Solving large linear systems in an implicit thermohaline ocean model

de Niet, Arie Christiaan

IMPORTANT NOTE: You are advised to consult the publisher's version (publisher's PDF) if you wish to cite from it. Please check the document version below.

Document Version
Publisher's PDF, also known as Version of record

Publication date:
2007

Link to publication in University of Groningen/UMCG research database

Citation for published version (APA):
Chapter 3

Solving large linear systems

In this chapter we treat the techniques from numerical linear algebra needed to solve large systems of linear equations. In the previous chapter we encountered two of such systems: the Jacobian matrix in THCM in Equation (2.34) and the saddle point problem in Equation (2.36). Both systems are large and sparse, that is they have only a few nonzeros on each row. The solution of large sparse systems of equations is the domain of numerical linear algebra. In this chapter we will discuss in short the two most important approaches to solve such a system: direct methods in Section 3.1 and iterative methods in Section 3.2. In the last section of this chapter we will explain the relation of numerical continuation of solutions of THCM and the next chapters.

In the next two sections we suppose that we want to solve the equation

\[ Ax = b, \]  

(3.1)

where \( A \in \mathbb{R}^{n \times n} \) and \( x, r \in \mathbb{R}^n \). Furthermore we assume that \( n \) is large (typically \( \mathcal{O}(10^{3-9}) \)) and \( A \) is sparse, i.e. per row there is a small number of nonzeros.

3.1 Direct methods

The approach in direct methods is to build a matrix factorization \( A = LDU \), where \( L \) is a unit lower triangular matrix, which means it has ones on the diagonal and only zeros above, \( U \) a unit upper triangular matrix (i.e. ones on the diagonal and zeros below) and \( D \) a diagonal matrix. If \( A \) is symmetric positive definite one can build a Cholesky factorization \( A = LDL^T \), so \( U = L^T \).

Given such a matrix factorization, we can solve the equation in three steps: solve \( Lz = b \), \( Dy = z \) and \( Ux = y \) respectively, which is easy, because equations with lower (or upper) triangular matrices can be solved via backward substitution.
3.1.1 Construction of an LDU-factorization

The LDU-factorization is built using Gaussian elimination. To illustrate the construction we show the elimination of the first row, where we assume that the first element of the matrix \( a_{11} \), is nonzero:

\[
A = A^{(0)} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ a_{21}^{-1}a_{11}^{-1}I & A^{(1)} \end{bmatrix} \begin{bmatrix} 1 & a_{11}^{-1}a_{12} \\ 0 & I \end{bmatrix},
\]

(3.2)

where \( A^{(1)} = A_{22} - a_{21}a_{11}^{-1}a_{12} \) is called the Schur complement of \( a_{11} \) in \( A \). Note that \( A^{(1)} \in \mathbb{R}^{n-1 \times n-1} \). For \( A^{(1)} \) we can build a similar factorization to obtain \( A^{(2)} \). Repeating this procedure we finally get the LDU-factorization. It is likely that the nonzero patterns of \( a_{21}a_{11}^{-1}a_{12} \) and \( A_{22} \) differ, so by the elimination we create new nonzeros in the Schur complement. In fact the growth of the number of nonzeros in the Schur complements can be huge. The factors \( L \) and \( U \) can be almost dense, even if \( A \) is sparse. Both storage (memory) and the construction time are related to the fill in the final factors, so we would like to avoid dense factors. Fortunately the fill in the factors can be reduced by simply changing the ordering of the rows and columns in \( A \). In direct methods much effort is put in the search for a renumbering of the variables \( q \) with corresponding permutation matrix \( Q \), such that the permuted matrix

\[
QAQ^T = LDU,
\]

(3.3)

has factors \( L \) and \( U \) that are as sparse as possible. Such an ordering \( q \) is called a fill reducing ordering.

3.1.2 Matrices and graphs

In the theory of fill reducing orderings graphs play an important role, see [22, 43, 52, 53], because the fill pattern of a symmetric matrix can be identified with an undirected graph. This adjacency graph has a number of vertices equal to the dimension of the matrix and has an edge for each off-diagonal nonzero, i.e. \( \{i, j\} \) is an edge if \( a_{ij} \neq 0 \). In Figure 3.1 we show an example of a fill pattern of symmetric matrix \( F(A) \) and its adjacency graph \( G(A) \) for the matrix

\[
A = \begin{bmatrix}
4 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & 0 \\
0 & 4 & -1 & 0 & 0 & 0 & -1 & -1 & 0 \\
0 & -1 & 4 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 4 & -1 & 0 & 0 & -1 & 0 \\
0 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -1 & 4 & 0 & -1 & -1 \\
-1 & -1 & 0 & 0 & 0 & 4 & -1 & 0 & 0 \\
0 & -1 & 0 & -1 & 0 & -1 & 4 & 0 & 0 \\
0 & -1 & 0 & 0 & 0 & -1 & 0 & 0 & 4
\end{bmatrix}.
\]

(3.4)
3.1 Direct methods

A cycle of length \( l \) is a sequence of vertices \( \{v_1, v_2, v_3, \ldots, v_l\} \) (\( v_i \neq v_j \) if \( i \neq j \)), such that \( \{v_i, v_{i+1}\} \) and \( \{v_l, v_1\} \) are edges of the graph. For example the sequence \( \{1, 8, 2, 7\} \) is a cycle in \( G(F) \).

A chord of a cycle is an edge between two non consecutive vertices, so there is an edge \( \{v_i, v_j\} \), with \( j \neq i \pm 1 \). This chord makes it possible to split the cycle in two subcycles. The cycle \( \{1, 8, 2, 7\} \) has a chord, namely the edge \( \{7, 8\} \).

A chordal graph is a graph with the special property that every cycle with length larger than 3 has a chord. If and only if a graph is chordal, there exists a perfect elimination ordering, i.e. an ordering such that no extra fill is created during the factorization [52]. The search for the ordering that gives the minimum fill can be translated to the question what is the smallest number of edges that need to be added to make a graph chordal.

In the example in Figure 3.1 we have only two cycles of length larger than 3. The first is \( \{1, 8, 2, 7\} \), which has a chord. The second cycle is \( \{4, 5, 6, 8\} \) and this cycle has no chord. Consequently a matrix with the fill pattern \( F \) has no perfect elimination ordering. The creation of new fill during factorization is inevitable.

To make the graph chordal we have to add only one edge \( \{4, 6\} \) (or \( \{5, 8\} \)). With this extra edge there would be a perfect elimination ordering, namely \( \{1, 3, 2, 7, 9, 5, 4, 6, 8\} \). If this ordering is used for \( A \), only two new entries will be created during factorization, while with the original ordering eight new nonzeros, the \( \circ \)'s in \( F(A) \) in Figure 3.1, were created during Gaussian elimination. Note that each new edge connects two vertices, hence gives two nonzeros in the matrix.

The last graph shown in the figure is the elimination tree of the matrix. The elimination tree is a special type of graph that is constructed in the following way. Each vertex is assigned precisely one parent (see [52]):

\[
\text{PARENT}[i] = \min_j \{i < j | l_{ji} \neq 0\},
\]
where \( l_{ji} \) is the entry on the \( j \)-th row and \( i \)-th column of the lower triangular matrix \( L \) of Equation (3.3). So to find the parent of \( i \) we search the first nonzero lower diagonal entry in the \( i \)-th column of \( L \). One gets the elimination tree by drawing an edge between each vertex and its parent.

Because all vertices only have one parent, we automatically get a graph without any cycles, which is called a tree. Note that one parent is allowed to have more than one child. The nodes that do not have a child are called the leafs of the tree. The tree \( T(A) \) in Figure 3.1 has three leafs: \( \{1,2,4\} \). The elimination of leafs can be done simultaneously, because they are independent from each other. This can be seen in \( F \) by the fact that the submatrix of rows and columns \( \{1,2,4\} \) form a diagonal matrix.

Operations on the matrix can be translated to operations on the adjacency graph and the elimination tree. This is important for Chapter 5, so we will give an example. Let us perform the first step of Gaussian elimination on matrix (3.4). We can construct the adjacency graph of the Schur complement from the adjacency graph in Figure 3.1: we remove node \( \{1\} \) and connect all its neighbours to each other. The first node has the neighbours \( \{7,8\} \), which are already connected, so in this case the graph hardly changes. The graph of the Schur complement is called the reduced adjacency graph. The construction of the elimination tree of the Schur complement is even simpler: we cut off the leave 1. If we continue Gaussian elimination we continue pruning the tree by cutting off the leaves one by one. A nice extended paper on the role of adjacency graphs and elimination trees in sparse matrix factorization is [52].

### 3.1.3 Fill reducing orderings

Although the fact that the computation of the ordering that gives the minimal fill is NP-complete [89], there are good heuristic algorithms available that give good approximations for that ordering. Well-known are the nested-dissection and minimum degree ordering algorithms.

Nested-dissection [32] uses a top-down approach. It searches in a graph for the smallest set of vertices that cuts the graph in at least two components of approximately equal size. The vertices in this set should be eliminated last. The procedure is repeated for the two components. For a more detailed description of nested dissection see [53, §3.8]. In the graph \( G(A) \) in Figure 3.1 nested-dissection would give node 8 the highest number in the ordering. If node 8 is removed from the graph we get two disconnected components, namely the sets of nodes \( \{1,7,2,3\} \) and \( \{4,5,6,9\} \).

The minimum degree approach starts at low level. The motto of this algorithm is that the vertices with the least degree (number of edges that start at a vertex) should be eliminated first [34]. In \( G(A) \) in Figure 3.1 the nodes 1, 3 and 9 all have degree 1 and are assigned a low number in the new ordering. This procedure is repeated on the reduced adjacency graph. The approximate minimum degree algorithm [2] is a variant that uses an upper bound for the degree, which is cheaper to compute than the degree itself.

Both algorithms appear to be able to produce an ordering that gives sparse factors for huge matrices. The ordering algorithms are quite fast, because they don’t use the size of the entries in \( A \). All necessary information is contained in the adjacency graph. Based on the adjacency graph
and the ordering a symbolic factorization can be build, that tells exactly what the fill pattern of 
$L$ will be. During the actual factorization only the size of the entries has to be computed, which 
is by far the most expensive part of the whole computation.

### 3.1.4 Factorizability

If $A$ is symmetric positive definite any ordering of $A$ will give a matrix that is factorizable, that 
is the factorization in (3.3) exists. However if we have an indefinite problem like the saddle 
point problem in Equation (2.35), the ordering has to be chosen with more care. If we blindly 
apply the approximate minimum degree algorithm to an indefinite matrix, it is possible that 
we get in trouble when the factorization is built: at some stage during Gaussian elimination as 
given by Equation (3.2), say at step $k$, we might get a Schur complement with $a_{11}^{(k)} = 0$, which 
was not allowed. The cause of this problem is that in the construction of the adjacency graph it 
was implicitly assumed that the diagonal entries in the matrix are nonzero. In Chapter 4 we will 
discuss the direct solution of saddle point matrices in detail. There we present a new approach 
to construct a fill reducing ordering for a certain class of saddle point matrices.

### 3.1.5 Numerical stability

Next to the existence of the factorization and the sparsity of the factors in direct methods 
numerical stability plays an important role. We would like to give bounds for the error in 
$x = L^{-T} D^{-1} L^{-1} b$, due to backward and forward substitution. As shown in [43, Ch.9] and [22, 
Ch.4,5], the error in $x$ can be bounded with norms in $L$ and $D$ and the growth factor $\rho$, that is 
declared as

$$
\rho = \frac{\max_{i,j,l} |a_{i,l}^{(l)}|}{\max_{i,j} |a_{i,j}|}. \quad (3.5)
$$

The growth factor is the largest entry that occurs in the Schur complements during Gaussian 
elimination divided by the largest entry in the matrix $A$. In case of symmetric positive definite 
matrices this growth factor is 1, so there is no problem with stability. However for indefinite 
matrices there is no better bound than $\rho \leq 2.57^{n-1}$ (see [43, §11.1]) which is a huge number if 
$n$ becomes large. In Chapter 4 we show that for a large class of saddle point problems we can 
compute a much smaller bound for the growth factor.

If the problem is two dimensional, direct solvers can compete in many cases with the iterative methods we will introduce in the next section. And in general direct methods are more robust than iterative methods. The disadvantage is that they often require more memory. Moreover the factorization time in direct methods is usually very high.
3.2 Iterative methods

As an alternative for the direct methods, one can use an iterative method. Instead of directly computing a solution that is as accurate as possible, the solution is approximated and at each step of the iterative method we try to improve the approximation. In this section we won’t pay attention to the classical iterative methods like Jacobi, Gauss-Seidel and SOR (see [90]), because we want to focus on Krylov-subspace methods.

Given some initial guess for the solution \( x_0 \) we have an initial residual \( r_0 = b - Ax_0 \). The \( i \)-th Krylov space is the space spanned by

\[
K^i(A; r_0) = \{ r_0, Ar_0, A^2r_0, \ldots A^{i-1}r_0 \}.
\]

There is a wide range of Krylov subspace methods available. They have in common that the \( i \)-th approximation of \( x \) satisfies \( x_i \in K^i(A; r_0) \), but there are differences in the way the approximation in this space is chosen. For example GMRES chooses the \( x_i \) for which the norm \( \| b - Ax_i \|_2 \) is minimal over \( K^i(A; r_0) \), whereas the conjugate gradient method minimizes the real error in the \( A \)-norm.

For a detailed overview on Krylov subspace methods, the differences between the methods, the construction of an orthogonal basis for \( K^i(A; r_0) \) and theories about convergence see [82].

In this thesis we mainly use GMRES [64] or its flexible variant FGMRES [62].

3.2.1 Preconditioning

For symmetric positive definite matrices \( A \) the convergence in Krylov subspace methods is related to the eigenvalues of \( A \), in particular the spectral condition number \( \kappa = \lambda_{\text{max}}/\lambda_{\text{min}} \). If this number is large the convergence of \( x_i \) to \( x \) will be slow. In that case a remedy is the use of a preconditioner \( P \) to speed up the convergence. This preconditioner has to satisfy three properties:

(i) \( P \) is a good approximation to \( A \) in a certain sense,
(ii) \( P \) is relative cheap to construct,
(iii) the equation \( Px = b \) is much easier to solve than \( Ax = b \).

To clarify (i): preferably the condition number of the preconditioned matrix, i.e. \( P^{-1}A \), should be small.

Again we have the problem that there is a wide range of methods that provide preconditioners: incomplete LU factorizations, multigrid methods, sparse approximate inverses, see for an overview [82, Ch.13] and [53, Ch.8].

The formulation of the preconditioner above suggests \( P \) is a matrix. However we allow \( P \) to be a more general operator on vectors. The preconditioners we describe in Chapters 6 and 7 can be iterative solvers itself. This nested iteration forces us to use a flexible Krylov method, i.e. a Krylov methods that allows variable preconditioning; for that we use FGMRES [62].
3.2.2 Incomplete LU factorization

An important class of preconditioners is formed by incomplete LU factorizations. This type of methods is based on the \textit{LDU}-factorization as we described it in Section 3.1. The major drawback of the construction of an exact \textit{LDU}-factorization is the growth of the fill, which has large impact on both memory requirements and construction time. The basic idea behind incomplete LU-factorizations is the reduction of the nonzeros in the Schur complements.

The general framework for the construction of an incomplete LU factorization is the following. First of all a subset of variables \( x_1 \subset x \) is selected; this set may consist of a single variable. Let \( x_2 = x \setminus x_1 \) be the complement of this set. The matrix \( A \) is reordered according to this partitioning and split into four parts,

\[
A = \begin{bmatrix}
A_{11} & A_{12} \\
A_{21} & A_{22}
\end{bmatrix}.
\]

The next steps are the approximation of \( A_{ij} \) with matrices \( \hat{A}_{ij} \) and the computation of the approximate Schur complement \( S = \hat{A}_{22} - \hat{A}_{21}\hat{A}_{11}^{-1}\hat{A}_{12} \). In most ILU algorithms the approximation will be such, that the Schur complement is easy to compute. This is the case if \( \hat{A}_{11} \) is a diagonal matrix. Now we have the following incomplete factorization for \( A \),

\[
\begin{bmatrix}
I & 0 \\
\hat{A}_{21}\hat{A}_{11}^{-1} & I
\end{bmatrix}
\begin{bmatrix}
\hat{A}_{11} & \hat{A}_{12} \\
0 & S
\end{bmatrix}.
\]

There are many variants of incomplete LU factorization. One can choose for example between a drop-by-position or a drop-by-size criterion, whether to apply Gustafsson-modification, design a sophisticated search algorithm to select the right \( x_1 \), and so on. For an overview of preconditioning methods in general and incomplete LU factorizations in particular we refer to [65] and [53, Ch.8].

Amongst the best incomplete LU factorization, there is \textit{MRILU} [10]. This method is a black-box solver. It is based on matrix entries only and is comparable to algebraic multigrid.

3.3 Solving large linear systems in THCM

Except for Chapter 4 all remaining chapters in this thesis are about preconditioning. The main goal is to find a good preconditioner for the Jacobian matrix (2.34). But on the way we study preconditioners for the simpler problem (2.35).

In previous version of the thermohaline ocean model [86] equations with the Jacobian matrix (2.34) were solved with \textit{GMRES} in combination with \textit{MRILU} as preconditioner. \textit{MRILU} used \textit{clustering of variables}, i.e. the variables \((u,v,w,p,T,S)\) that belong to the same cell are treated as a single variable. Although \textit{MRILU} is able to build a preconditioner for the matrix, where many other methods simply fail, there is a number of disadvantages. First of all we have to choose a small drop tolerance in order to get a reasonable preconditioner. Immediate consequence is that the factorization contains a large amount of fill and therefore the construction
of the preconditioner takes much time and memory. A second drawback is that MRILU does not see the structure of the matrix, just because of the clustering. However without clustering MRILU is not able to build a good preconditioner at all, because it gets in trouble with the zero diagonal blocks.

In an overview paper on the history of iterative methods [65] the authors state: “One ray of hope for those problems that are hard to solve by iterative techniques, is to combine techniques from direct and iterative solution technologies.” Hence at the start of the project we had the plan to reduce the memory requirements and construction time of MRILU by incorporation of the fill reducing orderings we discussed in Section 3.1.3. The result of the efforts to combine fill reducing orderings and incomplete LU factorizations can be found in Chapter 5. Unfortunately the results in that chapter were not as promising as we had hoped. For Poisson problems it works fine, but for coupled partial differential equations, like the Stokes equation, which gives a saddle point problem, the results are disappointing.

Chapter 4 can be considered as a byproduct of a study on direct solution methods for saddle point problems.

We searched the solution in a different direction and decided that we somehow had to exploit the structure of the matrix (2.34) in the design of a new preconditioner. As we will see in Chapter 6 for the saddle point problem and in Chapter 7 for the ocean matrix, it is useful to incorporate knowledge about different variables in the solution process. A heterogeneous system should be given a heterogeneous treatment.