Hierarchical LU decomposition based preconditioners for BEM

by

Mario Bebendorf

Preprint no.: 28 2004

Hierarchical *LU* decomposition based preconditioners for BEM

Mario Bebendorf
Universität Leipzig
Augustusplatz 10-11
D-04109 Leipzig, Germany
bebendorf@math.uni-leipzig.de

Abstract

The adaptive cross approximation method can be used to efficiently approximate stiffness matrices arising from boundary element applications by hierarchical matrices. In this article an approximative LU decomposition in the same format is presented which can be used for preconditioning the resulting coefficient matrices efficiently. If the LU decomposition is computed with high precision, it may even be used as a direct yet efficient solver.

AMS Subject Classification: 65F05, 65F30, 65F50
Key words: Data-sparse approximations, hierarchical matrices, fast solvers, boundary elements.

1 Introduction

We consider Fredholm integral equations $\lambda u + \mathcal{K}u = f$, $\lambda \in \mathbb{R}$, for the unknown function u, where the integral operator $\mathcal{K}$ is defined by

$$(\mathcal{K}u)(x) = \int_\Gamma \kappa(x,y)u(y)\,\mathrm{d}s_y, \quad x \in \mathbb{R}^3, \tag{1.1}$$

with a kernel function $\kappa : \mathbb{R}^3 \times \Gamma \to \mathbb{R}$ on a manifold $\Gamma \subset \mathbb{R}^3$. Note that the results of this article can easily be extended to volume integrals. It can be shown (cf. [8]) that this kind of integral equation arises for example from any direct boundary integral method applied to

$$\begin{aligned} Lu &= 0 \quad \text{in} \quad \Omega \\ R\gamma u &= g \quad \text{on} \quad \Gamma := \partial\Omega, \end{aligned}$$

where L is an elliptic partial differential operator of order $2m$ and R is an $m \times (2m)$ matrix of tangential derivatives applied to the Cauchy vector $\gamma u = [u, \partial_n u, \dots, \partial_n^{2m-1} u]$. As a consequence a differential equation on a domain Ω is reformulated as an integral equation on its boundary Γ. The advantage of applying the finite element method to this boundary integral equation, i.e., the advantage of the so-called *boundary element method* (BEM) over the usual finite element method, is that a discretisation of the volume Ω is avoided. Hence, the number of unknowns is reduced due to the reduction of the dimensionality. However, if $\mathcal{K}$ is discretised using a finite dimensional ansatz space $V_h := \operatorname{span}\{\varphi_i : i \in I\}$, where $I := \{1, \dots, n\}$, the arising stiffness matrix $K \in \mathbb{R}^{n \times n}$ with entries

$$K_{ij} := \int_\Gamma \int_\Gamma \varphi_i(x)\kappa(x,y)\varphi_j(y)\,\mathrm{d}s_x\,\mathrm{d}s_y, \quad i,j = 1,\dots,n, \tag{1.2}$$

is expected to be dense, since the support of the kernel κ is in general non-local. Since the introduction of data-sparse representations this disadvantage of the boundary element method has been overcome.

An important property that all data-sparse representations are based on is the *pseudo-locality property* of elliptic operators. Since $\mathcal{K}$ is an elliptic pseudo-differential operator, its kernel function κ fulfils the *Calderón-Zygmund* property, which states that κ is smooth everywhere on $\Gamma \times \Gamma$ apart from the diagonal $\{(x,x) : x \in \Gamma\}$. The Calderón-Zygmund property is sometimes called *asymptotical smoothness*, i.e., there are constants $c_1^{\text{as}}, c_2^{\text{as}} > 0$ and a real number $s \geq 0$ such that for all multi-indices $\alpha \in \mathbb{N}_0^3$ it holds that

$$|\partial_y^\alpha \kappa(x,y)| \leq c_1^{\text{as}} p!\, (c_2^{\text{as}}|x-y|)^{-s-p}, \quad p = |\alpha|, \tag{1.3}$$

where as usual we denote by ∂_y^α the partial derivative

$$\partial_y^\alpha = \left(\frac{\partial}{\partial y_1}\right)^{\alpha_1} \left(\frac{\partial}{\partial y_2}\right)^{\alpha_2} \left(\frac{\partial}{\partial y_3}\right)^{\alpha_3}.$$

Note that strongly singular kernels are not excluded. However, then the integral in (1.1) has to be defined by an appropriate regularisation.

If the boundary integral method is applied for example to Laplace's equation, κ is either the kernel of the *single layer*, the *double layer* or the *hypersingular* operator

$$\kappa_V(x,y) := \frac{1}{4\pi}\frac{1}{|x-y|}, \quad \kappa_K(x,y) := \frac{\partial}{\partial n_y}\kappa_V(x,y) \quad \text{and} \quad \kappa_D(x,y) := \frac{\partial}{\partial n_x}\frac{\partial}{\partial n_y}\kappa_V(x,y).$$

Here, n_x denotes the outer normal unit vector to the surface Γ at $x \in \Gamma$. It is important to remark that neither the smoothness of κ with respect to x nor smoothness properties of the surface Γ are necessary for the methods treated in this article.

The algorithmic complexity of computing and storing the dense matrix K from (1.2) is quadratic in the number of degrees of freedom. Therefore, different approaches have been introduced to avoid dense matrices: for rotational invariant geometries, the matrix K is likely to have Toeplitz structure, which can be exploited (cf. [24]) by algorithms based on the fast Fourier transformation. If the underlying geometry can be described by a small number of smooth maps, wavelet techniques can be used for compressing the resulting matrices, see [11]. Another class of methods exploits the fact that the kernel function κ can locally be approximated by degenerate functions, i.e.,

$$\kappa(x,y) \approx \sum_{i=1}^{k} u_i(x) v_i(y), \tag{1.4}$$

where k is a small number. This idea originating from the fast multipole method [14] and panel clustering [19] today is looked at more from an algebraic point of view, since on the discrete level (1.4) means that appropriate blocks of the matrix K can be approximated by matrices of low rank. This gave rise to the mosaic-skeleton method [27] and hierarchical matrices ($\mathcal{H}$-matrices) [16, 17, 12]. By the latter it is not only possible to (approximatively) store dense matrices and multiply them by a vector with almost linear complexity, they also provide the usual (approximative) operations like matrix addition, multiplication and inversion with almost the same complexity. The basis for the efficiency of this class of matrices is a hierarchical partition of the matrix into blocks and the low-rank representation of each block. Instead of generating these low-rank approximants from degenerate kernel approximations, e.g, by the multipole expansion or interpolation, it is more efficient and convenient to use the adaptive cross approximation (ACA) algorithm [1, 4], which finds such low-rank approximants from few of the original matrix entries.

After building an $\mathcal{H}$-matrix approximant $\tilde{K}$ of K the discrete system

$$(\lambda M + \tilde{K})x = b \tag{1.5}$$

has to be solved for a given right-hand side b, where $M \in \mathbb{R}^{n\times n}$ is the mass matrix with entries

$$M_{ij} = \int_\Gamma \int_\Gamma \varphi_i(x)\varphi_j(x)\, \mathrm{d}s_x, \quad i,j = 1,\dots,n.$$

M is sparse and can therefore be easily treated. A direct method for the solution of (1.5) like Gaussian elimination would require a computational complexity cubic in the size n of the matrix K. Iterative solvers that are based on matrix-vector multiplication, e.g., CG or GMRES, can exploit the efficiency of the $\mathcal{H}$-matrix-vector multiplication. However, the number of iterative steps can be prohibitively large, since it depends on the condition number of $\lambda M + \tilde{K}$. In order to improve the convergence rate a preconditioner is used. A *left* preconditioner is a regular and easily invertible matrix $C \approx \lambda M + \tilde{K}$ in the sense that the condition number of $C^{-1}(\lambda M + \tilde{K})$ is bounded by a constant, where instead of (1.5) one solves the linear system

$$C^{-1}(\lambda M + \tilde{K})x = C^{-1}b.$$

If C is used as a *right* preconditioner (1.5) is replaced by

$$(\lambda M + \tilde{K})C^{-1}\tilde{x} = b.$$

In the latter case the solution x can be computed from $Cx = \tilde{x}$. Hence, in addition to the matrix-vector multiplications needed during the iteration, in each step a linear system with coefficient matrix C has to be solved. In this article only right preconditioners are considered. However, a left preconditioner can be defined analogously.

There are many different approaches to obtain a preconditioner C. In a recently published idea [22] an algebraic multigrid procedure is constructed for boundary element matrices. Another possibility is based on the mapping properties of the operator only, see [26]. Let $\mathcal{A} : V \to V'$ be a V-coercive and $\mathcal{B} : V' \to V$ a V'-coercive operator. Then $\mathcal{A}$ and $\mathcal{B}^{-1} : V \to V'$ both are V-coercive, i.e., there are constants $\alpha_i, \beta_i > 0$, $i = 1, 2$, such that for all $v \in V$

$$\alpha_1\|v\|_V^2 \le (\mathcal{A}v, v)_{L^2} \le \alpha_2\|v\|_V^2 \quad \text{and} \quad \beta_1\|v\|_V^2 \le (\mathcal{B}^{-1}v, v)_{L^2} \le \beta_2\|v\|_V^2.$$

From this it already follows that $\mathcal{A}$ and $\mathcal{B}^{-1}$ are spectrally equivalent, i.e,

$$\frac{\alpha_1}{\beta_2}(\mathcal{B}^{-1}v, v) \le (\mathcal{A}v, v) \le \frac{\alpha_2}{\beta_1}(\mathcal{B}^{-1}v, v) \quad \text{for all } v \in V.$$

Hence, for preconditioning the single layer potential operator of the Laplacian for instance the hypersingular operator can be used. Since both operators can be approximated using ACA, one could easily obtain an efficient preconditioner. Although the idea of using operators with inverse mapping properties is attractive, the preconditioning effect can only be observed asymptotically, i.e., for fixed problem sizes n the condition number might still be large.

The approach we are going to pursue in this article is to use an approximative LU decomposition

$$C := LU \approx \lambda M + \tilde{K},$$

where the two triangular factors L and U are stored in $\mathcal{H}$-matrix format. Depending on the prescribed accuracy of the LU decomposition the condition number can be controlled also for fixed problem sizes. A constant accuracy will be sufficient to guarantee spectral equivalence. The arising equations with coefficient matrix C can then be solved by forward/backward substitution with the same complexity as the $\mathcal{H}$-matrix-vector multiplication. The setup of the factors L and U can be done with complexity $\mathcal{O}(n \log^2 n)$. Since constants are much smaller compared with the complexity of building $\tilde{K}$, the time for decomposing $\tilde{K}$ can be neglected. This is especially true, if the LU decomposition is computed with low precision.

The rest of this article is organised as follows. Section 2 gives a brief review of the structure of $\mathcal{H}$-matrices. In Section 3 we explain in detail how K is approximated by $\mathcal{H}$-matrices using ACA. We present a new strategy to choose the first cross in the ACA algorithm. This strategy improves the accuracy of the first steps of ACA. Based on the approximative $\mathcal{H}$-arithmetic in Section 4 an algorithm for computing an LU decomposition in $\mathcal{H}$-matrix format is introduced. Based on this decomposition we present a *black-box* preconditioner for boundary element stiffness matrices. If the LU decomposition is computed with high precision it may also be used as an efficient direct solver. Furthermore, we discuss how the hierarchical forward/backward substitution should be implemented. The last Section 5 is devoted to numerical tests for typical academic as well as realistic problems. It will be seen that the proposed preconditioner has almost linear complexity and leads to a bounded number of iterations. Furthermore, we will apply our preconditioner to an extremely ill-conditioned boundary element stiffness matrix of an asymptotically well-conditioned operator.

2 Hierarchical matrices

This section gives a brief overview over the structure of $\mathcal{H}$-matrices originally introduced by Hackbusch et al. [16, 17]. We will describe the two principles on which the efficiency of $\mathcal{H}$-matrices is based. These are the hierarchical partitioning of the matrix into blocks and the blockwise restriction to low-rank matrices. These principles were also used in the mosaic-skeleton method [27].

In contrast to other efficient methods like wavelet techniques [7, 9, 10], fast multipole and panel clustering, see [14], [19] and the references therein, $\mathcal{H}$-matrices concentrate on the matrix-level. They are purely algebraic in the sense that once the $\mathcal{H}$-matrix approximant is built, no further information about the underlying problem is needed.

$\mathcal{H}$-matrices mainly aim at non-local formulations of elliptic boundary value problems. The latter usually leads to integral operators with kernel functions $\kappa(x, y)$ that share the same kind of algebraic singularity $|x - y|^{-s}$, $s > 0$. Hence, approximations of type (1.4) cannot be expected unless x and y are well separated from each other. This gives a condition on the pairs of domains on which κ can be approximated.

2.1 Admissibility condition

Let $b = t \times s \subset I \times J$, $I := \{1, \ldots, n_1\}$, $J := \{1, \ldots, n_2\}$, be the indices of a subblock of $A \in \mathbb{R}^{I \times J}$ with entries

$$a_{ij} = a(\psi_j, \varphi_i), \quad i \in t, \, j \in s, \tag{2.1}$$

where φ_i and ψ_j are ansatz and trial functions with supports $X_i := \operatorname{supp} \varphi_i$ and $Y_j := \operatorname{supp} \psi_j$, and a is a bilinear form which also depends linearly on the kernel function κ. We assume that the supports X_i and Y_j are local. Galerkin stiffness matrices are of the form (2.1). If formally delta distributions are used as ansatz and trial functions also collocation and Nyström matrices are enclosed.

The existence of low-rank approximants on subblocks of A is equivalent to the existence of degenerate approximations of κ on a corresponding pair of subdomains, since for each subblock

$b = t \times s$ the kernel κ is evaluated only on the Cartesian product $X_t \times Y_s$ of the supports of the basis functions φ_i, $i \in t$, and ψ_j, $j \in s$, where

$$X_t := \bigcup_{i \in t} X_i, \quad Y_s := \bigcup_{j \in s} Y_j.$$

In the field of elliptic partial differential equations the following condition on $b = t \times s$ has proved useful:

$$\min\{\operatorname{diam}(X_t), \operatorname{diam}(Y_s)\} \leq \eta \operatorname{dist}(X_t, Y_s), \tag{2.2}$$

where $\eta > 0$ is a given real number. This so-called *admissibility condition* will be the criterion for deciding whether b belongs to the partition we are going to construct.

If (2.2) holds, the Green function of any elliptic operator even with non-smooth coefficients can be approximated degenerately on $X_t \times Y_s$, see [5, 3] for a proof.

Remark 2.1 *In the case of unstructured grids the computation of the distance in* (2.2) *between two supports X_t and Y_s is too costly. Therefore, for practical purposes the supports are enclosed into sets of a simpler structure, e.g. boxes or spheres.*

In order to exploit the fact that there is a partition P such that on each contained block a given matrix can be approximated by a matrix of low rank, we first have to find P from the set of possible subsets of $I \times J$. This set however is too large to be searched for a partition that will satisfy our needs. Therefore, the set of subsets $b = t \times s$ is restricted to those which consist of index sets t and s, the so-called *clusters*, stemming from cluster trees T_I and T_J, respectively. We will see that with this restriction still competitive partitions can be computed with small effort.

2.2 The cluster tree

A tree T_I satisfying the following conditions is called a *cluster tree* for I:

(*i*) I is the root of T_I

(*ii*) if $t \in T_I$ is not a leaf, then t has sons $t_1, t_2 \in T_I$, so that $t = t_1 \cup t_2$.

The set of sons of $t \in T_I$ is denoted by $S(t)$, while $\mathcal{L}(T_I)$ stands for the set of leaves of the tree T_I.

A cluster tree is usually generated by recursive subdivision of I. For practical purposes the recursion should be stopped if a certain cardinality $n_{\min}$ of the clusters is reached, rather than subdividing the clusters until only one index is left. The depth of T_I will be denoted by L. For reasonable cluster trees one would always expect $L = \mathcal{O}(\log |I|)$. A strategy for subdividing I recursively leading to a cluster tree with clusters of minimal extension is based on the *principle component analysis*, see [1]. In the same article the complexity of building the cluster tree in the case of quasi-uniform grids is estimated as $\mathcal{O}(|I| \log |I|)$.

Remark 2.2 *Sometimes the number of sons in the previous definition of a cluster tree is not restricted to two. However, this generalization has not proved useful in practice.*

2.3 The block-cluster tree

Based on cluster trees T_I and T_J which contain hierarchies of partitions of I and J, respectively, we are able to construct the so called *block-cluster tree* $T_{I \times J}$ describing a hierarchy of partitions of $I \times J$ by the following rule:

```
procedure build_block-cluster_tree($t \times s$)
begin
  if $t \times s$ does not satisfy (2.2) and $t \notin \mathcal{L}(T_I)$ and $s \notin \mathcal{L}(T_J)$ then begin
    $S(t \times s) := \{t' \times s' : t' \in S(t),\ s' \in S(s)\}$
    for $t' \times s' \in S(t \times s)$ do build_block-cluster_tree($t' \times s'$)
  end
  else $S(t \times s) := \emptyset$
end
```

Applying *build_block-cluster_tree* to $I \times J$ we obtain a cluster tree for the index set $I \times J$. The set of leaves $P := \mathcal{L}(T_{I\times J})$ is a partition of $I \times J$ with blocks $b = t \times s \in P$ either satisfying (2.2) or consisting of clusters t and s one of which is a leaf in T_I or T_J, respectively, i.e., $\#t \leq n_{\min}$ or $\#s \leq n_{\min}$. The complexity of building the block-cluster tree in the case of quasi-uniform grids can be estimated as $\mathcal{O}(\eta^{-2} n \log n)$, where $n = \max\{|I|, |J|\}$, cf. [1].

For a given partition P we are now in a position to define the set of $\mathcal{H}$-matrices with blockwise rank k

$$\mathcal{H}(P, k) := \{M \in \mathbb{R}^{I\times J} : \operatorname{rank} M_b \leq k \text{ for all } b \in P\},$$

where by M_b we denote the restriction of M to block b. Note that $\mathcal{H}(P, k)$ is not a linear space, since in general the sum of two rank-k matrices exceeds rank k.

Remark 2.3 *For a block $B \in \mathbb{R}^{t\times s}$ the low-rank representation $B = UV^T$, $U \in \mathbb{R}^{t\times k}$, $V \in \mathbb{R}^{s\times k}$, is only advantageous compared with the entrywise representation, if $k(|s|+|t|) < |s|\,|t|$. For the sake of simplicity in this article we will however assume that each block has the low-rank representation. Employing the entrywise representation for appropriate blocks will accelerate the algorithms.*

2.4 Storage and arithmetical operations

The cost of multiplying an $\mathcal{H}$-matrix $M \in \mathcal{H}(P, k)$ or its transposed M^T by a vector x is inherited from the blockwise matrix-vector multiplication:

$$Mx = \sum_{t\times s\in P} M_{t\times s} x_s \quad \text{and} \quad M^T x = \sum_{t\times s\in P} (M_{t\times s})^T x_t.$$

Since each block $t \times s$ has the representation $M_{t\times s} = UV^T$, $U \in \mathbb{R}^{t\times k}$, $V \in \mathbb{R}^{s\times k}$ (see Remark 2.3), $\mathcal{O}(k(|t| + |s|))$ units of memory are needed to store $M_{t\times s}$ and the matrix-vector products

$$M_{t\times s} x_s = UV^T x_s \quad \text{and} \quad (M_{t\times s})^T x_t = VU^T x_t$$

can be done with $\mathcal{O}(k(|t| + |s|))$ operations. Exploiting the hierarchical structure of M it can therefore be shown that both storing M and multiplying M and M^T by a vector has complexity $\mathcal{O}(\eta^{-2} kn \log n)$, where $n = \max\{|I|, |J|\}$. For a rigorous analysis the reader is referred to [1]. Therefore, $\mathcal{H}$-matrices are well suited for iterative schemes such as Krylov subspace methods.

In addition to the fast matrix-vector multiplication $\mathcal{H}$-matrices also provide approximative versions of the usual matrix operations such as addition and multiplication with almost linear complexity, see [12]. These operations can only be carried out approximatively since $\mathcal{H}(T_{I\times J}, k)$ is not a linear space. Hence, the result of the arithmetical operations $A + B$ and $A \cdot B$ have to be projected into the set $\mathcal{H}(T_{I\times J}, k')$, where $k' \leq 2k$ is the smallest index such that the projection lies inside an ε-neighbourhood of the exact result. Under assumptions that are valid for elliptic problems the hierarchical addition has complexity $\mathcal{O}(k^2 n \log n)$. For multiplying two $\mathcal{H}$-matrices $\mathcal{O}(k^2 n \log^2 n)$ operations are needed. These fast $\mathcal{H}$-operations will be used extensively to compute hierarchical LU decompositions.

3 Discretisation using ACA

We have seen in the previous section that $\mathcal{H}$-matrices can be handled efficiently. In this section we concentrate on a single block $b = t \times s \in P$ satisfying (2.2) in the rows t and columns s of a matrix A of the form (1.2). We assume that the kernel function κ is asymptotically smooth (see (1.3)). Under these assumptions it is possible to devise an algorithm which generates a low-rank approximant S of the subblock A_b. For all blocks $b \in P$ that do not satisfy (2.2) the entries of A_b are stored without approximation.

In the following we present the ACA (adaptive cross approximation) algorithm [1, 4]. The idea of this algorithm is as follows. Starting from an initial matrix $R_0 := A_{t\times s} \in \mathbb{R}^{t\times s}$ in the k-th step find a nonzero pivot in R_k, say (i_k, j_k), and subtract a scaled outer product of the i_k-th row and the j_k-th column:

$$R_{k+1} = R_k - [(R_k)_{i_k j_k}]^{-1}(R_k)_{t,j_k}(R_k)_{i_k,s}. \tag{3.1}$$

The pivot (i_k, j_k) is chosen to be the maximum element in modulus of the i_k-th row, i.e.,

$$|(R_{k-1})_{i_k j_k}| = \max_{j=1,\dots,n} |(R_{k-1})_{i_k j}|. \tag{3.2}$$

Note that in this formulation of ACA only the choice of j_k is specified through (3.2), since only this condition is necessary for the convergence proof of ACA. Hence, ACA allows many implementations differing by the choice of i_k. In the following Algorithm 3.1 the index i_k, $k > 1$, is chosen to be the maximum entry in modulus of the j_{k-1}-th column of R_{k-2}.

What remains is the choice of the row i_1 to start from. Due to the assumption (1.3) each kernel function $\kappa(x, y)$ is almost constant with respect to y on $X_t \times Y_s$. Hence, if the expression

$$\max_{y\in X_t} |\kappa(x, y) - \kappa(x, z)| = \max_{y\in X_t} |\int_y^z \partial_y \kappa(x, \xi)\, \mathrm{d}\xi| \le c \max_{y\in X_t} |y - z|$$

appearing in the remainder after one step of ACA is to be minimized with respect to z, one should choose z to be the Chebyshev center of X_t. Since the Chebyshev center of a set is quite complicated to compute, we use the center of mass m_t of X_t instead. Hence, from these arguments it seems promising to choose i_1 so that the center z_{i_1} of X_{i_1} is closest to m_t. In Figure 1 we compare this new strategy with the old one in which i_1 was chosen so that z_{i_1} is closest to the center of Y_s. In addition Figure 1 shows the accuracy of approximations with find rank obtained from the multipole expansion

$$\frac{1}{|x|}\sum_{k=0}^{\infty}\sum_{\ell=-k}^{k} Y_k^{\ell}(\hat{x})Y_k^{-\ell}(\hat{y})\frac{1}{2k+1}\left(\frac{|y|}{|x|}\right)^k, \quad \text{where } \hat{x} = \frac{x}{|x|},\ \hat{y} = \frac{y}{|y|} \text{ with } |x| > |y|,$$

and from the singular value decomposition applied to an admissible matrix block. The first k singular triplets give provably the best accuracy among all rank-k approximants. The influence of then new strategy on the quality of the approximation can be realized in the first four steps. In this part the new version gives almost optimal approximation. For all other steps the old and the new version behave almost the same.

Since in the k-th step only the entries in the j_k-th column and the i_k-th row of R_k are used to compute R_{k+1}, there is no need to calculate the whole matrix R_k. Taking advantage of this, the following Algorithm 3.1 is an efficient reformulation of (3.1). Note that the vectors u_k and $\tilde{v}_k$ coincide with $(R_{k-1})_{t,j_k}$ and $(R_{k-1})_{i_k,s}$, respectively.

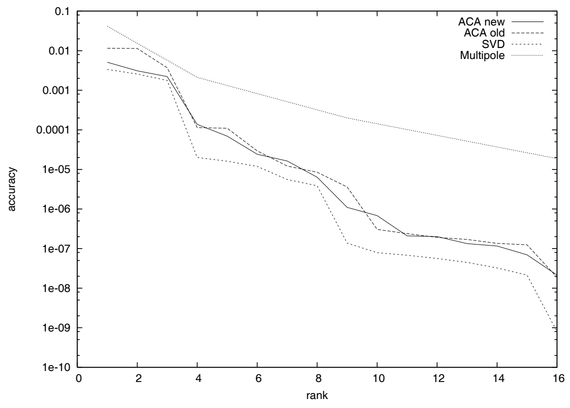

Figure 1: Accuracy of different approximants

```
Let $k = 1$;  $Z = \varnothing$ { $\textit{in } Z \textit{ the vanishing rows of } R_k \textit{ are collected}$ }
repeat
    if $k > 1$ then $i_k := \text{argmax}_{i \notin Z} |(u_{k-1})_i|$
    else choose $i_1$ so that $z_{i_1}$ is closest to the center of $X_t$
    $\tilde{v}_k := a_{i_k,s}$
    for $\ell = 1, \ldots, k-1$ do $\tilde{v}_k := \tilde{v}_k - (u_\ell)_{i_k} v_\ell$
    $Z := Z \cup \{i_k\}$
    if $\tilde{v}_k$ does not vanish then
        $j_k := \text{argmax}_{j=1,\ldots,n} |(\tilde{v}_k)_j|$;   $v_k := (\tilde{v}_k)_{j_k}^{-1} \tilde{v}_k$
        $u_k := a_{t,j_k}$
        for $\ell = 1, \ldots, k-1$ do $u_k := u_k - (v_\ell)_{j_k} u_\ell$.
        $k := k + 1$
    endif
until the stopping criterion (3.3) is fulfilled or $Z = t$
```

Algorithm 3.1: Adaptive Cross Approximation

The matrix $S_k := \sum_{\ell=1}^{k} u_\ell v_\ell^T$ is used as an approximant to A_b. Obviously, the rank of S_k is bounded by k. Let $\varepsilon > 0$ be given. The following condition on k

$$\|u_{k+1}\|_2 \, \|v_{k+1}\|_2 \leq \frac{\varepsilon(1-\eta)}{1+\varepsilon} \, \|S_k\|_F \tag{3.3}$$

can be used as a stopping criterion, see [6]. Here, $\|S_k\|_F$ denotes the Frobenius norm of S_k, which can be computed with $\mathcal{O}(k(|t|+|s|))$ operations. Therefore, the amount of numerical work required by Algorithm 3.1 is of the order $k^2(|t|+|s|)$. Hence, the cost of building an $\mathcal{H}$-matrix approximant with accuracy $\varepsilon > 0$ using ACA is of the order $n|\log \varepsilon|^4 \log n$, cf. [4].

A major advantage of this algorithm is that it adapts itself to the actual matrix properties, whereas the rank for instance in the fast multipole method is chosen beforehand and is hence often higher than necessary, compare Figure 1. The most important advantage of ACA, however, is a practical one. Since it is based on the matrix entries, ACA can be built on top of existing computer codes without changes. For applications see for instance [21], [22]. In [6] ACA is used to solve boundary integral equations in parallel.

Let $p \in \mathbb{N}$ and $n_p = \binom{d+p}{d}$ be the dimension of the space of polynomials in d variables of degree at most p. Then the following theorem gives a bound on the remainder R_k of the approximation. For a proof see [4].

Theorem 3.1 *Let $t \subset I$, $s \subset J$ and let X_t, Y_s fulfil condition* (2.2). *Furthermore, let κ be asymptotically smooth. In the case of collocation matrices*

$$a_{ij} = \int_\Gamma \kappa(x, y_i)\varphi_j(x) \,\mathrm{d}s_x, \quad i \in t,\ j \in s$$

it holds that

$$|(R_{n_p})_{ij}| \le c \operatorname{dist}^g(X_t, Y_s)\, \eta^p, \quad 0 < \eta < \frac{1}{4\sqrt{d}}. \tag{3.4}$$

Analogous results can be obtained for the Galerkin and the Nyström method, see [2, 1]. Note that the condition $\eta < (4\sqrt{d})^{-1}$ in (3.4) does not have to be satisfied in practice. Typically, η should be chosen 1.

The last theorem gives a bound on entries which have never been inspected. This seems impossible at first glance. The smoothness assumption (1.3) on the kernel function κ however provides a relation between the computed and neglected rows and columns in each block.

3.1 Recompressing $\mathcal{H}$-matrices

Although the quality of the approximant resulting from ACA is better than the quality of an approximant generated from kernel approximation, it can still be improved. Note that the major singular triplets will give the best approximation, see also Figure 1. However, for the computation of the singular value decomposition (SVD) the whole matrix block has to be computed beforehand. Since from ACA we already have an approximant UV^T to the entries A_b, we can now generate the SVD of the approximant instead of A_b by computing the QR decompositions $U = Q_U R_U$ and $V = Q_V R_V$ of $U \in \mathbb{R}^{t\times k}$ and $V \in \mathbb{R}^{s\times k}$, respectively. The SVD of A_b can then be reduced to the smaller SVD $Q_1 \Sigma Q_2^T$ of $R_U R_V^T \in \mathbb{R}^{k\times k}$, since

$$UV^T = Q_U R_U R_V^T Q_V^T = Q_U Q_1 \Sigma (Q_V Q_2)^T \tag{3.5}$$

is a singular value decomposition of UV^T.

In addition to recompressing each block one can also improve the block structure of the $\mathcal{H}$-matrix. Non-admissible blocks might be regarded as admissible, because the admissibility condition (2.2) is sufficient but not necessary. This was first observed in [18] under the term *weak admissibility.*

Rather than replacing the admissibility condition by another (improved) sufficient condition, one can reduce the storage and computational effort by inspecting the matrix directly, cf. [13]: If all four sons of a block b are leaves in the block cluster tree $T_{I\times J}$, one can try to unify them. The singular value decomposition of the union can be computed efficiently using a similar idea as (3.5). From the singular values one can obtain the rank required for a prescribed accuracy of the approximant. If the cost of the approximant of the union is less than the sum of the costs of the subblocks, the four blocks in the $\mathcal{H}$-matrix are replaced by one. This idea is then applied to the new leaves in the block cluster tree. Obviously, the accuracy of the approximant is kept during this procedure.

The combination of the previous two recompression techniques is of particular importance if the accuracy of the approximant is to be reduced. This happens for instance when computing a preconditioner, which does not require high precision.

Note that to apply the previous idea to an $\mathcal{H}$-matrix approximant, it is not necessary that it has been computed beforehand. If memory consumption is an issue, the computation of the $\mathcal{H}$-matrix approximant and the recompression procedure should be folded. It is obvious how this can be done.

4 Preconditioning in the $\mathcal{H}$-matrix format

The condition number of a Galerkin stiffness matrix arising from a pseudo-differential operator

$$\mathcal{K} : H^{\alpha}(\Gamma) \to H^{-\alpha}(\Gamma)$$

can be shown to be of the order $h^{-2|\alpha|}$, where $h := \min_{i=1,\dots,n} \operatorname{diam}(X_i)$ is the discretisation parameter. Hence, for $\alpha \neq 0$ the coefficient matrices are ill-conditioned if the number of unknowns n is large. In addition, a large condition number can result from the discretisation of the geometry even for small n, see Section 5. Therefore, if a linear system $Ax = b$ is to be solved iteratively, one has to incorporate a preconditioner C, since the convergence rate usually depends significantly on the condition number of the coefficient matrix.

In this article we propose to use $C = LU$, i.e.,

$$C^{-1} = (LU)^{-1} = U^{-1}L^{-1},$$

where L and U are lower and upper triangular $\mathcal{H}$-matrices such that $A \approx LU$ is an approximative LU decomposition. If A is symmetric positive definite, $C = LL^T$ is used as a preconditioner, where L is now the lower triangular $\mathcal{H}$-matrix from the approximative Cholesky decomposition $A \approx LL^T$. Hence, during an iterative scheme like GMRES in addition to multiplications of A and A^T by a vector, forward/backward substitutions have to be applied when multiplying with C^{-1}.

Note that in order to compute a spectrally equivalent preconditioner it is not necessary to compute the LU decomposition with high precision. Assume that we have computed a matrix C such that

$$\|I_n - AC^{-1}\|_2 \leq \delta < 1. \tag{4.1}$$

Since $\|AC^{-1}\|_2 \leq 1 + \|I_n - AC^{-1}\|_2 \leq 1 + \delta$ and since

$$\|CA^{-1}\|_2 \leq \sum_{j=0}^{\infty} \|I_n - AC^{-1}\|_2 \leq \frac{1}{1-\delta}$$

due to the Neumann series, we are led to

$$\operatorname{cond}_2(AC^{-1}) \leq \frac{1+\delta}{1-\delta}. \tag{4.2}$$

Hence, in order to obtain a spectrally equivalent preconditioner, the accuracy δ in (4.1) can be chosen independently of n, say $\delta = 0.1$.

In the following subsection it will be explained in detail how the factors L and U can be computed with almost linear complexity as well as how the forward/backward substitutions should be implemented. Hence, the hierarchical LU decomposition provides a spectrally equivalent preconditioner with almost linear complexity.

4.1 Hierarchical LU decomposition

Using the $\mathcal{H}$-matrix arithmetic it is possible to generate an LU decomposition of an $\mathcal{H}$-matrix $A \in \mathcal{H}(P)$ with any prescribed accuracy δ, i.e., a lower triangular matrix $L \in \mathcal{H}(P)$ and an upper triangular matrix $U \in \mathcal{H}(P)$ are to be found such that

$$\|I_n - A(LU)^{-1}\|_2 < \delta.$$

As we have seen in the previous section for a spectrally equivalent preconditioner it is sufficient to use $\delta \sim 1$. Hence, compared with building A the LU decomposition can be computed with lower precision. For higher efficiency A should therefore be copied and treated by the recompression procedure (see Section 3.1) with accuracy δ. The extra memory needed to hold the factors L and U is then typically much smaller than the memory needed for the stiffness matrix A. If however the $\mathcal{H}$-LU decomposition is to be used as a direct solver, see Section 4.2, then for both its computation and its decomposition the same accuracy $\delta = \varepsilon$ has to be used.

In order to define the $\mathcal{H}$-LU decomposition we exploit the hierarchical block structure of a block $(t,t) \in T_{I\times I} \setminus \mathcal{L}(T_{I\times I})$:

$$A_{tt} = \begin{bmatrix} A_{t_1t_1} & A_{t_1t_2} \\ A_{t_2t_1} & A_{t_2t_2} \end{bmatrix} = \begin{bmatrix} L_{t_1t_1} & \\ L_{t_2t_1} & L_{t_2t_2} \end{bmatrix} \begin{bmatrix} U_{t_1t_1} & U_{t_1t_2} \\ & U_{t_2t_2} \end{bmatrix},$$

where $t_1, t_2 \in T_I$ denote the sons of t in T_I. Hence, the LU decomposition of a block A_{tt} is reduced to the following four problems on the leaves of (t,t):

(i) Compute $L_{t_1t_1}$ and $U_{t_1t_1}$ from the LU decomposition $L_{t_1t_1}U_{t_1t_1} = A_{t_1t_1}$.

(ii) Compute $U_{t_1t_2}$ from $L_{t_1t_1}U_{t_1t_2} = A_{t_1t_2}$.

(iii) Compute $L_{t_2t_1}$ from $L_{t_2t_1}U_{t_1t_1} = A_{t_2t_1}$.

(iv) Compute $L_{t_2t_2}$ and $U_{t_2t_2}$ from the LU decomposition $L_{t_2t_2}U_{t_2t_2} = A_{t_2t_2} - L_{t_2t_1}U_{t_1t_2}$.

If a block $(t,t) \in \mathcal{L}(T_{I\times I})$ is a leaf, the usual pivoted LU decomposition is employed. For (i) and (iv) two LU decompositions of half the size have to be computed. In oder to solve (ii), i.e., solve $L_{tt}B_{ts} = A_{ts}$ for B_{ts}, where L_{tt} is a lower triangular matrix, we use a recursive block forward substitution: If the block $t \times s$ is not a leaf in $T_{I\times I}$, from the decompositions of the blocks A_{ts}, B_{ts} and L_{tt} into their subblocks

$$\begin{bmatrix} L_{t_1t_1} & \\ L_{t_2t_1} & L_{t_2t_2} \end{bmatrix} \begin{bmatrix} B_{t_1s_1} & B_{t_1s_2} \\ B_{t_2s_1} & B_{t_2s_2} \end{bmatrix} = \begin{bmatrix} A_{t_1s_1} & A_{t_1s_2} \\ A_{t_2s_1} & A_{t_2s_2} \end{bmatrix}$$

one observes that B_{ts} can be found from the following equations

$$\begin{aligned} L_{t_1t_1}B_{t_1s_1} &= A_{t_1s_1} \\ L_{t_1t_1}B_{t_1s_2} &= A_{t_1s_2} \\ L_{t_2t_2}B_{t_2s_1} &= A_{t_2s_1} - L_{t_2t_1}B_{t_1s_1} \\ L_{t_2t_2}B_{t_2s_2} &= A_{t_2s_2} - L_{t_2t_1}B_{t_1s_2}, \end{aligned}$$

which are again of type (ii). If on the other hand $t \times s$ is a leaf, the usual forward substitution is applied. Similarly, one can solve (iii) by a recursive block backward substitution.

In order to obtain an LU decomposition with almost linear complexity we take advantage of the $\mathcal{H}$-matrix arithmetic with truncation accuracy δ instead of the exact matrix operations. Hence, when computing the LU decomposition the accuracy of storing and of each arithmetical operation is δ. A result [20] on the stability analysis of the LU decomposition states that the product LU is backward stable, i.e.,

$$\|A - LU\|_2 < c\rho\delta\|A\|_2,$$

where

$$\rho := \frac{\max_{ij} |u_{ij}|}{\max_{i,j} |a_{ij}|},$$

is the so-called *growth factor*, which is bounded in practice. Hence, the accuracy of LU will be of order δ. Since the $\mathcal{H}$-LU decomposition is mainly determined by the $\mathcal{H}$-matrix multiplication, its complexity can be estimated to be of order $n|\log\delta|^4\log^2 n$. Thus, for the choice $\delta \sim 1$ we obtain a preconditioner with complexity $\mathcal{O}(n\log^2 n)$, while building the approximant using ACA has complexity $\mathcal{O}(n|\log\varepsilon|^4\log n)$. The LU decomposition of $\mathcal{H}$-matrices of a format that is too restrictive for our needs has already been used in [23].

In the case of positive definite matrices A we are also able to define an $\mathcal{H}$-version of the Cholesky decomposition of a block A_{tt}, $t \in T_{I\times I} \setminus \mathcal{L}(T_{I\times I})$:

$$A_{tt} = \begin{bmatrix} A_{t_1t_1} & A_{t_1t_2} \\ A_{t_1t_2}^T & A_{t_2t_2} \end{bmatrix} = \begin{bmatrix} L_{t_1t_1} & \\ L_{t_2t_1} & L_{t_2t_2} \end{bmatrix} \begin{bmatrix} L_{t_1t_1} & \\ L_{t_2t_1} & L_{t_2t_2} \end{bmatrix}^T .$$

This factorisation is recursively computed by

$$\begin{aligned} L_{t_1t_1}L_{t_1t_1}^T &= A_{t_1t_1} \\ L_{t_1t_1}L_{t_2t_1}^T &= A_{t_1t_2} \\ L_{t_2t_2}L_{t_2t_2}^T &= A_{t_2t_2} - L_{t_2t_1}L_{t_2t_1}^T \end{aligned}$$

using the usual Cholesky decomposition on the leaves. The equation $L_{t_1t_1}L_{t_2t_1}^T = A_{t_1t_2}$ is solved for $L_{t_2t_1}$ in a similar way as $U_{t_1t_2}$ has previously been obtained in the LU decomposition.

After the LU decomposition has been computed the solution of $Ax = b$ can be found by forward/backward substitution: $Ly = b$ and $Ux = y$. Since L and U are $\mathcal{H}$-matrices y_t, $t \in T_I \setminus \mathcal{L}(T_I)$, can be recursively computed by solving the following systems for y_{t_1} and y_{t_2}

$$L_{t_1t_1}y_{t_1} = b_{t_1} \quad \text{and} \quad L_{t_2t_2}y_{t_2} = b_{t_2} - L_{t_2t_1}y_{t_1}.$$

If $t \in \mathcal{L}(T_I)$ is a leaf, a usual triangular solver is used. The backward substitution can be done analogously. The complexity of this forward/backward substitution is determined by the complexity of the hierarchical matrix-vector multiplication, which is $\mathcal{O}(n\log n)$ for $\delta \sim 1$.

The computed $\mathcal{H}$-LU decomposition may be looked at as an incomplete LU decomposition. Instead of the sparsity pattern in the case of the $\mathcal{H}$-LU preconditioner the decomposition of the original matrix into subblocks is kept.

4.2 Direct solution

Since both generating the LU decomposition and the forward/backward substitution are efficient $\mathcal{H}$-operations, one can equally use the $\mathcal{H}$-LU decomposition as a direct solver. In this case the precision δ which the LU decomposition is generated with has to be of the same order as the accuracy ε of the approximant A. In this case no extra memory is needed, since the stiffness matrix A can be overwritten by the two factors L and U. The solution x is then given by $x := U^{-1}L^{-1}b$ without any iterative process. The disadvantage of this approach is that the time for setting up the two factors L and U has complexity $\mathcal{O}(n|\log\varepsilon|^4\log^2 n)$. Although the complexity estimate shows a logarithm more than the complexity estimate for building the approximant, it will be seen in the numerical experiments that the actual computational times are much lower. Once the two factors are computed, the system can be solved with $\mathcal{O}(n|\log\varepsilon|^4\log n)$ operations, such that if $Ax = b$ is to be solved for many right-hand sides the overall complexity of this approach might be less than an iterative solution using the LU preconditioner.

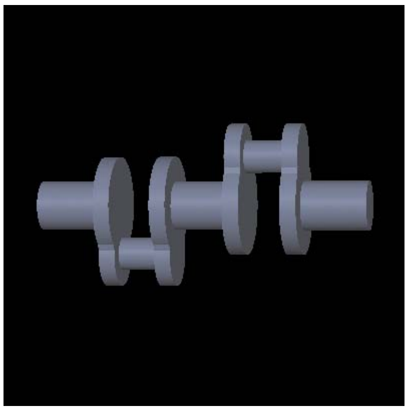
Figure 2: first geometry

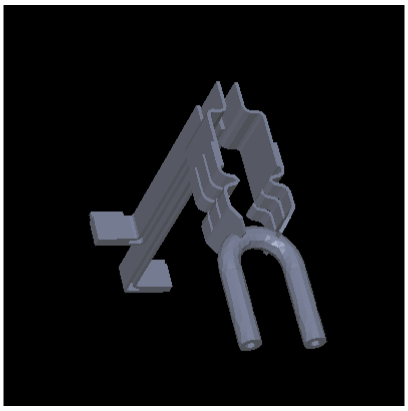
Figure 3: second geometry

5 Numerical examples

In this section we present numerical results for the algorithms proposed in this article. In the first test we apply the preconditioner to the inner Dirichlet problem of the Laplacian in $\mathbb{R}^3$. For the introduced *LU* decomposition we compare iterative solution with direct solution. The second test stems from real applications. Here, the Neumann problem is investigated. Although the double layer potential operator is asymptotically well-conditioned, the coefficient matrix is ill-conditioned even for a small problem size. All tests were computed on a dual Intel Xeon 2.666 GHz system[1].

5.1 Application: Inner Dirichlet Problem

In this section we are going to test Algorithm 3.1 on the boundary integral formulation of the inner Dirichlet problem

$$\Delta u = 0 \quad \text{in } \Omega \tag{5.1a}$$
$$u = g \quad \text{on } \Gamma := \partial\Omega \tag{5.1b}$$

for the Laplacian in three dimensions. The boundary $\Gamma \subset \mathbb{R}^3$, which was generated by NETGEN [25], is shown in Figure 2.

By Greens' formula and jump relations we obtain the following integral formulation of (5.1)

$$\frac{1}{2}u + \mathcal{K}u = \mathcal{V}(\partial_n u), \tag{5.2}$$

where the single layer potential operator $\mathcal{V} : H^{-1/2}(\Gamma) \to H^{1/2}(\Gamma)$ and the double layer potential operator $\mathcal{K} : H^{1/2}(\Gamma) \to H^{1/2}(\Gamma)$ are defined by

$$(\mathcal{V}u)(x) = \int_\Gamma s(x,y)u(y)\,\mathrm{d}s_y \quad \text{and} \quad (\mathcal{K}u)(x) = \int_\Gamma \partial_{n_y} s(x,y)u(y)\,\mathrm{d}s_y,$$

respectively. The kernel function s appearing under the last integrals is the fundamental solution $s(x,y) := (4\pi|x-y|)^{-1}$ of the Laplacian in three dimensions. The single layer potential operator is self adjoint with respect to $(\cdot,\cdot)_{L^2(\Gamma)}$ and is coercive, i.e.,

$$(\mathcal{V}u,u)_{L^2} \geq c\|u\|_{H^{-1/2}} \quad \text{for all } u \in H^{-1/2}(\Gamma).$$

[1]The $\mathcal{H}$-matrix library used is available under `http://www.mathematik.uni-leipzig.de/~bebendorf/AHMED.html`

Since the Dirichlet data $u = g$ is given on the boundary Γ, from (5.2) we obtain the following equation for the unknown $v := \partial_n u$:

$$\mathcal{V}v = (\frac{1}{2}\mathcal{I} + \mathcal{K})g.$$

A Galerkin discretisation with piecewise constants φ_i, $i = 1, \ldots, n$, and piecewise linears ψ_i, $i = 1, \ldots, n_\ell$, leads to the following linear system of equations

$$Ax = b, \quad b = (\frac{1}{2}M + B)\tilde{g}$$

where for $i = 1, \ldots, n$

$$a_{ij} = (\mathcal{V}\varphi_j, \varphi_i), \quad j = 1, \ldots, n \quad \text{and} \quad b_{ij} = (\mathcal{K}\psi_j, \varphi_i), \quad m_{ij} = (\psi_j, \varphi_i), \quad j = 1, \ldots, n_\ell$$

and $\tilde{g} \in \mathbb{R}^{n_\ell}$ is the vector minimizing

$$\|g - \sum_{i=1}^{n_\ell} \tilde{g}_i \psi_i\|_{L^2(\Gamma)}.$$

5.1.1 Building the $\mathcal{H}$-matrix approximants

In the first set of numerical tests we approximate the matrices A and B of the single and double layer operator using ACA (see Algorithm 3.1) with relative accuracy $\varepsilon = 10^{-4}$. Additionally the techniques from Section 3.1 were used to improve the compression rate. In all tests a minimal blocksize $n_{\min} = 10$ was chosen.

Since $\mathcal{A}$ is coercive its Galerkin stiffness matrix A is positive definite. Under sufficiently small perturbations such as the approximation error caused by ACA this property is preserved. Hence, in contrast to the $\mathcal{H}$-matrix approximant $B_\mathcal{H}$ to B we may only generate the upper triangular part of the approximant $A_\mathcal{H}$ to A.

The same operator is tested for two different discretisations of the surface from Figure 2. The first discretisation has $n = 28288$ degrees of freedom, while for the other n is 113152. Note that storing the single layer matrix entrywise without approximation would need 3052 MB in the first case and 48841 MB in the second. The matrix B of the double layer operator is a matrix of dimension 28288×14146 in the first case and 113152×56578 in the second. Hence, the amount of storage to hold B entrywise would be 3053 MB and 48843 MB, respectively.

In the following table the time for building and the memory consumption of $A_\mathcal{H}$ and $B_\mathcal{H}$ for different admissibility parameters η are shown.

	$n = 28288$				$n = 113152$			
	single layer		double layer		single layer		double layer	
η	MB	time	MB	time	MB	time	MB	time
0.6	76	132 s	154	772 s	378	698 s	756	3972 s
0.8	78	99 s	156	596 s	391	497 s	765	2971 s
1.0	83	79 s	164	491 s	422	397 s	807	2408 s
1.2	88	71 s	172	448 s	458	353 s	860	2195 s

We observe that for lower η the approximant needs less memory but takes more time for its computation. Since we have not used the most elaborate integration routines, the CPU times for building the approximants could be easily reduced, if faster routines were employed.

5.1.2 Computing the $\mathcal{H}$-LU decomposition

After the single layer potential matrix $A_{\mathcal{H}}$ has been built in $\mathcal{H}$-matrix format we recompress a copy of it to a prescribed accuracy δ and compute the hierarchical Cholesky decomposition (see Section 4.1) with precision δ. Depending on δ this decomposition will then either be used as a preconditioner or for direct solution. If it is to be used as a direct solver we use $\delta = \varepsilon$ and may overwrite the original matrix $A_{\mathcal{H}}$. Hence, in this case no additional memory is needed. If, however, the Cholesky decomposition will be used for preconditioning, then δ is chosen to be 0.1, which leads to a condition number of at most 2, see (4.2).

In the following table the CPU time for recompressing a copy of $A_{\mathcal{H}}$ to accuracy δ, the memory consumption of the generated preconditioner and the CPU time for the Cholesky decomposition are presented. For these tests $\eta = 1.0$ was used.

	$n = 28288$			$n = 113152$		
δ	recompr.	MB	decomp.	recompr.	MB	decomp.
$1e-1$	3.6 s	11	3.4 s	20.4 s	54	3.7 s
$1e-2$	8.1 s	40	5.7 s	40.1 s	224	53.0 s
$1e-3$	6.0 s	73	21.4 s	11.3 s	366	135.1 s
$1e-4$		81	26.5 s		410	173.0 s

Apparently, compared with build $A_{\mathcal{H}}$ the CPU time for generating the Cholesky decomposition can be neglected if $\delta \sim 0.1$. If $\delta \sim \varepsilon$ was used, we obtain a direct solver with relatively small effort.

5.1.3 Solving the system

In the final step we have to find the solution of the approximated system

$$A_{\mathcal{H}}x = \tilde{b}, \qquad \tilde{b} = (\frac{1}{2}M + B_{\mathcal{H}})\tilde{g}. \tag{5.3}$$

Using the computed Cholesky decomposition of $A_{\mathcal{H}}$ we may find it directly by $x = L^{-T}L^{-1}\tilde{b}$, if the precision δ of the decomposition is of the same order as the accuracy ε of $A_{\mathcal{H}}$. The following table shows the CPU time for solving (5.3) by hierarchical forward/backward substitution.

	time	error
28288	0.1 s	$7.2e-3$
113152	0.4 s	$1.9e-3$

Since for the Dirichlet data g we used the restriction of $s(x_0,\cdot)$ to the boundary, where $x_0 \notin \overline{\Omega}$, we know the exact solution $u = s(x_0,\cdot)$. Hence, in the last table we were able to computed the approximate L^2-error

$$\left(\sum_{i=1,\dots,n} |X_i|\,|\partial_n s(x_0,m_i) - x_i|^2\right)^{1/2}$$

as an error of the solution.

If on the other hand the Cholesky decomposition has been computed with low precision, we use it to solve (5.3) by the preconditioned conjugate gradient method. In the following table the number of iterations and the CPU time to obtain a solution with residual norm $1e-8$ are listed.

	$n = 28288$		$n = 113152$	
δ	steps	time	steps	time
$1e-1$	39	3.6 s	40	20.1 s
$1e-2$	21	2.6 s	21	14.1 s
$1e-3$	6	1.0 s	6	5.2 s

As one would expect, the number of iterations is less the more accurate the LU preconditioner is. In any case the number of iterations does not depend on n. For the L^2-error of the solution we obtained the same results as in the case of the direct solver.

As a conclusion the number of iterations using the proposed preconditioner is bounded independently of n. A direct solution using backward/forward substitution is attractively fast. Decomposing A with high precision, however, is only worthwhile if (5.3) has to be solved for many right hand sides $\tilde{b}$.

5.2 Application: Magnetostatic problem

The second example is an integral equation

$$\theta(x)\varphi(x) + \int_\Gamma \varphi(y) \frac{\langle n_y, x-y\rangle}{|x-y|^3}\, \mathrm{d}s_y = \int_\Gamma \partial_{n_y}\varphi(y) \frac{1}{|x-y|}\, \mathrm{d}s_y$$

with given Neumann boundary condition $\partial_n \varphi = g$ on the surface Γ from Figure 3. The dimension of the coefficient matrix A arising from a collocation method is $n = 3760$. Since the double layer operator has a one dimensional kernel (the surface is simply connected), the extended system

$$\begin{bmatrix} A & w \\ v^T & 0 \end{bmatrix} \begin{bmatrix} x \\ \lambda \end{bmatrix} = \begin{bmatrix} b \\ 0 \end{bmatrix}$$

with $v \in \operatorname{Ker} A^T$ and $w \notin \operatorname{Im} A$ has to be solved.

In the following table we compare the results obtained by a standard solution strategy, i.e., A is built without approximation and solved using Gaussian elimination, and the results obtained by fast methods.

	storage		time		
	matrix	precond.	matrix	precond.	solution
standard	108 MB		575.0 s		1108.0 s
Mbit	42 MB		149.0 s		1273.0 s
ACA	26 MB	12 MB	55.0 s	1.7 s	1.3 s

In the row "Mbit" the results using an implementation of the fast multipole method can be found. The memory needed to store the matrix is reduced compared with the standard solution strategy. But, although a fast method was used, it took longer to solve the system iteratively than solving it directly, since no preconditioner was available. In the last row the values of the methods from this article can be found. With the proposed preconditioner the iteration needed only few steps to converge.

Acknowledgement

The author wishes to thank ABB Switzerland AG for providing the second example.

References

[1] M. Bebendorf: *Approximation of boundary element matrices.* Numer. Math. 86, 565–589, 2000.

[2] M. Bebendorf: *Efficient Galerkin BEM using ACA.* in preparation.

[3] M. Bebendorf: *Efficient Inversion of the Galerkin Matrix of General Second Order Elliptic Operators with Non-smooth Coefficients.* to appear in Math. Comp.

[4] M. Bebendorf and S. Rjasanow: *Adaptive Low-Rank Approximation of Collocation Matrices.* Computing 70, 1–24, 2003.

[5] M. Bebendorf and W. Hackbusch: *Existence of $\mathcal{H}$-Matrix Approximants to the Inverse FE-Matrix of Elliptic Operators with L^∞-Coefficients.* Numer. Math. 95, 1–28, 2003.

[6] M. Bebendorf and R. Kriemann: *Fast Parallel solution of boundary element systems.* Preprint 10/2004, Max-Planck-Institut MIS, Leipzig.

[7] G. Beylkin, R. Coifman and V. Rokhlin: *Fast wavelet transforms and numerical algorithms. I.* Comm. Pure Appl. Math. 44:141–183, 1991.

[8] G. Chen and J. Zhou: *Boundary Element Methods.* Academic Press, 1992.

[9] W. Dahmen, S. Prössdorf and R. Schneider: *Wavelet approximation methods for pseudodifferential equations. II. Matrix compression and fast solution.* Adv. Comput. Math. 1: 259–335, 1993.

[10] W. Dahmen, S. Prössdorf and R. Schneider: *Wavelet approximation methods for pseudodifferential equations. I. Stability and convergence.* Math. Z. 215: 583–620, 1994.

[11] W. Dahmen and R. Schneider: *Wavelets on Manifolds I: Construction and Domain Decomposition.* SIAM Journal of Mathematical Analysis 31: 184–230, 1999.

[12] L. Grasedyck and W. Hackbusch: *Construction and arithmetics of $\mathcal{H}$-matrices,* Computing 70: 295–334, 2003.

[13] L. Grasedyck: *Adaptive Recompression of $\mathcal{H}$-matrices for BEM*, submitted to Computing.

[14] L. Greengard and V. Rokhlin: *A new version of the fast multipole method for the Laplace equation in three dimensions.* Acta numerica, 1997, pages 229–269. Cambridge Univ. Press, Cambridge, 1997.

[15] W. Hackbusch: *Theorie und Numerik elliptischer Differentialgleichungen.* B. G. Teubner, Stuttgart, 1996 - English translation: *Elliptic differential equations. Theory and numerical treatment.* Springer-Verlag, Berlin, 1992.

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

[17] W. Hackbusch and B. N. Khoromskij: *A sparse $\mathcal{H}$-matrix arithmetic. II. Application to multi-dimensional problems.* Computing 64, 21–47, 2000.

[18] W. Hackbusch, B. Khoromskij and R. Kriemann: *Hierarchical Matrices based on a Weak Admissibility Criterion* Max Planck Institute for Mathematics in the Sciences, Preprint 2/2003.

[19] W. Hackbusch and Z. P. Nowak: *On the fast matrix multiplication in the boundary element method by panel clustering.* Numer. Math. 54, 463–491, 1989.

[20] N. Higham: *Accuracy and Stability of Numerical Algorithms*, SIAM, 1996.

[21] S. Kurz, O. Rain and S. Rjasanow: *The Adaptive Cross Approximation Technique for the 3D Boundary Element Method.* IEEE Transaction on Magnetics 38, 421–424, 2002.

[22] U. Langer, D. Pusch and S. Reitzinger: *Efficient preconditioners for boundary element matrices based on grey-box algebraic multigrid methods*, Int. J. Numer. Meth. Engng. 58, 1937–1953, 2003.

[23] M. Lintner: *Lösung der 2D Wellengleichung mittels hierarchischer Matrizen.* Technische Universität München, Germany, 2002.

[24] S. Rjasanow: *Effective algorithms with circulant-block matrices.* Linear Algebra Appl., 202:55–69, 1994.

[25] J. Schöberl: *NETGEN - An advancing front 2D/3D-mesh generator based on abstract rules.* Comput.Visual.Sci, 1:41-52, 1997.

[26] O. Steinbach and W. Wendland: *The construction of some efficient preconditioners in the boundary element method*, Adv. Comput. Math. 9, 191–216, 1998.

[27] E. Tyrtyshnikov: *Mosaic-skeleton approximations.* Calcolo 33, 47–57 (1998), 1996.