Computing 70, 1–24 (2003)
Digital Object Identifier (DOI) 10.1007/s00607-002-1469-6

Computing
Printed in Austria

Adaptive Low-Rank Approximation of Collocation Matrices

M. Bebendorf, Leipzig, and **S. Rjasanow,** Saarbrücken

Received August 15, 2002; revised September 20, 2002
Published online: March 6, 2003

Abstract

This article deals with the solution of integral equations using collocation methods with almost linear complexity. Methods such as fast multipole, panel clustering and $\mathscr{H}$-matrix methods gain their efficiency from approximating the kernel function. The proposed algorithm which uses the $\mathscr{H}$-matrix format, in contrast, is purely algebraic and relies on a small part of the collocation matrix for its blockwise approximation by low-rank matrices. Furthermore, a new algorithm for matrix partitioning that significantly reduces the number of blocks generated is presented.

AMS Subject Classification: 41A63, 41A80, 65D05, 65D15, 65F05, 65F30.

Keywords: Integral equations, hierarchical matrices, low-rank approximation, fast solvers.

1 Introduction

This article is concerned with the efficient solution of Fredholm integral equations

$$\lambda u(y) + \int_\Gamma \kappa(x,y)u(x)\,\mathrm{d}s_x = f(y), \quad y \in \Gamma, \tag{1}$$

with a given right-hand side f on a $(d-1)$-dimensional manifold $\Gamma \subset \mathbb{R}^d$. This kind of integral equation arises for example from the boundary element method. However, it is easily possible to generalise the results of this paper to volume integral equations in $\mathbb{R}^d$.

In order to solve equation (1) numerically, the domain of integration Γ is divided into triangles $\Pi_h = \{\pi_i : i \in \mathscr{I}\}$, where $\mathscr{I}$ is an index set. Besides the *Galerkin method*, the *collocation method* and the *Nyström method* are commonly used; mathematically the Galerkin method is better understood than its competitors. In the Galerkin and the collocation method the solution u is approximated from a finite dimensional ansatz space V_h, i.e., the approximant $u_h \in V_h$ to u is sought of the form $u_h = \sum_{j=1}^N u_j\varphi_j$, where φ_j, $j \in I := \{1,\ldots,N\}$, is a basis of V_h. The corresponding supports $X_j := \operatorname{supp} \varphi_j \subset \Gamma$ are assembled from the sets π_i, i.e., there is $v \in \mathbb{N}$ independently of N and for each X_j an index set $\mathscr{I}_j \subset \mathscr{I}$, $\#\mathscr{I}_j \le v$, exists, so that

$$X_j = \bigcup_{i \in \mathscr{I}_j} \pi_i. \tag{2}$$

All three methods reduce (1) to a linear system of the form

$$(\lambda B + A)x = b, \quad A, B \in \mathbb{R}^{N \times N}, \; b \in \mathbb{R}^N, \tag{3}$$

where B is a sparse matrix and produces no numerical difficulties. However, A is a full matrix and therefore needs $\mathcal{O}(N^2)$ units of storage. The usual matrix-vector multiplication requires $\mathcal{O}(N^2)$ arithmetical operations. The latter is of particular importance if an iterative method is used for the solution of (3). The entries of A in the case of the Galerkin method are

$$a_{ij} = \int_\Gamma \int_\Gamma \kappa(x,y)\varphi_j(x)\varphi_i(y) \; \mathrm{d}s_x \, \mathrm{d}s_y, \tag{4}$$

in the case of the collocation method

$$a_{ij} = \int_\Gamma \kappa(x, y_i)\varphi_j(x) \; \mathrm{d}s_x \tag{5}$$

with *collocation points* $y_i \in X_i$ and in the case of the Nyström method

$$a_{ij} = \kappa(y_j, y_i) \quad \text{for } i \neq j \quad \text{and} \quad a_{ii} = c_i \tag{6}$$

with N pairwise distinct points $y_i \in \Gamma$ and N numbers c_i. In this article we will focus on collocation matrices, i.e. matrices of type (5). Matrices of type (6) were investigated in [1].

Modern numerical methods for the solution of (3) such as fast multipole [5, 10], panel clustering [9] and $\mathscr{H}$-matrices [7, 8] provide an approximation $\tilde{x}$ to the solution vector x in almost linear complexity by solving a perturbed linear system (3) in which A is easier to handle. The accuracy of $\tilde{x}$ is chosen so that the additional error for u is of the same size as the consistency error of the discretisation method. All these methods are based on kernel approximations by *degenerate kernels*, i.e.

$$\kappa(x,y) \approx \tilde{\kappa}(x,y) := \sum_{i=1}^{k} u_i(x) v_i(y). \tag{7}$$

In this article the kernel $\kappa : \Gamma \times \mathbb{R}^d \to \mathbb{R}$ in (1) is assumed to be *asymptotically smooth with respect to* y, i.e. $\kappa(x, \cdot) \in C^\infty(\mathbb{R}^d \setminus \{x\})$ for almost all $x \in \Gamma$, and there is a constant $g < 0$ so that for all multiindices $\alpha \in \mathbb{N}_0^d$ it holds that

$$|D_y^\alpha \kappa(x,y)| \le c_p |x - y|^{g-p}, \quad p = |\alpha|, \tag{8}$$

where c_p depends only on p. As usual we denote by D_y^α the partial derivative

$$D_y^\alpha = \left(\frac{\partial}{\partial y_1}\right)^{\alpha_1} \cdots \left(\frac{\partial}{\partial y_d}\right)^{\alpha_d}.$$

Strongly singular kernels are not excluded. However, then the integral in (1) has to be defined by an appropriate regularisation. E.g., the kernels $\kappa(x,y) = |x-y|^g$, $g < 0$, as well as their partial derivatives are asymptotically smooth. Furthermore, the kernel of the double-layer potential operator for the three-dimensional Laplace equation

$$\kappa(x,y) = -\frac{1}{4\pi}\frac{(x-y,n_x)}{|x-y|^3}$$

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

From (5) it can be seen that for the entry a_{ij} the kernel function κ is evaluated only on $X_i \times X_j$, i.e., a block $A_{t_1 t_2}$, $t_1, t_2 \subset I$, within the system matrix A corresponds to a pair of domains (X_{t_1}, X_{t_2}), where for $t \subset I$ we set $X_t = \bigcup_{j\in t} X_j$. For the latter assume that

$$\operatorname{diam} X_{t_2} \le \eta \operatorname{dist}(X_{t_1}, X_{t_2}), \quad \eta < 1, \tag{9}$$

holds. Then asymptotical smoothness guarantees the existence of degenerate kernel approximants (7) as can be seen from the truncated Taylor expansion. However, asymptotical smoothness is only sufficient for the existence of $\tilde{\kappa}$. For example in [3] it is proven that the Green function of the inverse of an uniformly elliptic partial differential operator with L^∞-coefficients can be approximated by a degenerate kernel.

$\mathscr{H}$-matrices obtain their efficiency in arithmetics and storage from a hierarchical partition of the matrix and using low-rank matrices as an approximation to each of the blocks. The blockwise low-rank approximant permits a fast matrix-vector multiplication, which can be exploited in iterative solvers, and can be stored efficiently. It has to be guaranteed that the perturbation made to the discrete operator does not lead to a loss of important properties such as solvability. The usual way to obtain a low-rank approximant from a degenerate kernel approximation in each of the cases (4)–(6) is to replace κ by $\tilde{\kappa}$. Obviously, the new block $\tilde{A}_{t_1 t_2}$ is an approximation to the entries $A_{t_1 t_2}$ from (4)–(6) respectively, and the rank of $\tilde{A}_{t_1 t_2}$ is bounded by the number k of terms in the sum (7).

The aim of this article is to present an algorithm for the generation of low-rank approximants from the matrix entries themselves without explicitly dealing with the kernel. The advantage of this technique is that the performance of already existing computer codes can be improved easily without changing the routines for the calculation of the matrix entries. Since only few entries are necessary to generate the approximant, the coefficient matrix does not have to be calculated in advance. Furthermore, the algorithm presented adapts the rank of the approximant to the respective needs, whereas in the existing methods the rank has to be fixed a priori from theoretical error estimates.

The structure of the rest of the article is as follows: for the existence of low-rank approximants in the case of asymptotically smooth kernels we have to impose condition (9). Since this condition cannot be fulfilled on $\Gamma \times \Gamma$, the coefficient matrix A has to be subdivided into blocks corresponding to domains that fulfil condition (9). Usually, algorithms as in [8, 1] are used to partition the matrix appropriately. In Section 2 we present a new algorithm that compared with the standard algorithms significantly reduces the number of blocks generated. For the sake of simplicity we will assume that the discretisation is quasi-uniform. Adaptive meshes require a more complicated complexity analysis for the matrix partitioning but do not affect the approximation results in Section 3. In this section an algorithm for the generation of low-rank approximants is described, which here will be referred to as ACA (Adaptive Cross Approximation). This algorithm is purely algebraic in the sense that it uses only entries from the original matrix A for the approximation of each block. The numerical examples in Section 4 show that this algorithm has almost linear complexity. Furthermore, not only can the approximant generated be stored efficiently, but multiplying it with a vector as a part of an iterative method also has almost linear complexity.

2 Matrix Partitioning

The aim of this section is to construct a partition of the coefficient matrix. We will divide the index set (I, I) into pairwise disjoint subsets $P = \{(t_1, t_2) : t_1, t_2 \subset I\}$, so that

$$I \times I = \bigcup_{(t_1,t_2)\in P} t_1 \times t_2$$

and for each pair $(t_1, t_2) \in P$ one of the following conditions holds:

$$\min\{\#t_1, \#t_2\} = 1 \quad \text{or} \tag{10a}$$

$$\operatorname{diam} X_{t_2} \leq \eta \operatorname{dist}(X_{t_1}, X_{t_2}). \tag{10b}$$

The parameter η is an upper bound for the relative distance of two clusters t_1 and t_2 and will be chosen later. If a pair (t_1, t_2) fulfils condition (10a) then each element from the corresponding block will be generated and stored. For all other pairs, condition (10b) is valid and in Section 3 we will investigate an algorithm for the approximation of the corresponding block by matrices of low rank. Both storage and multiplication of the resulting matrix by a vector can be done blockwise, taking advantage of the efficient representation of low-rank matrices.

2.1 The Algorithm

Given $S : \mathrm{Pot}(I) \to \mathrm{Pot}(\mathrm{Pot}(I))$ mapping an index set $t \subset I$ to a set $S(t)$ of pairwise disjoint subsets, so that

$$t = \bigcup_{s \in S(t)} s, \qquad \text{if } \#t > 1,$$

and $S(t) = \emptyset$, if $\#t \leq 1$, we define a cluster tree T by recursively applying S starting from the root I. S prescribes how an index set $t \subset I$ and therefore the domain X_t is subdivided into its sons. Hence, the cluster tree T contains a hierarchy of partitions of I. Cluster trees are frequently used in this field of research, cf. [9, 8]. For $t \subset I$ we assume that $S(t)$ can be evaluated in $\mathcal{O}(\#t)$ operations. An example for S can be found in [2]. By $q \in \mathbb{N}$ we denote the maximum degree, i.e. the maximum number of sons, of vertices $t \in T$. If we assume the ratios $\max\{\#t/\#s, s \in S(t)\}$, $t \in T$ and $S(t) \neq \emptyset$, to be bounded by $R > 1$ from below, the depth of the cluster tree is of order $L := \log_R N = \mathcal{O}(\log N)$.

Instead of searching all possible partitions of $I \times I$ for a candidate P that fulfils (10), in the following algorithm we are looking at only those partitions with pairs (t_1, t_2) for which t_2 stems from the cluster tree T. By this simplification it may happen that not the optimal P is found. However, as we will see, a partition P satisfying our needs can be computed in reasonable time.

For $t \in T$ let $z_t \in \mathbb{R}^d$ be an arbitrary but fixed point in X_t,

$$\tilde{F}(t) := \left\{ i \in I : \mathrm{diam}\, X_t \leq \frac{\eta}{1+\eta}\ \mathrm{dist}(X_i, z_t) \right\} \tag{11}$$

and $\tilde{N}(t) := I \setminus \tilde{F}(t)$. It can easily be seen that $\tilde{F}(t) \subset F(t)$, where

$$F(t) := \{ i \in I : \mathrm{diam}\, X_t \leq \eta\ \mathrm{dist}(X_i, X_t) \}$$

denotes the *farfield* of X_t. Obviously, the computation of $t_1 \cap \tilde{F}(t_2)$ can be performed with $\mathcal{O}(\#t_1 + \#t_2)$ operations, whereas the computation of $t_1 \cap F(t_2)$ would need $\mathcal{O}(\#t_1 \#t_2)$ operations.

Algorithm 2.1. *Partitioning* (t_1, t_2)

```
If $\min\{\#t_1, \#t_2\} = 1$ then $P := P \cup \{(t_1, t_2)\}$          // block $(t_1,t_2)$ fulfils (10a)
else begin
    $S_{t_2} := S(t_2)$;                                                // evaluate $S(t_2)$
    for $s \in S_{t_2}$ begin
        if $t_1 \cap \tilde{F}(s) \neq \emptyset$ then begin
            $P := P \cup \{(t_1 \cap \tilde{F}(s), s)\}$;          //block $(t_1 \cap \tilde{F}(s), s)$ fulfils (10b)
            if $t_1 \setminus \tilde{F}(s) \neq \emptyset$ then Partitioning$(t_1 \setminus \tilde{F}(s), s)$       // recursion
        end
    end
end
```

In Algorithm 2.1 each block (t_1, t_2) with $\#t_2 > 1$ is subdivided into blocks (t_1, s), $s \in S(t_2)$. The set $t_1 \cap \tilde{F}(s)$ is the largest possible subset of t_1 within the farfield $F(s)$ of s under the simplification we made when replacing F by $\tilde{F}$. Hence, $(t_1 \cap \tilde{F}(s), s)$ fulfils (10b) and is stored in P. The algorithm is then applied to the remaining parts $(t_1 \setminus \tilde{F}(s), s)$. The recursion stops if t_2 cannot be subdivided any more, i.e. $\#t_2 = 1$. It is obvious that a partition with the desired properties (10) is obtained by applying *Partitioning* to (I, I), if P was previously initialised by $P = \emptyset$. Notice that there is no need to calculate the cluster tree T in advance.

2.2 Computational complexity

In this subsection we will estimate the computational cost of the above algorithm. The domain of integration $\Gamma \subset \mathbb{R}^d$ is $(d-1)$-dimensional, i.e., there are two constants $c_1, c_2 > 0$ so that for all $z \in \Gamma$

$$c_1 r^{d-1} \le |\Gamma \cap B_r(z)| \le c_2 r^{d-1} \quad \text{for } r \to 0. \tag{12}$$

Here, $|\cdot|$ denotes the surface measure and $B_r(z) \subset \mathbb{R}^d$ is the Euclidean ball with centre z and radius r. We assume the triangulation Π_h to be *quasi-uniform* and *shape-regular*, i.e., that there is a constant c_u so that

$$c_u |X_i| \ge h^{d-1} \quad \text{for } i \in I, \tag{13}$$

where $h = \max_{i \in I} \operatorname{diam} X_i$. This guarantees in particular that we are able to find $c_3 > 0$ such that $\operatorname{diam} X_i \le c_3 \operatorname{diam} X_j$, $i, j \in I$. The supports X_i may overlap. In accordance with the standard finite element discretisation we require that each panel belongs to the support of a bounded number of basis functions, i.e., there is a constant $\mu > 0$ so that

$$\mu |X_t| \ge \sum_{i \in t} |X_i|. \tag{14}$$

These assumptions lead to

Lemma 2.2. *There is a constant* c_4 *such that* $\#t/c_4 \le |X_t| h^{-(d-1)} \le c_4 \#t$ *for all* $t \subset I$.

Proof. Using (12) and (13) we see that

$$|X_t| \le \sum_{i \in t} |X_i| \le c_2 \#t \max_{i \in I} \operatorname{diam}^{d-1} X_i = c_2 \#t\, h^{d-1}.$$

On the other hand $\mu |X_t| \ge \sum_{i \in t} |X_i| \ge \#t\, h^{d-1}/c_u$. □

Lemma 2.3. *Let* $t \subset I$. *Then* $\#\tilde{N}(t) \le c(\eta)\#t$, $c(\eta) = \tilde{c}(1 + 1/\eta)^{d-1}$ *holds, where* $\tilde{N}(t)$ *is the set from* (11).

Proof. From (13) follows $\operatorname{diam} X_i \le c_3 \operatorname{diam} X_t,\quad i \in I$. Since $\operatorname{dist}(X_i, z_t) < (1 + 1/\eta) \operatorname{diam} X_t$ for $i \in \tilde{N}(t)$ the ball $B_r(z_t)$ with $r = (c_3 + 1 + 1/\eta) \operatorname{diam} X_t$ covers $X_{\tilde{N}(t)} \supset X_t$. Now

$$|X_{\tilde{N}(t)}| \le c_2 r^{d-1} = c_2 (c_3 + 1 + 1/\eta)^{d-1} \operatorname{diam}^{d-1} X_t.$$

Using (12) again we obtain

$$|X_{\tilde{N}(t)}| \le \frac{c_2}{c_1} (c_3 + 1 + 1/\eta)^{d-1} |X_t|.$$

Lemma 2.2 leads to the desired estimate. □

Theorem 2.4. *The number of operations and the amount of storage needed to perform the recursion Partitioning (I, I) is of order $\eta^{-(d-1)} N \log N$. The number of blocks generated is of order N.*

Proof. The recursion *Partitioning* (I, I) descending the cluster tree T generates at most q blocks in each node. Remember that by assumption the maximum degree of vertices in T is q. The number of nodes is limited by qN, so the number of generated blocks is of order N.

Let N_{t_2} denote the number of operations needed to carry out a part *Partitioning* (t_1, t_2) of the whole recursion *Partitioning* (I, I). We will show that $N_{t_2} \le \bar{c}(\eta) \# t_2 \log_R \# t_2$, where $\bar{c}(\eta) = c_5(1 + q c(\eta))$. According to the remark preceding Algorithm 2.1 for the number of operations it holds that

$$N_{t_2} \le \sum_{s \in S(t_2)} c_5(\# t_1 + \# s) + N_s \le c_5 \# t_2 (1 + q c(\eta)) + \sum_{s \in S(t_2)} N_s.$$

For the last estimate we used Lemma 2.3 because *Partitioning* is only applied to pairs (t_1, t_2) for which $t_1 \subset \tilde{N}(t_2)$ is valid. Thus

$$\begin{aligned} N_{t_2} &\le \sum_{s \in S(t_2)} \bar{c}(\eta) \# s \log_R \# s + \bar{c}(\eta) \# t_2 \\ &= \bar{c}(\eta) \# t_2 \sum_{s \in S(t_2)} \frac{\# s}{\# t_2} \left(\log_R \# t_2 - \log_R \frac{\# t_2}{\# s} \right) + \bar{c}(\eta) \# t_2 \\ &= \bar{c}(\eta) \# t_2 \log_R \# t_2 + \bar{c}(\eta) \# t_2 \left(1 - \sum_{s \in S(t_2)} \frac{\# s}{\# t_2} \log_R \frac{\# t_2}{\# s} \right) \\ &\le \bar{c}(\eta) \# t_2 \log_R \# t_2, \end{aligned}$$

because we have assumed that $\# t_2 \ge R \# s$. □

In Section 3 it will be shown that it is possible to generate a rank-k approximant of a single block (t_1, t_2) in $\mathcal{O}(k^2(\#t_1 + \#t_2))$ operations. Storing and multiplying it by a vector takes $\mathcal{O}(k(\#t_1 + \#t_2))$ operations. By the same arguments as in the proof of Theorem 2.4 the numerical effort for approximating the whole matrix is $\mathcal{O}(\eta^{-(d-1)}k^2 N \log N)$. The cost for storing and multiplying the whole matrix by a vector amounts to $\mathcal{O}(\eta^{-(d-1)}kN \log N)$.

3 Incomplete Low-Rank Approximation

In the preceding section we explained how to partition a matrix into blocks (t_1, t_2) such that for the corresponding domains X_{t_1} and X_{t_2}

$$\operatorname{diam} X_{t_2} \le \eta \operatorname{dist}(X_{t_1}, X_{t_2}) \tag{15}$$

holds or the block degenerates to a vector. In this section we may therefore concentrate on a single block $B \in \mathbb{R}^{m\times n}$ with entries

$$b_{ij} = \int_\Gamma \kappa(x, y_i)\varphi_j(x)\, \mathrm{d}s_x, \quad i = 1, \ldots, m,\ j = 1, \ldots, n, \tag{16}$$

satisfying (15). We introduce functions $\mathscr{L}_j\kappa$ on $\mathbb{R}^d \setminus \bar{X}_{t_1}$ by

$$(\mathscr{L}_j\kappa)(y) = \int_\Gamma \kappa(x, y)\varphi_j(x)\ \mathrm{d}s_x.$$

Note that (8) guarantees that the functions $\mathscr{L}_j\kappa$ are well defined for all $g < 0$.

Let $k \in \mathbb{N}$. The aim of this section is to devise an algorithm which decomposes the block B in the following way:

$$B = U_k V_k^T + R_k,$$

where $U_k \in \mathbb{R}^{m\times k}$ and $V_k \in \mathbb{R}^{n\times k}$, and $R_k \in \mathbb{R}^{m\times n}$ is the approximation error. If we succeed in proving that the norm of R_k is small, B is obviously approximated by a matrix of rank at most k.

3.1 The Algorithm

In the course of this section an integer m' and points ζ_i, $i = 1, \ldots, m'$, will be chosen appropriately. We combine the yet undefined points ζ_i with the collocation points y_j in the $(m' + m)$-tuple

$$z_i = \begin{cases} \zeta_i, & 1 \le i \le m' \\ y_{i-m'}, & m' < i \le m' + m \end{cases}$$

and extend B to $\hat{B}$ by setting $\hat{b}_{ij} = (\mathscr{L}_j\kappa)(z_i)$, $i = 1, \ldots, m' + m$, $j = 1, \ldots, n$. The idea of the following algorithm is to approximate $\hat{B}$ by the outer product $\hat{u}v^T$ of one column $\hat{u}$ and one (scaled) row v of $\hat{B}$. Higher order approximations are obtained by approximating the remainder of the previous approximation step, respectively. Once m' and ζ_i are given, the following algorithm, in which $\hat{u}_k$ and v_k are successively generated, can be applied.

Algorithm 3.1.

```
$k := 1;\ i_1 = 1$
repeat
    $(\tilde{v}_k)_j := (\mathscr{L}_j\kappa)(\zeta_{i_k}) - \sum_{\ell=1}^{k-1}(\hat{u}_\ell)_{i_k}(v_\ell)_j, \quad j = 1,\dots,n$
  if $\tilde{v}_k$ *vanishes* then $i_k := i_k + 1$
  else begin
    $j_k := \operatorname{argmax}_{j=1,\dots,n}|(\tilde{v}_k)_j|;\ \gamma_k := (\tilde{v}_k)_{j_k}^{-1};\ v_k := \gamma_k\tilde{v}_k;$
      $(\hat{u}_k)_i := (\mathscr{L}_{j_k}\kappa)(z_i) - \sum_{\ell=1}^{k-1}(\hat{u}_\ell)_i(v_\ell)_{j_k}, \quad i = 1,\dots,m'+m$
    $i_{k+1} := i_k + 1;\ k := k + 1$
  end
until $i_k > m'$
```

Since the $\hat{B}$ is evaluated only in the first m' rows and the columns $j_1,\dots,j_k$, there is no need to generate the whole block $\hat{B}$ for its approximation. Due to this we call this approximation by low-rank matrices incomplete. It is worth remarking that Algorithm 3.1 differs from the algorithm in [1] for the approximation of matrices of type (6) only with respect to the initial matrix $\hat{B}$. Define

$$\hat{S}_k = \sum_{\ell=1}^{k}\hat{u}_\ell v_\ell^T$$

and $\hat{R}_k = \hat{B} - \hat{S}_k$. It is easy to see that $(\hat{u}_k)_i = (\hat{R}_{k-1})_{ij_k}$, $1 \le i \le m' + m$ and $(\tilde{v}_k)_j = (\hat{R}_{k-1})_{i_kj}$, $1 \le j \le n$. The rank of $\hat{S}_k$ is bounded by k. For the calculation of $\hat{S}_k$ we need at most $k(m' + m + n)$ units of memory and

$m'n + k(m' + m)$	evaluations of $\mathscr{L}_j\kappa$,
$2km'n + k^2(m' + m)$	additions and multiplications,
kn	divisions.

Under the assumption that the evaluation of $(\mathscr{L}_j\kappa)(y)$ can be done in $\mathcal{O}(1)$ operations the cost for the generation of the approximant $\hat{S}_k$ sum up to $\mathcal{O}(km'(m' + m + n))$ operations.

We have already pointed out in [1] that each step of Algorithm 3.1 may be understood as the generation of an approximant using the column-pivoted *LU* decomposition. To see this let us assume that $i_\ell = j_\ell = \ell$, $\ell = 1, \dots, k$. In this case we have

$$\hat{R}_k = (I - \gamma_k \hat{R}_{k-1} e_k e_k^T)\hat{R}_{k-1} = L_k \hat{R}_{k-1}.$$

The $(m' + m) \times (m' + m)$ matrix L_k

$$L_k = \begin{bmatrix} 1 & & & & & \\ & \ddots & & & & \\ & & 1 & & & \\ & & & 0 & & \\ & & & -\frac{(\hat{R}_{k-1})_{k+1,k}}{(\hat{R}_{k-1})_{k,k}} & 1 & \\ & & & \vdots & & \ddots & \\ & & & -\frac{(\hat{R}_{k-1})_{m'+m,k}}{(\hat{R}_{k-1})_{k,k}} & & & 1 \end{bmatrix}$$

differs from a *Gaussian matrix* only in position (k, k).

In the rest of this section the entries of $\hat{R}_k$ will be estimated. To this end we will relate $\hat{R}_k$ with functions r_k constructed by the following rule. Let $r_0(x, y) = \kappa(x, y)$, $s_0(x, y) = 0$ and for $k = 1, 2, \dots$

$$r_k(x, y) = r_{k-1}(x, y) - \gamma_k (\mathscr{L}_{j_k} r_{k-1})(y) r_{k-1}(x, \zeta_{i_k}), \tag{17a}$$
$$s_k(x, y) = s_{k-1}(x, y) + \gamma_k (\mathscr{L}_{j_k} r_{k-1})(y) r_{k-1}(x, \zeta_{i_k}). \tag{17b}$$

The following relation between $\hat{R}_k$ and r_k is obvious.

Lemma 3.2. *For* $1 \le i \le m' + m$, $1 \le j \le n$ *and* $k \ge 1$ *it holds that* $(\hat{R}_k)_{ij} = (\mathscr{L}_j r_k)(z_i)$.

The number of zeros of $\mathscr{L}_j r_k$ increases correspondingly with increasing k.

Lemma 3.3. *For* $k \ge 0$ and $1 \le \ell \le k$ *it holds that*

$$(\mathscr{L}_{j_\ell} r_k)(y) = 0 \quad \text{for all } y \in \mathbb{R}^d \setminus X_{t_1}.$$

The preceding lemma can be proven by inductively applying (17). We define

$$\hat{A}_k = \begin{bmatrix} (\mathscr{L}_{j_1}\kappa)(\zeta_{i_1}) & \dots & (\mathscr{L}_{j_k}\kappa)(\zeta_{i_1}) \\ \vdots & & \vdots \\ (\mathscr{L}_{j_1}\kappa)(\zeta_{i_k}) & \dots & (\mathscr{L}_{j_k}\kappa)(\zeta_{i_k}) \end{bmatrix}. \tag{18}$$

Let $\hat{A}_k(\ell,j) \in \mathbb{R}^{k\times k}$ be the matrix resulting from $\hat{A}_k$ by replacing the ℓth column with the vector $(\mathscr{L}_j\kappa)([\zeta]_k)$. Here and in the rest of this article we use the abbreviations

$$(\mathscr{L}_j\kappa)([\zeta]_k) := \begin{bmatrix} (\mathscr{L}_j\kappa)(\zeta_{i_1}) \\ \vdots \\ (\mathscr{L}_j\kappa)(\zeta_{i_k}) \end{bmatrix} \quad \text{and} \quad ([\mathscr{L}]_k\kappa)(y) := \begin{bmatrix} (\mathscr{L}_{j_1}\kappa)(y) \\ \vdots \\ (\mathscr{L}_{j_k}\kappa)(y) \end{bmatrix}.$$

Let $M_k(\ell,y) \in \mathbb{R}^{k\times k}$ be the matrix that forms after replacing the ℓth row of $\hat{A}_k$ with the last vector $([\mathscr{L}]_k\kappa)(y)$. Especially, $\hat{A}_k = \hat{A}_k(\ell,j_\ell) = M_k(\ell,\zeta_{i_\ell})$ holds. For the determinants of the matrices $\hat{A}_k(\ell,j)$ the following recurrence relation can be obtained.

Lemma 3.4. *For* $1 \le \ell < k$ *we have*

$$\det\hat{A}_k(\ell,j) = \gamma_k^{-1}\det\hat{A}_{k-1}(\ell,j) - (\mathscr{L}_j r_{k-1})(\zeta_{i_k})\det\hat{A}_{k-1}(\ell,j_k),$$

$\det\hat{A}_1(1,j) = (\mathscr{L}_j\kappa)(\zeta_{i_1})$ *and* $\det\hat{A}_k(k,j) = (\mathscr{L}_j r_{k-1})(\zeta_{i_k})\det\hat{A}_{k-1}$ *for* $k > 1$. *Especially,*

$$\det\hat{A}_k = \prod_{\ell=1}^{k}\gamma_\ell^{-1}.$$

Proof. It is easy to see that there are coefficients $\alpha_\mu^{(k-1)}, \mu = 1,\dots,k-1$, so that

$$(\mathscr{L}_\nu r_{k-1})(\zeta_{i_k}) = (\mathscr{L}_\nu\kappa)(\zeta_{i_k}) - \sum_{\mu=1}^{k-1}\alpha_\mu^{(k-1)}(\mathscr{L}_\nu\kappa)(\zeta_{i_\mu}), \quad \nu = 1,\dots,n.$$

Thus it is possible to replace each entry $(\mathscr{L}_\nu\kappa)(\zeta_{i_k})$ in the last row of $\hat{A}_k(\ell,j)$ by $(\mathscr{L}_\nu r_{k-1})(\zeta_{i_k})$ and obtain $\tilde{A}_k(\ell,j)$ without changing the determinant. From the last lemma it can be seen that $(\mathscr{L}_{j_\ell}r_{k-1})(\zeta_{i_k}) = 0, 1 \le \ell < k$. Hence, only the ℓth and the kth entry of the last row of $\tilde{A}_k(\ell,j)$ may not vanish. Using Laplace's theorem we end up with the assertion. □

Not only does the last lemma guarantee that $\hat{A}_k$ is non-singular, we also notice that the larger the product of the values γ_ℓ^{-1}, $\ell = 1,\dots,k$, the larger the determinant of $\hat{A}_k$.

3.2 Type of Approximation

We are now in a position to show that the decomposition of κ into s_k and r_k has a representation that can be exploited for the error analysis.

Lemma 3.5. *The sequences* $\{s_k\}_k$ *and* $\{r_k\}_k$ *generated in* (17) *satisfy*

$$s_k(x,y) + r_k(x,y) = \kappa(x,y),$$

where for $k \geq 1$

$$s_k(x,y) = ([\mathscr{L}]_k\kappa)(y)^T \hat{A}_k^{-1}\kappa(x,[\zeta]_k). \tag{19}$$

Provided $\{\zeta_{i_1},\dots,\zeta_{i_k}\} \subset \{y_1,\dots,y_m\}$ the decomposition (19) constitutes the analytic form of a pseudo skeleton decomposition in the sense of [6]. In contrast to multipole [5, 10] and panel clustering [9] methods we do not approximate the functions $\mathscr{L}_j\kappa$ by using appropriate approximations to the kernel κ. Instead, (19) shows that we employ a small number of functions chosen from $\{\mathscr{L}_1\kappa,\dots,\mathscr{L}_n\kappa\}$ to approximate $\mathscr{L}_j\kappa$.

Proof of Lemma 3.5. The lemma is obviously true for $k=1$. We continue by induction. From the definition of r_k and s_k we can see that

$$s_k(x,y) + r_k(x,y) = s_{k-1}(x,y) + r_{k-1}(x,y),$$

which according to the assumption coincides with $\kappa(x,y)$.

For the sake of simplicity we set

$$a_k = \hat{A}_{k-1}^{-1}(\mathscr{L}_{j_k}\kappa)([\zeta]_{k-1}) \quad \text{and} \quad b_k = \hat{A}_{k-1}^{-T}(([\mathscr{L}]_{k-1}\kappa)(\zeta_{i_k}).$$

Since

$$\begin{aligned} s_k(x,y) &= s_{k-1}(x,y) + \gamma_k(\mathscr{L}_{j_k}r_{k-1})(y)r_{k-1}(x,\zeta_{i_k}) \\ &= \begin{bmatrix} ([\mathscr{L}]_{k-1}\kappa)(y) \\ (\mathscr{L}_{j_k}\kappa)(y) \end{bmatrix}^T \begin{bmatrix} \hat{A}_{k-1}^{-1} + \gamma_k a_k b_k^T & -\gamma_k a_k \\ -\gamma_k b_k^T & \gamma_k \end{bmatrix} \begin{bmatrix} \kappa(x,[\zeta]_{k-1}) \\ \kappa(x,\zeta_{i_k}) \end{bmatrix} \end{aligned}$$

and

$$\hat{A}_k \begin{bmatrix} \hat{A}_{k-1}^{-1} + \gamma_k a_k b_k^T & -\gamma_k a_k \\ -\gamma_k b_k^T & \gamma_k \end{bmatrix} = I_k$$

together with the induction assumption for s_{k-1}, the property

$$s_k(x,y) = ([\mathscr{L}]_k\kappa)(y)^T \hat{A}_k^{-1}\kappa(x,[\zeta]_k)$$

can also be deduced for s_k. □

Using Cramer's rule we see that

$$\left(([\mathscr{L}]_k\kappa)(y)^T \hat{A}_k^{-1}\right)_\ell = \frac{\det M_k(\ell,y)}{\det \hat{A}_k},$$

where $M_k(\ell,y)$ is the matrix defined from (18). Hence from (19) we obtain

$$(\mathscr{L}_j s_k)(y) = \sum_{\ell=1}^{k} \frac{\det M_k(\ell, y)}{\det \hat{A}_k} (\mathscr{L}_j \kappa)(\zeta_{i_\ell}). \tag{20}$$

The last representation shows that $\mathscr{L}_j s_k$ is the uniquely defined interpolant of $\mathscr{L}_j \kappa$ at the nodes ζ_{i_ℓ} in the span of the functions $\mathscr{L}_{j_\ell} \kappa$, $\ell = 1, \ldots, k$. In order to see this, let Φ be a k-dimensional space of functions $\phi : \mathbb{R}^d \to \mathbb{R}$ equipped with a basis $\phi_1, \ldots, \phi_k$, $x_i \in \mathbb{R}^d$ and $f_i \in \mathbb{R}$, $i = 1, \ldots, k$. Define

$$M_\ell(x) = \begin{array}{cc} \begin{bmatrix} \phi_1(x_1) & \ldots & \phi_k(x_1) \\ \vdots & & \vdots \\ \phi_1(x) & \ldots & \phi_k(x) \\ \vdots & & \vdots \\ \phi_1(x_k) & \ldots & \phi_k(x_k) \end{bmatrix} & \begin{array}{c} \\ \\ \leftarrow \ell. \\ \\ \\ \end{array} \end{array}$$

Furthermore, let $M = M_\ell(x_\ell) \in \mathbb{R}^{k \times k}$. It is obvious that provided M is non-singular the Lagrange functions

$$\chi_\ell(x) := \frac{\det M_\ell(x)}{\det M}$$

are in Φ and $\chi_\ell(x_i) = \delta_{i\ell}, 1 \leq i, \ell \leq k$, where δ denotes Kronecker's symbol. The function

$$L_k^\phi(x) = f_1 \chi_1(x) + \cdots + f_k \chi_k(x) \in \Phi$$

solves the interpolation problem

$$L_k^\phi(x_i) = f_i, \quad 1 \leq i \leq k, \tag{21}$$

and is uniquely defined.

3.3 Error Analysis

We need an expression for the interpolation error. To this end, we will relate the error $\mathscr{L}_j r_k$ to the error in another system of functions. Let $\psi_1, \ldots, \psi_{m'}$ be any system of functions with

$$\det[\psi_j(\zeta_i)]_{1 \leq i, j \leq m'} \neq 0, \tag{22}$$

and define their span by Ψ. For this system let the error

$$E_{m'}^\psi[\mathscr{L}_j \kappa] := \mathscr{L}_j \kappa - L_{m'}^\psi[\mathscr{L}_j \kappa]$$

be known.

It is now possible to relate the remainder $\mathscr{L}_j r_k$ of our approximation to the remainder $E^{\psi}_{m'}$ of the interpolation in the ψ-system. Notice that the points ζ_i, $i_k < i < i_{k+1}$, which were omitted during the construction of the sequences $\{r_k\}_k$ and $\{s_k\}_k$, are not lost for the approximation error, since in (23) the error $E^{\psi}_{m'}$ is the error for an interpolation at m' nodes.

Lemma 3.6. *Let $\{i_1,\ldots,i_k\}$ be generated by Algorithm 3.1. Then the functions $\mathscr{L}_j r_k$ satisfy*

$$(\mathscr{L}_j r_k)(y) = E^{\psi}_{m'}[\mathscr{L}_j\kappa](y) - \sum_{\ell=1}^{k} \frac{\det \hat{A}_k(\ell,j)}{\det \hat{A}_k} E^{\psi}_{m'}[\mathscr{L}_{j_\ell}\kappa](y), \quad y \in \mathbb{R}^d \setminus \overline{X}_{t_1}. \qquad (23)$$

Proof. Let χ_i be the ith Lagrange function in Ψ, i.e., for $1 \le i, \ell \le m'$ we have $\chi_i(\zeta_\ell) = \delta_{i\ell}$. Set

$$\chi(y) = \begin{bmatrix} \chi_1(y) \\ \vdots \\ \chi_{m'}(y) \end{bmatrix}.$$

For the interpolant $L^{\psi}_{m'}[\mathscr{L}_{j'}\kappa](y) = (\mathscr{L}_{j'}\kappa)([\zeta]_{m'})^T\chi(y)$, $1 \le j' \le n$, we obtain

$$L^{\psi}_{m'}[\mathscr{L}_{j'}\kappa](y) = \sum_{i=1}^{i_1-1} (\mathscr{L}_{j'}\kappa)(\zeta_i)\chi_i(y) + \sum_{\ell=1}^{k}\left\{ (\mathscr{L}_{j'}\kappa)(\zeta_{i_\ell})\chi_{i_\ell}(y) + \sum_{i=i_\ell+1}^{i_{\ell+1}-1} (\mathscr{L}_{j'}\kappa)(\zeta_i)\chi_i(y)\right\},$$

where we set $i_{k+1} = m'+1$. Since $(\mathscr{L}_{j'} r_\ell)(\zeta_i) = 0$ for all $i_\ell < i < i_{\ell+1}$ we obtain with the aid of Lemma 3.5

$$0 = (\mathscr{L}_{j'} r_\ell)(\zeta_i) = (\mathscr{L}_{j'}\kappa)(\zeta_i) - ([\mathscr{L}]_\ell\kappa)(\zeta_i)^T \hat{A}_l^{-1} (\mathscr{L}_{j'}\kappa)([\zeta]_\ell).$$

Thus

$$(\mathscr{L}_{j'}\kappa)(\zeta_i) = \sum_{\nu=1}^{\ell} \Big(([\mathscr{L}]_\ell\kappa)(\zeta_i)^T \hat{A}_\ell^{-1} \Big)_\nu (\mathscr{L}_{j'}\kappa)(\zeta_{i_\nu}).$$

From this it follows that

$$\begin{aligned}\sum_{i=i_\ell+1}^{i_{\ell+1}-1} (\mathscr{L}_{j'}\kappa)(\zeta_i)\chi_i(y) &= \sum_{i=i_\ell+1}^{i_{\ell+1}-1} \sum_{\nu=1}^{\ell} \Big(([\mathscr{L}]_l\kappa)(\zeta_i)^T \hat{A}_\ell^{-1} \Big)_\nu (\mathscr{L}_{j'}\kappa)(\zeta_{i_\nu})\chi_i(y) \\ &= \sum_{\nu=1}^{\ell} (\mathscr{L}_{j'}\kappa)(\zeta_{i_\nu})\alpha_\nu^{(\ell)}(y),\end{aligned}$$

where $\alpha_\nu^{(\ell)}(y) = \sum_{i=i_\ell+1}^{i_{\ell+1}-1} \chi_i(y) \Big(([\mathscr{L}]_\ell\kappa)(\zeta_i)^T \hat{A}_\ell^{-1} \Big)_\nu$.

From $(\mathscr{L}_{j'}\kappa)(\zeta_i) = 0,\ 1 \le i < i_1$, we see that

$$\begin{aligned}
L_{m'}^{\psi}[\mathscr{L}_{j'}\kappa](y) &= \sum_{\ell=1}^{k}\left\{(\mathscr{L}_{j'}\kappa)(\zeta_{i_\ell})\chi_{i_\ell}(y) + \sum_{v=1}^{\ell}(\mathscr{L}_{j'}\kappa)(\zeta_{i_v})\alpha_v^{(\ell)}(y)\right\}\\
&= \sum_{\ell=1}^{k}(\mathscr{L}_{j'}\kappa)(\zeta_{i_\ell})\chi_{i_\ell}(y) + \sum_{v=1}^{k}\sum_{\ell=v}^{k}(\mathscr{L}_{j'}\kappa)(\zeta_{i_v})\alpha_v^{(\ell)}(y)\\
&= \sum_{\ell=1}^{k}(\mathscr{L}_{j'}\kappa)(\zeta_{i_\ell})q_\ell(y) = (\mathscr{L}_{j'}\kappa)([\zeta]_k)^T q(y),
\end{aligned}$$

where $q \in \mathbb{R}^k$ is the vector with components $q_\ell(y) = \chi_{i_\ell}(y) + \sum_{v=\ell}^{k}\alpha_\ell^{(v)}(y)$. Using Lemma 3.5 we obtain

$$\begin{aligned}
(\mathscr{L}_j r_k)(y) &= (\mathscr{L}_j\kappa)(y) - ([\mathscr{L}]_k\kappa)(y)^T\hat{A}_k^{-1}(\mathscr{L}_j\kappa)([\zeta]_k)\\
&= E_{m'}^{\psi}[\mathscr{L}_j\kappa](y) + (\mathscr{L}_j\kappa)([\zeta]_k)^T q(y) - ([\mathscr{L}]_k\kappa)(y)^T\hat{A}_k^{-1}(\mathscr{L}_j\kappa)([\zeta]_k)\\
&= E_{m'}^{\psi}[\mathscr{L}_j\kappa](y) - \big(([\mathscr{L}]_k\kappa)(y) - \hat{A}_k^T q(y)\big)^T\hat{A}_k^{-1}(\mathscr{L}_j\kappa)([\zeta]_k)\\
&= E_{m'}^{\psi}[\mathscr{L}_j\kappa](y) - \sum_{\ell=1}^{k}E_{m'}^{\psi}[\mathscr{L}_{j_\ell}\kappa](y)\big(\hat{A}_k^{-1}(\mathscr{L}_j\kappa)([\zeta]_k)\big)_\ell.
\end{aligned}$$

According to Cramer's rule

$$\big(\hat{A}_k^{-1}(\mathscr{L}_j\kappa)([\zeta]_k)\big)_\ell = \frac{\det\hat{A}_k(\ell,j)}{\det\hat{A}_k}.$$

The assertion follows. □

3.4 Choice of Pivots

We are now able to control the approximation error by estimating the coefficients in (23). In general, the choice of j_k in Algorithm 3.1 does not produce a submatrix of maximum determinant in modulus. However, we can see from Lemma 3.4 that the maximum element strategy is the best possible choice with respect to maximum determinants if we keep all previously chosen indices $j_1, \ldots, j_{k-1}$ fixed.

Lemma 3.7. *Assume that in each step we choose j_k so that*

$$|(\mathscr{L}_{j_k}r_{k-1})(\zeta_{i_k})| \ge |(\mathscr{L}_j r_{k-1})(\zeta_{i_k})| \quad \text{for all } 1 \le j \le n.$$

Then for $1 \le \ell \le k$ and $j = 1, \ldots, n$ *it holds that*

$$|\det\hat{A}_k(\ell,j)| \le 2^{k-\ell}|\det\hat{A}_k|. \tag{24}$$

Proof. Apply Lemma 3.4. For details see [1].

Thus we obtain

$$|(\mathscr{L}_j r_k)(y)| \leq (1+2^k) \sup_{y \in X_{t_2}} |E^{\psi}_{m'}(\mathscr{L}_j \kappa)(y)|. \tag{25}$$

It is well known that elements may grow during a column-pivoted *LU* decomposition. The term 2^k in (25) is therefore not a consequence of overestimation.

3.5 Error Estimates

In order to estimate the error of our interpolation and also, according to Lemma 3.2, the error of matrix approximation, we have to specify the system of functions $\psi_1, \ldots, \psi_{m'}$ so that the interpolation error for it is known. The easiest choice are the monomials

$$\psi_{\mathbf{i}}(x) = x^{\mathbf{i}} = x_1^{i_1} \cdots x_d^{i_d}, \quad \mathbf{i} \in \mathbb{N}_0^d \text{ with } \|\mathbf{i}\|_\infty \leq p-1.$$

Accordingly, we choose $m' = p^d$ as the dimension of the spanned space Ψ. The set of points $\{\zeta_1, \ldots, \zeta_{m'}\}$ from the construction of $\{s_k\}_k$ is chosen to be the tensor-product

$$\zeta_{\mathbf{i}} = (\xi_{i_1}, \ldots, \xi_{i_d}) \in \mathbb{R}^d, \quad \mathbf{i} \in \mathbb{N}^d \text{ with } \|\mathbf{i}\|_\infty \leq p,$$

of the zeros of Chebyshev polynomials on $[-a, a]$

$$\xi_\nu = a \cdot \cos\left(\frac{\pi}{2} \frac{2\nu - 1}{p}\right), \quad \nu = 1, \ldots, p.$$

The uniqueness of interpolation is inherited from one-dimensional interpolation, so the condition $\det[\psi_j(\zeta_i)]_{ij} \neq 0$ from (22) is fulfilled. The polynomial $L_p f \in \Pi_{p-1}[-a, a]$ interpolating a function $f \in C^p[-a, a]$ at the points ξ_ν, $\nu = 1, \ldots, p$, satisfies, cf. [12],

$$\|f - L_p f\|_\infty \leq \frac{a^p}{2^{p-1}} \frac{\|f^{(p)}\|_\infty}{p!} \quad \text{and} \quad \|L_p f\|_\infty \leq c \log p \, \|f\|_\infty. \tag{26}$$

For multivariate functions $f : [-a, a]^d \to \mathbb{R}$ we use the tensor-product interpolation polynomial $\mathbf{L}_{m'}[f]$

$$\mathbf{L}_{m'}[f] = L_p^{(1)} \cdots L_p^{(d)} f \in \Psi.$$

Then using standard tensor-product arguments we obtain with (26)

$$\|f - \mathbf{L}_{m'}[f]\|_\infty \leq \tilde{c}_p \, a^p \max_{\ell=1,\ldots,d} \|\partial_\ell^p f\|_\infty, \quad \tilde{c}_p = \frac{1 + d(c \log p)^{d-1}}{2^{p-1} p!}. \tag{27}$$

We are now ready to estimate the remainder $\hat{R}_k$ in Algorithm 3.1. To this end we remove the virtual points ζ_i by letting $S_k \in \mathbb{R}^{m \times n}$ be the last m rows of $\hat{S}_k$, i.e.

$$S_k = \sum_{\ell=1}^{k} u_\ell v_\ell^T,$$

where $u_\ell \in \mathbb{R}^m$ are the last m entries of $\hat{u}_\ell$.

Theorem 3.8. *Let* (X_{t_1}, X_{t_2}) *fulfil condition* (15) *and* κ *be an asymptotically smooth kernel. In the case of collocation matrices*

$$a_{ij} = \int_D \kappa(x, y_i)\varphi_j(x)\,\mathrm{d}s_x, \quad i = 1, \ldots, m,\ j = 1, \ldots, n,$$

with supp $\varphi_j \subset X_{t_1}$, $\|\varphi_j\|_\infty = 1$ *and* $y_i \in X_{t_2}$ *it holds that*

$$|(R_k)_{ij}| \le C_p \operatorname{dist}^g(X_{t_1}, X_{t_2})|X_{t_1}|\eta^p, \quad 0 < \eta < \frac{1}{4\sqrt{d}}, \tag{28}$$

where $R_k = A - S_k$.

Proof. We can find a cube Q_a having side length $a = 2 \operatorname{diam} X_{t_2}$ such that $X_{t_2} \subset Q_a$, for which we may assume that $Q_a = \{x \in \mathbb{R}^d : \|x\|_\infty \le a\}$. It is easy to check that

$$2 \operatorname{dist}(X_{t_1}, Q_a) \ge \operatorname{dist}(X_{t_1}, X_{t_2}) \quad \text{for } \eta < \frac{1}{4\sqrt{d}}.$$

From this it follows that $a \le 4\eta \operatorname{dist}(X_{t_1}, Q_a)$. By assumption (see (7)), the derivatives of κ are bounded on $X_{t_1} \times Q_a$:

$$\sup_{y \in Q_a} \|(\partial_y^\alpha \kappa)_y\|_{L^\infty(X_{t_1})} \le c_p \operatorname{dist}^{g-p}(X_{t_1}, Q_a), \quad |\alpha| = p.$$

According to (27) we have

$$\|E_{m'}[\mathscr{L}_j\kappa]\|_{Q_a} \le \tilde{c}_p\, a^p \max_{\ell=1,\ldots,d} \|\partial_\ell^p \mathscr{L}_j\kappa\|_{Q_a}.$$

The derivatives of $\mathscr{L}_j\kappa$ can be estimated by

$$\begin{aligned}
\|\partial_{y_\ell}^p \mathscr{L}_j\kappa\|_{Q_a} = \|\mathscr{L}_j\partial_{y_\ell}^p \kappa\|_{Q_a} &= \sup_{y \in Q_a}\left|\int_\Gamma \partial_{y_\ell}^p \kappa(x,y)\varphi_j(x)\mathrm{d}s_x\right| \\
&\le |X_{t_1}| \sup_{y \in Q_a} \|(\partial_{y_\ell}^p \kappa)_y\|_{L^\infty(X_{t_1})} \\
&\le c_p |X_{t_1}| \operatorname{dist}^{g-p}(X_{t_1}, Q_a)
\end{aligned}$$

and thus

$$\|E_{m'}[\mathscr{L}_j\kappa]\|_{Q_a} \leq \tilde{c}_p c_p |X_{t_1}| \operatorname{dist}^g(X_{t_1}, Q_a)(4\eta)^p. \tag{29}$$

Using (23) and (25) we are finally led to

$$|(R_k)_{ij}| \leq C_p |X_{t_1}| \operatorname{dist}^g(X_{t_1}, X_{t_2})\eta^p,$$

where $C_p = \tilde{c}_p c_p 2^{2p-g}(1 + 2^{m'})$. $\square$

Finally, we will estimate how a prescribed accuracy $\varepsilon > 0$ for the approximation error $\|A - \tilde{A}\|_F < \varepsilon$ affects the cost of the algorithm. For an increasing p the term C_p in (28) grows faster than η^p. Hence, we have to keep p and therefore the rank k constant and control the error by η. Theorem 3.8 states that for each block $M \in \mathbb{R}^{m\times n}$ the approximant $\tilde{M}$ satisfies

$$\|M - \tilde{M}\|_F \leq C_p \sqrt{mn} \operatorname{dist}^g(X_{t_1}, X_{t_2})\eta^p$$

Since (X_{t_1}, X_{t_2}) fulfils (15) and $\operatorname{diam} X_{t_2} \geq h/c_u$ we obtain $\operatorname{dist}^g(X_{t_1}, X_{t_2}) \leq c\eta^g h^g$. Thus $\|A - \tilde{A}\|_F \leq cC_p N h^g \eta^{p+g}$. Setting $\eta^{p+g} = \varepsilon/(cC_p N h^g)$ we get $\|A - \tilde{A}\|_F \leq \varepsilon$. Note that from (13) it follows that $h^{d-1} \sim N^{-1}$. Hence, the overall complexity $\mathcal{O}(\eta^{-(d-1)} N \log N)$ calculated at the end of Section 2 reads $\mathcal{O}(\varepsilon^{-\alpha} N^{1+\alpha} \log N)$ for any $\alpha > 0$.

4 Numerical Experiments

Algorithm 3.1 may be stopped if the approximation reaches a certain accuracy. For this purpose the error estimator from [1] can be used. It is based on the idea that R_{p^d} is approximated by $\sum_{\ell=p^d}^{(p+1)^d} u_\ell v_\ell^T$ and that the latter can be evaluated efficiently.

4.1 Implementation Aspects

In this section we discuss two possible implementations of Algorithm 3.1. For computational purposes it is possible to use collocation points y_i for the interpolation points ζ_i, i.e. for practice we do not have to extend the block B to $\hat{B}$ in Algorithm 3.1. Each of the following algorithms produces vectors $u_\ell \in \mathbb{R}^m$ and $v_\ell \in \mathbb{R}^n$, $\ell = 1, \ldots, k$, from which the approximant S_k can be formed:

$$S_k = \sum_{\ell=1}^{k} u_\ell v_\ell^T.$$

We call the following algorithm *fully pivoted ACA*, since in each step the whole error matrix R_k is inspected for its maximal entry.

Algorithm 4.1. (**fully pivoted ACA**).

```
$R_0 := B;\ k := 0$
repeat
        $k := k+1$
        $(i_k, j_k) = \mathrm{argmax}_{i,j}|(R_{k-1})_{i,j}|;$
        $u_k = R_{k-1}e_{j_k};\ v_k = R_{k-1}^T e_{i_k};$
        $\gamma_k = \left((R_{k-1})_{i_k,j_k}\right)^{-1};$
        $R_k = R_{k-1} - \gamma_k u_k v_k^T$
until *the stopping criterion* (30) *is fulfilled*
```

Algorithm 4.1 may be stopped at step k if for a given $\varepsilon > 0$ the relative accuracy ε is reached:

$$\|R_k\|_F \le \varepsilon \|B\|_F. \tag{30}$$

Hence, the number of operations required to generate the approximant is $\mathcal{O}(k\ m\ n)$. Memory requirements for the algorithm are $\mathcal{O}(nm)$. Thus the algorithm is rather expensive and cannot be used for large matrices.

If the system matrix A has not yet been generated but there is a possibility of generating its entries individually then the following *partially pivoted ACA method* can be used for the approximation.

Algorithm 4.2 (**partially pivoted ACA**).

```
$i_1 = 1;\ \ Z = \emptyset$
repeat                                              //*find first non-zero row in B*
        for $j = 1, \ldots, n$ do $(\tilde{v}_1)_j := b_{i_1 j}$
        if $\tilde{v}_1$ *vanishes* then begin
            $Z := Z \cup \{i_1\};\ i_1 := i_1 + 1$        //*in Z the vanishing rows of $R_k$ are collected*
        end
until $\tilde{v}_1$ *does not vanish or* $i_1 > m$
if $i_1 > m$ then *exit*
$j_1 := \mathrm{argmax}_{j=1,\ldots,n}|b_{i_1 j}|;\ \gamma_1 = (\tilde{v}_1)_{j_1}^{-1};\ v_1 = \gamma_1 \tilde{v}_1$
for $i = 1, \ldots, m$ do $(u_1)_i = b_{ij_1}$
$k := 2$
```

```
repeat
    $i_k := \mathrm{argmax}_{i \notin Z} |(u_{k-1})_i|$
            $(\tilde{v}_k)_j := b_{i_k j} - \sum_{\ell=1}^{k-1} (u_\ell)_{i_k} (v_\ell)_j, \quad j = 1, \ldots, n$
    $Z := Z \cup \{i_k\}$
    if $\tilde{v}_k$ does not vanishes then begin
        $j_k := \arg\max_{j=1,\ldots,n} |(\tilde{v}_k)_j|; \quad \gamma_k := (\tilde{v}_k)_{j_k}^{-1}; \; v_k := \gamma_k \tilde{v}_k$
            $(u_k)_i := b_{i j_k} - \sum_{\ell=1}^{k-1} (u_\ell)_i (v_\ell)_{j_k}, \quad i = 1, \ldots, m$
        $k := k + 1$
    end
until the stopping criterion (32) is fulfilled.
```

With regard to stopping criteria, the following considerations can be made. Since the block B will not be generated completely, only the norm of the approximant S_k can be used instead. Its value can be recursively computed in the following way:

$$\|S_k\|_F^2 = \|S_{k-1}\|_F^2 + 2 \sum_{j=1}^{k-1} u_k^T u_j v_j^T v_k + \|u_k\|_F^2 \|v_k\|_F^2. \tag{31}$$

An appropriate stopping criterion is to terminate the iteration, if for a given $\varepsilon > 0$ at step k it holds that

$$\|u_k\|_F \|v_k\|_F \leq \varepsilon \|S_k\|_F. \tag{32}$$

The amount of numerical work required by Algorithm 4.2 is of order $k^2(m + n)$.

The last two algorithms can also be applied to collocation matrices for which the kernel κ is degenerate but not asymptotically smooth.

Lemma 4.3. *rank* $R_{k+1} =$ *rank* $R_k - 1$.

Proof. Without loss of generality we may assume that $i_k = 1$. Hence,

$$R_{k+1} = R_k - \frac{1}{(R_k)_{11}} R_k e_1 e_1^T R_k = \begin{bmatrix} 0 & \ldots & 0 \\ \vdots & \tilde{R}_k & \\ 0 & & \end{bmatrix}$$

with an $(m - 1) \times (n - 1)$ matrix $\tilde{R}_k$, the rank of which equals the rank of R_{k+1}. On the other hand

$$R_k \begin{bmatrix} 1 & & & \\ -\frac{(R_k)_{21}}{(R_k)_{11}} & 1 & & \\ \vdots & & \ddots & \\ -\frac{(R_k)_{m1}}{(R_k)_{11}} & & & 1 \end{bmatrix} = \begin{bmatrix} (R_k)_{11} & \dots & (R_k)_{1n} \\ 0 & & \\ \vdots & \tilde{R}_k & \\ 0 & & \end{bmatrix}$$

gives rank $R_k = \text{rank } \tilde{R}_k + 1$. □

Hence, if rank $B = r$, then both algorithms will reproduce B in r steps, i.e. $S_r = B$.

4.2 Numerical Results

We first apply the algorithm to a family of surfaces converging to the unit sphere. This sequence is generated by recursive refinement of the icosahedron dividing each of the surface triangles in four and projecting the new knots to the unit sphere as shown in Figure 1.

The following numerical tests were performed for the boundary integral formulation

$$\mathscr{A}v = \left(\frac{1}{2}\mathscr{I} + \mathscr{B}\right)f,$$

where

$$(\mathscr{A}\varphi)(x) = \int_{\partial\Omega} s(x,y)\varphi(y)\, \mathrm{d}s_y \quad \text{and} \quad (\mathscr{B}\varphi)(x) = \int_{\partial\Omega} \partial_{n_y} s(x,y)\varphi(y)\, \mathrm{d}s_y,$$

of the Dirichlet problem for Laplace's equation

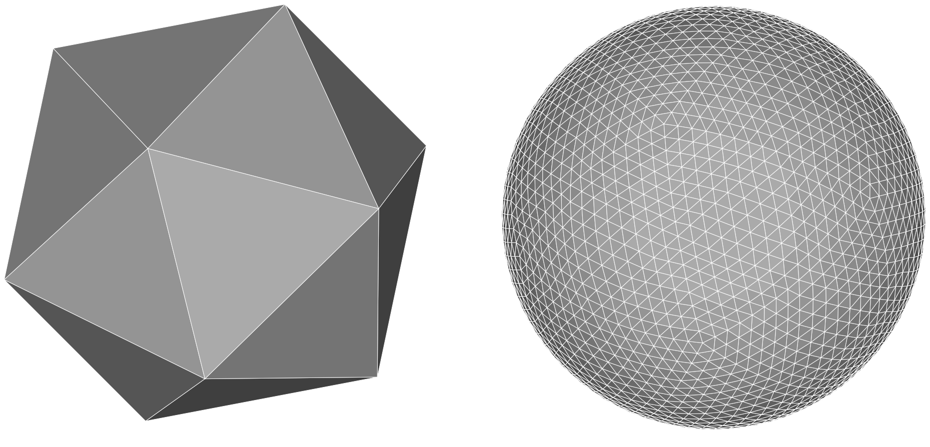

Fig. 1. Icosahedron ($n = 20$) and its refinement ($n = 5120$)

$$\Delta u = 0 \text{ in } \Omega, \tag{33a}$$

$$u = f \text{ on } \partial\Omega \tag{33b}$$

using collocation with piecewise constant ansatz functions. In the above the function s is the so-called singularity function $s(x,y) = \frac{1}{4\pi}|x-y|^{-1}$.

Table 1 shows the compression factors for different problem sizes, i.e., the ratios of the amount of storage needed when using the approximant and the amount of storage for the original matrix, for the single layer and double potential matrix, the number of iterations when using unpreconditioned GMRES and the accuracy

$$\left(\sum_{\pi\in\Pi_h} |\pi||\partial_n s(x_0, m_\pi) - v_h(\pi)|^2\right)^{1/2}, \quad m_\pi \text{ centre of } \pi$$

of the solution v_h. Since we choose $f = s(x_0,\cdot)|_{\partial\Omega}$, $x_0 \notin \overline{\Omega}$, the solution of (33) is known to be $u = s(x_0,\cdot)$.

For the approximation of the blocks we use Algorithm 4.2 and in the stopping criterion (32) ε is chosen to be 10^{-6}, while the relative accuracy of GMRES is 10^{-8}.

An example of the partition of the BEM matrix is presented in Figure 2. Note that the numbering of the columns in the matrix corresponds to the permutation obtained during the construction of the cluster tree, whereas the numbering of the rows is individual for each block due to Algorithm 2.1. The grey scale in Figure 2 indicates the quality of the approximation of the block as a percentage. Thus the big light blocks are very well approximated while the compression of the small dark blocks is either not possible or the compression rate is low.

In the remaining tests the aim is to compare different methods for the generation of low-rank approximants for the following mesh which consists of $n = 19712$ elements. This mesh comes from the TEAM 10 benchmark problem (see [11]) frequently used in the computational electrodynamics community. The speciality of this multiply connected mesh is an extremely thin split in the middle and mesh refinement on the edges.

Table 2 shows the numerical results. For the relative accuracy in each case $\varepsilon = 10^{-4}$ is used. Generating the whole matrix without approximation would lead to 2964.5 MB of storage.

A further example is the kernel of the radiation heat transfer operator for convex domains

$$\kappa(x,y) = \frac{1}{4\pi}\frac{(x-y,n_x)(x-y,n_y)}{|x-y|^4},$$

which is not asymptotically smooth in any variable, but can be approximated by a degenerate kernel. Though this kind of kernel is not covered by our theory, the

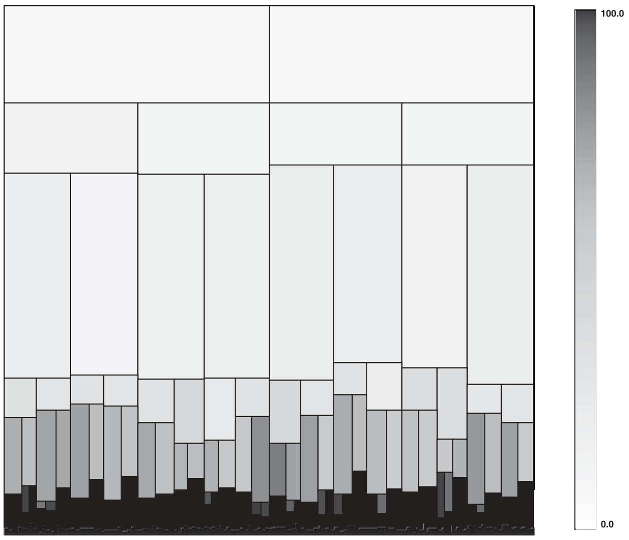

Fig. 2. Block structure of the matrix for $n = 1280$

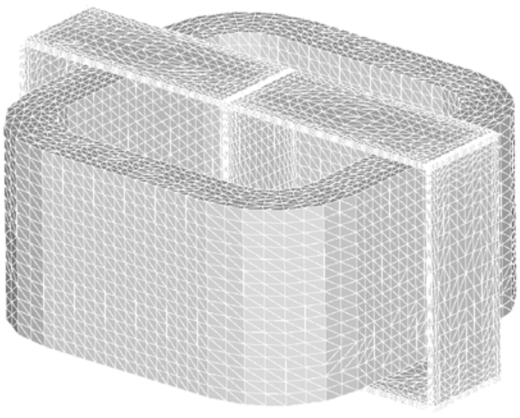

Fig. 3. TEAM 10 problem ($n = 19712$)

Table 1. Numerical results for the first example

n	single layer	double layer	# It	accuracy
80	100 %	100 %	14	0.791e-2
320	96 %	100 %	19	0.297e-2
1280	57 %	64 %	24	0.927e-3
5120	25 %	27 %	28	0.268e-3
20480	9 %	10 %	34	0.796e-4
81920	3 %	3 %	39	0.263e-4

Table 2. Numerical results for the second example

	single layer		double layer		CPU-time
SVD	199.11 MB	6.72 %	310.98 MB	10.49 %	100 h
ACA full	277.54 MB	9.36 %	410.29 MB	13.84 %	20.5 h
ACA partial	242.46 MB	8.18 %	376.19 MB	12.69 %	10 min

algorithms seem to work well. Some numerical examples can be found in our previous article [4].

Acknowledgement

The authors wish to thank Dr. S. Kurz, Bosch GmbH, Stuttgart for providing the mesh for the second numerical example.

References

[1] Bebendorf, M.: Approximation of boundary element matrices. Numer. Math. *86*, 565–589 (2000).
[2] Bebendorf, M.: Effiziente numerische Lösung von Randintegralgleichungen unter Verwendung von Niedrigrang-Matrizen. dissertation.de, Verlag im Internet, 2001. ISBN 3-89825-183-7.
[3] Bebendorf, M., Hackbusch W.: Existence of $\mathcal{H}$-Matrix Approximants to the Inverse FE-Matrix of Elliptic Operators with L^∞-Coefficients. Technical Report 21, Max-Planck-Institute, Leipzig, 2002. to appear in Numer. Math.
[4] Bebendorf, M., Rjasanow, S.: Matrix Compression for the Radiation Heat Transfer in Exhaust Pipes. In: Multifield Problems: state of the art, A.-M. Sändig, W. Schielen, W. L. Wendland (eds.), pp. 183–192. Springer Verlag, 2000.
[5] Cheng, H., Greengard, L., Rokhlin, V.: A fast adaptive multipole algorithm in three dimensions. J. Comput. Phys. *155(2)*, 468–498 (1999).
[6] Goreinov, S. A., Tyrtyshnikov, E. E., Zamarashkin, N. L.: A theory of pseudoskeleton approximations. Linear Algebra Appl. *261*, 1–21 (1997).
[7] Hackbusch, W.: A sparse matrix arithmetic based on $\mathcal{H}$-matrices. I. Introduction to $\mathcal{H}$-matrices. Computing *62(2)*, 89–108 (1999).
[8] Hackbusch, W., Khoromskij, B. N.: A sparse $\mathcal{H}$-matrix arithmetic. II. Application to multi-dimensional problems. Computing *64(1)*, 21–47 (2000).
[9] Hackbusch, W., Nowak, Z. P.: On the fast matrix multiplication in the boundary element method by panel clustering. Numer. Math. *54(4)*, 463–491 (1989).
[10] Nabors, K., Phillips, J., Korsmeyer, F. T., White J.: Multipole and precorrected-FFT accelerated iterative methods for solving surface integral formulations of three-dimensional Laplace problems. In: Domain-based parallelism and problem decomposition methods in computational science and engineering, pp. 193–215. SIAM, Philadelphia, PA: 1995.
[11] Nakata, T., Takahashi, N., Fujiwara, K.: Summary of results for benchmark problem 10 (steel plates around a coil). COMPEL *11*, 335–344 (1992).
[12] Schönhage, A.: Approximationstheorie. de Gruyter, Berlin 1971.

M. Bebendorf
Max-Planck-Institute for Mathematics
in the Sciences Inselstraße 22–26
04103 Leipzig, Germany
e-mail: bebendorf@mis.mpg.de

S. Rjasanow
Fachrichtung 6.1 – Mathematik,
Universität des Saarlandes
'Postfach 151150
66041 Saarbrücken, Germany
e-mail: rjasanow@num.uni-sb.de