \( A \) that will be evaluated in the matrix-vector product \( A m_j \). Next we define yet another set by:

\[
I := \{ i | A(i,J) \text{ is a nonzero row} \}
\] (4.8)

where \( A(:,J) \) is the submatrix formed by columns of \( A \) that corresponds to the set \( J \). With the set \( I \) we determine the submatrix \( \hat{A} := A(I,J) \) in order to drop the zero rows that may occur due to sparsity. The aforementioned matrix-vector product is computed more efficiently by \( \hat{A} \hat{m}_j \) where \( \hat{m}_j := m_j(J) \). Now we can replace each least square problem from equation (4.6) by much smaller least square problems of the form:

\[
\min \| \hat{e}_j - \hat{A} \hat{m}_j \|_2^2 \text{ for } j = 1, \ldots, n
\] (4.9)

where \( \hat{e}_j := e_j(I) \). Assuming that due to sparsity each system is relatively small, we will solve each system with a direct method such as a QR factorization. Thus, for a known structure it is straightforward to compute an approximated sparse inverse. Methods in which a structure is pre-determined are called static pattern techniques. Unfortunately, for a general matrix we don’t know a suitable sparse structure that captures the large entries of its inverse. Therefore we will describe a method in which we start with an initial sparse structure, usually an empty or diagonal structure, which we will iteratively augment such that the new approximated inverse preconditioner is improved. These type of methods are called adaptive pattern techniques. The most successful approach in this category is known as the Sparse Approximate Inverse algorithm (SPAI), proposed by Grote and Huckle [11].

### 4.2.1 SPAI

Suppose that we have defined a sparse structure \( S \) that we aim to improve. From equation (4.9) we define the corresponding residuals:

\[
r_j = e_j - A(:,J)\hat{m}_j \text{ for } j = 1, \ldots, n
\] (4.10)

By augmenting the structure \( S \) we aim to reduce the residuals in the Euclidean norm; \( \| r_j \|_2 \). We therefore determine the following set:

\[
L = \{ \ell | r_j(\ell) \neq 0 \}
\]

In general we expect that due to sparsity most entries of \( r_j \) are zero. Also, due to finite precision, we expect most entries of \( \hat{r}_j = \hat{e}_j - \hat{A} \hat{m}_j \) to be nonzero. Hence, typically \( L \) will equal \( I \). To
reduce \( \|r_j\|_2 \) it makes sense to evaluate columns of \( A \) that were not evaluated before and besides that corresponds to the set \( \mathcal{L} \). To this end we define the following set:

\[
\hat{\mathcal{J}} = \{ i \mid i \in \mathcal{L}, i \notin \mathcal{J} \}
\]

and for each \( j \in \hat{\mathcal{J}} \) we want to determine whether the \( j^{th} \) column of \( A \) can make a significant contribution to the reduction in \( \|r_j\|_2 \). This is done in a cheap way by considering the following minimization problems:

\[
\min_{\mu_j} \|r_j - \mu_j A e_j\|_2 \quad \text{for each } j \in \hat{\mathcal{J}}
\]

where \( \mu_j \in \mathbb{R} \). Notice \( A e_j \) is the \( j^{th} \) column of \( A \). The solution is simply determined by performing a projection of \( r \) on the space spanned by \( A e_j \). Hence \( \mu_j = -r^T_j A e_j / \|A e_j\|_2 \). Next we compute the Euclidean norm \( \rho_j \) of the new residuals \( r_j + \mu_j A e_j \):

\[
\rho_j^2 = \|r\|_2^2 - \left( r^T A e_j \right)^2 / \|A e_j\|_2^2.
\]

In practical implementations we will make a selection of the most profitable indices \( j \), these are the indices that corresponds to the smallest values of \( \rho_j \). We also restrict the memory storage by allowing a maximal amount of fill-in. As a consequence some indices are dropped in \( \hat{\mathcal{J}} \). Next we determine the corresponding set \( \hat{\mathcal{I}} \) in a similar way as in equation (4.8). Now we consider the augmented matrix \( A(\hat{\mathcal{I}} \cup \hat{\mathcal{J}}) \) for which we again solve the minimization problem of equation (4.9) by computing a QR factorization. Finally we compute the new residual norm \( \|r_j\|_2 \). The process is repeated till the residual meets a certain tolerance or till a maximal amount of fill-in is achieved. The method is summarized in the algorithm [1].

We mention an important theoretical property for the SPAI algorithm. We denote by \( r_k \) the residual for every \( m_k \) and assume that it satisfies:

\[
\|r_k\|_2 = \|A m_k - e_k\| < \epsilon.
\]

It is derived that for the spectral properties of the preconditioned matrix \( AM \) the following theorem holds [11].

**Theorem 4.4** Let \( p = \min_{1 \leq k \leq n} \{ \text{number of nonzero elements of } r_k \} \). Then, the eigenvalues \( \lambda_k \) of \( AM \) are clustered at 1 and lie inside a circle of radius \( \sqrt{\epsilon}p \). Furthermore, if \( \sqrt{\epsilon}p < 1 \), then \( \lambda_{\text{max}} \) and \( \lambda_{\text{min}} \) satisfy

\[
\left| \lambda_{\text{max}} / \lambda_{\text{min}} \right| \leq \frac{1 + \sqrt{\epsilon}p}{1 - \sqrt{\epsilon}p}
\]

In conclusion, depending on the user-defined parameters \( \epsilon \) and the maximal amount of fill-in, we may expect good results from the SPAI algorithm. However, we mention that the construction of the SPAI preconditioner may be expensive. To reduce cost we could use a static technique by a priori pattern selection, which in many cases seems to be an equally effective approach [6].

\[4\] In practice we will expand the QR factorization of \( A(\hat{\mathcal{I}}, \hat{\mathcal{J}}) \) to reduce cost [11]
Algorithm 4.1 SPAI algorithm

1: for every column \( m_j \) of \( M \) do
2:  (a) Choose an initial sparsity pattern \( \mathcal{J} \).
3:  (b) Determine the set of corresponding row indices \( \mathcal{I} \).
4:  Compute a QR decomposition of \( \hat{A} = A(\mathcal{I}, \mathcal{J}) \) and compute the solution \( \hat{m}_j \) from equation \([4.9]\).
5:  Set \( r_j = e_j - A(:, \mathcal{J}) \hat{m}_j \)
6:  while \( \|r_j\|_2 > \epsilon \) do
7:  (c) Set \( L \) equal to the set of indices \( \ell \) for which \( r(\ell) \neq 0 \)
8:  (d) Set \( \mathcal{J} \) equal to the set of all new column indices that appear in all \( L \) but not in \( \mathcal{J} \).
9:  for each \( k \in \mathcal{J} \) do
10:     (e) Set \( \rho_k^2 = \|r_j\|_2^2 - (r_j^T Ae_k)^2/\|Ae_k\|_2^2 \)
11:  end for
12:  Delete from \( \mathcal{J} \) all but the most profitable indices.
13:  (f) Determine the new indices \( \tilde{\mathcal{I}} \) and update the QR decomposition of the submatrix \( A(\tilde{\mathcal{I}} \cup \mathcal{I}, \mathcal{J} \cup \mathcal{J}) \). Next solve for new \( \hat{m}_j \).
14:  Set \( r_j = e_j - A(:, \mathcal{J}) \hat{m}_j \)
15:  Set \( \mathcal{I} = \mathcal{I} \cup \hat{\mathcal{I}} \)
16:  Set \( \mathcal{J} = \mathcal{J} \cup \hat{\mathcal{J}} \)
17: end while
18: end for

4.3 Factorized sparse approximate inverses

A different type of sparse approximate inverse preconditioner is constructed by incomplete inverse factorization; suppose that \( A \) is nonsingular and admits the factorization \( A = LDU \) where \( L \) and \( U^T \) are unit lower triangular and \( D \) is diagonal, that is \( A \) admits a scaled LU factorization. Hence the inverse is factorized as follows: \( A^{-1} = U^{-1}D^{-1}L^{-1} = Z^T D^{-1}W \) where \( Z := U^{-T} \) and \( W := L^{-1} \) are unit lower triangular matrices. We expect \( Z \) and \( W \) to be dense, even when the original matrix \( A \) is sparse. In methods of sparse approximate inverses we compute sparse approximations \( \tilde{Z} \approx Z \) and \( \tilde{W} \approx W \) such that the preconditioner is defined by \([2]\):

\[
M = \tilde{Z}^T \tilde{D}^{-1} \tilde{W} \approx A^{-1}
\]

(4.11)

where \( \tilde{D} \) is a non singular diagonal matrix such that \( \tilde{D} \approx D \). In this subsection we will treat two well-known methods that are based on this concept. The first one is the Factorized Sparse Approximate Inverse (FSAI) which computes a direct approximation of the inverse factors, the method was introduced by Kolotilina and Yeremin [13, 16]. The second method constructs the Approximate Inverse preconditioner of factors with an incomplete bi-conjugation procedure. This method is known by AINV and was proposed by Benzi and Tůma [3, 4].

4.3.1 FSAI

The FSAI method is particularly useful for SPD matrices because, as we will see later, the obtained preconditioner will be SPD as well. The same goes for the preconditioned matrix and hence the preconditioned system can be solved with the CG method. However, we will first
treat general nonsymmetric matrices \[16\]. Next we will derive that this method leads to SPD preconditioners for SPD matrices \[2, 13\].

In FSAI we assume \(A\) to be a non singular matrix that allows an unscaled LU factorization, i.e. \(A = LU\) and \(A^{-1} = U^{-1}L^{-1}\). The aim is to construct an incomplete inverse factorization of the form \(M = G_LG_U\) where \(G_L\) and \(G_U\) are lower triangular matrices that have some sparse structure. The sparsity pattern must be described beforehand; we require \(G_L\) and \(G_U\) to be constrained to a certain triangular sparsity zero pattern prescribed by \[16\]:

\[
\{(i, j) \mid i < j\} \subseteq S_L \subseteq \{(i, j) \mid 1 \leq i \neq j \leq n\}. \tag{4.12}
\]

Initially we will construct unscaled factors \(G_L\) and \(G_U\) that are constrained to the same set. When we denote the indices of matrix \(K\) by \(\{K\}_{ij}\), then these factors are constrained to \(S_L\) in the following way:

\[
\begin{cases}
\{G_LA\}_{ij} = \delta_{ij}, & (i, j) \notin S_L \quad \text{and} \quad \{AG_U\}_{ij} = \delta_{ij}, & (j, i) \notin S_L \\
\{G_L\}_{ij} = 0 & (i, j) \in S_L \quad \text{and} \quad \{G_U\}_{ij} = 0 & (j, i) \in S_L
\end{cases} \tag{4.13}
\]

The factor \(G_L\) is computed by rows. Denote the \(i^{\text{th}}\) row by \(l_i\), from equation \(4.13\) it follows that:

\[
\{l_iA\}_{ij} = \delta_{ij} \quad \text{for} \quad (i, j) \notin S_L, \quad \text{for} \quad i = 1, \ldots, n. \tag{4.14}
\]

If we additionally define the set

\[
\mathcal{M} = \{j \mid (i, j) \notin S_L\}
\]

where \(i\) is fixed, then the systems of equation \(4.14\) are equivalent to the systems of submatrices

\[
\tilde{l}_i\tilde{A}_i = \tilde{e}_i \quad \text{for} \quad i = 1, \ldots, n
\]

where \(\tilde{l}_i := l_i(\mathcal{M}), \tilde{A}_i := A(\mathcal{M}, \mathcal{M})\) and \(\tilde{e}_i := e_i(\mathcal{M})\). Notice that the each system is independent, thus the method is just like SPAI inherently parallelizable. Also notice that if the matrices \(A_i\) are nonsingular, then the factor \(G_L\) is uniquely determined.

For the factor \(G_U\) we can make a similar reasoning, its entries are computed by column. Denote the columns by \(u_j\) and define the set:

\[
\mathcal{N} = \{i \mid (j, i) \notin S_L\}
\]

where \(j\) is fixed. Then the columns of \(G_U\) are determined by solving the systems of submatrices:

\[
\tilde{A}_i\tilde{u}_i = \tilde{e}_i \quad \text{for} \quad i = 1, \ldots, n
\]

where \(\tilde{u}_i := u_j(\mathcal{N}), \tilde{A}_i := A(\mathcal{N}, \mathcal{N})\) and \(\tilde{e}_i := e_i(\mathcal{N})\). Again notice that if the matrices \(\tilde{A}_i\) are nonsingular, then the factor \(G_U\) is uniquely determined. Also, in this case it can be shown that the diagonal entries of \(G_U\) and \(G_L\) are the same \[16\].

Assume that the diagonal entries of both \(G_L\) and \(G_U\) are all nonzero. Now the actual factors \(G_L\) and \(G_U\) are defined by:

\[
G_L := \text{diag}([\{G_L\}_{11}]^{-1/2}, \ldots, [\{G_L\}_{nn}]^{-1/2})G_L \quad \tag{4.15}
\]

and

\[
G_U := \text{diag}(\text{sign}([\{G_U\}_{11}]), [\{G_U\}_{11}]^{-1/2}, \ldots, \text{sign}([\{G_U\}_{nn}]), [\{G_U\}_{nn}]^{-1/2}) \quad \tag{4.16}
\]

29
The preconditioner $M = G_L G_U$ can be applied as a two-sided preconditioner (see equation (3.10)). In this case the diagonal entries of the preconditioned matrices are all equal to 1, that is $\{G_L A G_U\}_{ii} = 1$ for $i = 1, \ldots, n$.

We will show that FSAI is a symmetry preserving preconditioner; since for symmetric matrices it holds that $(A G_U)^T = G_U^T A$ it follows from equation (4.13) that the factors $G_L$ and $G_U^T$ are constrained to $S_L$ in the same way, hence $\bar{G}_L = G_U^T$. If both factors are uniquely determined, it is obvious from equations (4.15) and (4.16) that the resulting preconditioner $M = G_L G_U$ is symmetric. If in addition the matrix is also positive definite, it can be shown that the diagonal entries of $\bar{G}_L$ are all positive\footnote{2}. Now we only require the computation of the following factor:

$$G_L := \text{diag}(\{G_L\}_{ii}^{-1/2}, \ldots, \{G_L\}_{nn}^{-1/2}) \bar{G}_L \quad (4.17)$$

where $\bar{G}_L$ is constructed according to equation (4.13). It follows that $G_U = G_U^T$ and the preconditioner is given by $M = G_L G_U^T$. Moreover, the preconditioned matrix $G_L A G_U^T$ is SPD as well\footnote{2}. The main benefit of FSAI is that for SPD matrices we can apply the CG method to the preconditioned matrix.

Finally, we mention that the following theorem can be derived for SPD matrices\footnote{10}.

**Theorem 4.5** Let $A$ be a SPD matrix and define the following nonnegative quadratic functional:

$$F(X) := \|I - X L\|_F^2.$$  

If we require for the unscaled factors that $\bar{G}_L = G_U^T$ constrained to the triangular sparsity pattern $S_L$, then the computation of the unscaled factor for $A$ of the FSAI preconditioner prescribed in subsection 4.3.1 is equivalent to the following minimization problem:

$$\min F(X) \quad \text{with } X \text{ constrained to } S_L$$

In conclusion, for SPD matrices the FSAI algorithm is equivalent to the Frobenius norm minimization of the Cholesky factor $L$ of $A$, i.e. the lower triangular matrix $L$ such that $A = LL^T$ which is known to exist for SPD matrices.

**4.3.2 AINV**

The AINV method is based on an incomplete biconjugation procedure. The complete biconjugation procedure is an alternative method for explicitly determining $A^{-1}$. The following is a definition on $A$-biconjugation.

**Definition 4.6** Two sets of vectors $\{z_i\}_{i=1}^n$ and $\{w_i\}_{i=1}^n$ are $A$-biconjugate if:

$$w_i^T A z_j = 0 \quad \text{if and only if } i \neq j$$

In the biconjugation procedure we construct two sets of vectors that are $A$-biconjugate, assuming that two such sets exist. Let two matrices $Z$ and $W$ be defined by:

$$Z := [z_1, \ldots, z_n]$$

$$W := [w_1, \ldots, w_n]$$
By definition 4.6 we can also define \( p_i := w_i^T A z_i \neq 0 \) for \( i = 1, \ldots, n \). Then it follows that:

\[
W^T A Z = D := \text{diag}(p_1, \ldots, p_n).
\]

When \( A \) is nonsingular, then so are \( W \) and \( Z \) and it follows that:

\[
A^{-1} = Z D^{-1} W^T = \sum_{i=1}^{n} \frac{z_i w_i^T}{p_i}
\]  \hspace{1cm} (4.18)

We conclude that when we are able to construct two sets of vectors which are \( A \)-biconjugate, we have explicitly determined \( A^{-1} \). The two sets can be constructed from any two linear independent sets \( \{ u_i \}_{i=1}^{n} \) and \( \{ v_i \}_{i=1}^{n} \) by performing a generalized Gram-Schmidt orthogonalization \[4\]. For numerical applications one would generally choose \( u_i = v_i = e_i \). The biconjugation procedure is given in algorithm 4.2. Here we have denoted the rows of \( A \) and \( A^T \) by \( a_i \) and \( c_i \) respectively.

**Algorithm 4.2** Biconjugation algorithm

1: Let \( w_i^{(0)} = z_i^{(0)} = e_i \) for \( i = 1, \ldots, n \)
2: for \( i = 1, \ldots, n \) do
3: \hspace{1cm} for \( j = i, \ldots, n \) do
4: \hspace{2cm} \( p_j^{(i-1)} := a_{ij} z_j^{(i-1)} \)
5: \hspace{2cm} \( q_j^{(i-1)} := c_{ij} w_j^{(i-1)} \)
6: \hspace{1cm} end for
7: If \( i = n \) go to (13)
8: for \( j = i+1, \ldots, n \) do
9: \hspace{2cm} \( z_j^{(i)} := z_j(i-1) - (p_j^{(i-1)}/p_i^{(i-1)}) z_i^{(i-1)} \)
10: \hspace{2cm} \( w_j^{(i)} := w_j(i-1) - (q_j^{(i-1)}/q_i^{(i-1)}) w_i^{(i-1)} \)
11: end for
12: end for
13: Let \( z_i := z_i^{(i-1)} \), \( w_i := w_i^{(i-1)} \) and \( p_i := p_i^{(i-1)} \) for \( i = 1, \ldots, n \)
14: Set \( Z = [z_1, \ldots, z_n] \), \( W = [w_1, \ldots, w_n] \) and \( \text{diag}(p_1, \ldots, p_n) \)

Notice the process could fail in line 9 and 10 due to a zero or very small division. The following theorem tells us when the biconjugation algorithm does not break down.

**Theorem 4.7** In exact arithmetic, algorithm 4.2 does not break down if and only if all \( n \) leading leading minors of \( A \) are nonzero.

Recall the definition of leading minors:

**Definition 4.8** The \( k^{th} \) leading minor of a matrix \( A \) is the determinant of its upper-left \( k \times k \) submatrix.

In case the process does not break down, the matrices \( Z \) and \( W \) are unit upper triangular. Hence \( A = W^{-T} D Z^{-1} \) is a LDU factorization and the algorithm computes \( W = L^{-T} \) and \( Z = U^{-1} \) explicitly. Again we expect the factors \( Z \) and \( W \) to be dense, which makes the complete biconjugation procedure in general too expensive. By an incomplete biconjugation we aim to produce, in a relative cheap way, the factors \( \tilde{W} \), \( \tilde{Z} \) and \( \tilde{D} \) such that we obtain the preconditioner:

\[
M := \tilde{W} \tilde{D}^{-1} \tilde{Z} \approx A^{-1}
\]  \hspace{1cm} (4.19)
which is of the form of equation (4.11). The incomplete biconjugation procedure can be realized by producing a priori sparsity patterns, see e.g. subsection 4.1.1. Next we drop elements in algorithm 4.2 that are outside the sparsity pattern.

The factors $Z$ and $W$ are typically dense. However for sparse matrices it is observed that very often many entries have small magnitude. Hence, the sparsity pattern can also be determined by an adaptive technique; we can drop elements according to a user-defined tolerance and allow a maximal amount of fill-in, similar as in the ILUT algorithm (A.1). Although in practice the algorithm is implemented in a more complex way to prevent break downs due to small pivots. This is done by transforming the system with a diagonal shift $A_1 = A + \alpha I$ and performing pivot modifications. The interested reader is referred to the article by Benzi and Tůma [4].

4.3.3 Factorized vs unfactorized preconditioners

We end this subsection with a discussion on factorized- versus unfactorized preconditioners. We already mentioned that with factorized preconditioners we are able to preserve symmetry. However, we are also interested in the construction cost. Let $m$ be the number of nonzeros in a row of $M$. Then for the described methods, the required amount of flops in constructing this row is of the order $O(m^3)$. For the preconditioner in factorized form, $M = M_1M_2$, the same holds for the separate factors $M_1$ and $M_2$. However, in the factorized case, the number of nonzeros per row are normally halved [6]. Therefore the construction of the factorized preconditioner may be cheaper to construct. We conclude that it is advantageous to compute factorized preconditioners for symmetric and indefinite systems as well.

4.4 Approximate inverses of ILU-factorizations

In this subsection we will treat the type of sparse approximate inverse preconditioners that are constructed by an approximate inversion of an incomplete LU factorization. The factorization may be obtained by techniques such as ILU($p$) or ILUT, but for the purpose of this discussion we assume that the factorization $\tilde{L}\tilde{U} \approx A$ is given. The preconditioner $M \approx A^{-1}$ is obtained by constructing an approximation of the product $\tilde{U}^{-1}\tilde{L}^{-1}$. The approximation is constructed by considering the incomplete factors separately. We will treat shortly how the approximated inverse of $\tilde{L}$ is determined (notice that the process for $\tilde{U}$ will be similar).

Given the factor $\tilde{L}$, we could construct its inverse column-wise. The $i$th column of the inverse can be denoted by $\tilde{L}^{-1}e_i$. Hence, this column is the solution to the following linear system:

$$\tilde{L}x_i = e_i$$ (4.20)

By solving the system for $i = 1, \ldots, n$ we have constructed $\tilde{L}^{-1}$. In principle the columns can be constructed parallel. But in this method there are some issues we need to take in account. First of all, when we construct the $i$th column we have to solve a system with the submatrix $\tilde{L}(n-i+1:n, n+i-1:n)$. Thus the cost for constructing $x_1$ are much higher than the cost for constructing $x_n$, which means that we have to apply load balancing. Secondly, we note that there is some need for communication in the process.
The system of equation (4.20) will obviously be solved approximately. This is for example done by applying forward substitution while forcing a sparsity pattern to the solution, either by applying a drop tolerance or by setting a sparsity structure and perhaps a maximal amount of fill-in. For practical reasons it is best to apply dropping rules during the substitution. In determining a structure for the approximate inverse of the incomplete, there will be some communication in process.

When the factors from an incomplete LU factorization are ill-conditioned, they can still be used by these type of techniques. However, they are less popular than other approaches because they are more difficult to implement: The two levels of incompleteness, the ILU factorization and the approximate inverse of the factors, requires more user-defined parameters. Moreover, both phases of the process are not readily implemented in parallel.
5 Software Implementation

In parallel software implementations the work is distributed over multiple processors to achieve high performance. In the previous section we discussed that for sparse approximated inverse preconditioning techniques, such as SPAI and FSAI, the rows or columns are computed independently from each other. Hence, the construction of the rows and columns can be distributed over multiple processors. The parallel implementation does, however, add to the complexity of the process. In order to achieve high performance we require specialized programming techniques. We will briefly mention the most important ones.

The computation of a sparse approximated inverse that contains many nonzero entries per row may be excessive. Assume that the preconditioner is computed by rows. In order to construct the rows efficiently, we have to perform a load balancing, i.e. we have to distribute the work in such a way that each processor has approximately the same workload. Even when each processor owns the same number of rows of the matrix, the construction phase may be unbalanced because the sparsity pattern may be irregular. The load balancing is realized by a repartitioning of the distribution of the work. In this process the data transfer should be reduced as much as possible, as this slows down the process as well.

A consequence of parallel implementation is the need for communication between the processors. For example, the construction of an a priori zero pattern may be determined by powers of the sparsified matrix $\tilde{A}$ as described in section 4.1.1. Rows of $\tilde{A}^i$ are constructed by the merging of rows of $\tilde{A}$. In this process it will occur that a processor requires rows that are stored on a different processor. The first processor will have to communicate with the other, this is called one-sided communication. This type of communication might become asynchronous and slow down the process. By predetermining the number of request a processor will receive, this behavior is avoided. This technique is called global communication. One can imagine that the communication becomes even more complex for highly sequential processes such as incomplete LU factorizations.

Finally we have to take into account the matrix data structure. If in the construction of a preconditioner the matrix is mainly accessed by rows, then the data structure of $A$ should allow for an efficient access by row. It might be necessary to store $A$ twice, such that is efficiently accessed by both rows and columns.

5.1 Software libraries and packages

In this section we discussed the difficulties in parallel implementations of large systems. For a wide range of problems, there are libraries and packages readily available from the internet that deal with this issues. We will discuss the ones that are relevant to the thesis. First we will take a look at the library PETSc, a large library which can be used both for parallel implementation of parallel Krylov subspace solvers as for parallel preconditioning (including sparse approximated inverse preconditioners). Next we will treat some packages that are more specific to certain sparse approximated inverse techniques.
5.1.1 PETSc

**Name** PETSc - The Portable, Extensible Toolkit for Scientific Computation. [1]

**Author(s)** S. Balay, J. Brown, K. Buschelman, V. Eijkhout, W. Gropp, D. Kaushik, M. Knepley, L. Curfman McInnes, B. Smith, and H. Zhang.

**Language** The software is written in C, C++, Fortran and Python.

**MATLAB** PETSc is directly callable from MATLAB.

**Description** PETSc is a large library of data structures and routines, specialized for parallel implementations on high-performance computers. It includes parallel Krylov subspace solvers as well as sparse approximated inverse preconditioners.

**Website** http://www.mcs.anl.gov/petsc/index.html

PETSc is a large library for data structures and routines in parallel computation on high-performance computers. Many of the routines have been derived from computations on numerical solutions of partial differential equations and related problems. The experience from the authors have led to efficient parallel implementations; PETSc deals with the practical issues that are associated with parallel programming and computations with large systems, some examples are[1]:

- Index sets (IS), including permutations, for indexing into vectors, renumbering, etc.
- Vectors (Vec).
- Matrices (Mat)(generally sparse).
- Krylov subspace methods (KSP)(over fifteen)
- Dozens of preconditioners, including sparse direct solvers (PC).

In conclusion, PETSc provides a rich environment of efficient algorithms and data structures for a user that has to perform parallel computations. The library provides parallel implementation of Krylov subspace solvers and is there especially useful for the theory of this thesis.

5.1.2 SPAI

**Name** SPAI - SParse Approximate Inverse Preconditioner [7].

**Author(s)** Marcus Grote, Stephen Barnard. The configuration was programmed by Oliver Bröker and Michael Hagemann.
The software is written in C/MPI.

MATLAB

SPAI provides a MATLAB interface.

Description

Given a sparse matrix A the SPAI Algorithm computes a sparse approximate inverse M by Frobenius norm minimization. The algorithm can be applied both as a static as an adaptive technique. The preconditioner can be applied to an iterative method. The package includes a Bi-CGSTAB routine which is only intended for testing.

Website

http://cccs.unibas.ch/en/education/software-packages

The SPAI algorithm computes, in parallel, a sparse approximated inverse preconditioner by Frobenius norm minimization, see equation (4.5). The algorithm can be used as a static technique. In this case the sparsity pattern is fixed by either a banded structure or a subset of the sparsity pattern of A. This simple approach may not be robust and as an alternative we can determine the sparsity iteratively by an adaptive technique, as is done in algorithm 4.1; this algorithm was also suggested by Marcus Grote, one of the main authors of the package.

If SPAI is used as an adaptive technique, the algorithm proceeds until the Frobenius norm of AM − I is less than a user defined threshold eps. The threshold must be between 0 and 1. It is suggested to start with a relative large value, like 0.7, and next to decrease eps until one finds an accurate preconditioner. The user must also define a parameter ns: this is the maximum number of improvement steps per column in SPAI. From section 4.2 we know that the preconditioner M is constructed by column. If the maximum number of improvement steps is reached and the residual is not less than eps, then SPAI uses the best approximation obtained so far. It is suggested to use values of ns such that some columns do not achieve the required accuracy eps.

The included Bi-CGSTAB routine can be used for testing, but it is mentioned that this routine is not efficient in a parallel environment. For practical implementations, there is a PETSc interface to SPAI. The SPAI algorithm is robust, inherently parallel, ordering independent and effective on nonsymmetric and ill-conditioned problems.

5.1.3 ParaSails

Name

ParaSails - Parallel sparse approximate inverse preconditioner, using a priori sparsity patterns and least-squares (Frobenius norm) minimization (symmetric positive definite and general versions) [7].

Author(s)

Edmond Chow.

Language

The software is written in C.

MATLAB

ParaSails does not provide a MATLAB interface.

---

5Message Passing Interface (MPI) is a standardized and portable message-passing system designed by a group of researchers from academia and industry to function on a wide variety of parallel computers.
Description

ParaSails is a parallel implementation of a sparse approximate inverse preconditioner, using a priori sparsity patterns and least-squares (Frobenius norm) minimization. The package includes parallel CG and GMRES solvers and a parallel matrix class. SPD problems are preconditioned with a factorized SPD preconditioner and general (nonsymmetric and/or indefinite) problems are handled with an unfactorized preconditioner. ParaSails uses post-filtering techniques to reduce cost of applying the preconditioner.

Website

http://www.llnl.gov/CASC/parasails

The a priori sparsity pattern in ParaSails is determined by constructing, in parallel, powers of the binary matrix \( \tilde{A} \), see equation (4.3). In tests performed by Chow, powers of levels up to 4 and thresholds less than 0.3 were suitable parameters in producing an accurate preconditioner [6].

For SPD problems a factorized sparse approximated inverse \( M = G_L D_L T_L G_L \) is constructed by Frobenius norm minimization of the Cholesky factor. General problems are preconditioned with a factorized sparse approximated inverse \( M \) of the original matrix, see equation (4.5).

The preconditioning step is executed more efficiently by a post-filtering technique on \( M \) (or \( G_L \)). Entries of the scaled preconditioner that are smaller than a user defined threshold, called the filter value, are dropped. In tests performed by Chow, filter values between 0.05 and 0.1 have shown to give good results.

5.1.4 HSL MI12

Name

HSL MI12.

Author(s)

Numerical Analysis Group at the STFC Rutherford Appleton Laboratory

Origin

N.I.N. Gould and J.A. Scott

Language

The software is written in Fortran 77.

MATLAB

HSL MI12 does not provide a MATLAB interface.

Description

The routine finds a sparse approximated inverse \( M \) by Frobenius norm minimization. The process may be improved by first performing a block triangularization of \( A \) and then computing approximate inverses to the resulting diagonal blocks.

Website

http://www.hsl.rl.ac.uk/catalogue/mi12.xml

The abbreviation HSL stands for Harwell Subroutine Library, it is a collection of packages that were written and developed by the Numerical Analysis Group at the STFC Rutherford Appleton Laboratory and other experts. The MI packages are specialized for iterative methods of sparse matrices. The MI12 package computes an sparse approximated inverse preconditioner for sparse unsymmetric matrices. The process is similarly to the SPAI package, based on Frobenius
norm minimization of $AM - I$. In addition, the package includes an option to perform a block triangularization of $A$ as follows:

$$PAQ = \begin{pmatrix} A_{11} & A_{12} & \cdots & A_{1b} \\ A_{21} & A_{22} & \cdots & A_{2b} \\ \vdots & \vdots & \ddots & \vdots \\ A_{bb} & \cdots & & A_{bb} \end{pmatrix}$$

where $P$ and $Q$ are permutation matrices. When a block triangularization is performed, the preconditioner is constructed by sparse approximated inverses of the diagonal blocks. These are determined by a Frobenius norm minimization of $A_{jj}M_{jj} - I_j$ for $j = 1, \ldots, b$. In both cases the sparsity pattern is determined iteratively starting with a complete zero pattern. The algorithm allows a maximum amount $l$ of nonzero entries in each column of $M$ according to a user defined parameter. The sparsity pattern is augmented in a similar way as in the SPAI algorithm [4.1]. As in the SPAI package, a threshold $\epsilon$ should be defined such that the process is terminated whenever $\|A_{jj}m_{jj}^i - e_i\|_F < \epsilon$, where $m_{jj}^i$ is the $i^{th}$ column of $M_{jj}$. 

38
6 Numerical experiments

In this section we describe some numerical experiments with sparse approximate inverse techniques. First we consider test problems with two matrices from practical implementations. We will test the effectiveness of SPAI preconditioners in comparison to the more standard used ILUT preconditioners. Secondly, we will see a practical implementation of SPAI preconditioners in electromagnetic scattering problems.

6.1 Comparative study on SPAI and ILUT preconditioners

In this section we report on experiments with SPAI and ILUT preconditioners applied to Krylov methods. In this comparative study on preconditioners we show that the sparse approximate inverse technique SPAI gives good results for two indefinite and unsymmetric systems. In the second case we consider an ill-conditioned system. Even then SPAI has shown to produce an accurate preconditioner and fast convergence was achieved. The more general used ILUT algorithm leads to bad convergence behavior and for the ill-conditioned the process breaks down because we encounter a zero pivot in the construction phase of the preconditioner.

The two matrices that we considered in the experiments originate from physical problems and they are available for free on the website http://math.nist.gov/MatrixMarket. The first matrix belongs to the set denoted by BRUSSEL, a set of matrices obtained from the reaction-diffusion Brusselator Model. The matrix that we consider has the matrix name RDB2048, for the sake of both notation as brevity we will refer to this matrix by $A_{rdb}$. The matrix is supplied by K. Meerbergen from the Katholieke Universiteit Leuven and A. Spence from the University of Bath. The field of application is is chemical engineering. The matrix is unsymmetric indefinite, is of size $n = 2048$ and has condition number $K(A_{rdb}) = 1.81 \cdot 10^3$, see figure 4 (left). The nonzero structure of the matrix is represented with a density plot; this is a coloured plot of the sparsity pattern from which we readily observe the pattern of the largest entries.

![Figure 4: Matrices RDB2048 (left) and FIDAP032 (right).](image)

The second matrix belongs to the set FIDAP; a set of matrices that is generated by the software package FIDAP. The matrix that we consider has the matrix name FIDAP032 and we
refer to it by $A_{fdp}$. The matrix is supplied by Isaac Hasbani from Fluid Dynamics International and Barry Rackner from Minnesota Supercomputer Center. The field of application is finite element modeling. The matrix is unsymmetric indefinite and is of size $n = 1159$, see figure 4 (right). With condition number $K(A_{fdp}) = 5.24 \cdot 10^{20}$ we consider the corresponding system to be ill-conditioned.

We have explicitly determined the inverse of $A_{rdb}$ in MATLAB. The density plot is shown in figure 5. We notice that, although the original matrix is very sparse, the inverse is completely dense. This is typically observed for sparse matrices. However in many cases, and also in this particular case, a lot of the entries outside the sparsity pattern of the original matrix have small magnitude. We conclude that an inverse may be effectively approximated by a sparse matrix. We mention that $A_{fdp}$ is too ill-conditioned to invert in MATLAB.

![Figure 5: Density plot of the explicit inverse $A_{rdb}^{-1}$.](image)

With the package SPAI, described in subsection 5.1.2, we have constructed sparse approximate inverses of the two matrices. The right-hand side of the considered systems was computed by taking for $x$ the vector of all ones, so that the exact solution is always known. SPAI is implemented as an adaptive technique; for the threshold $\epsilon_p$ we have chosen 0.6, for the parameter $n_s$ we choose 5 and we allowed a maximum number of 5 nonzero entries in each column. The meaning of these parameter is described in subsection 5.1.2. The parameters were chosen such that we obtained good results with the included Bi-CGSTAB routine from the SPAI package. Although we could compute the forward error explicitly, since we know the exact solution, we used the residuals for the stop criterion as this is standard for the Bi-CGSTAB routine. We have applied right-preconditioning such that the residuals are not changed. The tolerance for the stop criterion was $10^{-8}$.

The obtained preconditioners for $A_{rdb}$ and $A_{fdp}$ ($M_{rdb}$ and $M_{fdp}$ respectively), are shown in figure 6 (the top left en -right corner, respectively). Notice that both preconditioners have a very sparse pattern, comparable with the sparsity pattern of the original matrix. Also notice that $M_{rdb}$ is much sparser than we may have suspected from the density plot of $A_{rdb}^{-1}$ (figure 5).

We represented convergence plots of the Bi-CGSTAB routine applied to the preconditioned systems $A_{rdb}M_{rdb}$ and $A_{fdp}M_{fdp}$, see figure 6 (bottom left and -right corner, respectively). Convergence was achieved in 146 iterations steps for $A_{rdb}$ and in 255 steps for $A_{fdp}$. In practice, however, it is possible that we require a lower tolerance of say $10^{-3}$ and we could have achieve convergence in approximately 50 and 100 iteration steps respectively.
Figure 6: Results from numerical experiments with the package SPAI to matrices $A_{rd}$ and $A_{fd}$. 

We tested the effectiveness of the SPAI preconditioners $M_{rd}$ and $M_{fd}$ in comparison to the popular and more general used ILUT preconditioners. We used the ITSOL package, this package provides a routine for the standard ILUT algorithm described in subsection 3.2.4. In the package there is included a routine for the flexible GMRES with restart $m$ (FGMRES($m$)). First, the routine of GMRES($m$) is prescribed as follows: In each iteration step we construct a Krylov subspace of size up to $m$ and check whether the solution is accurate enough. In the next iteration step, the previous Krylov subspace is destroyed and GMRES($m$) restarts the process with the latest approximation as an initial guess. For unpreconditioned systems we know from the Arnoldi

\footnote{Available on the website \url{http://www-users.cs.umn.edu/saad/software/ITSOL/index.html}, the software is written in C.}
Table 1: Results from GMRES(60) applied to the preconditioned systems RDB2048L and FIDAP032. By "iter" we refer to the outer iterations, the residuals and errors are given in 2-norm and the abbreviation "b.d." stands for a break down in the construction of the preconditioner.

<table>
<thead>
<tr>
<th></th>
<th>GMRES(60), Toler. $1e-8$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RDB2048L</td>
</tr>
<tr>
<td>iter</td>
<td>residual</td>
</tr>
<tr>
<td>SPAI</td>
<td>3</td>
</tr>
<tr>
<td>ILUT</td>
<td>5000</td>
</tr>
</tbody>
</table>

method that for each iteration of GMRES($m$) we perform $m$ matrix-vector products. Therefore we refer to this process by the outer iteration of GMRES. Notice that an outer iteration of GMRES($m$) is, in terms of computational work, equivalent to $m$ iterations of GMRES. We call the algorithm FGMRES flexible because it allows for a variable preconditioner in each outer iteration step, however, in this case the preconditioner remains the same. For this reason we interpret the included solver as GMRES($m$).

The right-hand sides of the systems are again computed by taking for $x$ the vector of all ones. For the construction of the ILUT preconditioners we have chosen a drop tolerance of 0.3 and a level of fill of 50. For the dimension of the Krylov subspace in outer FGMRES we have chosen $m = 60$. The SPAI preconditioners $M_{rb}$ and $M_{fb}$ were stored in MATLAB and we solved the system with a similar GMRES(60) routine. We present the results in table 1.

For the well-conditioned system RDB2048L we notice that the SPAI preconditioner is very effective, whereas the ILUT preconditioner did not manage to converge within 5000 outer iterations (the maximal amount of iterations allowed). At this point the forward error is still significant. The ill-conditioned system FIDAP032 converges in 23 outer iterations when we use the SPAI preconditioner. However, from the forward error we observe that the system did not converge to an accurate solution. Since the system is ill-conditioned it is prone to numerical errors and we can not always fix this with a preconditioner. The SPAI preconditioner still performs much better, as for the second matrix the construction of the ILUT preconditioner breaks down due to a zero pivot. In this comparative study on preconditioners we have shown that sparse approximate inverse techniques may provide accurate preconditioners for unsymmetric indefinite system and that they should be valued as alternatives for standard techniques based on incomplete LU decomposition.
6.2 Electromagnetic Scattering - the Satellite Problem

In this subsection we review the electromagnetic scattering problem for a simplified satellite, see figure 7. The problem addresses the physical issue of detecting the diffraction pattern of the electromagnetic radiation scattered from the satellite when it is illuminated by an incident incoming wave. Electromagnetic scattering problems of large and complex bodies are of interest in the design of many industrial devices like radars, antennae, computer microprocessors, cellular telephone and so on. Maxwell’s equations describe how electric and magnetic fields are generated and altered by each other, and by charges and currents. However, for computational purposes the problem is converted to an integral formulation of the surface current of the object. Once the surface current is known, we could derive the diffraction pattern of the electromagnetic radiation using Maxwell’s equations. Hence, the surface current is the unknown quantity. The linear systems that are derived from these type of problems are among the most difficult problems to solve by iterative methods. The need for preconditioning of these type of problems, and especially the effectiveness of sparse approximate inverse techniques based on Frobenius norm minimization in comparison to other preconditioning techniques, is described by B. Carpentieri [5].

Figure 7: Simplified model of a satellite with a mesh of triangles on its surface

The discretization of the problem is realized by constructing a mesh of triangles on the surface of the satellite, see figure 7. Now, the unknowns of the system are associated with the vectorial flux across an edge in the mesh. Hence, each column of the matrix that arises from this problem corresponds to such an edge in the mesh. The right-hand side depends on the frequency and the direction of the incoming wave. For the purpose of this thesis, we will not go in to detail in the construction of the linear system. We do mention that the frequency $\omega$ is related to the wavelength $\lambda$ by $\omega = \frac{2\pi c}{\lambda}$, where the constant $c$ is the speed of light. For physical consistency we require about ten discretization points per wavelength, so that the linear systems arising from
high-frequency scattering problems can be very large.

For the SPAI preconditioner we compute the sparsity pattern in advance, next the preconditioner is constructed by solving the $n$ least square problems of eqn (4.9). For the a priori pattern selection we consider a model problem: The electromagnetic scatter problem of a sphere, see figure 8. The model problem is representative of the general trend and we will use the same approach for the satellite problem.

![Figure 8: Model problem of a sphere with a mesh of triangles on its surface.](image)

In the upper left corner of figure 9 we present the density plot of matrix that arises from the electromagnetic scatter problem of the sphere. The matrix is completely dense, however, it contains many small entries. We sparsify the matrix by dropping these small values, see figure 9 (the bottom left corner). In this figure we also present a density plot of the explicit inverse and the nonzero structure of a sparsified version of the explicit inverse (the upper- and bottom right corner, respectively). The general trend that is observed for these type of problems is that the nonzero structure of the sparsified matrix and the sparsified explicit inverse are very similar. Since the most important information is maintained in the sparsified matrices, it makes sense to choose as a sparsity pattern, for the preconditioner $M \approx A^{-1}$, the nonzero structure of the sparsified matrix $A$.

For the satellite problem we use the same approach. The sparsity pattern that we obtained for the preconditioner is shown in figure 10 (left); the size of the matrix is $n = 1699$. The preconditioner is obtained by Frobenius norm minimization and is applied as a left-sided preconditioner, eqn (3.8). We solve the system with GMRES(60) with a tolerance of $1.0e - 10$ for the stop criterion. Notice that the stop criterion is based on the preconditioned residual $Mr^{(k)} := M(b - Ax^{(k)})$, which we call the Arnoldi residual. The method converges within 121 iterations at which point the actual residual, for the unpreconditioned system, takes value $0.43e - 08$. We present the convergence plot in figure 10 (right).
6.2.1 Spectral deflation

In this subsection we propose a refinement technique for the SPAI preconditioner. The goal is to remove the effect of the smallest eigenvalues (in magnitude) from the preconditioned matrix (we know from section 3.1 that eigenvalues close to the origin are bad for convergence). This technique is known as spectral deflation. We apply this technique to the satellite problem to see how the convergence improves.

We first consider the model problem of the sphere as it is representative for the general trend. The eigenvalue distribution of the corresponding matrix is presented in figure 11 (left). We observe that spectrum is widely spread in the complex plane, which explains the need for preconditioning. When we precondition the system with a Frobenius norm minimization method,
as described earlier for the model problem, the eigenvalues become clustered at the point one, see figure 11 (right). However, we observe that there are still a few eigenvalues located near the origin and we expect these eigenvalues to slow down the convergence significantly.

We are considering a system that is left-preconditioned, i.e. the system $M_1Ax = M_1b$ where $M_1$ is the preconditioner obtained by Frobenius norm minimization. Assume for simplicity that the preconditioned matrix $M_1A$ is diagonalizable, this means that there exists a diagonal matrix $D = \text{diag}(\lambda_i)$, where $|\lambda_1| \leq \ldots \leq |\lambda_n|$, and a nonsingular matrix $V = [v_1, \ldots, v_n]$, where $v_i$ is an associated right eigenvector, such that:

$$M_1A = VDV^{-1}.$$
we then have $U^H V = \text{diag}(u_i^H v_i)$, with $u_i^H v_i \neq 0$ for every $i$. Here $U^H$ denotes the Hermitian of $U$ as it may contain complex eigenvectors. Now let $V_\epsilon$ denote the set of right eigenvectors associated to the eigenvalues with $|\lambda_i| \leq \epsilon$ and define $U_\epsilon$ in a similar way. It is shown that the following theorem holds [5].

**Theorem 6.1** Let

\[
A_\epsilon = U_\epsilon^H M_1 A V_\epsilon, \\
M_\epsilon = V_\epsilon A_\epsilon^{-1} U_\epsilon^H M_1
\]

and

\[
M = M_1 + M_\epsilon.
\]

Then $MA$ is diagonalizable and we have $MA = V \text{diag}(\eta_i) V^{-1}$ with

\[
\begin{cases}
\eta_i = \lambda_i & \text{if } |\lambda_i| > \epsilon, \\
\eta_i = 1 + \lambda_i & \text{if } |\lambda_i| \leq \epsilon.
\end{cases}
\]

We do not provide a proof for the theorem, but it is easy to derive that $A_\epsilon = \text{diag}(\lambda_i \cdot u_i^H v_i)$; a matrix of size $k \times k$ where $k$ is the amount of eigenvalues that are smaller (in absolute value) than $\epsilon$. By using the diagonalization of $M_1 A$ multiple times, the result follows.

Computing the eigenvalues and the associated eigenvectors is expensive and we want to reduce the cost as much as possible. The following theorem will help to reduce cost by obtaining a preconditioner that is similar to the preconditioner obtained in theorem 6.1. We state the theorem without proof and refer the interested reader to the article by B. Carpentieri [5].

**Theorem 6.2** Let $W$ be such that $\tilde{A}_\epsilon = W^H A V_\epsilon$ has full rank, $\tilde{M}_\epsilon = V_\epsilon \tilde{A}^{-1}_\epsilon W^H$

and

$\tilde{M} = M_1 + \tilde{M}_\epsilon$.

Then $\tilde{M} A$ is similar to a matrix whose eigenvalues are

\[
\begin{cases}
\eta_i = \lambda_i & \text{if } |\lambda_i| > \epsilon, \\
\eta_i = 1 + \lambda_i & \text{if } |\lambda_i| \leq \epsilon.
\end{cases}
\]

We expect $V_\epsilon$ to be of full rank $k$; consequently, a natural choice for the matrix $W$ is to choose the same matrix $V_\epsilon$. Now, without computing the left eigenvectors, we are able to construct a preconditioner $M$ such that the preconditioned matrix $MA$ has the desired properties; in general we assume $\epsilon \ll 1$ so that the smallest eigenvalues become clustered at the point one as well.

We have applied spectral deflation to the satellite problem. With the package ARPACK we have computed the smallest eigenvalues and their corresponding approximate eigenvectors. By successively shifting the eigenvalues, as described in theorem 6.2, we observe how the convergence behavior of GMRES(60) improves. The results are shown in table 2; the convergence behavior
Table 2: Effect of shifting the eigenvalues nearest zero on the convergence of GMRES(60). We show the magnitude of the successively shifted eigenvalues and the number of iterations required when these eigenvalues are shifted.

<table>
<thead>
<tr>
<th>Nr. of shifted Eigenvalues</th>
<th>GMRES(60)</th>
<th>Toler. $1\times 10^{-10}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>121</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>112</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>107</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>97</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>91</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>85</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>80</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>76</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>66</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>60</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>59</td>
<td></td>
</tr>
</tbody>
</table>

keeps improving by successively shifting the smallest eigenvalues. By shifting the ten smallest eigenvalues, GMRES(60) already converges twice as fast.

We mention some remarks on spectral deflation. First of all, the preconditioning phase of GMRES(60) requires a matrix-vector (M-V) product with the matrix $M = M_1 + \tilde{M}_c$. The costs of the correction update $\tilde{M}_c v$ for the M-V product $M v$ are $2nk + k^2$, where $k$ is the amount of shifted eigenvalues. Secondly, the ARPACK package requires a significant amount of M-V products for the computation of the smallest eigenvalues and the associated eigenvectors. For example, in order to shift 12 eigenvalues for the model problem of the sphere, we require 131 such products. For the slightly more complex problem of the cylinder, this already increases to 597. This should be compared with the approximate $120N$ M-V products that are required for $N$ steps of the preconditioned GMRES(60) routine. However, the additional cost for constructing the preconditioner can be amortized if the matrix is reused for solving linear system with the same coefficient matrix and several right-hand sides. This is actually a typical scenario for realistic electromagnetic simulations [5].
7 Conclusion

In my thesis I have reviewed techniques for computing a sparse approximation to the inverse of a matrix. We have seen that the inverse of a sparse matrix is typically dense, but may contain a lot of entries with small magnitude. Sparse approximate inverse techniques can be effectively used in many applications of numerical analysis, however, the main focus in this thesis was on the preconditioning of linear systems. We have discussed the concept of preconditioning and reviewed some well-known standard preconditioning techniques based on incomplete LU decomposition. These type of techniques are effectively used in many applications, however, we have seen that might be ineffective, or might even fail, on unsymmetric and indefinite systems. The most important causes for failure are a break down due to a zero pivot and unstable linear solves in the preconditioning phase. We have shown that sparse approximate inverse techniques do not suffer from these problems and can be effectively used on indefinite and unsymmetric systems. Another issue we addressed to in this thesis is the need for parallel implementation on high-performance computers. The standard techniques are in general highly sequential, both in construction as in implementation, whereas sparse approximate inverse techniques are in general inherently parallelizable. Preconditioning is still the main stumbling block in achieving high performance for large, sparse linear systems and sparse approximate inverse techniques are promising techniques to overcome this.

In a comparative study on preconditioners, we have shown that sparse approximate inverse techniques are competitive and sometimes even superior to the standard techniques based on incomplete LU decomposition. The focus was mainly on preconditioners obtained by Frobenius norm minimization of the inverse, i.e. the SPAI preconditioner. With the SPAI preconditioner we achieved good results on unsymmetric and indefinite (and in a particular case even ill-conditioned) systems from practical implementations. With the ILUT preconditioner convergence was either slow or the construction of the preconditioner broke down due to a zero pivot.

Finally, we have reviewed the use of SPAI preconditioners in practical implementations of electromagnetic scattering problems. Linear systems arising from electromagnetic scattering problems are among the most difficult systems to solve by iterative methods. According to a model problem of a sphere, we have studied the scattering problem of a simplified satellite. With the a priori pattern selection obtained from the nonzero structure of the sparsified matrix to the original problem, we achieved good results when the system was solved with GMRES(60). In addition we proposed a refinement technique by spectral deflation and showed that the convergence was significantly improved by shifting up to ten eigenvalues of smallest value.
References


Appendices

A  ILUT algorithm

**Algorithm A.1 ILUT algorithm, IKJ variant**

1: for $i = 2, \ldots, n$ do
2:   $w := a_{i\ast}$
3:   for $k = 1, \ldots, i - 1$ and when $w_k \neq 0$ do
4:     $w_k = w_k / a_{kk}$
5:     Apply a dropping rule to $w_k$
6:     if $w_k \neq 0$ then
7:       $w := w - w_k * u_{k\ast}$
8:     end if
9:   end for
10: Apply a dropping rule to row $w$
11: $l_{ij} = w_j$ for $j = 1, \ldots, i - 1$
12: $u_{ij} = w_j$ for $j = i, \ldots, n$
13: $w := o$
14: end for

B  MATLAB code for GMRES(m)

The GMRES(m) routine is implemented in MATLAB with the following code.

```matlab
function [x, error, iter, mvprod, flag] = gmres_spai(A, x, b, M, restrt, ...
  max_it, tol )
% --- Iterative template routine ---
% Univ. of Tennessee and Oak Ridge National Laboratory
% October 1, 1993
% Details of this algorithm are described in "Templates for the
% Solution of Linear Systems: Building Blocks for Iterative
% Methods", Barrett, Berry, Chan, Demmel, Donato, Dongarra,
% Eijkhout, Pozo, Romine, and van der Vorst, SIAM Publications,
% 1993. (ftp netlib2.cs.utk.edu; cd linalg; get templates.ps).
% [x, error, iter, mvprod, flag] = gmres_spai(A, x, b, M, restrt, max_it, tol )
% gmres.m solves the linear system Ax=b
% using the Generalized Minimal residual ( GMRESm ) method with restarts .
% % input  A   REAL nonsymmetric positive definite matrix
% %        x   REAL initial guess vector
% %        b   REAL right hand side vector
% %        M   REAL preconditioner matrix
% %        restrt INTEGER number of iterations between restarts
% %        max_it INTEGER maximum number of iterations
% %        tol   REAL error tolerance
```
% output x REAL solution vector
% error REAL error norm
% iter INTEGER number of iterations performed
% mvprod INTEGER number of actual M-V products
% flag INTEGER: 0 = solution found to tolerance
% 1 = no convergence given max_it

iter = 0; % initialization
flag = 0;
mvprod = 0;

bnrm2 = norm(b);
if (bnrm2 == 0.0), bnrm2 = 1.0; end

r = M * (b - A*x);
mvprod = mvprod + 1;
error = norm(r) / bnrm2;
if (error < tol) return, end

[n,n] = size(A);
% initialize workspace
m = restrt;
V(1:n,1:m+1) = zeros(n,m+1);
H(1:m+1,1:m) = zeros(m+1,m);
cs(1:m) = zeros(m,1);
sn(1:m) = zeros(m,1);
e1 = zeros(n,1);
e1(1) = 1.0;

for iter = 1:max_it, % begin iteration
    r = M * (b - A*x);
mvprod = mvprod + 1;
    V(:,1) = r / norm(r);
s = norm(r)*e1;
    for i = 1:m, % construct orthonormal
        w = M * (A*V(:,i)); % basis using Gram-Schmidt
        mvprod = mvprod + 1;
        for k = 1:i,
            temp = cs(k)*H(k,i) + sn(k)*H(k+1,i);
            H(k+1,i) = -sn(k)*H(k,i) + cs(k)*H(k+1,i);
            H(k,i) = temp;
        end
        [cs(i),sn(i)] = rotmat(H(i,i), H(i+1,i)); % form i-th rotation matrix
        temp = cs(i)*s(i);
        s(i+1) = -sn(i)*s(i);
        s(i) = temp;
        H(i+1,i) = cs(i)*H(i,i) + sn(i)*H(i+1,i);
        H(i,i+1) = 0.0;
        error = abs(s(i+1)) / bnrm2;
        if (error < tol), % update approximation
            y = H(1:i+1,i) \ s(1:i);
            y = x + V(:,1:i)*y;
            break;
        end
    end
end
The routine is applied to the systems RDB2049 and FIDAP032, \( m = 60 \), with the following script.

```plaintext
clear all; close all; clc
load matrices
fprintf('
');
fprintf('* RDB2048L problem *
');
A=rdb2048;
M=rdb2048_spai;
n=size(A,1);
restrt=60;
tol=1e-8;
b=A*ones(n,1);
x0=zeros(n,1);
max_it=n;
[x, error, iter, mvprod, flag] = gmres_spai( A, x0, b, M, 60, max_it, tol);

fprintf('
');
fprintf('* FIDAP032 problem *
');
pause(0)
A=fidap032;
M=fidap032_spai;
n=size(A,1);
restrt=60;
tol=1e-8;
b=A*ones(n,1);
x0=zeros(n,1);
max_it=n;
[x, error, iter, mvprod, flag] = gmres_spai( A, x0, b, M, 60, max_it, tol);
fprintf('
');
```

53