Approaches to reduce the computational cost when solving linear systems of equations arising in Boundary Element Method discretizations

Erik Bängtsson and Maya Neytcheva

November 5, 2003

Abstract

Preconditioned iterative solution methods are compared with the direct Gaussian elimination method to solve dense linear systems $A\mathbf{x} = \mathbf{b}$ which originate from crack propagation problems, modeled and discretized by boundary element (BEM) techniques.

Numerical experiments are presented and compared with the direct solution method available in a commercial BEM package. The experiments show that the preconditioned iterative schemes are competitive compared to the direct solver with respect to both arithmetic operations required and memory demands.

1 Introduction

The need to solve linear systems of equations arises in many numerical applications. The systems may arise from discretizations of differential equations, optimization problems, etc. When the discretization of the original problem is done using a method such as for instance Finite Elements, Finite Differences, Finite Volumes, the matrix of the arising system of equations is large and sparse. In certain application fields it turns out to be more advantageous to discretize with another technique, such as the Boundary Element Method (BEM). The BEM technique reformulates the original differential equation, defined in some domain $\Omega \subset \mathbb{R}^n$ to another differential problem over $\partial\Omega \subset \mathbb{R}^{n-1}$, reducing in this way the number of unknowns and the size of the linear systems to be solved. This approach is particularly attractive for 3D applications where the sparse linear systems may become extremely large. Another class of problems which are successfully treated with BEM are those where the modeled phenomenon propagates in the interior of the finite elements or volumes, such as crack propagation, and expensive re-meshing has to be done in order to resolve the solution.

The price to be paid when using BEM is that, although the problem is solved in a space of one dimension less than the original space, the arising systems of equations are dense. The latter entails high demands on the computer resources both in terms of memory and computational power.

Consider the solution of a linear system of equations

$$A\mathbf{x} = \mathbf{b}, \tag{1}$$

where $\mathbf{x}$ and $\mathbf{b}$ are vectors of length N and A is an $N \times N$ dense matrix.

In recent years much research has been done in two main directions:

(i) how to store the matrix in a compressed form in order to reduce the memory demands (see Section 5 for a brief survey) and

(ii) how to solve the arising system of equations (1).

This work addresses the solution approaches (ii) and how to reduce the corresponding computational cost, measured in terms of arithmetic operations.

Traditionally, direct solution methods such as Gaussian elimination or its symmetric version, the Cholesky factorization, are used to solve systems of equations with dense matrices. The direct methods are known to be robust with respect to various problem parameters and their implementation in the case of dense matrices is straight forward. However, the computational cost in this case is of order $\mathcal{O}(N^3)$, which severely limits the size of the problems which can be treated even on high performance computers.

The need to solve large and complex problems brings another solution technique into focus, namely, the iterative solution methods. A fast convergent iterative solution method, which would require a number of iterations much less than N, will decrease the computational cost to $\mathcal{O}(N^2)$, i.e., the total solution cost will become proportional to the cost of a matrix-vector multiplication with the dense matrix A.

The performance of the iterative solution methods depends to a high extent on the conditioning of the matrix A. For ill-conditioned problems the convergence may be very slow, the methods can stagnate or even diverge. Therefore, the so-called preconditioned form of the iterative solution methods is mostly used, where the matrix A is made better conditioned with the help of an additional matrix (denoted here by M) which is referred to as a preconditioner for A.

The classical way to introduce preconditioning is to observe that the solution to the original system $A\mathbf{x} = \mathbf{b}$ is equivalent to the solution of any of the following systems

$$M^{-1}A\mathbf{x} = M^{-1}\mathbf{b}, \tag{2}$$

$$MA\mathbf{x} = M\mathbf{b}, \tag{3}$$

$$[M]A\mathbf{x} = [M]\mathbf{b}. \tag{4}$$

Formulation (2) indicates that M should be chosen to approximate the matrix A itself. If $M = A$ exactly, the iterative method will converge in one iteration. The latter remark has a theoretical value only since clearly there will be no reduction in the solution costs. In this case each iteration step of the iterative solution method will require one solution with the matrix M. Formulation (3) indicates that M can be sought as an approximation of A^{-1}. In this case each iteration step of the iterative solution method will require one vector multiplication with the matrix M and therefore the preconditioner is referred to as of multiplicative type. Formulation (4) indicates that M does not have to be an explicitly computable matrix, but a procedure, such as in the case of the multilevel/multigrid methods, where we need only the action of M on a vector. In all cases, a number of requirements are imposed on M.

(a) The preconditioned system $M^{-1}A$ (or MA, respectively $[M]A$) must be (much) better conditioned than A. The best would be to achieve a condition number of the preconditioned system, bounded from above independently of the number of degrees of freedom N. In this case the preconditioner is optimal with respect to rate of convergence.

(b) Solutions with M (or multiplications with M, respectively the action of $[M]$ on a vector) must be computationally cheap. The optimal computational cost in this context is proportional to the degrees of freedom N.

(c) The cost to construct the preconditioner has to be bounded, for example, proportional to the cost of the matrix-vector multiplication.

(d) To construct and apply the preconditioner should be efficient on high-performance parallel computer facilities.

Some of the above goals are contradicting (for example (a) versus (b) and (c)). A preconditioner which efficiently improves the condition number is often more complicated to compute and implement efficiently. A simple and easy-to-implement preconditioner on the other hand may not improve substantially the condition number of the system. Therefore, in practice, a trade-off between these is aimed. However, reaching a good balance for the above mentioned criteria is far from trivial and the choice of preconditioner is still an open question for many problems.

In this report numerical experiments with some algebraically constructed preconditioners are performed. All the tests are on matrices obtained from BEM approximations of crack propagation problems, using the so-called Displacement Discontinuity Method (DDM), developed by Crouch [8], see for instance also [1] and the references therein.

In [6] sparse symmetric preconditioners for complex symmetric dense systems arising in computational electromagnetism have been studied. There, the factorized sparse approximate inverse preconditioners are found to be inefficient for the considered class of problems, while Frobenius norm based preconditioners have shown to be efficient and robust.

Another experience in solving BEM systems of equations using an iterative solution method is reported in [16]. There it is reported that the conjugate gradient method produces a convergent, stable and consistent numerical solution when the number of degrees of freedom is increased.

Brief problem formulation and description of the discretization technique

For simplicity, consider a homogeneous elastic body in $\mathbb{R}^2$ with a boundary S and one fracture (crack boundary S_c) as depicted in Figure 1. The crack propagation problem is solved numerically using various discretization approaches. As a competitive alternative to the DDM method we mention here a method developed in [24], where conventional finite elements are used. By introducing 'discontinuity jumps', the part of the discontinuity is made completely independent of the finite element mesh, which enables tracing the fracture propagations within the interior of the finite elements, without changing the initial mesh. The matrices of the arising systems of equations are sparse and all the available machinery for solving sparse linear systems is directly applicable.

When using the DDM technique, the fracture propagation is directly presented in terms of fracture elements, without introducing fracture surfaces. The displacement discontinuities D_i^s in shear direction and D_i^ν in normal direction across a fracture are unknowns.

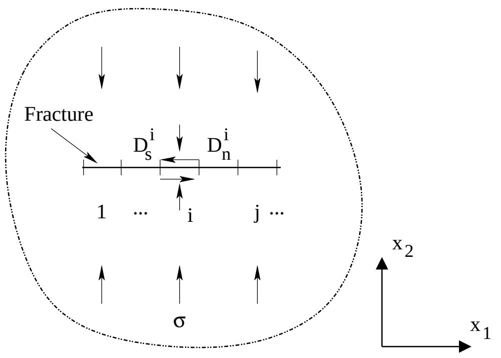


Figure 1: A fracture inside an elastic medium and DDM discretization

Let S and S_c be discretized in N and M elements, correspondingly. The general form of the algebraic system to be solved reads as

$$\begin{array}{l} \sum\limits_{n=1}^{N} \varphi_n^i \int\limits_{S_c} T_{ki}(\mathbf{x}^j, \mathbf{x}) dS_n + \sum\limits_{m=1}^{M} D_m^i \int\limits_{S_c} T_{ki}(\mathbf{x}^j, \mathbf{x}_c) dS_c = \widetilde{t}_k^j, \quad j = 1, \cdots, N \\ \sum\limits_{n=1}^{N} \varphi_n^i \int\limits_{S_c} T_{ki}(\mathbf{x}_c^j, \mathbf{x}) dS_n + \sum\limits_{m=1}^{M} D_m^i \int\limits_{S_c} T'_{ki}(\mathbf{x}_c^j, \mathbf{x}_c) dS_c = \widetilde{t'}_k^j, \quad j = 1, \cdots, M \end{array} \tag{5}$$

where T_{ki} and T'_{ki} are the fundamental solutions in the interior and on the crack surface, respectively, $\varphi^i(\mathbf{x})$ are fictitious loads at point $\mathbf{x} \in S$, $\widetilde{t}_k^j$ and $\widetilde{t'}_k^j$, $k = 1, 2$ are traction boundary conditions on S and S_c, correspondingly.

It is also assumed that the crack is aligned with the coordinate axis $x_2 = 0$, in which case and explicit simplified formula for the fundamental solution T'_{ki} can be used.

In general, three states of a fracture can be distinguished - open, elastic contact or sliding contact, leading to different formulations for the governing system of algebraic equations. The displacement discontinuities D^i of the fracture are obtained by solving the system (5) using some numerical method. The system matrix in (5) is dense and to obtain its solution, in practice direct Gaussian elimination is mostly used.

The matrices are nonsymmetric and are solved by the generalized conjugate gradient method (GCG), see e.g. [2], or the generalized minimal residual method (GMRES), see e.g. [21].

The report is organized as follows. In Section 2 we briefly describe the problem and present some methods to construct preconditioners for its iterative solution. In Section 2.2.1 a strategy to efficiently solve a sequence of linear systems of equations with some special properties is presented. In Section 3 the computational complexity of the different approaches is derived. Section 4 contains experimental results. Some techniques how to store efficiently BEM matrices are briefly mentioned in Section 5. In Section 6 some conclusions are drawn.

2 Preconditioning techniques

When used with a robust preconditioner, iterative solution methods offer substantial savings in computational time and successfully compete with the direct solution methods.

Using preconditioned iterative solution methods for dense systems of equations, as arising in BEM applications, is an active field of research. The so-called (”problem-given”) approach is studied in the works of many authors. One such approach, used to construct preconditioners for singular boundary integral equations, is the operator splitting technique. Recently (see [14]) efficient algebraic multigrid preconditioners have been constructed and applied for the iterative solution of BEM equations, arising from standard Galerkin boundary element discretization of first kind boundary integral operators.

In this report we survey strategies to construct a preconditioner, which are based only on the matrix A and do not use any knowledge of the underlying problem. We refer this strategy as to ”matrix-given”.

2.1 Sparse approximate inverse preconditioning

The idea of the sparse approximate inverse preconditioning technique is to construct the matrix M as an approximation of A^{-1}, such that M has an a priori given sparsity pattern ($\mathcal{S}$), such as a band matrix for instance. The approach is used in many applications. It is first developed for block band matrices and originates in [3, 9] and later is further developed in [13, 12].

There exist various methods to compute the entries of M. One approach is to compute M having a prescribed sparsity structure $\mathcal{S}$ by solving the least square problem

$$\min_{M^{-1}\in\mathcal{S}} \|AM^{-1}-I\|_F^2 = \sum_{j=1}^{n} \min_{m_j\in\mathcal{S}} \|Am_j - e_j\|_2^2,$$

where m_j and e_j are the columns of M and the identity matrix I, correspondingly. Here $\|\cdot\|_F$ denotes the Frobenius norm.

Another simple idea is to require that for all indices $i, j \in \mathcal{S}$ there holds

$$(MA)_{ij} = \delta_{ij}, \tag{6}$$

where δ_{ij} is the Kronecker symbol. For band matrices A the computational cost to compute the entries of M from (6) is equal to solving N small systems of equations of order equal to the number of nonzero elements in the rows of A. Furthermore, the computations for the nonzero entries in each row of M can be performed completely in parallel. We note here that this explicit approach produces a nonsymmetric M even if A is symmetric. The idea in (6), applied for dense matrices can be found in the literature under different names, one of them being the diagonal block approximate inverse (DBAI) technique (see [7]). DBAI constructs M as a matrix with k diagonals (for odd positive integers k) which approximates the inverse of the corresponding band part of A.

As is readily understood from the construction of the approximate inverse, it is important that the entries of the true inverse of A decay quickly away from the main diagonal. Estimates of the accuracy of the constructed preconditioner can be based on this rate of decay and such estimates for some classes of (sparse) matrices are found in [2]. When the BEM-operator has a strongly singular behavior the matrix A is diagonally dominant and numerical evidence shows similar behavior for its inverse. Even exponential decay of the entries in A^{-1} can be shown for strictly diagonally dominant matrices. Therefore, for certain problems it can be expected that the DBAI preconditioner will capture the most significant part of A^{-1}.

The DBAI (or DBAI(k)) approximate inverse preconditioner is constructed column by column by solving N $k \times k$ - linear systems of equations of the form

$$\begin{pmatrix} A_{j-\ell,j-\ell} & \cdots & A_{j-\ell,j} & \cdots & A_{j-\ell,j+\ell} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ A_{j,j-\ell} & \cdots & A_{j,j} & \cdots & A_{j,j+\ell} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ A_{j+\ell,j-\ell} & \cdots & A_{j+\ell,j} & \cdots & A_{j+\ell,j+\ell} \end{pmatrix} \begin{pmatrix} M_{j-\ell,j} \\ \vdots \\ M_{j,j} \\ \vdots \\ M_{j+\ell,j} \end{pmatrix} = \begin{pmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{pmatrix},$$

where $\ell = \lfloor k/2 \rfloor$, $k \in \mathbb{N}$.

The matrix M is computed as a band matrix with or without a wraparound (without a wraparound for the tests presented in this report). The bandwidth of M is determined by the parameter k and the efficiency of this preconditioner is largely dependent on its size. If $k = 1$, the preconditioner only consists of the diagonal of A, $M = diag(A)$. This is the Jacobi preconditioner and it has been applied to precondition dense systems arising in various applications. The main advantage of Jacobi preconditioner is its simplicity. Its efficiency can vary, however, see for example [17], [5] and [15]. While for small k the preconditioner could be a poor approximation of A^{-1}, for large k M becomes expensive to construct and apply. Furthermore, as already mentioned, the efficiency of this preconditioner depends on the rate of decay of the off-diagonal elements in A^{-1}. If the inverse of the matrix contains significant off-diagonal elements out of structure of M, which is chosen in advance, the preconditioner will not be able to capture these, and will be less efficient.

On the other hand, there exist clear advantages using this preconditioner, namely, it approximates the inverse of A directly and no solution of systems but only matrix-vector multiplications with M are required during each iteration. The matrix-vector multiplication with the preconditioner is in general cheaper compared to the matrix-vector multiplication with A itself, and as already mentioned, the construction of M can be fully parallelized. For a theoretical justification of this approach, see [7] and the references therein.

2.2 Incomplete LU-factorization preconditioning

When solving a linear system of equations by a direct method, by applying standard Gauss elimination, the original matrix A is first factored into a lower (L) and upper (U) triangular factors. The factorization phase is followed by a solution phase, consisting of two solutions of systems with the triangular matrices U and L. As is well known, even if A is sparse, the L and U factors can be full matrices.

The idea of the *incomplete* LU (ILU) factorization methods is to save memory and computation by rejecting some entries in the L and U factors based on some criterion. One such criterion can be to neglect entries out of a prescribed sparsity pattern. Another criterion can be to prescribe a certain threshold value τ (called a drop tolerance) and omit all entries in the triangular factors which value is smaller than τ. Clearly, the smaller τ is, the more nonzero elements in the factors are retained and the more accurate the approximation becomes. The threshold parameter balances between the accuracy of the preconditioner and the cost to construct and apply it. Detailed derivations of ILU preconditioners can be found in [19] and [2], among others.

In this report we consider ILU preconditioners constructed using the latter ('by-value') strategy, i.e., the preconditioner is based on the size of the elements and not their position in the matrix. For matrices which have large off-diagonal elements away from the main diagonal it is expected to perform better than the DBAI-preconditioner. However, the number of non-zero elements and the structure of the factors can not be determined beforehand, and neither the computational complexity and memory demands. Furthermore, the implementation of the ILU preconditioners on a parallel computer architecture is known to be not very efficient.

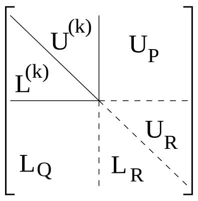
The ILU 'by-value' technique is one of the simplest methods to construct incomplete factorization preconditioners. A more advanced approach is the combination of ILU and algebraic multilevel iteration methods, the so-called Algebraic Recursive Multilevel Solver (ARMS), developed in a number of works, originating in [22]. ARMS exploits the so-called 'independent sets' or 'group-independent sets'. The latter idea targets sparse matrices and up to the knowledge of the authors is not applied for dense matrices so far.

2.2.1 Successive LU/ILU factorizations

Let a sequence of matrices $A^{(k)} \in \mathbb{R}^{N^{(k)}\times N^{(k)}}$ have to be factorized, where each matrix $A^{(k+1)}$ is obtained from $A^{(k)}$ by augmenting it with a number of rows and columns,

$$A^{(k+1)} = \begin{pmatrix} A^{(k)} & P \\ Q & R \end{pmatrix},$$

where $P \in \mathbb{R}^{N^{(k)}\times n^{(k)}}$, $Q \in \mathbb{R}^{n^{(k)}\times N^{(k)}}$, $R \in \mathbb{R}^{n^{(k)}\times n^{(k)}}$ and $N^{(k+1)} = N^{(k)} + n^{(k)}$. Let $A^{(k)}$ be factored as $L^{(k)}U^{(k)}$. Clearly, to factorize $A^{(k+1)}$ we can reuse the factors $L^{(k)}$ and $U^{(k)}$



The entries in U_P are computed as follows

$$\begin{array}{l} u^{(k+1)}_{1,j+N^{(k)}} = p_{1,j} \\ u^{(k+1)}_{i,j+N^{(k)}} = p_{i,j} - \sum_{m=1}^{i-1} l^{(k)}_{i,m} u^{(k+1)}_{m,j+N^{(k)}} \\ \text{for} \quad i = 2,\dots,N^{(k)} \quad \text{and} \quad j = 1,\dots,n_k, \end{array}$$

where $p_{i,j}$, $l^{(k+1)}_{i,j}$ and $u^{(k+1)}_{i,j}$ denotes the elements of P, $L^{(k+1)}$ and $U^{(k+1)}$, respectively. Similarly, the entries in L_Q are computed as

$$\begin{array}{l} l^{(k+1)}_{j+N^{(k)},1} = q_{j,1} \\ l^{(k+1)}_{j+N^{(k)},i} = \left(q_{j,i} - \sum_{m=1}^{i-1} l^{(k+1)}_{j+N^{(k)},m} u^{(k)}_{m,j}\right) / (u^{(k)}_{ii}) \\ \text{for} \quad i = 2,\dots,N^{(k)} \quad \text{and} \quad j = 1,\dots,n. \end{array} \tag{7}$$

The final contributions to $L^{(k+1)}$ and $U^{(k+1)}$, L_R and U_R, come from the LU-factorization of R which has first been modified as L_Q.

In this way the computational complexity to fully factorize $A^{(k+1)}$, instead of $O\left((N^{(k+1)})^3\right)$, becomes proportional to $3n^{(k)}N^{(k)} + 3n^{(k)}(N^{(k)} + n^{(k)}) + {n^{(k)}}^3$.

2.3 Full block-factorized preconditioners with approximated blocks

In some applications the system matrix A admits in a natural form a block 2-by-2 structure

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}, \tag{8}$$

which can then be factored as

$$A = \begin{bmatrix} I & 0 \\ A_{21}A_{11}^{-1} & I \end{bmatrix} \begin{bmatrix} A_{11} & A_{12} \\ 0 & S_2 \end{bmatrix} = \begin{bmatrix} A_{11} & 0 \\ A_{21} & I \end{bmatrix} \begin{bmatrix} I & A_{11}^{-1}A_{12} \\ 0 & S_2 \end{bmatrix}, \tag{9}$$

or

$$A = \begin{bmatrix} I & A_{12} \\ 0 & A_{22} \end{bmatrix} \begin{bmatrix} S_1 & 0 \\ A_{22}^{-1}A_{21} & I \end{bmatrix} = \begin{bmatrix} S_1 & A_{12} \\ 0 & A_{22} \end{bmatrix} \begin{bmatrix} I & 0 \\ A_{22}^{-1}A_{21} & I \end{bmatrix}, \tag{10}$$

where $S_1 = A_{11} - A_{12}A_{22}^{-1}A_{21}$ and $S_2 = A_{22} - A_{21}A_{11}^{-1}A_{12}$ are the so-called Schur complements of A. In this way solution of systems with A can be replaced by solutions of two systems with block triangular matrices. The latter is not recommendable to be implemented straightforwardly in practice, however, since we have to form S_1 or S_2 explicitly, which includes inversion of the corresponding diagonal block and to solve two systems with A_{ii} and one with S_i, $i = 1, 2$. Instead, utilizing the structure of A, we can construct preconditioners in the form (9) or (10), where some of the blocks are approximated in some way. For example, the block A_{22} can be approximated by a sparse matrix and the block A_{11} can be replaced by its ILU factorization. In some applications one of the diagonal blocks A_{11} or A_{22} may be well approximated by a band or even a diagonal matrix and in such cases the corresponding Schur complement can be explicitly computed on low cost. In some cases one can solve the reduced (Schur complement) system instead, which is always better conditioned than A itself.

This technique can be extra beneficial for some particular problems, the solution of which evolve, for example with time, and after each evolution step the system matrix is augmented by a number of newly computed rows and columns. In such a case the block A_{11} remains unchanged and its approximation (or factorization) can be computed only once and reused during each consecutive evolution (time) step.

3 Computational complexity

When solving a dense system $A\mathbf{x} = \mathbf{b}$ by Gaussian elimination, the number of arithmetic operations to be performed is of order $\mathcal{O}(N^3)$, where N is the size of the system.

The computational complexity of one step of the iterative solver consists of one (two in some cases) matrix-vector multiplication, vector updates, scalar products and the cost to apply the preconditioner which has been chosen. We provide below the preconditioned

GMRES(m) algorithm to illustrate its complexity, which is very similar to that of the GCG-MR method. Here GMRES(m) stands for truncated GMRES (as in [20]) where maximum m search directions are used per iteration (it). The complexity of the full GMRES is obtained for $m = it$.

```
 1  For $it = 1, 2, \cdots$
 2      Solve $\mathbf{r}$ from $M\mathbf{r} = \mathbf{b} - A\mathbf{x}_0$
 3      Normalize residual: $\beta = \|\mathbf{r}_0\|, \mathbf{v}^{(1)} = \beta\mathbf{r}$
 4      Define $(m+1) \times m$ matrix $H_m = \{h_{i,k}\}, 1 \le i \le m+1, 1 \le k \le m$. Set $H_m = 0$.
 5      For $i = 1, 2, \cdots, m$
 6          Solve $\mathbf{w}_i$ from $M\mathbf{w} = A\mathbf{v}_i$
 7              For $k = 1, \cdots, i$
 8                  $h_{k,i} = (\mathbf{w}_i, \mathbf{v}^{(k)})$
 9                  $\mathbf{w}_i = \mathbf{w}_i - h_{k,i}\mathbf{v}^{(k)}$
10              End
11              $h_{i+1,i} = \|\mathbf{w}_i\|_2$. If $h_{i+1,i} = 0$, set $m = i$ and goto 12
12              $\mathbf{v}_{i+1} = \mathbf{w}_i / h_i + 1, i$
13          End
14          Compute $y_m$ as a minimizer of $\|\beta\mathbf{e}_1 - H_m\mathbf{y}\|_2$ and $x_m = x_0 + \mathbf{V}_m\mathbf{y}_m$.
15  End
```

As is seen from the algorithm, the work per one GMRES(m) iteration involves one matrix multiplications with A, one action of the preconditioner M, $2m$ vector updates and scalar products, and the cost to solve the minimization problem in step 14,

$$W_{GMRES_{it}} = \mathcal{O}\left(N^2 + W_{prec} + 2mN\right) \tag{11}$$

The computational complexity of applying the DBAI preconditioner is proportional to the number of nonzero elements, i.e.,

$$W_{DBAI} = \mathcal{O}(kN) \tag{12}$$

and for the ILU preconditioner the costs to solve two triangular systems is also proportional to the number of nonzero elements in the L and U factors,

$$W_{ILU} = \mathcal{O}(nnz(L) + nnz(U)) \tag{13}$$

It is clear from (11), (12) and (13) that the total computational cost for the iterative solver grows as N^2 as long as $it < N$, which is less than the work required for a direct solver. It can be shown (see for instance [2]) that it will be proportional to the number of distinct eigenvalues of the matrix $M^{-1}A$. Therefore, a good preconditioner will be the one which clusters the spectrum of the preconditioned matrix around a few points as tightly as possible.

4 Numerical experiments

The target problem is crack propagation in a homogeneous, brittle material discretized using the DDM method. The system matrix A is dense and due to the singular nature of the BEM operator, is strongly diagonally dominant. Depending on the geometry of the domain and the ordering of the unknowns, A may also have some large off-diagonal elements far away from the main diagonal.

All numerical experiments are performed on a Dell Optoplex GX260 computer with 512 MB RAM memory under Windows XP Professional. The computer is equipped with a 2.66 GHz Intel Pentium 4 processor. The test problems are modeled and discretized within a commercial BEM-code which uses the DDM technique, amended with modified criteria for crack opening detection (see [4] and [25]). The direct solution experiments are performed using the direct solver available in the commercial package in FORTRAN90 compiled with a Lahey compiler on the same computer.

After generation the matrices and the corresponding right-hand-side vectors are exported to MATLAB where the iterative solution methods are run. For the iterative solution tests all matrices are first symmetrically scaled to unit diagonal. The iterations of the GCG-MR and the GMRES method are terminated when the norm of the residual has decreased with six orders of magnitude.

We note that since MATLAB is an interpreter, MATLAB codes are in general less efficient that FORTRAN codes. Still, for large enough problems, the MATLAB timing results are smaller than these for the direct solution method, which is a clear indication that iterative solution methods are the methods of choice for dense BEM matrices.

Numerical experiments on three different test problems are performed.

PROBLEM 1 (SINGLE CRACK) *The problem setting describes the propagation of a short horizontal crack in an infinite homogeneous solid.*

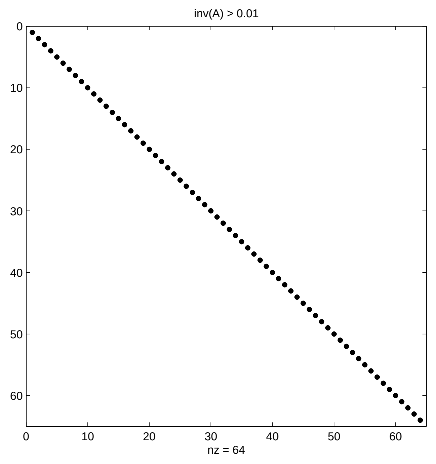
As can be observed in Figure 2, the inverse of A is strongly diagonally dominant. The elements of the first off-diagonal are several orders of magnitude smaller than the elements on the main diagonal and the size of the elements further away decay very fast. This indicates that the DBAI-preconditioner can be efficient.

PROBLEM 2 (BOREHOLE WITH FOUR CRACKS) *The problem setting describes a circular borehole in homogeneous infinite media which is subjected to uniaxial stress and in the wall of the hole four radial cracks are situated.*

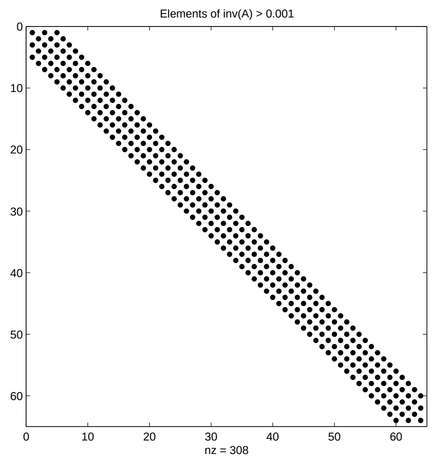
The increased geometrical complexity is reflected in A, which now contains a number of large off-diagonal elements away from the main diagonal.

PROBLEM 3 (GALLERY) *This problem models a gallery in fractured rock at a depth of 500 m. The geometry of the problem is shown in Fig. 3.*

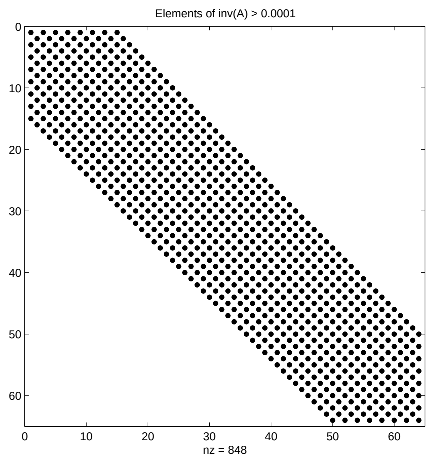
First we illustrate the performance of the FORTRAN-coded direct solver, available in the commercial BEM-package. Table 1 shows CPU-time spent to solve $A\mathbf{x} = \mathbf{b}$ for different problem sizes using Gaussian elimination. The timings show fairly good agreement with the theoretical prediction $\mathcal{O}(N^3)$, although effects due to hierarchical memory management, such as page misses, cause additional increase of the computing time for the larger problems.



(a) $A^{-1} > 0.01$



(b) $A^{-1} > 0.001$



(c) $A^{-1} > 0.0001$

Figure 2: Problem 1: Rate of decay of the elements in A^{-1}

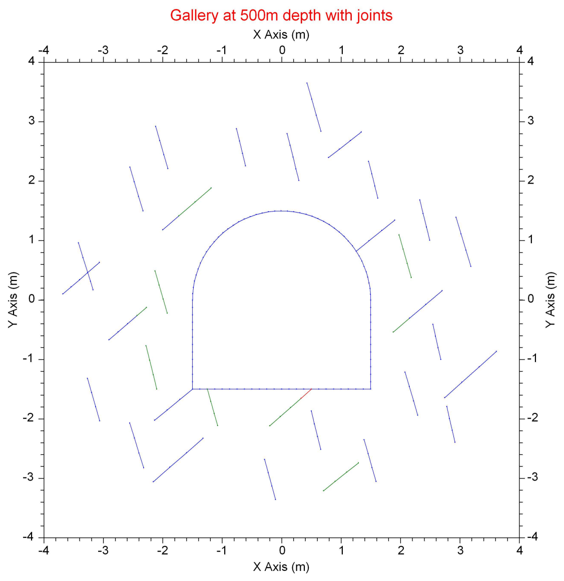


Figure 3: Problem 3: The geometry of the problem.

Problem 1					
Degrees of freedom N	128	256	512	1024	2048
CPU-time (s)	0.0156	0.0625	0.750	8.313	311.047

Problem 3			
Degrees of freedom N	384	762	1510
CPU-time (s)	0.282	2.875	72.047

Table 1: CPU-time required to solve the systems using Gaussian elimination

4.1 Problem 1: Single crack

Table 2 shows the CPU-timings and iteration counts required for both the unpreconditioned and DBAI(k) preconditioned GCG-MR solver to converge. The indication 'no' in the first column of the table indicates that no preconditioner has been applied.

Since the problem is quite simple and relatively well conditioned, even the unpreconditioned iterative method outperforms the direct solver. For larger problems the usefulness of the preconditioned method becomes more evident.

For all N the iterative solver is as fast or faster than the direct solver. Note that for $N = 2048$ and the largest bandwidth, the GCG-MR solver is more than 700 times faster. For an iterative solver coded in `FORTRAN`, for instance, this difference will be further increased in favor of the iterative approach.

4.2 Problem 2: Borehole with four cracks

Table 3 shows CPU-timings and iteration counts for the solutions of the borehole problem using unpreconditioned and DBAI(k) preconditioned GCG-MR. The number of iterations

k	$N = 128$		$N = 256$	
	CPU-time	iter	CPU-time(s)	iter
no	0.002	4	0.003	4
3	0.009	4	0.016	4
5	0.009	3	0.017	3
7	0.007	3	0.021	3
9	0.010	3	0.021	3

k	$N = 512$		$N = 1024$		$N = 2048$	
	CPU-time	iter	CPU-time(s)	iter	CPU-time	iter
no	0.017	5	0.067	6	0.335	7
3	0.040	5	0.115	6	0.415	7
5	0.042	4	0.102	4	0.354	5
7	0.045	4	0.111	4	0.368	5
9	0.047	3	0.119	4	0.391	5

Table 2: Problem 1: CPU-time when using DBAI(k)-preconditioned GCG-MR

required for convergence has increased considerably, compared to the unpreconditioned solutions in Problem 1 and the effect of the preconditioner is clearly seen.

The geometry of Problem 2 is somewhat more complicated than in Problem 1, which is reflected in the structure of A. The matrix contains large elements away from the main diagonal and the DBAI-preconditioner must contain many diagonals to catch these contributions. This is depicted in Table 3. As the bandwidth of the preconditioner increases from 3 to 39, the iteration count is decreased with a factor between 2.5 and 3.5. The time per iteration increases, however.

The cost to apply the DBAI(k) preconditioner with $k = 3, 9, 39$ turns out to be comparable with that of the ILU preconditioner with $\tau = 0.1, 0.01, 0.001$, because pairwise the nonzero elements is about the same.

Tables 4 and 5 illustrate the ILU preconditioner for Problem 2. Table 4 shows the density and bandwidth of the L-factors of the incomplete LU-factorization of A. Here the density of the matrix is defined as the fraction of the nonzero elements and the bandwidth is the average number of non-zero elements per row. The upper triangular matrices have about the same number of nonzero elements as the corresponding L factors.

Table 5 contains CPU-timings and iteration counts when the borehole problem is solved using GCG-MR with an ILU-preconditioner. The table shows that when the threshold of the preconditioner is decreased with four orders of magnitude, the iteration count decreases about ten times.

The differences in time are mainly due to the construction of the preconditioner. Execution time is influenced by the locality of the memory accesses and the known fact that indirect addressing in `MATLAB` is time consuming and will be much faster if done in `FORTRAN`. Here, the construction of the ILU-preconditioner when $N = 1920$ requires about 20 % of the total CPU-time. For the DBAI-preconditioner, the same number varies between 0.77 % and 28 % depending choice of k.

Problem 2 exhibits in a natural way a beneficial 2×2-block structure, where the A_{22}

k	$N = 120$		$N = 240$	
	CPU-time	iter	CPU-time(s)	iter
no	0.021	30	0.068	42
3	0.024	31	0.074	42
9	0.017	16	0.045	23
39	0.042	11	0.087	13

k	$N = 480$		$N = 960$		$N = 1920$	
	CPU-time	iter	CPU-time(s)	iter	CPU-time	iter
no	0.229	54	1.078	75	5.304	107
3	0.269	58	1.102	75	5.365	107
9	0.146	31	0.561	42	2.571	56
39	0.210	18	0.597	25	2.146	33

Table 3: Problem 2: CPU-time when using DBAI(k)-preconditioned GCG-MR

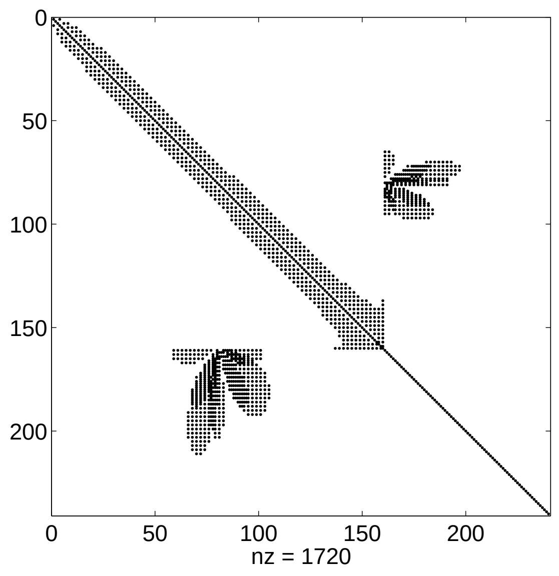


Figure 4: Problem 2:$A > 0.001$. Note the 2×2-block structure of the matrix.

N	$N = 120$		$N = 240$		$N = 480$		$N = 960$		$N = 1920$	
drop tol	Density	Bandwidth	Density	Bandwidth	Density	Bandwidth	Density	Bandwidth	Density	Bandwidth
.1	0.0169	2.0	0.0084	2.0	0.0042	2.0	0.0021	2.0	0.0010	2.0
.01	0.0474	5.7	0.0215	5.2	0.0110	5.3	0.0054	5.2	0.0026	5.1
.001	0.2085	25.0	0.0888	21.3	0.0397	19.1	0.0181	17.3	0.0085	16.3
.0001	0.4341	52.1	0.3653	87.7	0.1800	86.4	0.0734	70.5	0.0329	61.6

Table 4: Problem 2: Properties of the ILU preconditioner

N	$N = 120$		$N = 240$		$N = 480$		$N = 960$		$N = 1920$	
drop tol	CPU-time(s)	iter	CPU-time	iter	CPU-time	iter	CPU-time(s)	iter	CPU-time	iter
.1	0.028	16	0.149	22	1.054	30	5.354	41	27.210	53
.01	0.021	9	0.088	12	0.630	17	3.161	23	16.868	32
.001	0.016	5	0.075	8	0.436	10	2.004	13	9.887	17
.0001	0.017	3	0.080	4	0.425	6	1.823	8	8.599	11

Table 5: Problem 2: CPU-time when using ILU-preconditioned GCG-MR

block is very strongly diagonally dominant. This structure is illustrated in Figure 4 showing all elements of A larger than 0.001. Both forms of the full block-factorized preconditioner (formulas (9) and (10)) are tested for Problem 2. In the sequel, the preconditioners are referred to as the S_2- and S_1-preconditioner. The blocks A_{22}, respectively A_{11}, are approximated by their diagonal, or by an ILU factorization. Then the corresponding approximated Schur complement is computed explicitly and used in the numerical experiments.

Table 6 shows iteration counts and CPU-times for the the S_1-preconditioner. The approximation strategies for the A_{22} block lead to convergence rates, nearly independent of the problem size.

When the ILU threshold parameter is decreased, the iteration counts and timings decrease as well. In this case the quality of the preconditioner is less sensitive to the choice of τ compared with the ILU preconditioner constructed for A itself.

Table 7 show results from the S_2-approach and it is evident that this preconditioner does not perform as well as S_1. Both iteration counts and CPU-times are higher, how much

	$N = 120$		$N = 240$		$N = 480$		$N = 960$		$N = 1920$	
	CPU-time	iter	CPU-time(s)	iter	CPU-time	iter	CPU-time(s)	iter	CPU-time	iter
diag	0.022	8	0.084	10	0.572	14	4.266	19	36.722	26
ILU(0.1)	0.019	5	0.078	7	0.419	8	2.972	11	22.478	14
ILU(0.01)	0.016	5	0.062	5	0.356	6	2.500	7	15.816	9
ILU(0.001)	0.016	4	0.059	4	0.322	5	1.975	6	13.544	7
ILU(0.0001)	0.013	2	0.056	3	0.306	4	1.891	5	11.441	5

Table 6: Problem 2: CPU-time for the S_1-preconditioned GCG-MR

depends on the problem size and the approximation of A_{11}.

One observation from Table 7 is that the iteration count for the ILU(0.1)-approximation is half of the iteration count for the diag-approximation. This reflects the structure of the matrix A. A_{22} can be well approximated with its diagonal because its off-diagonal elements decay very fast. This does not hold for A_{11}.

The timings for the versions (9) and (10) do not differ very much, even though the iteration counts are much smaller in the first case. This is due to the fact that in our implementation the Schur-complement is solved by a direct method and also because the S_1-complement is two times larger in size than the S_2-complement.

4.3 Problem 3: Gallery problem

Problem 3 is solved with DBAI(k) and ILU(τ) preconditioners. The parameters k and τ are chosen such that both preconditioners contains approximately the same number of nonzero elements.

Table 8 shows the relative number of non-zero elements of the matrices (density) and the number of non-zeros per row (bandwidth) in the L-factor of A.

For this problem the `MATLAB` implementation of the GMRES method is used. Both GCG-MR and GMRES are similar and require about the same amount of work per iteration. Timings and iteration counts required for the ILU-preconditioned GMRES are shown in Table 9. It is clearly seen that the number of iterations required to solve the problem decrease with decreasing τ. The CPU-time, however, does not decrease with the same rate since the density of the preconditioner increases. Such a behavior is often observed for ILU preconditioners, where decreasing the drop tolerance entails work which is not always compensated by a corresponding decrease if the iteration count. Table 9 indicates that for this problem the optimal choice of τ is in the interval between 0.001 and 0.0001.

It is also seen from Table 9 that for this problem the DBAI-preconditioner performs

	$N = 120$		$N = 240$	
	CPU-time	iter	CPU-time(s)	iter
diag	0.046	30	0.156	41
ILU(0.1)	0.038	16	0.169	21
ILU(0.01)	0.031	10	0.116	13
ILU(0.001)	0.025	7	0.106	10
ILU(0.0001)	0.034	7	0.122	8

	$N = 480$		$N = 960$		$N = 1920$	
	CPU-time	iter	CPU-time(s)	iter	CPU-time	iter
diag	0.759	53	5.091	73	32.434	104
ILU(0.1)	0.825	29	4.847	40	26.572	53
ILU(0.01)	0.550	17	3.059	23	17.416	32
ILU(0.001)	0.444	12	2.275	15	11.497	18
ILU(0.0001)	0.484	9	2.500	12	12.184	15

Table 7: Problem 2: CPU-time required to solve using S_2-preconditioned GCG-MR

N	$N = 384$		$N = 762$		$N = 1510$	
drop tol	Density	Bandwidth	Density	Bandwidth	Density	Bandwidth
.1	0.0039	1.5	0.0020	1.5	0.0011	1.7
.01	0.0113	4.3	0.0053	4.0	0.0026	3.9
.001	0.0846	32.5	0.0257	19.6	0.0102	15.4
.0001	0.3300	126.7	0.2172	165.5	0.0814	123.1

Table 8: Problem 3: Properties of the ILU-preconditioner

worse that ILU, the reason being that larger elements in A^{-1} away from the diagonal are not captured in M.

Problem 3 does not exhibit an evident 2 $\times$ 2 block structure (see Figure 5(a)). However, one can first apply a symmetric permutation, which aims at bringing the largest in absolute value entries of A closer to the main diagonal. Then the desired block structure is visible and the A_{22} block can be again approximated by a diagonal matrix (see Figure 5(b)). The iteration counts and CPU-timings for the GMRES method, preconditioned by a full block-factorized preconditioner are also found in Table 9. It is evident that the S_1-preconditioner outperforms the other three preconditioning strategies significantly.

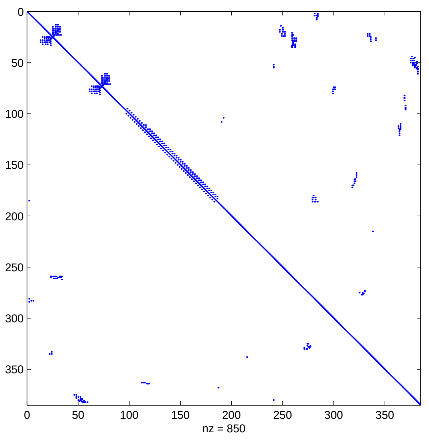
4.4 Effect of the preconditioners on the spectrum of the preconditioned matrix

We illustrate the effect of the preconditioning on the eigenvalues of the preconditioned matrix $B = M^{-1}A$. If M is well chosen, it is expected to cluster the eigenvalues of B insuring in this way fast convergence. The matrix in Figures 6, 7, 8, 9 and 10 is that from Problem 2 and is of size 240×240.

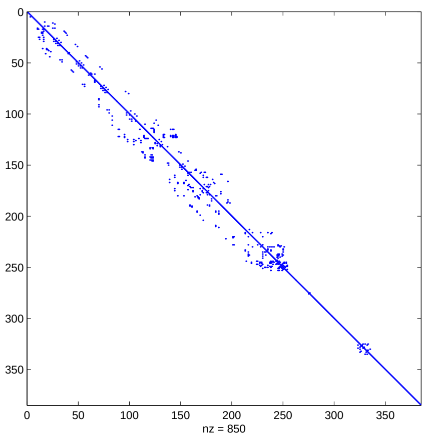
Figure 6 shows the spectrum and the GCG-MR convergence for the unpreconditioned

N	$N = 384$		$N = 762$		$N = 1510$	
ILU-preconditioning						
τ	CPU-time(s)	iter	CPU-time	iter	CPU-time	iter
.1	0.500	44	2.265	61	11.750	84
.01	0.344	25	1.360	34	6.925	47
.001	0.281	14	0.969	19	4.369	25
.0001	0.406	6	1.828	9	6.256	14
DBAI(k)-preconditioning						
k	CPU-time(s)	iter	CPU-time	iter	CPU-time	iter
3	0.453	89	1.843	123	13.022	169
7	0.297	58	1.172	81	8.319	111
49	0.359	33	0.891	36	3.719	44
279	6.047	31	17.156	34	37.341	33
S_1-preconditioning						
A_{22}	CPU-time(s)	iter	CPU-time	iter	CPU-time	iter
diag	0.328	13	1.019	9	5.906	11
ILU(0.1)	0.409	13	1.381	9	6.694	9
ILU(0.01)	0.331	9	0.200	7	5.691	7
ILU(0.001)	0.312	8	1.109	6	5.291	6
ILU(0.0001)	0.275	6	1.084	5	5.178	5
S_2-preconditioning						
A_{11}	CPU-time(s)	iter	CPU-time	iter	CPU-time	iter
diag	0.731	89	3.606	122	18.812	167
ILU(0.1)	1.041	46	5.634	60	36.038	83
ILU(0.01)	0.744	29	3.416	34	21.316	47
ILU(0.001)	0.562	19	2.444	21	13.912	28
ILU(0.0001)	0.587	14	3.538	14	13.613	19

Table 9: Problem 3: CPU-time and iteration counts for the preconditioned GMRES

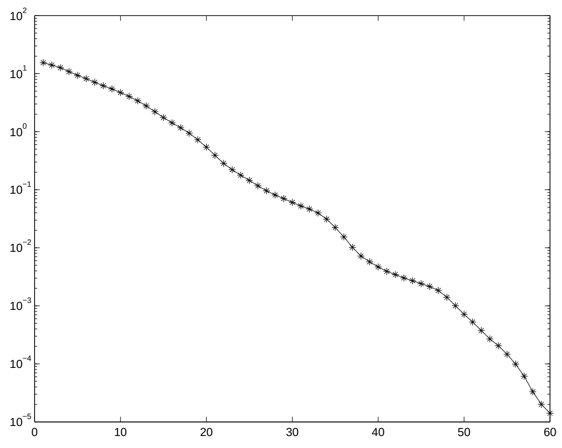


(a) Original structure $A > 0.01$

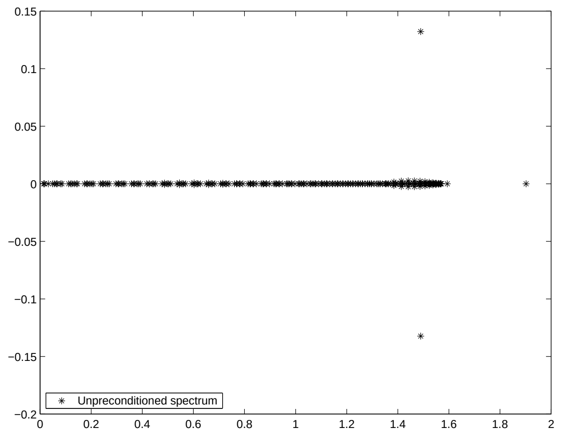


(b) $A > 0.01$ after permutation

Figure 5: Problem 3



(a) Convergence of the unpreconditioned GCG-MR



(b) Spectrum of A

Figure 6: Problem 2

matrix. All the eigenvalues of A except three are spread along the real line between zero and two. The spectrum is not clustered and the convergence of the iterative solver is not very fast.

Figure 7 shows how the spectrum of B changes for the different preconditioners. All preconditioners contain approximately the same number of nonzeros. The ILU and DBAI preconditioning strategies affects the spectrum of the iteration matrix very differently. The ILU preconditioner clusters the spectrum around unity as the accuracy of the preconditioner increases. The DBAI preconditioner, on the other hand clusters most of the eigenvalues around one as the number of nonzeros increase, but the extreme eigenvalues are not much affected by the preconditioner. Hence, the condition number of the iteration matrix in this case is almost the same as the condition number of A itself.

The convergence of the GCG-MR method for the DBAI and ILU preconditioner with different accuracy is shown in Figure 8.

The spectra for the S_1 and S_2 preconditioners with $diag$ and ILU approximations of the A_{11}/A_{22}-block are shown in Figures 9 and 10. The incomplete factors are denoted by $\tilde{L}$ and $\tilde{U}$, respectively. The threshold parameters are chosen as the ones used for ILU preconditioning of A. The achieved spectra are well clustered, which is confirmed by the corresponding fast convergence of the iterative method.

Convergence rates for the different full block-factorized preconditioners are shown in Figure 11.

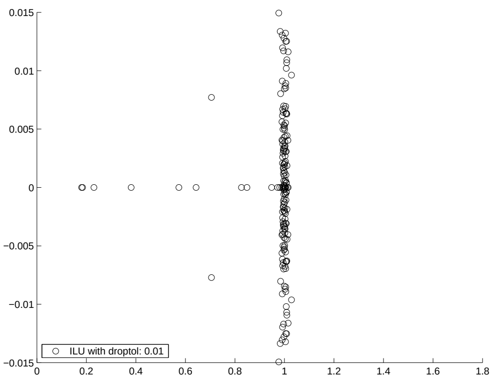
5 Some techniques for a compressed storage of the BEM matrix

It is clear from (11), that the most significant term in the expression for the computational complexity of the iterative solver is the N^2-term coming from the matrix-vector multiplication. In order to further reduce the cost of solving the system of equations, one has to make this operation more efficient. This has to be done already at the stage of discretization the problem and generating the system matrix.

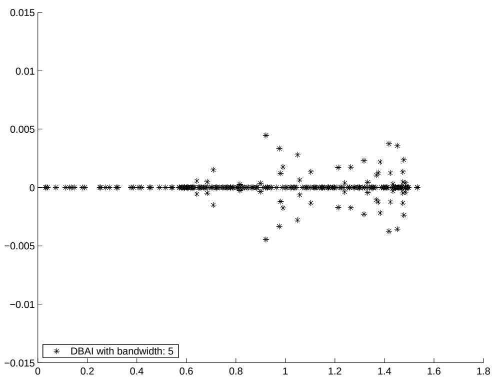
One approach could be to apply the Fast Multipole Method (FMM), developed by Greengard and Rokhlin [18]. FMM is an algorithm originally developed for efficient calculation of the potential in N-body problems, but it has shown to be an efficient means for the computation of dense matrix-vector multiplications as well.

Another efficient way to compress the matrix A is to sparsify it by using the so-called Mosaic-Skeleton approximation technique, developed in [23]. The dense matrix is sparsified by omitting the least significant elements and bounds of the error of the action of the approximation are shown. The matrix is decomposed in rank-one terms, which makes the multiplication with a vector much cheaper than $O(N^2)$.

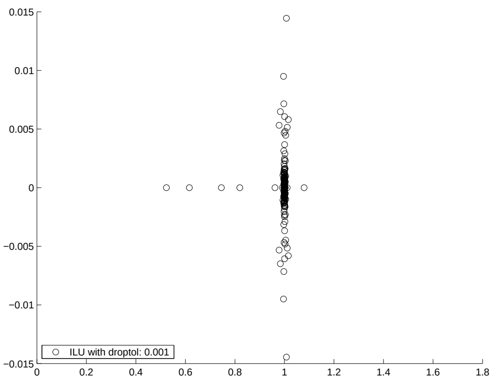
Recently (see [10], [11] and the references therein) a class of hierarchical data-sparse matrices ($\mathcal{H}$-matrices) has been developed which allow to approximate nonlocal operators such as boundary/volume integrals, as well as the solution operators of elliptic, parabolic and hyperbolic partial differential problems. This approach turns out to be very useful in constructing sparse approximate inverse preconditioners, for example. Furthermore, in $\mathcal{H}$-matrix arithmetics, the essential matrix operations (matrix-matrix and matrix-vector multi-



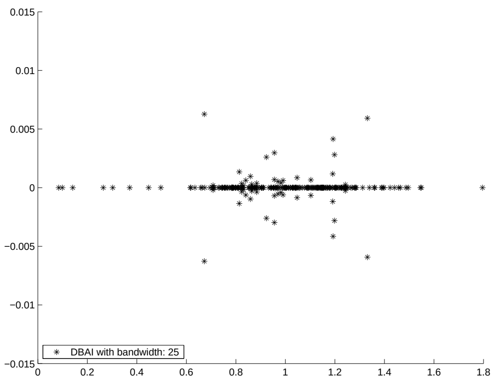
(a) $ILU(0.01)$



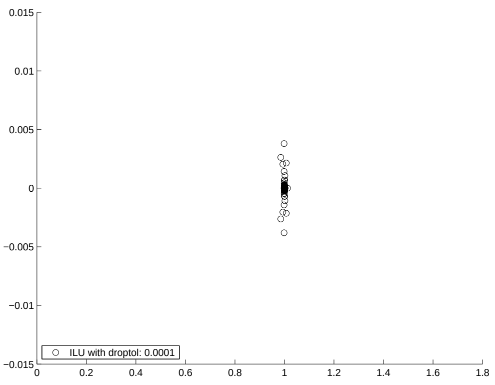
(b) $DBAI(5)$



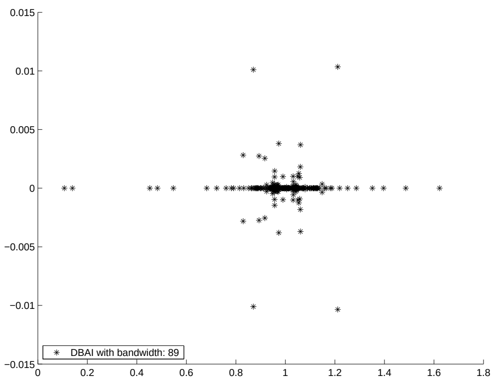
(c) $ILU(0.001)$



(d) $DBAI(25)$

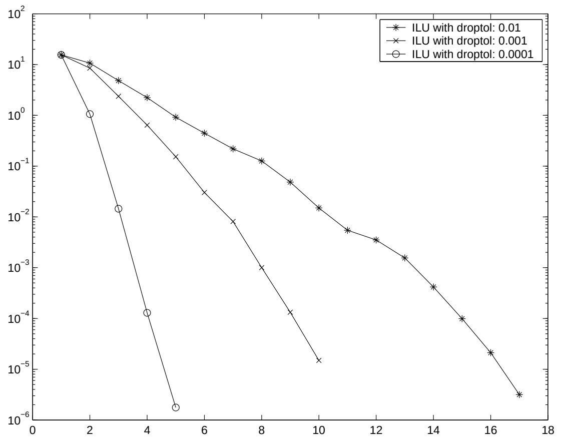


(e) $ILU(0.0001)$

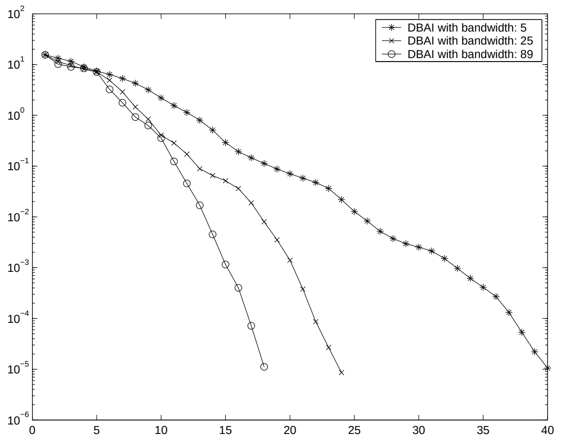


(f) $DBAI(89)$

Figure 7: Problem 2: The spectrum of the iteration matrix $B = M^{-1}A$



(a) ILU preconditioner



(b) DBAI preconditioner

Figure 8: Convergence history for GCG-MR

plication, addition and inversion) can be executed in nearly optimal (linear) complexity, up to a logarithmic factor. These significant achievements are possible only if the problem formulation, choice of suitable discretization method, data structures and solution method are considered together, aiming at decreasing the demands on computer resources needed for the numerical simulations.

6 Conclusions

Preconditioned iterative solution techniques are an efficient and competitive alternative to solve dense linear systems of equations arising from BEM discretizations of crack propagation problems. The arithmetic cost to achieve the solution from these solvers are proportional to itN^2 and for $it \ll N$, the savings in computations are substantial compared to direct solution methods.

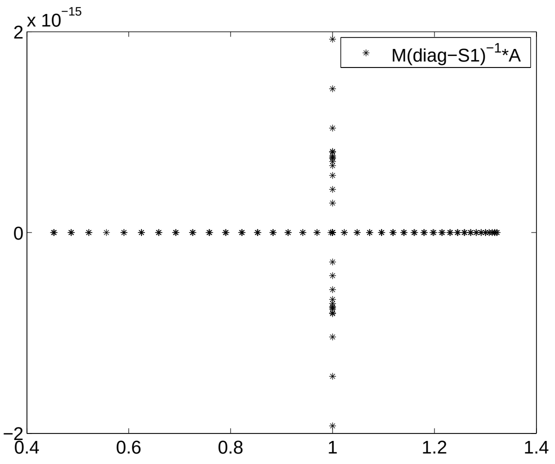
For simple problems, where the inverse of the system matrix has a band structure with fast decaying entries away from the main diagonal, the DBAI(k) preconditioner performs quite satisfactory and is computationally cheap.

The ILU-preconditioner has shown to be numerically efficient for all test problems. Compared to ILU, the approximate inverse DBAI(k) preconditioner requires in general more iterations to converge.

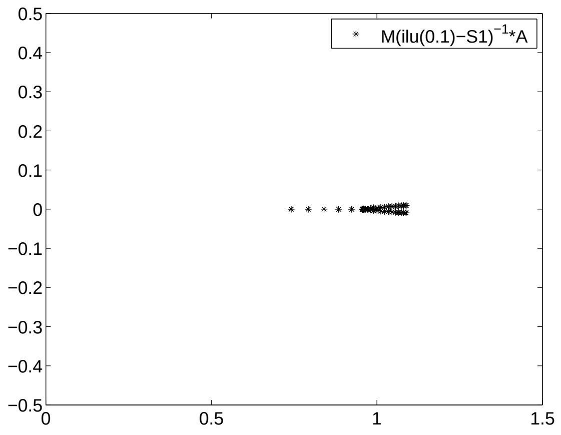
The full block-factorized preconditioner turns out to be the numerically most efficient, whenever the system matric can be put in a proper 2-by-2 block form.

For large enough problems the preconditioned iterative solvers are found faster than the direct solver even though the iterative methods are implemented in `MATLAB` (interpreting language) while the direct solution method is implemented in `FORTRAN` (compiled routine).

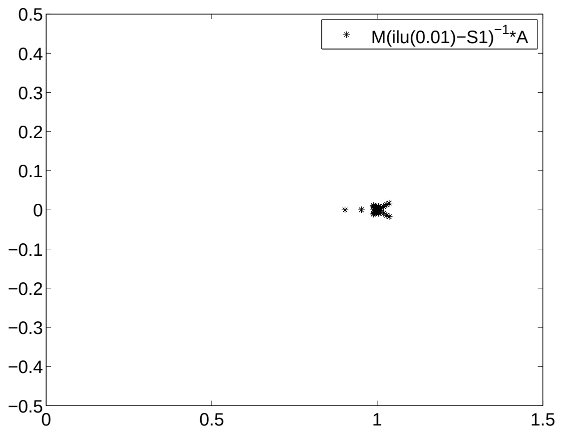
Further speedup of the solution process is achievable if the system matrix is either sparsified or handled in a hierarchical format, reducing in this way the matrix-vector multiplication



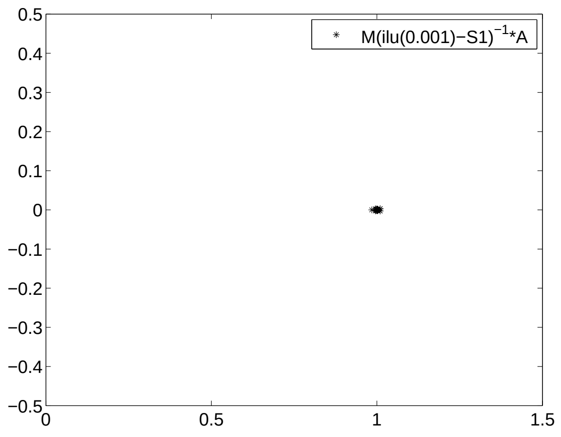
(a) $A_{22} \approx diag(A_{22})$



(b) $A_{22} \approx \tilde{L}\tilde{U}$

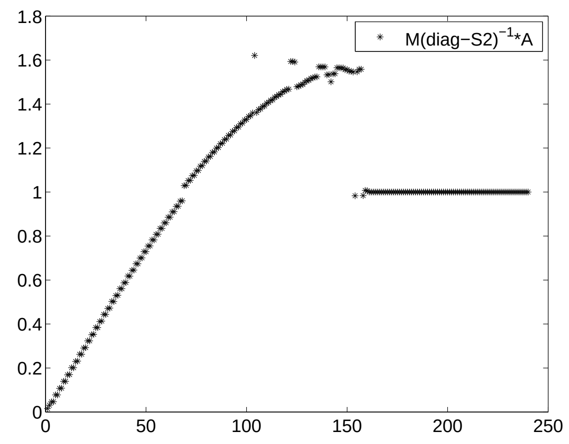


(c) $A_{22} \approx \tilde{L}\tilde{U}$

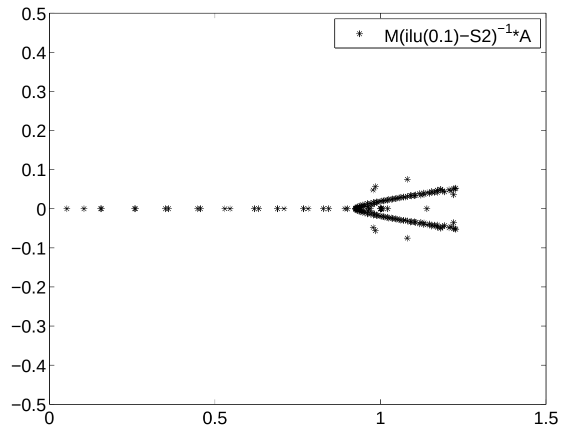


(d) $A_{22} \approx \tilde{L}\tilde{U}$

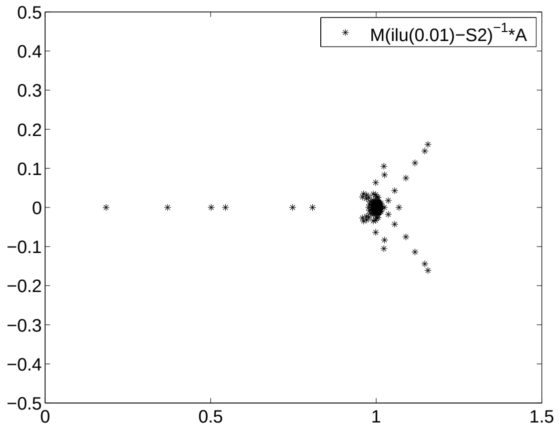
Figure 9: Problem 2: $N = 240$. The spectrum of $M^{-1}A$ for various S_1-preconditioners



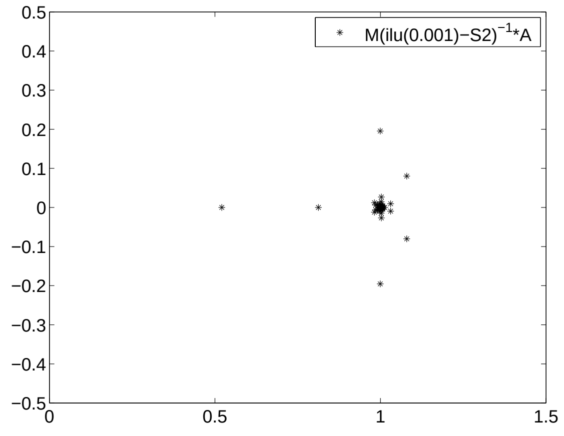
(a) $A_{11} \approx diag(A_{11})$



(b) $A_{11} \approx \tilde{L}\tilde{U}$

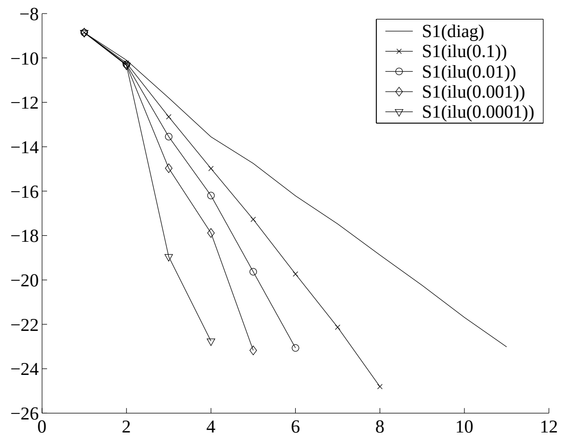


(c) $A_{11} \approx \tilde{L}\tilde{U}$

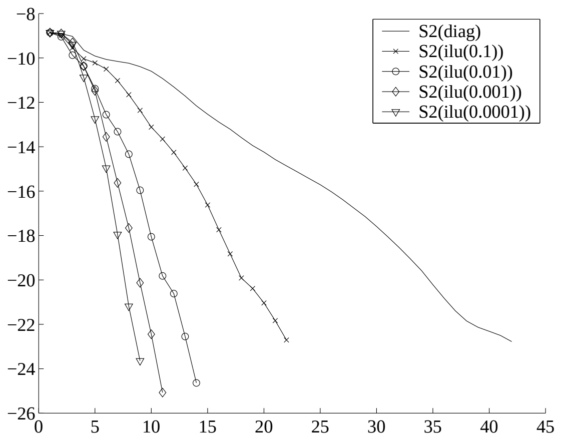


(d) $A_{11} \approx \tilde{L}\tilde{U}$

Figure 10: Problem 2: $N = 240$. The spectrum of $M^{-1}A$ for various S_2-preconditioners



(a) S_1



(b) S_2

Figure 11: Problem 2: $N = 240$. Convergence for S_1- and S_2-preconditioned GMRES. The scale on the y-axis is logarithmic.

cost from $\mathcal{O}(N^2)$ to almost linear complexity $\mathcal{O}(N \log N)$.

Acknowledgments

The authors express their thanks to Prof. Owe Stephanson, Dr. Lanru Jing and Dr. Hee-Suk Lee, Engineering Geology and Geophysics , Royal Institute of Technology, Sweden, for initiating and supporting this project.

The authors are much indebted to Tech. Lic. Mikael Rinne at Fracom ltd, (www.fracom.fi) for providing the BEM-discretization package Fracod for DDM crack analysis. His continuous support and interest in this work is thankfully acknowledged. The authors also thank Dr. Baotang Shen, Fracom and CSIRO Exploration and Mining, for answering the numerous technical questions regarding Fracod.

Thanks go to Dr. Jingyu Shi, Fracom and School of Mathematical Sciences, Queensland University of Technology, for his valuable comments on this report.

References

[1] Aliabadi MH. *The boundary Element Method*, Volume II: Applications in solids and structures. John Wiley& Sons Ltd, 2002.

[2] Owe Axelsson. *Iterative Solution Methods*. Cambridge University Press, 1996.

[3] Axelsson O, Brinkkemper S, Il'in V. On some versions of incomplete block-matrix factorization iterative methods. *Linear Algebra and its Applications*, 1984, 58:3-15.

[4] Baotang Shen. Mechanics of fractures and intervening bridges in hard rocks. Doctoral thesis, Engineering Geology, Royal institute of Technology, Stockholm, Sweden, 1993.

[5] L. P. S. Barra, W. J. Coutinho, W. J. Mansur, and J. C. F. Telles. Iterative solution of BEM equations by GMRES algorithm. *Computers and structures*, 44(6):1249–1253, 1992.

[6] B. Carpentieri, I.S. Duff, L. Giraud, M. Magolu monga Made. Sparse symmetric preconditioners for dense linear systems in electromagnetism. CERFACS Technical Report TR/PA/01/35. Accepted for publication in *Numerical Linear Algebra with Application*.

[7] Ke Chen. An analysis of sparse approximate inverse preconditioners for boundary integral equations. *SIAM J. Matrix Anal. Appl.*, 22(4):1058–1078, 2001.

[8] S. L. Crouch. Solution of plane elasticity problems by the displacement discontinuity method. *International journal for numerical methods in engineering*, 10:301–343, 1976.

[9] Concus P, Golub G, Meurant G. Block preconditioning for the conjugate gradient method. *SIAM Journal on Statistical and Scientific Computing* 1985, 6:220-252.

[10] Hackbusch W. A sparse matrix arithmetic based on $\mathcal{H}$-matrices. Part I. Introduction to $\mathcal{H}$-matrices. *Computing*, 1999; 62:89-108.

[11] Hackbusch W, Khoromskij B. A sparce $\mathcal{H}$-matrix arithmetic: General complexity estimates. *journal of Computational and Applied Mathematics*, 2000; 125:479-501.

[12] Kolotilina L. On approximate inverses of block H-matrices. *Numerical Analysis and Mathematical modelling*, Moscow, 1989 (in Russian).

[13] Kolotilina L, Yeremin A. Factorized sparse approximate inverse preconditionings. *SIAM Journal on Matrix Analysis and Applications* 14 (1993), 45–58.

[14] Langer U, Pusch D, Reitzinger S. Efficient preconditioners for boundary element matrices based on grey-box algebraic multigrid methods, *International Journal for Numerical Methods in Engineering*, 2003; **57**:1937–1953.

[15] C. Y. Leung and Walker S. P. Iterative solution of large three-dimensional BEM elastostatic analyses using the GMRES technique. *International Journal for numerical methods in engineering*, 40:2227–2236, 1997.

[16] L. Martin, Dinh Nho Hao, D. Lesnic. Conjugate gradient - boundary element method for the Cauchy problem in elasticity, *Q. JI Mech. Appl. Math.*, 2002; 55:227–247.

[17] M. Merkel, V. Bulgakov, R. Bialecki, and G. Kuhn. Iterative solution of large-scale 3d-BEM industrial problems. *Engineering Analysis with Boundary Elements*, 22:183–197, 1998.

[18] V. Rokhlin and L. Greengard. A fast algorithm for particle simulations. *Journal of Computational Physics*, 73:325–348, 1987.

[19] Youcef Saad. Ilut: A dual threshold incomplete LU factorization. *Numerical linear algebra with applications*, 1(4):387–402, 1994.

[20] Youcef Saad. *Iterative solution methods for sparse linear systems*, 2000. `http://www-users.cs.umn.edu/ saad/books.html`

[21] Youcef Saad and Martin H. Schulz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal of Scientific and Statistical Computing*, 1986; 7:856–869.

[22] Yousef Saad, Brian Suchomel. ARMS: an algebraic recursive multilevel solver for general sparse linear systems, *Numerical Linear Algebra with Applications*, 9 (2002), 359–378.

[23] Eugene Tyrtyshnikov. Mosaic-skeleton approximations. *Calcolo*, 33(1–2):47 – 57, 1996.

[24] Wells GN, Sluys LJ. A new method for modelling cohesive cracks using finite elements, *International Journal for Numerical Methods in Engineering* 2001; 50:2667-2682.

[25] FRACOD^{2D} Two Dimesional Fracture Propagation Code, version 1.1, User's manual, `http://www.fracom.fi`