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):

Copyright
Other than for strictly personal use, it is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license (like Creative Commons).

Take-down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Downloaded from the University of Groningen/UMCG research database (Pure): http://www.rug.nl/research/portal. For technical reasons the number of authors shown on this cover page is limited to 10 maximum.
Chapter 5

A fill-reducing ordering for use in incomplete LU factorizations

5.1 Introduction

In Chapter 3 we described the two methods to solve a linear system $Ax = b$, with $A \in \mathbb{R}^{n \times n}$ sparse and $x, b \in \mathbb{R}^{n}$: we can choose between direct and iterative methods. In general, these approaches are quite different, however, in this chapter we will combine elements of both.

Direct methods compute an exact factorization of the matrix, hence, $A = LU$ with $L$ a lower triangular matrix and $U$ an upper triangular matrix. Herewith the linear system can easily be solved using forward and backward substitution. The amount of work depends on the amount of non-zeros (the fill) in the factors $L$ and $U$. Smart ordering of the variables in $x$ can reduce the fill substantially and it is attractive to incorporate this idea in incomplete factorizations.

On the other hand there are the iterative methods. These methods generate a sequence $\{x_1, x_2, ..., x_n\}$ of approximations to the solution of $Ax = b$; popular are the Krylov subspace methods. Preconditioning, e.g. by incomplete LU factorization, can improve the performance of the methods considerably. Thus constructed iterative methods appear to be competitive to direct ones for large linear systems.

An historical overview of iterative methods can be found in [65]. In the conclusion of that paper it is remarked that ideas from direct and iterative methods should be combined to solve problems that are hard to solve by iterative methods.

In this chapter we explore the possibilities to use orderings from direct methods in incomplete LU factorizations. The idea of combining both worlds is not completely new. In [24] a huge number of orderings is combined with a ICCG(0) factorization in preconditioned Conjugate Gradient method and in [16] the effect of ordering in ILU factorizations with a high level of fill is considered.

In the next two sections we will shortly introduce the techniques from the two worlds that we will use: first the fill-reducing ordering as used in direct methods and thereafter the incomplete LU factorization as preconditioner in iterative methods. The most important and most
Chapter 5. A fill-reducing ordering for use in incomplete LU factorizations

<table>
<thead>
<tr>
<th>ordering</th>
<th>fill ($\times 10^6$)</th>
<th>relative fill</th>
<th>construction time (sec)</th>
<th>solve time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>natural ordering</td>
<td>33.1</td>
<td>102.1</td>
<td>34.44</td>
<td>1.02</td>
</tr>
<tr>
<td>reverse Cuthill-McKee (SYMRCM)</td>
<td>22.2</td>
<td>68.6</td>
<td>18.74</td>
<td>0.72</td>
</tr>
<tr>
<td>minimum degree (SYMMMD)</td>
<td>3.7</td>
<td>11.5</td>
<td>3.00</td>
<td>0.21</td>
</tr>
<tr>
<td>approximate minimum degree (SYMAMD)</td>
<td>3.6</td>
<td>11.1</td>
<td>2.66</td>
<td>0.22</td>
</tr>
</tbody>
</table>

Table 5.1: Number of nonzeros (fill) in the factors of a Poisson matrix on a $255 \times 255$ grid for several orderings. Between brackets: the corresponding MATLAB functions.

An extensive section of this chapter is Section 5.4. There we will explain in detail how to combine both techniques in a new algorithm for ordered incomplete LU factorization. In Section 5.5 the algorithm is applied to a few problems and the chapter closes with some conclusions in Section 5.6.

5.2 Fill-reducing orderings in direct methods

In the construction of an exact LU factorization the ordering of the variables in the process of Gaussian elimination is very important. A smart ordering can reduce the fill (the number of non-zeros) in the factors. Fill reduction has several advantages: less memory to store the factors is needed, computations with the factors become cheaper and the construction of the factors becomes cheaper as well.

Because of these advantages one would like to have the special ordering that generates the least fill. Unfortunately the minimum fill problem is NP-complete [89], so it is not practical to compute that ordering. This is not a big issue, because fortunately there are efficient heuristic algorithms that produce a near-minimum fill ordering. In [53, Ch.3] several of these algorithms for ordering in sparse linear systems are presented. We introduced the two most important in Section 3.1.

In this chapter we will avoid going into further details, instead we will show the importance of an appropriate ordering by an example. In Table 5.1 we applied four different ordering algorithms to a Poisson matrix on a square grid of $255 \times 255$ nodes. We computed the fill and the relative fill, i.e. the fill of the factors divided by the fill of the original matrix. Furthermore we measured the construction time and the time needed to solve an equation with the factors. We did the computations in MATLAB and the ordering functions used are mentioned in the table in brackets. The table not only shows that fill reduction reduces the amount of work in the solution phase, but it does even more seriously in the construction phase.

There is one important disadvantage in the use of direct methods: even if the best fill-reducing ordering is used, the fill in the factors grows with the problem size. If we increase the problem size the fill in the factorization will have a seriously larger increase. Therefore, with direct methods the memory limits of any computer are easily reached.
5.3 Preconditioning via incomplete LU factorization

The general idea of preconditioning is to construct a matrix \( \hat{A} \), that is a good approximation for \( A \), but such that equations with \( \hat{A} \) are much easier to solve than equations with \( A \). With a good approximation we mean that the eigenvalues of \( \hat{A}^{-1}A \) are not far from one.

One way to obtain a preconditioner is the construction of an incomplete LU factorization. A lower triangular matrix \( \hat{L} \) and an upper triangular matrix \( \hat{U} \) are computed such that \( \hat{A} = \hat{L}\hat{U} \) is close to \( A \). The incomplete factors \( \hat{L} \) and \( \hat{U} \) should be sparser than the exact factors in any fill-reducing ordering. The sparser the incomplete factors, the cheaper we can solve equations with \( \hat{A} \).

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,

\[
A = \begin{bmatrix}
A_{11} & A_{12} \\
A_{21} & A_{22}
\end{bmatrix}.
\] (5.1)

The next steps are the approximation of \( A_{ij} \) by 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 \( A \),

\[
\begin{bmatrix}
I & 0 \\
\hat{A}_{21}\hat{A}_{11}^{-1} & I
\end{bmatrix}
\begin{bmatrix}
\hat{A}_{11} & \hat{A}_{12} \\
0 & S
\end{bmatrix}.
\] (5.2)

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].

The use of Krylov subspace methods in combination with preconditioning via incomplete LU factorization is a powerful tool as is shown in [10] and [9], where MRILU is used.

5.4 Fill-reducing orderings in incomplete LU factorization

In this section we will propose a method to combine fill-reducing orderings and incomplete factorization. The central idea is quite simple: compute some fill-reducing ordering for the matrix and start performing Gaussian elimination on the matrix according to this ordering, interrupt the process a number of times for a search for a set of nodes that can be eliminated inexacty (see Algorithm 5.4.1).
Algorithm 5.4.1. Ordered incomplete LU-factorization.

1. Compute a fill-reducing ordering for the matrix.

2. Alternate the following steps:
   (a) Reduce the matrix by exact elimination according to the ordering.
   (b) Reduce the matrix by inexact elimination.

In Section 3.1 we explained that Gaussian elimination on a matrix is related to pruning of the elimination tree. The pruning at step 2(a) of the algorithm above is regular, so we cut off a number of leaves of the elimination tree. Clearly this does not affect the structure of the tree. It loses some of the leaves and might become less deep, but the nodes that remain keep the same parents.

The pruning at step 2(b) is totally different. The nodes that are eliminated here can be spread all over the tree. Pruning in this case means that we cut out nodes in the middle of a branch. Even the root is candidate for pruning. Obviously this pruning will change the tree: nodes lose their parents and get new ones somewhere in the tree; whole branches will move. The inexact elimination has to be done very carefully, otherwise we totally ruin the tree and we make the effort we made in step 1, where we construct the fill reducing ordering and the corresponding tree, worthless.

As an example we will apply the Algorithm 5.4.1 to the matrix in Equation (3.4). Suppose that the fill reducing ordering computed in step 1 is the trivial fundamental ordering $q_F = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$. The elimination tree that corresponds to this ordering is plotted in Figure 3.1. A possible evaluation at step 2 could be the following: 2(a) the nodes $\{1, 2\}$ are eliminated exactly, 2(b) the nodes $\{5, 7\}$ are eliminated inexactly, 2(a) nodes $\{3, 4\}$ exactly, 2(b) node $\{8\}$ inexactly, finally 2(a) nodes $\{6, 9\}$ exactly. In fact we changed the ordering to $q' = \{1, 2, 5, 7, 3, 4, 8, 6, 9\}$, where some eliminations won't be carried out exactly.

In the second half of this section we will pay attention to the conditions for inexact elimination. First we want to focus on the question when to apply steps 2(a) and 2(b). For an answer to that question we have to study the ordering of the elimination tree.

### 5.4.1 Tree ordering

In step 1 of Algorithm 5.4.1 we compute an ordering that reduces the fill in the factors of an exact factorizations. We will call this ordering the fundamental ordering. Given this ordering there is some freedom to change it without changing the structure of the elimination tree, which determines the fill in the factors. This is caused by the fact that we are free to renumber the nodes in the elimination tree, as long as parents have higher numbers than the children. Orderings that give elimination trees with the same structure are called equivalent.

Matrix (3.4) with the trivial fundamental ordering $q_F = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$ gives the elimination tree in Figure 3.1. However the nodes 1 and 2 are both leaves, so they can be
interchanged without changing the structure of the elimination tree. Consequently the ordering $q'_{F} = \{2, 1, 3, 4, 5, 6, 7, 8, 9\}$ is equivalent to $q_{F}$.

An elimination tree generally has many equivalent orderings. However most algorithms prefer the *postordering* for the elimination tree, which numbers all the subtrees consecutively. For our example the elimination tree with postordering is the left tree in Figure 5.1. The subtree rooted at node 7 is numbered first. The postordering permutation equivalent to the trivial ordering is $q_{P} = \{1, 2, 3, 7, 4, 5, 6, 8, 9\}$.

In Algorithm 5.4.1 we want to jump a few times between exact elimination (step 2(a)) and inexact elimination (step 2(b)). In that case the postordering for the elimination tree is not very convenient. The problem is that during the exact elimination all changes happen on a small subtree, while during the inexact elimination changes appear all over the tree. In the example one can see that the elimination of the first three nodes in the postordering (\{1, 2, 3\}) does not affect any of the nodes in the subtree rooted at node 6. The point is that we would like to balance the pruning of step 2(a) better over the whole tree, because in that case the changes in the underlying adjacency graph are well spread over the graph. Then a new search in step 2(b) for a set that can be eliminated inexacty is really different from the previous search.

We would like to have an ordering of the elimination tree, such that elimination of a number of nodes changes the dependencies of the remaining nodes in as many subtrees as possible. This obviously happens if all leaves are eliminated. The leaves form an independent set, so they can be eliminated simultaneously. If the leaves are removed from the tree, we get a new tree, that of course has a new set of leaves. That set can be eliminated as the next level. In this way we get an ordering that we will call a *level ordering* where all leaves are numbered first, then the leaves of the reduced tree, etcetera. Furthermore we define a *level* as the set of all leaves of a (reduced) tree. The middle tree in Figure 5.1 has a level ordering. There are five levels, namely the sets of nodes \{1, 2, 4\}, \{3, 5\}, \{7, 6\}, \{8\} and \{9\}, which gives the equivalent ordering $q_{L} = \{1, 2, 4, 3, 5, 7, 6, 8, 9\}$.

The attempt to reduce the system via inexact elimination could be naturally placed between
the regular exact elimination of two consecutive levels. Unfortunately in the level ordering the number of levels is equal to the depth of the tree, which becomes a large number for large problems. This means that we have to perform the relative expensive procedure of a search for inexact elimination many times. Furthermore the last levels exist of very few nodes. Exact elimination of these levels change only a few dependencies. A new attempt for inexact elimination seems vain in advance.

This problem can be tackled by merging consecutive levels, but we prefer a more sophisticated approach. To reduce the number of levels we will define a new type of equivalent ordering. The elimination tree contains many chains of parents with only one child. In general close to the root the chains are long, whereas close to the leaves the chains are short. We use the idea of chains in a variant of the level ordering in order to reduce the number of levels. In the bare-branch ordering, we will number chains instead of leaves. In this new kind of equivalent ordering we start again at the leaves. We number a leaf and if this leaf is an only child we number the parent too. All only-child ancestors are numbered until we find an ancestor with more then one child. We do this for all leaves and put them with their only-child-ancestors in the first level. The second level exists of the leaves and only-child-ancestors of the reduced elimination tree and so on. The chains of parents with only one child can be seen as bare branches of the elimination tree, which explains the name of the ordering. Figure 5.1 shows a bare-branch ordering. The tree has only two levels, namely the sets \( \{1, 2, 3, 4, 5, 6\} \) and \( \{7, 8, 9, 10\} \). Consequently the bare-branch ordering is \( q_B = \{1, 2, 3, 4, 5, 6, 7, 8, 9\} \) and we see that the fundamental ordering is more than equivalent to \( q_B \), it is exactly the same.

In the bare-branch ordering the number of levels no longer depends on the depth of the tree, but on its width. The width of a tree is equal to the number of leaves, which is related to the number of splittings. A splitting is a node of the elimination tree that has at least two children. The number of splittings determines the number of levels in the bare-branch ordering. In a balanced tree the splittings will be equally spread over the subtrees and the number of levels will be approximately \( \log_2(\text{#leaves}) \). In fact this number is an overestimate for the number of levels, because splittings can have more than two branches and probably the tree is not perfectly balanced, which also reduces the number of levels.

For the example trees in Figure 5.1 the differences between the orderings are small. However if we want to apply Algorithm 5.4.1 to large matrices it is important to use the bare-branch ordering. The advantage over the postordering is that the bare branch orderings ensures that in exact elimination the pruning is well balanced over the tree. In one level all bare-branches are eliminated at the same time. The advantage over the level ordering is that we have a small number of levels even if we have a large matrix. Numerical experiments support this. In Table 5.2 we compare the number of levels of the bare-branch ordering with that of the level ordering. The matrix is a Poisson matrix for which an approximate minimum degree ordering is constructed with the MATLAB-function SYMAMD. We also added a column with the estimate \( \lceil \log_2(\text{#leaves}) \rceil \), which appears to be almost equal to the number of levels in the bare-branch ordering.

We note that it is easy to transform a postordering into one of the other orderings; only one sweep through the elimination tree is required. Hence the bare-branch ordering can be
Table 5.2: Number of levels in two equivalent orderings for the Poisson matrix on several grid-sizes. The fundamental ordering is approximate minimum degree. Shown are the level ordering (level) and the bare-branch ordering (branch). The last column (est) contains the estimate for the number of levels based on the width of the tree.

<table>
<thead>
<tr>
<th>grid size</th>
<th>number of nonzeros</th>
<th>tree depth</th>
<th>tree width</th>
<th>number of levels</th>
<th>bare-branch</th>
<th>est</th>
</tr>
</thead>
<tbody>
<tr>
<td>3×3</td>
<td>33</td>
<td>6</td>
<td>4</td>
<td>6</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>7×7</td>
<td>217</td>
<td>19</td>
<td>13</td>
<td>19</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>15×15</td>
<td>1065</td>
<td>50</td>
<td>50</td>
<td>5</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>31×31</td>
<td>4681</td>
<td>115</td>
<td>443</td>
<td>115</td>
<td>7</td>
<td>9</td>
</tr>
<tr>
<td>63×63</td>
<td>19593</td>
<td>315</td>
<td>1915</td>
<td>315</td>
<td>8</td>
<td>11</td>
</tr>
<tr>
<td>127×127</td>
<td>80137</td>
<td>678</td>
<td>7931</td>
<td>678</td>
<td>11</td>
<td>13</td>
</tr>
<tr>
<td>255×255</td>
<td>324105</td>
<td>1484</td>
<td>32251</td>
<td>1484</td>
<td>12</td>
<td>15</td>
</tr>
<tr>
<td>511×511</td>
<td>1303561</td>
<td>3299</td>
<td>130043</td>
<td>3299</td>
<td>14</td>
<td>17</td>
</tr>
<tr>
<td>1023×1023</td>
<td>5228553</td>
<td>6349</td>
<td>522235</td>
<td>6349</td>
<td>16</td>
<td>19</td>
</tr>
<tr>
<td>2047×2047</td>
<td>20942857</td>
<td>14730</td>
<td>2093051</td>
<td>14730</td>
<td>18</td>
<td>21</td>
</tr>
</tbody>
</table>

5.4.2 Inexact elimination

Now we have introduced the bare-branch ordering it is natural to apply step 2(b) in Algorithm 5.4.1 directly after the exact elimination of a level in step 2(a). Then we try to reduce the size of the matrix by inexact elimination between the exact elimination of two consecutive levels. As the number of levels is relative small, we don’t have to apply step 2(b) too often.

As we stated before, the inexact elimination of nodes should be done carefully. The elimination of nodes that are not a leave, will change the tree. For example, if we remove node 7 in a tree in Figure 5.1, the tree will definitely change, e.g. due to exact elimination of node 7 in the adjacency graph in Figure 3.1 a new edge is created between the nodes 1 and 2. The nodes 1 and 3 lose 7 as their parent and get a new one somewhere else in the tree: 2 becomes the parent of 1 and 8 the parent of 3. The structure of the tree is lost, because leave 1 moves to a different branch. As we want to maintain the structure of the tree as much as possible, we demand that the new parent is one of the ancestors of node 7: node 8 or 9. Hence the edge between nodes 1 and 2 in the adjacency graph, which causes 2 to become the parent of 1, is unwanted. In general the nodes in disjoint subtrees are not connected in the adjacency graph and are never allowed to become connected by inexact elimination. If such edges are created by the elimination of a node in step 2(b), we have to drop them immediately.

This demand has some consequences for the framework of incomplete LU factorization as sketched in Section 5.3. Suppose we have chosen the set \( x_1 \subset x \), computed \( x_2 = x \setminus x_1 \) and partitioned the matrix \( A \) as in Equation 5.1. We will use a diagonal approximation for \( A_{11} \), so \( \hat{A}_{11} = D_{11} \), but the blocks \( A_{12} \) and \( A_{21} \) will not be approximated. Now we can compute the
Schur-complement $S = A_{22} - A_{21}D_{11}^{-1}A_{12}$. The matrix $S$ has probably fill, where $A_{22}$ has not. Hence we drop all new entries and approximate $S$ with $\hat{S}$, that has the same fill pattern as $A_{22}$. Of course the more fill is dropped the worse the approximation will be. So we have to look for a set such that the dropping in both $A_{11}$ and the Schur-complement is limited. This is what the algorithm does: look for a set $x_1 \subset x$ such that the $E_{11} = A_{11} - D_{11}$ and $E_{22} = S - \hat{S}$ is “relatively” small. In the current algorithm this dropping is computed exactly, which of course is expensive. Smart estimates might make the algorithm much faster, but because at the moment we are mainly interested in the possibility at all we won’t pay attention to that in this chapter.

The procedure sketched here will end up in the following incomplete LU factorization.

$$
\begin{bmatrix}
A_{11} & A_{12} \\
A_{21} & A_{22}
\end{bmatrix}
= 
\begin{bmatrix}
I & 0 \\
D_{11} & A_{12}
\end{bmatrix}
\begin{bmatrix}
D_{11} & A_{12} \\
0 & \hat{S}
\end{bmatrix}
+ 
\begin{bmatrix}
E_{11} & 0 \\
0 & E_{22}
\end{bmatrix}.
\tag{5.3}
$$

In general this incomplete factorization will be a good one if the error matrix $E$ is small relative to $A$. Of course we cannot compute $A^{-1}E$, so we look at $D^{-1}E$, where $D$ is the diagonal of $A$. We can do this without much concern, because we still relate the error to the size of the entries in $A$. In the search algorithm we look for a set $x_1 \subset x$ such that $D^{-1}E$ is small in some norm. The existence of such a set depends on the drop tolerance and the type of problem. If a nonempty $x_1$ is found, this set is eliminated in the way sketched above.

Because the fill patterns of $\hat{S}$ and $A_{22}$ are the same, we ensure that no entries are created between disjoint subtrees. If we remove the eliminated nodes from the tree, connecting orphans to the first ancestor of the eliminated parent, we get a new tree, that is an elimination tree for $\hat{S}$. The corresponding ordering is still fill reducing, so the original exact factorization procedure can be continued.

Applying the algorithm sketched so far we observe that for large values of the drop tolerance the elimination tree falls apart. After a few times of exact and inexact elimination we get a forest instead of a single tree. This is caused by the fact that the Schur complement becomes reducible or in graph terminology the grid is no longer connected: it consists of two or more components. With the tree falling apart the number of iterations grows seriously and zeros appear on the diagonal of $U$, so the factorization becomes singular. Of course these zeros can be easily replaced by a nonzero number, but it will still have a bad effect on the number of iterations.

This motivates to put an extra demand on the selection of the sets, namely $x_2 = x \setminus x_1$ should be connected. Unfortunately it is not easy to check whether the deletion of a node will create two or more components. Therefore we make an even stronger demand: a node is considered for inexact elimination only if its neighbours in $x_2$ are connected. This is easy to check because the involved grid is rather small. For a five point stencil the demand will prevent any inexact elimination on the first level, so at that level we relax the demand. The neighbours of the neighbours in $x_2$ should be connected.

With this extra restriction the method becomes quite robust. For any drop tolerance the factorization is non singular. For the problems we treat in the next section this restriction has
5.5 Numerical results

hardly any effect on the fill, but the number of iterations reduces with a factor two or more.

The order of the nodes in the search for the set \( x_1 \) appears to have some influence on the factorization also. The best search order seems to be the opposite of the exact elimination order. The nodes in the highest level are considered for inexact elimination first. This order will have the largest effect on the fill of the factors.

5.4.3 Implementation

The algorithm sketched in this section has been implemented in MATLAB. The current version of the algorithm is not very efficient. This is mainly caused by the fact that in the search for incomplete elimination the errors are computed exactly. However in this chapter we are mainly interested in the possibility at all of combining fill-reducing orderings and incomplete factorization. We expect that smart estimates for the error and a detailed time study of the algorithm can make it competitive to the exact factorization.

The algorithm requires a fill-reducing ordering with a balanced elimination tree and a large width. This means that methods like minimum degree or nested dissection are appropriate. The reversed Cuthill McKee ordering reduces the bandwidth of the matrix, which means that often the first off-diagonals are completely filled. A tridiagonal matrix has small bandwidth, but its tree is not interesting: it exists of a single chain of nodes. As the reversed Cuthill McKee ordering gives an extremely unbalanced tree, we cannot use it in Algorithm 5.4.1.

5.5 Numerical results

In this section we will show numerical results for three problems: a Poisson equation, a convection-diffusion equation and a Stokes equation. In all cases we apply a priori a (block) diagonal scaling. We use the approximate minimum degree ordering that is available in MATLAB and in the incomplete factorization we use Gustafsson-modification, i.e. we lump dropped elements on the diagonal. We will focus on the fill and the quality of the factorization. The quality is measured by the number of iterations in a Krylov subspace method that is needed to reach an accuracy of \( 10^{-6} \).

We will compare our method to the incomplete LU or incomplete Cholesky factorization of MATLAB with natural and approximate minimum degree ordering.

All experiments are run on a PC with 2 GB memory and a 3 GHz processor.

5.5.1 Laplace equation

First the method is tested on the Laplace equation on the unit square with Dirichlet conditions \( u = 1 \) on the boundary. Hence, the solution is simply the constant value 1. We use a standard
five-point discretization on a uniform grid. So we have the equations
\[
\Delta u = 0 \text{ in } \Omega, \\
u = 1 \text{ on } \partial \Omega.
\]

For the starting vector in the Krylov subspace method (here PCG) we disturb the solution. The distortion exists of a single hump, a smooth function. The starting vector is
\[
u_0(x,y) = 1 + (xy(1-x)(1-y))^2 e^{x^2y}.
\]

The problem is equivalent to the “Uniform-2D” problem in [9]; the initial error is exactly the same. In Table 5.3 it is shown how the fill and the number of iterations behave if the problem size increases for various values of the dropping parameter \(\varepsilon\). The problem size is just the square of the number of grid points in \(x\)- and \(y\)-direction and the fill is the average number of non zeros in \(L+U\) per row. The matrix \(A\) itself has approximately 5 nonzeros per row. In the first column one observes how the fill increases using the minimum degree ordering without dropping. So for the largest problem the fill of the preconditioner is more than 11 times as much as that of the original matrix. In the other columns one observes also an increase of the fill but it becomes less pronounced with increasing \(\varepsilon\) (look for instance to the ratio of the fill for the largest and of that for the smallest problem). Of course, for the exact elimination the number of iterations is precisely one. For the case \(\varepsilon = 0.01\) the fill is still rather high resulting in an accurate factorization which amounts to only 2 iterations to gain 6 digits. The case \(\varepsilon = 0.2\) has a more attractive fill for practical purposes whereas the number of iterations is still rather low. Because of the demand of connectivity the algorithm cannot generate a factorization with less fill than 6.9 in the last column of Table 5.3. A further reduction of the fill would cause the tree to fall apart, which is not allowed.

In Table 5.4 the number of iterations and the number of nonzeros per row in \(L+U\) are shown for the incomplete Choleski factorization with natural and approximate minimum degree ordering. The drop tolerances are chosen such that the level of fill is more or less the same as that in the last row of Table 5.3. Comparing the results we conclude that the factorization of \(OILU\) (an acronym for Algorithm 5.4.1) is better than an ordinary incomplete LU factorization with natural ordering as long as we want a fill that is more than one and a half times the fill of the original matrix.

The amount of work per unknown to solve the system during the iteration is roughly the product of the fill and the number of iterations where the fill is simply 5 times the relative fill. So for the largest problem the amount of work is respectively 55, 86, 105, 84, 95, 157, 220, 255, 323 and 669 with increasing \(\varepsilon\).

The last two values are rather high due to a large number of iterations. We attribute this to the dropping criterion. It can happen that in successive search sweeps connections from the same node are removed. This may accumulate to a rather large amount of dropped connections for such a node, if it is one of the last nodes to be eliminated. Due to this accumulation of local errors, the global error of the factorization can become too large, which results in a bad
5.5 Numerical results

<table>
<thead>
<tr>
<th>problem size</th>
<th>drop tolerance</th>
<th>0</th>
<th>0.01</th>
<th>0.02</th>
<th>0.04</th>
<th>0.1</th>
<th>0.2</th>
<th>0.3</th>
<th>0.4</th>
<th>0.6</th>
<th>0.8</th>
</tr>
</thead>
<tbody>
<tr>
<td>961</td>
<td>nnz per row</td>
<td>21.6</td>
<td>19.2</td>
<td>18.7</td>
<td>17.9</td>
<td>14.6</td>
<td>12.1</td>
<td>10.2</td>
<td>8.1</td>
<td>7.9</td>
<td>6.4</td>
</tr>
<tr>
<td></td>
<td>iterations</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>7</td>
<td>11</td>
<td>12</td>
<td>19</td>
</tr>
<tr>
<td>3969</td>
<td>nnz per row</td>
<td>30.6</td>
<td>26.1</td>
<td>24.6</td>
<td>22.4</td>
<td>16.9</td>
<td>13.4</td>
<td>11.2</td>
<td>8.5</td>
<td>8.2</td>
<td>6.8</td>
</tr>
<tr>
<td></td>
<td>iterations</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>6</td>
<td>10</td>
<td>14</td>
<td>16</td>
<td>30</td>
</tr>
<tr>
<td>16129</td>
<td>nnz per row</td>
<td>41.5</td>
<td>34.1</td>
<td>30.1</td>
<td>25.6</td>
<td>18.3</td>
<td>14.0</td>
<td>11.8</td>
<td>8.7</td>
<td>8.4</td>
<td>6.9</td>
</tr>
<tr>
<td></td>
<td>iterations</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>9</td>
<td>14</td>
<td>19</td>
<td>26</td>
<td>49</td>
</tr>
<tr>
<td>65025</td>
<td>nnz per row</td>
<td>55.3</td>
<td>42.8</td>
<td>35.1</td>
<td>28.0</td>
<td>19.1</td>
<td>14.3</td>
<td>12.2</td>
<td>8.8</td>
<td>8.5</td>
<td>6.9</td>
</tr>
<tr>
<td></td>
<td>iterations</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>5</td>
<td>11</td>
<td>18</td>
<td>29</td>
<td>38</td>
<td>97</td>
</tr>
</tbody>
</table>

Table 5.3: Results of OILU for approximate minimum degree ordering on Laplace equation as preconditioner in PCG

<table>
<thead>
<tr>
<th>natural ordering</th>
<th>drop tolerance</th>
<th>2e-4</th>
<th>4e-4</th>
<th>7e-4</th>
<th>1e-3</th>
<th>3e-3</th>
<th>5e-3</th>
<th>8e-3</th>
<th>2e-2</th>
<th>4e-2</th>
<th>1e-1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>nnz per row</td>
<td>55.9</td>
<td>43.2</td>
<td>35.7</td>
<td>30.2</td>
<td>18.7</td>
<td>14.8</td>
<td>12.9</td>
<td>8.9</td>
<td>7.0</td>
<td>5.0</td>
</tr>
<tr>
<td></td>
<td>iterations</td>
<td>5</td>
<td>7</td>
<td>8</td>
<td>10</td>
<td>19</td>
<td>24</td>
<td>28</td>
<td>41</td>
<td>55</td>
<td>97</td>
</tr>
<tr>
<td>amd ordering</td>
<td>drop tolerance</td>
<td>1e-5</td>
<td>5e-5</td>
<td>1e-4</td>
<td>4e-4</td>
<td>2.5e-3</td>
<td>8e-3</td>
<td>1.5e-2</td>
<td>5e-2</td>
<td>1e-1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>nnz per row</td>
<td>45.4</td>
<td>38.8</td>
<td>35.4</td>
<td>28.1</td>
<td>18.9</td>
<td>14.6</td>
<td>11.7</td>
<td>9.0</td>
<td>7.0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>iterations</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>8</td>
<td>19</td>
<td>31</td>
<td>44</td>
<td>73</td>
<td>105</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.4: Results of incomplete Choleski factorization in MATLAB with natural and approximate minimum degree (amd) ordering on Laplace equation as preconditioner in PCG. The problem size is 65025.

factorization. In MRILU [10] this is prevented by a sum criterion. We did not implement such a criterion which explains the deterioration of the results for large $\varepsilon$.

So far as solving is concerned this is done fastest by the direct solver. We don’t need an iterative method to compute the solution and we can apply the factorization only once. This complies with the experiments in [9] where a similar observation was made in a comparison of methods to solve Poisson like equations. The bottleneck appears to be the construction phase and the amount of memory needed to store the factorization. From [10] we know that the amount of work for the construction goes down drastically when the fill decreases. For a similar case in [10] Table 1, the decrease of the relative fill from 8.7 to 3.6 brought the amount of work down by a factor 6. We expect a similar decrease here.

### 5.5.2 Convection-diffusion equation with grid stretching

To show that the idea can be as easily applied to a less trivial problem, we apply the method to a convection-diffusion equation on a unit square. The equation is given by

$$-\Delta u + u_x + 0.5u_y = 0 \text{ in } \Omega$$
with the same boundary conditions as used for the Poisson equation. In the discretization we use central differences. The Cartesian grid is refined regularly towards the left and lower boundary in such a way that the ratio of the largest and smallest mesh size is 100. In this case the iterations are started with the zero vector.

The results, given in Table 5.5, show that the method behaves very similar on this problem, however the number of iterations is slightly higher, which we attribute to the difficulty of convection and stretching.

### 5.5.3 Stokes

The third problem is the two-dimensional Stokes equation in a driven cavity. For that the following set of equations have to be solved on the unit square \( \Omega \)

\[
\begin{align*}
- \nu \Delta u + \nabla p &= 0 \text{ in } \Omega, \\
\nabla \cdot u &= 0 \text{ in } \Omega,
\end{align*}
\]

(5.4)

where \( u(x, y) \) is the velocity field and \( p(x, y) \) the pressure field; the parameter \( \nu \) controls the amount of viscosity. The non-trivial solution is determined by the boundary conditions that are zero on three sides of the unit square. At the upper boundary \( (y = 1) \) we have a horizontal velocity \( u(x, 1) = 1 \).

We can get rid of the parameter \( \nu \) by defining a new pressure variable \( \tilde{p} = p/\nu \). If the first equation is divided by \( \nu \), we can substitute \( p \) by \( \tilde{p} \) and the parameter \( \nu \) is gone. So we may assume that \( \nu = 1 \).

The equations are discretized on a staggered grid (an Arakawa C-grid), which results in a system of linear equations \( K\tilde{u} = \tilde{b} \), where \( K \) is of the form

\[
\begin{bmatrix}
A & B \\
B^T & 0
\end{bmatrix}.
\]

Because this matrix is indefinite, we use clustering of variables, i.e. the velocity and pressure nodes of the same cell are treated as one variable. Thereby the \( 3 \times 3 \) blocks on the diagonal are invertible. We compute an approximate minimum degree ordering for the block-matrix and

<table>
<thead>
<tr>
<th>problem</th>
<th>( \varepsilon = 0 )</th>
<th>( \varepsilon = 0.01 )</th>
<th>( \varepsilon = 0.1 )</th>
<th>( \varepsilon = 0.2 )</th>
<th>( \varepsilon = 0.3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>size</td>
<td>r.fill</td>
<td>it</td>
<td>r.fill</td>
<td>it</td>
<td>r.fill</td>
</tr>
<tr>
<td>576</td>
<td>3.9</td>
<td>1</td>
<td>3.2</td>
<td>2</td>
<td>2.7</td>
</tr>
<tr>
<td>2401</td>
<td>5.3</td>
<td>1</td>
<td>4.3</td>
<td>3</td>
<td>3.6</td>
</tr>
<tr>
<td>9801</td>
<td>6.9</td>
<td>1</td>
<td>5.6</td>
<td>3</td>
<td>4.2</td>
</tr>
<tr>
<td>39601</td>
<td>8.7</td>
<td>1</td>
<td>7.6</td>
<td>3</td>
<td>5.1</td>
</tr>
</tbody>
</table>

Table 5.5: Results of \( \text{OILU} \) for approximate minimum degree ordering on convection-diffusion equation as preconditioner in \( \text{PCG} \)
apply the ordered incomplete LU factorization. The quality of the factorization is tested in BICGSTAB [81]. The results for several values of the drop tolerance can be found in Table 5.6.

Also here one observes that for large $\epsilon$ the increase of the fill with the problem size is less pronounced than for small $\epsilon$. Unfortunately the number of iterations also increases rather fast for both increasing problem size and increasing $\epsilon$. Even stagnation of convergence occurs for the largest problem for $\epsilon = 0.7$. This is due to a too crude dropping criterion. Since the matrix is indefinite, the preconditioner might become close to singular. Clearly further research on robust dropping criteria is needed here.

Some readers may wonder why we are interested at all in preconditioners for $K$ since there is a number of alternative approaches in which often only a system with $A$ and one with a crude approximation, usual a diagonal matrix, to $B^T A^{-1} B$ needs to be solved. Since the system with $A$ is of Poisson type, many methods are available to solve it, e.g. multi grid, MRILU, and also the current method (see Section 5.5.1). We will see some of these approaches in the next chapter. Our motivation to apply the ordered incomplete LU factorization is that by an accurate factorization of $K$ we can use it as a preconditioner in eigenvalue problems in the way demonstrated in [73].

### 5.6 Conclusions

From the investigations in this chapter we may conclude the following.

1. The bare-branch ordering for a balanced elimination tree is useful for the construction of an ordered incomplete LU factorization. First of all it guarantees that the pruning by exact elimination is well balanced over the elimination tree. Furthermore it has a small number of levels even for large matrices. The inexact elimination can be placed naturally between the exact elimination of two consecutive levels. Any ordering of an elimination tree can be easily transformed to an equivalent bare-branch ordering.

2. The demand that no new fill is created in the Schur complement during the inexact elimination is sufficient to guarantee that the nodes in different subtrees do not get connected and that we do not ruin the fundamental fill-reducing ordering.

<table>
<thead>
<tr>
<th>problem size</th>
<th>$\epsilon = 0$ r.fill</th>
<th>$\epsilon = 0.1$ r.fill</th>
<th>$\epsilon = 0.3$ r.fill</th>
<th>$\epsilon = 0.5$ r.fill</th>
<th>$\epsilon = 0.7$ r.fill</th>
<th>it</th>
<th>it</th>
<th>it</th>
</tr>
</thead>
<tbody>
<tr>
<td>147</td>
<td>4.1</td>
<td>1</td>
<td>4.0</td>
<td>2</td>
<td>3.7</td>
<td>3</td>
<td>3.4</td>
<td>6</td>
</tr>
<tr>
<td>675</td>
<td>6.4</td>
<td>1</td>
<td>6.3</td>
<td>3</td>
<td>5.9</td>
<td>5</td>
<td>5.2</td>
<td>9</td>
</tr>
<tr>
<td>2883</td>
<td>9.4</td>
<td>1</td>
<td>9.0</td>
<td>4</td>
<td>8.3</td>
<td>7</td>
<td>6.9</td>
<td>14</td>
</tr>
<tr>
<td>11907</td>
<td>13.7</td>
<td>1</td>
<td>13.1</td>
<td>4</td>
<td>11.0</td>
<td>8</td>
<td>8.2</td>
<td>21</td>
</tr>
<tr>
<td>48387</td>
<td>18.8</td>
<td>1</td>
<td>17.7</td>
<td>6</td>
<td>13.6</td>
<td>17</td>
<td>9.1</td>
<td>81</td>
</tr>
</tbody>
</table>

Table 5.6: Results for OILU for block approximate minimum degree ordering on Stokes equation as preconditioner in BICGSTAB.
3. For the Poisson and the convection-diffusion problem we observe that the fill obtained by approximate minimum degree can be reduced significantly by inexact elimination, while the work in the solution phase increases only slightly (except when due to the too crude dropping criterion, too much is dropped in part of the matrix). From other sources [10] it is known that the reduction of the fill usually means a more than proportional reduction of the time needed for the construction of the factorization.

4. For the Stokes problem similar fill reductions can be obtained. However, the simple dropping criterion we used may lead to singularities in the preconditioner resulting in loss of speed of convergence. More sophisticated dropping criteria should remedy this situation.

At the end of this chapter we can conclude that the combination of incomplete LU factorizations and fill reducing orderings might be useful in case of Laplace type equations. However if this still holds for coupled partial differential equations is questionable. We made several attempts to improve the dropping criteria, but it hardly changed the results. Hence even if we find more sophisticated dropping criteria, it is unlikely the results in Table 5.6 will improve considerably. We didn’t even mention the much more complex ocean system in equation (2.34). In both cases an approach that uses the structure of the equations (that is not seen due to clustering) might be better. We will study such methods in the next two chapters.