ELSEVIER

Engineering Analysis with Boundary Elements 27 (2003) 405–422

ENGINEERING ANALYSIS *with* BOUNDARY ELEMENTS

www.elsevier.com/locate/enganabound

Introduction to hierarchical matrices with applications

Steffen Börm[a,*], Lars Grasedyck[b], Wolfgang Hackbusch[a]

[a]*Max Planck Institute for Mathematics in the Sciences, Inselstrasse 22-26, Leipzig 04103, Germany*
[b]*Lehrstuhl Praktische Mathematik, University Kiel, Kiel, Germany*

Received 31 January 2002; revised 4 April 2002; accepted 9 May 2002

Abstract

We give a short introduction to methods for the data-sparse approximation of matrices resulting from the discretisation of non-local operators occurring in boundary integral methods, as the inverses of partial differential operators or as solutions of control problems.

The result of the approximation will be so-called *hierarchical matrices* (or short $\mathcal{H}$-matrices). These matrices form a subset of the set of all matrices and have a data-sparse representation. The essential operations for these matrices (matrix-vector and matrix–matrix multiplication, addition and inversion) can be performed in, up to logarithmic factors, optimal complexity.

We give a review of specialised variants of $\mathcal{H}$-matrices, especially of $\mathcal{H}^2$-matrices, and finally consider applications of the different methods to problems from integral equations, partial differential equations and control theory.

Keywords: 65F05; 65F30; 65F50; 65N50

1. Introduction

1.1. Overview

$\mathcal{H}$-matrices are based on two observations:

- Integral operators can be efficiently treated by using separable expansions of the corresponding kernel functions [20,26].
- The inverse of an elliptic partial differential operator can be cast in the form of an integral operator by using Green's function as Schwartz kernel.

This article consists of seven sections: the present section gives a short over-view of the basic ideas. Section 2 will establish *cluster trees* and *block partitions* that play a crucial role in the subsequent approximation schemes. Section 3 is concerned with operations on low-rank matrices, the basic building blocks of $\mathcal{H}$-matrices. In Section 4, we introduce the original $\mathcal{H}$-matrices and a set of algorithms for performing basic algebraic operations on them. Section 5 is concerned with a specialisation of $\mathcal{H}$-matrices, namely uniform $\mathcal{H}$-matrices and $\mathcal{H}^2$-matrices, which can be used to significantly improve the performance and reduce memory requirements, especially for applications in the field of integral equations. Other specialised variants of $\mathcal{H}$-matrices are described in Section 6. Section 7 presents several applications of $\mathcal{H}$- and $\mathcal{H}^2$-matrices together with numerical results.

1.2. Model problem: integral equation

Let us consider an integral operator of the form

$$\mathcal{L}[u](x) = \int_\Omega g(x,y)u(y)\mathrm{d}y \tag{1}$$

on a submanifold or subdomain Ω of $\mathbb{R}^d$ with a kernel function

$$g : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}.$$

In typical applications, g is non-local, so, contrary to the treatment of differential operators, the finite element discretisation of the operator $\mathcal{L}$ does not lead to a sparse matrix. Due to the lack of sparsity, operations on the discrete matrix are prohibitively expensive.

There are different methods for avoiding the necessity of working with the full matrix: for some domains and some

* Corresponding author.
E-mail address: sbo@mis.mpg.de (S. Börm).

0955-7997/03/$ - see front matter © 2003 Elsevier Science Ltd. All rights reserved.
doi:10.1016/S0955-7997(02)00152-2

operators, it may be possible to diagonalise the matrix by applying the fast Fourier transform. In a more general setting, it may be possible to approximate the integral operator by means of a Wavelet basis.

$\mathcal{H}$-matrices are based on the fact that, at least for typical kernel functions $g(\cdot,\cdot)$, singularities only occur at the diagonal and the function is smooth everywhere else. In order to describe this property precisely, we introduce the notion of *asymptotic smoothness*: the kernel function $g(\cdot,\cdot)$ is called *asymptotically smooth*, if there are constants $C_{\mathrm{as1}}, C_{\mathrm{as2}} \in \mathbb{R}_{>0}$ satisfying

$$|\partial_x^\alpha \partial_y^\beta g(x,y)| \leq C_{\mathrm{as1}}(C_{\mathrm{as2}}\|x-y\|)^{-|\alpha|-|\beta|}\alpha+\beta|g(x,y)| \quad (2)$$

for all multi-indices $\alpha, \beta \in \mathbb{N}_0^{\mathrm{d}}$ and all $x, y \in \mathbb{R}^{\mathrm{d}}$ with $x \neq y$.

The idea of the *panel clustering method* [20] is to exploit this smoothness in order to replace $g(\cdot,\cdot)$ by an approximation: if we fix $x \in \Omega$ and a subset ω of Ω satisfying

$$\mathrm{dist}(x,\omega) := \inf\{\|x-y\| : y \in \omega\} \geq C\mathrm{diam}(\omega),$$

we can replace $g(x,y)$ by its truncated Taylor expansion at a point $y_0 \in \omega$ in order to get the *degenerate approximation*

$$\tilde{g}(x,y) = \sum_{\nu=1}^{k} p_\nu^x q_\nu(y). \quad (3)$$

If we use a collocation scheme, this translates into a *low-rank approximation* of part of the row corresponding to the collocation point x. By organising the sets ω in a hierarchical way, a data-sparse approximation of the full matrix can be constructed.

A closer investigation shows that the coefficients p_ν^x can be used for a neighbourhood of x without reducing the approximation properties. This leads to schemes based on low-rank approximations of blocks of the matrix [3,14,26], that we will call $\mathcal{H}$-matrix schemes.

These methods can be refined by choosing different degenerate approximations, leading to multipole expansions, uniform $\mathcal{H}$-matrix or $\mathcal{H}^2$-matrix techniques.

1.3. Elliptic partial differential equations

Neglecting boundary conditions, we can represent the inverse of an elliptic partial differential operator by an integral operator of the form (1). This suggests that it will be possible to apply the $\mathcal{H}$-matrix technique to these inverses and even, since we assume that a finite element discretisation scheme will give us a good approximation of the infinite-dimensional operator, to store the inverse of the matrix resulting from a finite element discretisation in the form of an $\mathcal{H}$-matrix.

Since the 'real' kernel function, i.e. Green's function, will depend on the (variable) coefficients and on the shape of the boundary in a complicated way, it is not feasible to discretise it directly in order to find an approximation of the inverse. Instead, we will introduce approximations of basic matrix operations like addition and multiplication in Section 4.4 and derive an approximative algorithm for the computation of the inverse of the FE stiffness matrix.

1.4. Matrix equations

The approximative matrix operations, namely addition, multiplication and inversion, introduced in Section 4.4 can replace the corresponding standard operations in many algorithms working on matrices.

We can approximate matrix functions like $\exp(L)$ by using the approximative matrix–matrix multiplication in combination with a Taylor expansion of the function or by using the approximative inversion in combination with Dunford–Cauchy integrals [6].

We can also use the approximative matrix arithmetic to improve the performance of Newton's method for solving nonlinear matrix equations like Riccati's equation arising, e.g., in control theory (Section 7.4).

2. Construction of the cluster tree and block partition

While wavelet techniques can be employed to deal directly with problems in a continuum, $\mathcal{H}$-matrix techniques require a discrete subspace together with the finite element or boundary element basis $(\varphi_i)_{i\in I}$. The corresponding Ritz–Galerkin matrix (stiffness matrix) L is given by

$$L_{ij} = \langle \varphi_i, \mathcal{L}\varphi_j \rangle_{L^2}. \quad (4)$$

In the following, we identify subblocks $\tau \times \sigma \subset I \times I$ such that the submatrix $(L_{ij})_{i\in\tau, j\in\sigma}$ allows a low rank approximation. Since the number of possible subsets of $I \times I$ is considerably large we restrict ourselves to a small set of candidates τ, σ that are constructed and organised in a tree structure.

2.1. Cluster tree

Let $\mathcal{T}_I$ be a tree and denote by T_I the set of its nodes. $\mathcal{T}_I$ is called a *cluster tree* corresponding to an index set I, if the following conditions hold:

1. $T_I \subseteq \mathcal{P}(I)\backslash\{\emptyset\}$, i.e. each node of $\mathcal{T}_I$ is a subset of the index set I. Here, $\mathcal{P}(I)$ denotes the set of all subsets of I.
2. I is the root of $\mathcal{T}_I$.
3. If $\tau \in T_I$ is a leaf, then $|\tau| \leq C_{\mathrm{leaf}}$, i.e. the leaves consist of a relatively small number of indices (here, $|\tau|$ denotes the number of elements in the set τ).
4. If $\tau \in T_I$ is *not* a leaf, then it has two sons and is their disjoint union.

For each $\tau \in T_I$, we denote the set of its sons by $S(\tau) \subseteq T_I$. The leaves of the tree $\mathcal{T}_I$ are denoted by $\mathcal{L}_I$.

The restriction of $\mathcal{T}_I$ to binary trees serves only the purpose of simplifying the presentation of some steps of the algorithms. The extension to more general trees is straightforward.

The *support* of a cluster $\tau \in T_I$ is given by the union of the supports of the basis functions corresponding to its elements, i.e.

$$\Omega_\tau := \bigcup_{i\in\tau} \Omega_i, \qquad \text{where } \Omega_i := \operatorname{supp} \varphi_i \text{ for all } i \in I.$$

Example 2.1 (Construction of cluster trees by bisection). A simple method of building a cluster tree is based on geometry-based splittings of the index set. We associate each degree of freedom $i \in I$ with a suitable point $x_i \in \mathbb{R}^d$, e.g., the centre of the support of the corresponding basis function or the corresponding Lagrange point, if Lagrangian finite elements are used.

Let $\{e_1, \ldots, e_d\}$ be an orthonormal basis of $\mathbb{R}^d$, e.g. the basis $\{e_x, e_y, e_z\}$ of the canonical unit vectors in 3D. The following algorithm will split a given cluster $\tau \subseteq I$ into two sons such that the points x_i are separated by a hyper-plane (Fig. 1).

```
procedure Split(τ, var τ1, var τ2);
begin
  {Choose a direction for geometrical splitting of the
  cluster τ}
  for j := 1 to d do begin
    α_j := min{⟨e_j, x_i⟩ : i ∈ τ}; {⟨·,·⟩ is the R^d Eucli-
    dean product}
    β_j := max{⟨e_j, x_i⟩ : i ∈ τ}
  end;
  j_max := argmax{β_j − α_j : j ∈ {1,...,d}};
  {Split the cluster τ in the chosen direction}
  γ := (α_jmax + β_jmax)/2;
  τ1 := ∅; τ2 := ∅;
  for i ∈ τ do
    if ⟨e_jmax, x_i⟩ ≤ γ then
      τ1 := τ1 ∪ {i}
    else
      τ2 := τ2 ∪ {i};
  end
```

In theory each node of the tree is split into two sons until the cardinality of the node is 1. In practice one should stop the splitting if the cardinality of a node is less or equal to a threshold parameter C_{leaf}. On typical machines, setting $C_{\text{leaf}} = 32$ leads to good performance.

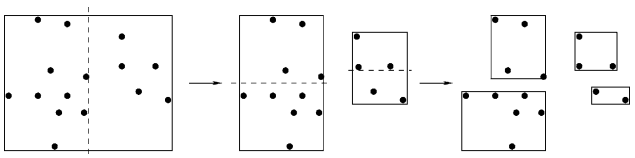


Fig. 1. The bounding box to the left containing the points x_i is divided into two parts in x-direction. In the next step the new bounding boxes are divided in y-direction.

2.2. Admissibility condition

Next, we need an *admissibility condition* that allows us to check if a candidate $(\tau, \sigma) \in T_I \times T_I$ allows for a suitable low rank approximation.

If we assume asymptotically smooth kernels, this requirement will lead to an admissibility condition of the form

$$\min\{\operatorname{diam}(\Omega_\tau), \operatorname{diam}(\Omega_\sigma)\} \le \eta \operatorname{dist}(\Omega_\tau, \Omega_\sigma), \tag{5}$$

where $\eta \in \mathbb{R}_{>0}$ is some parameter controlling the trade-off between the number of admissible blocks, i.e. the algorithmic complexity, and the speed of convergence, i.e. the quality of the approximation [20].

In typical applications for unstructured grids, the computation of the diameter of a cluster and especially of the distance of two clusters will be too complicated or too time-consuming, so the 'minimal' condition (5) will be replaced by a stronger variant, for example, by using super-sets of Ω_τ and Ω_σ, such as bounding sets, that are of a simpler structure.

Example 2.2 (Admissibility by bounding boxes). A relatively general and practical admissibility condition for clusters in $\mathbb{R}^d$ can be defined by using *bounding boxes*. We define the canonical coordinate maps

$$\pi_k : \mathbb{R}^d \to \mathbb{R}, \qquad x \mapsto x_k,$$

for all $k \in \{1, \ldots, d\}$. The bounding box for a cluster $\tau \in T_I$ is then given by

$$Q_\tau := \prod_{k=1}^{d} [a_{\tau,k}, b_{\tau,k}],$$

where $a_{\tau,k} := \min(\pi_k \Omega_\tau)$ and $b_{\tau,k} := \max(\pi_k \Omega_\tau)$.

Obviously, we have $\Omega_\tau \subseteq Q_\tau$, so we can define the admissibility condition

$$\min\{\operatorname{diam}(Q_\tau), \operatorname{diam}(Q_\sigma)\} \le \eta \operatorname{dist}(Q_\tau, Q_\sigma) \tag{6}$$

that implies Eq. (5). We can compute the diameters and distances of the boxes by

$$\operatorname{diam}(Q_\tau) = \left(\sum_{k=1}^{d} (b_{\tau,k} - a_{\tau,k})^2 \right)^{1/2}$$

and

$$\operatorname{dist}(Q_\tau, Q_\sigma) = \left(\sum_{k=1}^{d} (\max(0, a_{\tau,k} - b_{\sigma,k}))^2 + (\max(0, a_{\sigma,k} - b_{\tau,k}))^2 \right)^{1/2}.$$

2.3. Block tree

The cluster tree can be used to define a *block tree* by forming pairs of clusters recursively. The block tree $\mathcal{T}_{I\times I}$

corresponding to a cluster tree $\mathcal{T}_I$ and an admissibility condition is constructed by the following procedure:

```
procedure BuildBlockTree(τ×σ);
begin
  if τ×σ is not admissible and |τ| > C_leaf and |σ| >
  C_leaf then begin
    S(τ×σ) := {τ'×σ' : τ' ∈ S(τ), σ' ∈ S(σ)};
    for τ'×σ' ∈ S(τ×σ) do
    BuildBlockTree(τ'×σ')
  end
  else
    S(τ×σ) := ∅
end
```

By calling this procedure with $\tau = \sigma = I$ and a parameter C_{leaf} that determines the minimal size of a cluster (Definition 4.1), we create a block tree with root $I \times I$. The leaves of the block tree are denoted by $\mathcal{L}_{I\times I}$ and form a partition of $I \times I$.

The suitability of a block tree can be measured by:

Definition 2.3 (Sparsity). We define the sparsity constant C_{sp} of a block tree $\mathcal{T}_{I\times I}$ by

$$C_{\text{sp}} := \max_{\tau\in T_I} |\{\sigma \in T_I;\ \tau\times\sigma \in \mathcal{L}_{I\times I}\}|.$$

A block tree $\mathcal{T}_{I\times I}$ is called sparse if $C_{\text{sp}} = \mathcal{O}(1)$ (does not depend on $|I|$).

The constant C_{sp} is a measure for the sparsity of the block structure imposed by the partitioning $\mathcal{L}_{I\times I}$ of the product index set $I \times I$ (Fig. 2).

The complexity of algorithms for the creation of a cluster tree and block tree has been analysed in detail in Ref. [10]: for typical quasi-uniform grids, a 'good' cluster tree can be created in $\mathcal{O}(n \log n)$ operations, the computation of the block tree can be accomplished in $\mathcal{O}(n)$ operations. Even for arbitrary grids where we only assume that the supports Ω_i of the corresponding basis functions are local and satisfy some weak assumptions, one can generate a sparse block tree in $\mathcal{O}(n \log n)$ operations [11,13]. An alternative algorithm for constructing admissible block-partitions for integral equations is described in Ref. [3]. Further analysis concerning the approximation on graded meshes can be found in Ref. [16].

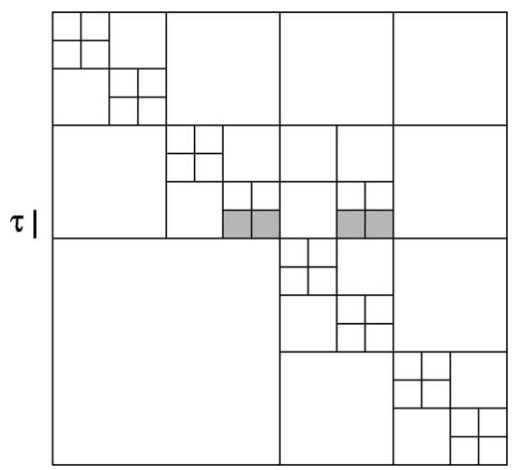


Fig. 2. The maximum of 4 in the definition of C_{sp} is achieved by τ.

3. Rk-matrices

The basic building blocks for $\mathcal{H}$-matrices (defined in Section 4) are **R**k-matrices which are a straightforward representation of low rank matrices. These matrices form subblocks of the $\mathcal{H}$-matrix corresponding to subsets $\tau\times\sigma \subset I\times I$.

Definition 3.1 (**R**k*-matrix*)**.** A matrix of the form

$$R = AB^{\mathrm{T}}, \qquad A \in \mathbb{R}^{\tau\times k},\ B \in \mathbb{R}^{\sigma\times k}$$

is called an **R**k-matrix.

Here, $\tau\times k$ is a short notation for $\{(i,j)|i \in \tau,\ j \in \{1,\ldots,k\}\}$.

Any matrix of rank atmost k can be represented as an **R**k-matrix and each **R**k-matrix has at most rank k. **R**k-matrices have some nice properties, e.g., only $k(n+m)$ numbers are needed to store an **R**k-matrix.

3.1. Discretisation

In the $\mathcal{H}$-matrix representation of matrices, **R**k-matrices will occur only as a representation of admissible blocks.

If $\mathcal{L}$ is a differential operator, we have $\operatorname{supp}(\mathcal{L}\varphi_j) \subseteq \operatorname{supp}\varphi_j$, so the matrix blocks corresponding to admissible pairs of clusters are zero.

The situation is more complicated if $\mathcal{L}$ is an integral operator of the type (1): let $\tau\times\sigma$ be an admissible pair of clusters. Without loss of generality, we may assume that $\operatorname{diam}(\Omega_\tau) \le \operatorname{diam}(\Omega_\sigma)$.

In order to construct a rank-k approximation of the block $\tau\times\sigma$, we use an m-th order interpolation scheme with interpolation points $(x_j^\tau)_{j=1}^k$ and corresponding Lagrange polynomials $(p_j^\tau)_{j=1}^k$ and approximate the original kernel function $g(\cdot,\cdot)$ by its interpolant

$$\tilde{g}(x,y) := \sum_{\iota=1}^{k} p_\iota^\tau(x) g(x_\iota^\tau, y). \tag{7}$$

Combining the asymptotical smoothness assumption (2) with standard interpolation error estimates, we get

$$|g(x,y) - \tilde{g}(x,y)| \le C\left(C_{\text{int}}C_{\text{as2}}\frac{\operatorname{diam}(\Omega_\tau)}{\operatorname{dist}(\Omega_\tau,\Omega_\sigma)}\right)^m \|g\|_{\infty,\Omega_\tau\times\Omega_\sigma},$$

which combined with the admissibility condition (5) reads

$$|g(x,y) - \tilde{g}(x,y)| \leq C(C_{\mathrm{int}}C_{\mathrm{as2}}\eta)^m \|g\|_{\infty,\Omega_\tau\times\Omega_\sigma},$$

so if we choose $\eta < 1/(C_{\mathrm{int}}C_{\mathrm{as2}})$, we get *exponential* convergence of the interpolation if we increase the order m.

By replacing $g(\cdot,\cdot)$ by $\tilde{g}(\cdot,\cdot)$ in Eq. (4), we find

$$L_{ij} \approx \sum_{\iota=1}^{k} \int_\Omega p_\iota^\tau(x)\varphi_i(x)\mathrm{d}x \int_\Omega g(x_\iota^\tau, y)\varphi_j(y)\mathrm{d}y. \tag{8}$$

Notation 3.2. For a vector v and a subset $\tau \subset I$, $v|_\tau$ is the restriction to the vector $(v_j)_{j\in\tau}$, while for a matrix L and subsets τ, $\sigma \subset I$ the notation $L|_{\tau\times\sigma}$ is used for the block $(L_{ij})_{i\in\tau,j\in\sigma}$.

We define matrices $A \in \mathbb{R}^{\tau\times k}$ and $B \in \mathbb{R}^{\sigma\times k}$ by setting

$$A_{i\iota} := \int_\Omega p_\iota^\tau(x)\varphi_i(x)\mathrm{d}x$$

and

$$B_{j\iota} := \int_\Omega g(x_\iota^\tau, y)\varphi_j(y)\mathrm{d}y$$

and rewrite Eq. (8) as

$$L|_{\tau\times\sigma} \approx AB^{\mathrm{T}},$$

so we have approximated $L|_{\tau\times\sigma}$ by an $\mathbf{R}k$-matrix.

Remark 3.3 (Collocation instead of Galerkin techniques). This interpolation-based approach is not limited to Galerkin discretisations. In the case of a collocation scheme, the matrix entries L_{ij}^{Coll} take the form

$$L_{ij}^{\mathrm{Coll}} := (\mathscr{L}\varphi_j)(c_i) = \int_\Omega g(c_i,y)\varphi_j(y)\mathrm{d}y,$$

where $(c_i)_{i\in I}$ is the family of collocation points. Replacing the kernel function $g(\cdot,\cdot)$ by its interpolant $\tilde{g}(\cdot,\cdot)$, we find

$$L_{ij}^{\mathrm{Coll}} \approx \sum_{\iota=1}^{k} p_\iota^\tau(c_i) \int_\Omega g(x_\iota^\tau, y)\varphi_j(y)\mathrm{d}y$$

and can introduce

$$A'_{i\iota} := p_\iota^\tau(c_i)$$

in order to get

$$L^{\mathrm{Coll}}|_{\tau\times\sigma} \approx A'B^{\mathrm{T}}.$$

Remark 3.4 (Double layer potential). The *double layer potential*

$$\mathscr{L}^{\mathrm{DLP}} : [u](x) := \int_\Gamma \langle \nabla_y g(x,y), n(y)\rangle u(y)\mathrm{d}y$$

for a one-codimensional submanifold Γ of $\mathbb{R}^{\mathrm{d}}$ plays an important role in boundary element techniques. Since the effective kernel function depends on the normal vector of the manifold, it is not defined in $\mathbb{R}^{\mathrm{d}}$, so we are not able to establish the estimate Eq. (2) directly.

Instead, we replace $g(\cdot,\cdot)$ by $\tilde{g}(\cdot,\cdot)$ from Eq. (7) and use the normal derivative of the result to get

$$L_{ij}^{\mathrm{DLP}} \approx \sum_{\iota=1}^{k} \int_\Gamma p_\iota^\tau(x)\varphi_i(x)\mathrm{d}x \int_\Gamma \langle \nabla_y g(x_\iota^\tau, y), n(y)\rangle \varphi_j(y)\mathrm{d}y,$$

so we can again find a low-rank approximation

$$L^{\mathrm{DLP}}|_{\tau\times\sigma} \approx AB'^{\mathrm{T}}$$

by setting

$$B'_{j\iota} := \int_\Gamma \langle \nabla_y g(x_\iota^\tau, y), n(y)\rangle \varphi_j(y)\mathrm{d}y.$$

Remark 3.5 (Adaptive low-rank approximation). The complexity can be significantly improved by using *adaptive* low-rank approximations instead of the analytically derived ones given above [1].

3.2. Matrix-vector multiplication

The matrix-vector multiplication $x \mapsto y := Rx$ of an $\mathbf{R}k$-matrix $R = AB^{\mathrm{T}}$ with a vector $x \in \mathbb{R}^\sigma$ can be done in two steps:

1. Calculate $z := B^{\mathrm{T}}x \in \mathbb{R}^k$.
2. Calculate $y := Az \in \mathbb{R}^\tau$.

The transposed $R^{\mathrm{T}} = BA^{\mathrm{T}}$ can be treated analogously and the complexity of the matrix-vector multiplication is $\mathcal{O}(k(|\sigma| + |\tau|)$.

3.3. Truncation

A best approximation of an arbitrary matrix $M \in \mathbb{R}^{\tau\times\sigma}$ by an $\mathbf{R}k$-matrix $\tilde{M} = \tilde{A}\tilde{B}^{\mathrm{T}}$ (w.r.t. the spectral and Frobenius norm) can be computed using the (truncated) singular value decomposition as follows:

1. Calculate a singular value decomposition $M = U\Sigma V^{\mathrm{T}}$ of M.
2. Set $\tilde{U} := [U_1 \cdots U_k]$ (first k columns), $\tilde{\Sigma} := \mathrm{diag}(\Sigma_{11}, \ldots, \Sigma_{kk})$ (first (largest) k singular values), $\tilde{V} := [V_1 \cdots V_k]$ (first k columns).
3. Set $\tilde{A} := \tilde{U}\tilde{\Sigma} \in \mathbb{R}^{\tau\times k}$ and $\tilde{B} := \tilde{V} \in \mathbb{R}^{\sigma\times k}$.

We call $\tilde{M}$ a truncation of M to the set of $\mathbf{R}k$-matrices. The complexity of the truncation is $\mathcal{O}((|\tau| + |\sigma|)^3)$.

If the matrix M is an $\mathbf{R}K$-matrix $M = AB^{\mathrm{T}}$ then the truncation can be computed in $\mathcal{O}(K^2(|\tau| + |\sigma|) + K^3)$ by the following procedure:

1. Calculate a truncated QR-decomposition $A = Q_A R_A$ of A, $Q_A \in \mathbb{R}^{\tau\times K}$, $R_A \in \mathbb{R}^{K\times K}$.
2. Calculate a truncated QR-decomposition $B = Q_B R_B$ of B, $Q_B \in \mathbb{R}^{\sigma\times K}$, $R_B \in \mathbb{R}^{K\times K}$.
3. Calculate a singular value decomposition $R_A R_B^{\mathrm{T}} = U\Sigma V^{\mathrm{T}}$ of $R_A R_B^{\mathrm{T}}$.
4. Set $\tilde{U} := [U_1 \cdots U_k]$ (first k columns), $\tilde{\Sigma} := \mathrm{diag}(\Sigma_{11}, \ldots, \Sigma_{kk})$ (first (largest) k singular values), $\tilde{V} := [V_1 \cdots V_k]$ (first k columns).
5. Set $\tilde{A} := Q_A \tilde{U}\tilde{\Sigma} \in \mathbb{R}^{\tau\times k}$ and $\tilde{B} := Q_B \tilde{V} \in \mathbb{R}^{\sigma\times k}$.

3.4. Addition

Let $R_1 = AB^{\mathrm{T}}$, $R_2 = CD^{\mathrm{T}}$ be $\mathbf{R}k$-matrices. The sum

$$R_1 + R_2 = [A \quad C][B \quad D]^{\mathrm{T}}$$

is an $\mathbf{R}2k$-matrix. We define the *formatted addition* $\oplus$ of two $\mathbf{R}k$-matrices as a best approximation (in the spectral and Frobenius norm) of the sum in the set of $\mathbf{R}k$-matrices, which can be computed as in Section 3.3 with complexity $\mathcal{O}(k^2(|\tau| + |\sigma|) + k^3)$. The *formatted subtraction* $\ominus$ is defined analogously.

3.5. Multiplication

The multiplication of an $\mathbf{R}k$-matrix $R = AB^{\mathrm{T}}$ by an arbitrary matrix M from the left or right yields again an $\mathbf{R}k$-matrix:

$$RM = AB^{\mathrm{T}}M = A(M^{\mathrm{T}}B)^{\mathrm{T}},$$

$$MR = MAB^{\mathrm{T}} = (MA)B^{\mathrm{T}}.$$

To calculate the product one has to perform the matrix-vector multiplication $M^{\mathrm{T}}B_i$ for the k columns $i = 1, \ldots, k$ of B with the transposed of M or MA_i for the k columns $i = 1, \ldots, k$ of A with the matrix M.

4. $\mathscr{H}$-matrices

Based on the cluster (binary) tree $\mathcal{T}_I$ and the block (quad-) tree $\mathcal{T}_{I\times I}$ we define the $\mathscr{H}$-matrix structure.

Definition 4.1 ($\mathscr{H}$-matrix). Let $L \in \mathbb{R}^{I\times I}$ be a matrix and $\mathcal{T}_{I\times I}$ a block tree of $I \times I$ consisting of admissible and non-admissible leaves. Let $k \in \mathbb{N}$. L is called an *$\mathscr{H}$-matrix* of blockwise rank k, if for all admissible leaves $\tau \times \sigma \in T_{I\times I}$

$$\mathrm{rank}(L|_{\tau\times\sigma}) \le k,$$

i.e. each admissible subblock of the matrix can be represented as an $\mathbf{R}K$ - matrix while the non-admissible subblocks corresponding to leaves do not have to bear any specific structure.

Remark 4.2. If $\tau \times \sigma$ is a non-admissible leaf of $\tau_{I\times I}$, then either $|\tau| \le C_{\mathrm{leaf}}$ or $|\sigma| \le C_{\mathrm{leaf}}$ (Section 2.3), which means that the rank is bounded by C_{leaf}.

The storage requirements for an $\mathscr{H}$-matrix are $\mathcal{O}(nk\log(n))$ for the one- and two-dimensional block tree in Refs. [14,17]. The same bound holds for any $\mathscr{H}$-matrix based on a sparse block tree [10,11].

4.1. Matrix-vector multiplication

Let $L \in \mathbb{R}^{I\times I}$ be an $\mathscr{H}$-matrix. To compute the matrix-vector product $y := y + Lx$ with $x, y \in \mathbb{R}^I$, we use the following procedure that performs the matrix-vector multiplication in each leaf of the block tree:

```
procedure MVM(L, τ×σ, x, var y);
begin
  if S(τ×σ) ≠ ∅ then
    for τ'×σ' ∈ S(τ×σ) do MVM(L, τ'×σ', x, y)
  else
    y|_τ := y|_τ + L|_{τ×σ} x|_σ; {unstructured or Rk-matrix}
end
```

The starting index sets are $\tau = \sigma = I$.

The complexity for the matrix-vector multiplication (sparse block tree) is $\mathcal{O}(nk\log(n))$ [10,11]. For some model problems the complexity can be estimated by exploiting the recursive structure as in Refs. [14,17].

4.2. Addition

Let L, $L^{(1)}$, $L^{(2)} \in \mathbb{R}^{I\times I}$ be $\mathscr{H}$-matrices. The sum $L := L^{(1)} + L^{(2)}$ is an $\mathscr{H}$-matrix with blockwise rank $2k$. The *formatted sum* $\tilde{L} := L^{(1)} \oplus L^{(2)}$ is defined by using the formatted addition for the $\mathbf{R}k$-subblocks and the standard addition for unstructured (full) matrices in the non-admissible leaves:

```
procedure Add(var L̃, τ×σ, L(1), L(2));
begin
  if S(τ×σ) ≠ ∅ then
    for τ'×σ' ∈ S(τ×σ) do Add(L̃, τ'×σ', L(1),
    L(2))
  else
    L̃|_{τ×σ} := L(1)|_{τ×σ} ⊕ L(2)|_{τ×σ} {unstructured or Rk-
    matrices}
end
```

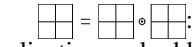
Calling the procedure with $\tau = \sigma = I$ and $\tilde{L} := 0$ yields $\tilde{L} = L^{(1)} \oplus L^{(2)}$.

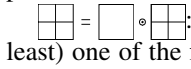
The complexity of the formatted addition (sparse block tree) is $\mathcal{O}(nk^2\log(n))$.

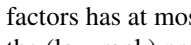
4.3. Multiplication

Let L, $L^{(1)}$, $L^{(2)} \in \mathbb{R}^{I\times I}$ be $\mathscr{H}$-matrices. The matrix $L := L + L^{(1)} \cdot L^{(2)}$ is (under moderate assumptions that are further investigated in Refs. [10,11]) an $\mathscr{H}$-matrix with blockwise

rank $\mathcal{O}(k\log(n))$. The *formatted product* $\tilde{L} := L \oplus L^{(1)} \odot L^{(2)}$ is defined by using the formatted addition in the $\mathbf{R}k$-subblocks. We distinguish between three cases.

 : all matrices are subdivided. The multiplication and addition is done in the subblocks.

 : the target matrix is subdivided and (at least) one of the factors is not subdivided. Then one of the factors has at most rank $\max\{k, C_{\text{leaf}}\}$. We have to compute the (low rank) product as in Section 3.5 and add the product to the target matrix.

 : the target matrix is not subdivided. This case will be treated in a separate procedure *MulAddRk*.

```
procedure MulAdd(var L̃, τ, ζ, σ, L^(1), L^(2));
begin
  if S(τ×ζ) ≠ ∅ and S(ζ×σ) ≠ ∅ and S(τ×σ) ≠ ∅
  then
    { Case 1: All matrices are subdivided }
    for τ' ∈ S(τ), ζ' ∈ S(ζ), σ' ∈ S(σ) do
      MulAdd(L̃, τ', ζ', σ', L^(1), L^(2))
  else begin
    if S(τ×σ) ≠ ∅ then begin
      { Case 2: The target matrix is subdivided }
      Calculate the product L' := L^(1)|_{τ×ζ} L^(2)|_{ζ×σ}
      and add L' to L̃|_{τ×σ} { formatted addition in
      subblocks of τ×σ }
    end
    else begin
      { Case 3: The target matrix is not subdivided}
      MulAddRk(L̃, τ, ζ, σ, L^(1), L^(2))
    end
  end
end
```

Calling this procedure with $\tau = \zeta = \sigma = I$, $\tilde{L} := 0$ yields $\tilde{L} = L^{(1)} \odot L^{(2)}$.

To cover Case 3 we have to multiply two subdivided matrices, truncate the product to the set of $\mathbf{R}k$-matrices and add the result to the target matrix. To do this we first calculate the products in the subblocks and truncate them to the set of $\mathbf{R}k$-matrices. Afterwards all four $\mathbf{R}k$-submatrices are added to the target matrix (extending them by zeros such that all matrices are of the same size) using the formatted addition.

```
procedure MulAddRk(var L̃, τ, ζ, σ, L^(1), L^(2));
begin
  if S(τ×ζ) = ∅ or S(ζ×σ) = ∅ then
    Calculate the product L' := L^(1)|_{τ×ζ} L^(2)|_{ζ×σ}
    and add L' to L̃|_{τ×σ} {formatted addition}
  else begin
    for each τ' ∈ S(τ), σ' ∈ S(σ) do begin
      Initialise L'_{τ',σ'} := 0;
      for each ζ' ∈ S(ζ) do MulAddRk(L'_{τ',σ'}, τ', ζ',
      σ', L^(1), L^(2));
      {L'_{τ',σ'} is smaller than L and extended by
      zeros}
      L̃ := L ⊕ Σ_{τ'∈S(τ)} Σ_{σ'∈S(σ)} L'_{τ',σ'}
    end
  end
end
```

The complexity for the formatted multiplication is $\mathcal{O}(nk^2\log(n)^2)$ for the block trees from Refs. [14,17].

If $\mathcal{T}_{I\times I}$ is an arbitrary block tree that is sparse and *almost idempotent* (defined in Ref. [11]) then the complexity is again $\mathcal{O}(nk^2\log(n)^2)$ [10,11].

4.4. Inversion

The inverse of a 2×2 block-matrix can be computed by use of the Schur complement [14] if the matrix is, e.g., positive definite. The exact sums and products are replaced by the formatted operations $\oplus, \odot$ and recursively one can define the *formatted inverse* $\tilde{L}$ of L by the following procedure:

```
procedure Invert(var L̃, τ, σ, L);
begin
  if S(τ×σ) = ∅ then
    Calculate the inverse L̃ := L^{-1} exactly {Small
    matrix}
  else begin
    {S(τ) = {τ1, τ2}, S(σ) = {σ1, σ2}, L = [L11 L12; L21 L22]}
    Invert(Y, τ1, σ1, L|_{τ1×σ1});
    S := L22 ⊖ (L21 ⊙ (Y ⊙ L12));
    Invert(L̃|_{τ2×σ2}, τ2, σ2, S);
    L̃|_{τ1×σ1} := Y ⊕ (Y ⊙ (L12 ⊙ (L̃|_{τ2×σ2} ⊙ (L21 ⊙ Y))));
    L̃|_{τ1×σ2} := −Y ⊙ (L12 ⊙ L̃|_{τ2×σ2});
    L̃|_{τ2×σ1} := −L̃|_{τ2×σ2} ⊙ (L21 ⊙ Y)
  end
end
```

The starting index sets are $\tau = \sigma = I$. Note that two auxiliary matrices Y, S are needed in the procedure.

The complexity for the computation of the formatted inverse is the same as for the multiplication of two $\mathcal{H}$-matrices: $\mathcal{O}(nk^2\log(n)^2)$. This is proven in Ref. [14] for a one-dimensional model problem but also holds for arbitrary block trees $\mathcal{T}_{I\times I}$ that are sparse and *almost idempotent* [10,11].

5. $\mathcal{H}^2$-matrices

The matrix-vector multiplication for $\mathcal{H}$-matrices (Section 4.1) has almost linear complexity. While this is sufficient for most applications, it is not optimal, so we will now consider improvements of the basic $\mathcal{H}$-matrix technique that lead to a better complexity. For some problems, even the optimal complexity of $\mathcal{O}(n)$ can be reached.

5.1. Uniform $\mathcal{H}$-matrices

Let us recall the approximation (8):

$$L_{ij} \approx \sum_{\iota=1}^{k} \int_\Omega p_\iota^\tau(x)\varphi_i(x)\mathrm{d}x \int_\Omega g(x_\iota^\tau, y)\varphi_j(y)\mathrm{d}y.$$

It was derived by interpolating the kernel function $g(\cdot,\cdot)$ in one argument, and the first argument x was chosen because of the smaller size of the support Ω_τ corresponding to the cluster τ. If Ω_σ had been smaller, we would have interpolated with respect to the second argument y.

We will now refine this approach along the lines described in Ref. [8] and introduce *uniform $\mathcal{H}$-matrices*, a subset of the general $\mathcal{H}$-matrices, that allow us to improve the complexity for the matrix-vector multiplication and of the storage requirements.

If the diameters of Ω_τ and Ω_σ are of comparable size, we can go one step further and interpolate in *both* variables, leading to the interpolant

$$\tilde{g}(x,y) := \sum_{\iota=1}^{k_\tau}\sum_{\kappa=1}^{k_\sigma} g(x_\iota^\tau, x_\kappa^\sigma)p_\iota^\tau(x)p_\kappa^\sigma(y). \tag{9}$$

Since we are now interpolating in both arguments, we have to replace the standard admissibility condition (5) by the modified condition

$$\max\{\operatorname{diam}(\Omega_\tau), \operatorname{diam}(\Omega_\sigma)\} \le \eta \operatorname{dist}(\Omega_\tau, \Omega_\sigma) \tag{10}$$

in order to get approximation results similar to those derived for the $\mathcal{H}$-matrix technique. For most applications, the more restrictive new condition (10) and the original condition (5) will be equivalent [11].

By replacing $g(\cdot,\cdot)$ by $\tilde{g}(\cdot,\cdot)$ in Eq. (4), we get

$$L_{ij} \approx \sum_{\iota=1}^{k_\tau}\sum_{\kappa=1}^{k_\sigma} g(x_\iota^\tau, x_\kappa^\sigma) \int_\Omega p_\iota^\tau(x)\varphi_i(x)\mathrm{d}x \int_\Omega p_\kappa^\sigma(y)\varphi_j(y)\mathrm{d}y,$$

which can be rewritten in the form

$$L|_{\tau\times\sigma} \approx V_\tau S_{\tau,\sigma} V_\sigma^{\mathrm{T}} \tag{11}$$

with

$$\begin{aligned}(V_\tau)_{i\iota} &:= \int_\Omega p_\iota^\tau(x)\varphi_i(x)\mathrm{d}x,\\ (V_\sigma)_{j\kappa} &:= \int_\Omega p_\kappa^\sigma(y)\varphi_j(y)\mathrm{d}y\end{aligned} \tag{12}$$

and

$$(S_{\tau,\sigma})_{\iota\kappa} := g(x_\iota^\tau, x_\kappa^\sigma). \tag{13}$$

The important difference between the $\mathbf{R}k$-representation used for standard $\mathcal{H}$-matrices and the representation (14) is the fact that the matrix V_τ depends only on the row cluster τ and the matrix V_σ depends only on the column cluster σ, while all the information about their interaction via the kernel function is stored in the, typically relatively small, matrix $S_{\tau,\sigma}$.

Definition 5.1 (Cluster basis). Let $\mathbf{k} = (k_\tau)_{\tau\in T_I} \in \mathbb{N}^{T_I}$. A family $V = (V_\tau)_{\tau\in T_I}$ with $V_\tau \in \mathbb{R}^{\tau\times k_\tau}$ for each $\tau \in T_I$ is called a *cluster basis* with respect to the *rank distribution* $\mathbf{k}$.

Definition 5.2 (Uniform $\mathcal{H}$-matrix). Let $L \in \mathbb{R}^{I\times I}$ be a matrix and $\mathcal{T}_{I\times I}$ a block tree of $I\times I$ consisting of admissible and non-admissible leaves.
Let V be a cluster basis with respect to a rank distribution $\mathbf{k}$. L is called *uniform $\mathcal{H}$-matrix* with respect to V and the coefficient family $(S_{\tau,\sigma})_{\tau\times\sigma\in\mathcal{L}_{I\times I}}$, if

$$L|_{\tau\times\sigma} = V_\tau S_{\tau,\sigma} V_\sigma^{\mathrm{T}} \tag{14}$$

holds for all $\tau\times\sigma \in \mathcal{L}_{I\times I}$.

Remark 5.3 (Subspace property). While the $\mathcal{H}$-matrices are not closed under addition (therefore we have to use the formatted addition mentioned above), the uniform $\mathcal{H}$-matrices corresponding to a fixed cluster basis V form a subspace of $\mathbb{R}^{I\times I}$.

The decomposition (14) gives rise to a new algorithm for computing the matrix-vector product $y := Lx$:

if we denote the set of all columns in a row corresponding to $\tau \in T_I$ (Fig. 3) by

$$R_\tau := \{\sigma \in T_I : \tau\times\sigma \in \mathcal{L}_{I\times I}\},$$

we find

$$y_i = (Lx)_i = \sum_{\tau\ni i}\left(\sum_{\sigma\in R_\tau} L|_{\tau\times\sigma}x|_\sigma\right)_i = \sum_{\tau\ni i}\left(\sum_{\sigma\in R_\tau} V_\tau S_{\tau,\sigma}V_\sigma^{\mathrm{T}}x|_\sigma\right)_i$$

$$= \sum_{\tau\ni i}\left(V_\tau \sum_{\sigma\in R_\tau} S_{\tau,\sigma}V_\sigma^{\mathrm{T}}x|_\sigma\right)_i.$$

This representation of the result vector leads to the desired new algorithm: in a first step, we compute the vectors

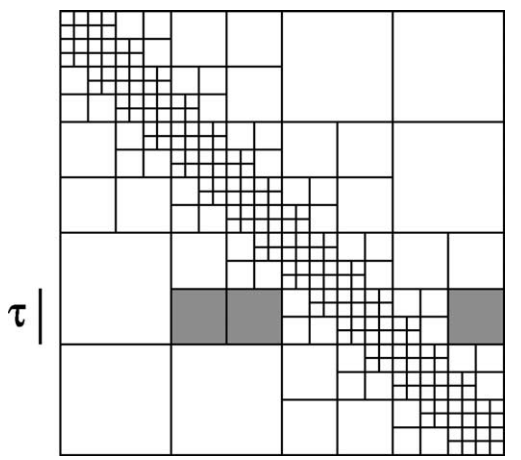


Fig. 3. Blocks in the cluster row corresponding to τ.

$\hat{x}_\sigma := V_\sigma^{\mathrm{T}} x|_\sigma$, leading to the simplified equation

$$y_i = \left(\sum_{\tau \ni i} V_\tau \sum_{\sigma \in R_\tau} S_{\tau,\sigma} \hat{x}_\sigma \right)_i .$$

In the second step, we compute the inner sum

$$\hat{y}_\tau := \sum_{\sigma \in R_\tau} S_{\tau,\sigma} \hat{x}_\sigma .$$

In the last step, we compute the outer sum

$$y_i = \left(\sum_{\tau \ni i} V_\tau \hat{y}_\tau \right)_i .$$

The resulting algorithm takes the following form:

```
procedure UniformMVM(x, var y);
  begin
  { Forward transformation }
  for σ ∈ T_I do x̂_σ := V_σ^T x|_σ;
  { Multiplication }
  for τ ∈ T_I do begin
    ŷ_τ := 0;
    for σ ∈ R_τ do ŷ_τ := ŷ_τ + S_{τ,σ} x̂_σ
  end;
  { Backward transformation }
  y := 0;
  for τ ∈ T_I do y|_τ := y|_τ + V_τ ŷ_τ
end
```

Remark 5.4 (Complexity). In typical applications with 'constant' rank $k_\tau = \min\{|\tau|, k_0\}$, the computation of the forward or backward transformation to a given vector can be computed in $\mathcal{O}(nk_0 \log n)$ operations. The multiplication requires only $\mathcal{O}(nk_0)$ operations.

Although the complexity for the entire matrix-vector multiplication for uniform $\mathcal{H}$-matrices remains the same as in the case of $\mathcal{H}$-matrices, the new class of matrices has other advantages: they form a subspace, i.e. they can be added without truncation and the addition requires usually only $\mathcal{O}(nk_0)$ operations. If the cluster basis V, i.e. the subspace of the set of uniform $\mathcal{H}$-matrices, is fixed, only $\mathcal{O}(nk_0)$ units of memory are required to store an element of this class, therefore uniform $\mathcal{H}$-matrices can be used to significantly reduce memory requirements when working with more than one discretised operator, e.g. when storing resolvents $(A - z_i I)^{-1}$ for several shifts z_i.

5.2. $\mathcal{H}^2$-matrices

The asymptotic complexity of the matrix-vector multiplication for uniform $\mathcal{H}$-matrices is dominated by the forward and backward transformation. In order to come closer to the optimal complexity $\mathcal{O}(n)$, we have to find a way of improving these transformations. The method of doing this was introduced in Ref. [19] under the name of $\mathcal{H}^2$-matrices. While the former paper applies the Taylor expansion to approximate the kernel function, our presentation will be based on the interpolation approaches described in Refs. [5,8].

Let us consider the case of an approximation of the type (9) with constant approximation order, i.e.

$$\mathrm{span}\{p_\iota^{\tau_1} : \iota \in \{1, \ldots, k_{\tau_1}\}\} = \mathrm{span}\{p_\iota^{\tau_2} : \iota \in \{1, \ldots, k_{\tau_2}\}\}$$

holds for all $\tau_1, \tau_2 \in T_I$. This means that each polynomial $p_\iota^{\tau_1}$ corresponding to a cluster τ_1 can be expressed in the polynomial basis corresponding to any other cluster τ_2 and that we have constant rank $k_\tau = k_0$ (this implies $|\tau| \geq k_0$ for all clusters τ).

Since each $p_\iota^{\tau_2}$ is a Lagrange polynomial corresponding to an interpolation point $x_\iota^{\tau_2}$, we even find

$$p_\iota^{\tau_1}(x) = \sum_{\kappa=1}^{k_{\tau_2}} p_\iota^{\tau_1}(x_\kappa^{\tau_2}) p_\kappa^{\tau_2}(x).$$

Let us apply this equation to the case of a cluster τ with a son cluster τ'. For each $i \in \tau'$, we find

$$(V_\tau)_{i\iota} = \int_\Omega p_\iota^\tau(x) \varphi_i(x) \mathrm{d}x = \sum_{\kappa=1}^{k_{\tau'}} p_\iota^\tau(x_\kappa^{\tau'}) \int_\Omega p_\kappa^{\tau'}(x) \varphi_i(x) \mathrm{d}x$$

$$= \sum_{\kappa=1}^{k_{\tau'}} p_\iota^\tau(x_\kappa^{\tau'}) (V_{\tau'})_{i\kappa},$$

i.e. we can express V_τ in terms of $V_{\tau'}$. We introduce the *transfer matrix* $B_{\tau',\tau}$ by setting

$$(B_{\tau',\tau})_{\kappa\iota} := p_\iota^\tau(x_\kappa^{\tau'})$$

and rewrite the above relation in the form

$$V_\tau|_{\tau' \times k_\tau} = V_{\tau'} B_{\tau',\tau}. \tag{15}$$

This means that we only have to store V_τ for clusters without sons, while for all other clusters, the transfer matrices $B_{\tau',\tau}$ of size $\mathrm{K}_0 \times \mathrm{K}_0$ are sufficient.

Definition 5.5 (Nested cluster basis). A cluster basis V with respect to a rank distribution k is called *nested*, if there is a family $B = (B_{\tau',\tau})_{\tau \in T_I, \tau' \in S(\tau)}$ of *transfer matrices* satisfying

$$V_\tau|_{\tau' \times k_\tau} = V_{\tau'} B_{\tau',\tau}$$

for all $\tau \in T_I$ and $\tau' \in S(\tau)$.

Definition 5.6 ($\mathcal{H}^2$-matrix). Let $L \in \mathbb{R}^{I \times I}$ be a uniform $\mathcal{H}$-matrix with respect to a cluster basis V. The matrix L is called $\mathcal{H}^2$-matrix, if V is nested.

The name '$\mathcal{H}^2$-matrix' is motivated by the fact that for this class of matrices, two hierarchies are involved: first the hierarchy of the clusters already exploited for $\mathcal{H}$-matrices, second the hierarchy of the cluster bases.

In addition to the fact that the memory requirements of $\mathcal{H}^2$-matrices are lower than those of $\mathcal{H}$- or even uniform $\mathcal{H}$-matrices, we can even speed up the forward and backward transformation: if τ is a cluster with sons $\{\tau_1, \ldots, \tau_s\}$, we have

$$V_\tau = \begin{pmatrix} V_{\tau_1} B_{\tau_1,\tau} \\ \vdots \\ V_{\tau_s} B_{\tau_s,\tau} \end{pmatrix} = \begin{pmatrix} V_{\tau_1} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & V_{\tau_s} \end{pmatrix} \begin{pmatrix} B_{\tau_1,\tau} \\ \vdots \\ B_{\tau_s,\tau} \end{pmatrix}, \tag{16}$$

and therefore

$$\hat{x}_\tau = V_\tau^{\mathrm{T}} x|_\tau = (B_{\tau_1,\tau}^{\mathrm{T}} V_{\tau_1}^{\mathrm{T}} \quad \cdots \quad B_{\tau_s,\tau}^{\mathrm{T}} V_{\tau_s}^{\mathrm{T}}) x|_\tau = \sum_{j=1}^{s} B_{\tau_j,\tau}^{\mathrm{T}} V_{\tau_j}^{\mathrm{T}} x|_{\tau_j}$$

$$= \sum_{j=1}^{s} B_{\tau_j,\tau}^{\mathrm{T}} \hat{x}_{\tau_j}.$$

This equation leads to the following recursive procedure that, given an input vector and a root cluster τ, will compute the coefficient vectors corresponding to all descendants of τ in the cluster tree:

```
procedure FastForward(τ, x, var x̂);
begin
  if S(τ) = ∅ then
    x̂_τ := V_τ^T x|_τ
  else begin
    x̂_τ := 0;
    for τ' ∈ S(τ) do begin
      FastForward(τ', x, x̂);
      x̂_τ := x̂_τ + B_{τ',τ}^T x̂_{τ'}
    end
  end
end
```

The backward transformation can, too, be rewritten as a recursive procedure:

```
procedure FastBackward(τ, var y, var ŷ);
begin
  if S(τ) = ∅ then
    y := V_τ ŷ_τ
  else
    for τ' ∈ S(τ) do begin
      ŷ_{τ'} := ŷ_{τ'} + B_{τ',τ} ŷ_τ;
      FastBackward(τ', y, ŷ)
    end
  end
end
```

Remark 5.7 (Complexity). The matrices $B_{\tau',\tau}$ are in $\mathbb{R}^{k_{\tau'} \times k_\tau}$, so the recursion steps in the new transformation algorithms require only $\mathcal{O}(k_{\tau'} k_\tau)$ operations, leading to a total of $\mathcal{O}(nk_0)$. In typical applications, the start of the iteration, i.e. the multiplication with V_τ in the leaves of $\mathcal{T}_I$, requires the same amount of work, leading to a total of $\mathcal{O}(nk_0)$.

The following algorithm results from combining the fast forward and backward transformations with the matrix-vector multiplication approach used for uniform matrices:

```
procedure FastMVM(x, var y);
begin
  { Recursive forward transformation }
  FastForward(I, x, x̂);
  { Multiplication }
  for τ ∈ T_I do begin
    ŷ_τ := 0;
    for σ ∈ R_τ do ŷ_τ := ŷ_τ + S_{τ,σ} x̂_σ
  end;
  {Recursive backward transformation}
  FastBackward(I, y, ŷ)
end
```

Remark 5.8 (Complexity). By using the recursive algorithms, we have reduced the complexity of the forward and backward transformation to $\mathcal{O}(nk_0)$. In standard applications, the multiplication step requires the same amount of work, so the new matrix-vector multiplication algorithm has a complexity of $\mathcal{O}(nk_0)$.

Remark 5.9 (Variable order). Even the improved complexity of $\mathcal{O}(nk_0)$ is not optimal, since usually k_0 will be chosen to be proportional to $\log n$.
In Ref. [25] a further refinement of $\mathcal{H}^2$-matrices is described: the order of approximation is no longer constant, but increases as the clusters become larger. This approach reduces the complexity for the matrix-vector multiplication and storage to the optimal class of $\mathcal{O}(n)$, while leaving the approximation properties intact, if the original kernel function $k(\cdot,\cdot)$ satisfies some additional assumptions.

5.3. Adaptive choice of the cluster basis

As soon as a cluster basis is fixed, the best uniform $\mathcal{H}$- or $\mathcal{H}^2$-approximation of a given matrix $L \in \mathbb{R}^{I \times I}$ in the Frobenius norm can be computed by solving the variational problem given by the projection equation

$$\langle V_\tau S_{\tau,\sigma} V_\sigma^{\mathrm{T}} - L|_{\tau\times\sigma}, V_\tau X V_\sigma^{\mathrm{T}} \rangle_{\mathrm{F}} = 0$$

for all $X \in \mathbb{R}^{k_\tau \times k_\sigma}$. The solution is given by

$$S_{\tau,\sigma} = (V_\tau^{\mathrm{T}} V_\tau)^{-1} V_\tau^{\mathrm{T}} L|_{\tau\times\sigma} V_\sigma (V_\sigma^{\mathrm{T}} V_\sigma)^{-1}. \tag{17}$$

If the bases V_τ and V_σ are orthogonal, this expression takes the simple form

$$S_{\tau,\sigma} = V_\tau^{\mathrm{T}} L|_{\tau\times\sigma} V_\sigma. \tag{18}$$

This means that, in order to find a good approximation, we 'only' have to determine a suitable cluster basis, preferably orthogonal, and compute the corresponding coefficient matrices by Eq. (17) or (18).

The a priori choice of a cluster basis will, in most applications, not lead to an optimal $\mathscr{H}^2$-representation of the operator. In some applications, e.g., when dealing with the inverses of finite element matrices or pseudo-differential operators, the theoretical construction of a suitable cluster basis will be much too complicated.

In both cases, an algorithm for creating a cluster basis a posteriori is desirable. We will now describe an algorithm (introduced in Ref. [4]) for doing this.

In the case of uniform $\mathscr{H}$-matrices, we can use the following straightforward approach: in order to find an optimal rank-k-basis of a general matrix Y, we compute the singular value decomposition $U\Sigma V^{\mathrm{T}} = Y$ of Y and use the first k columns of U as basis vectors. This is equivalent to choosing the eigenvectors corresponding to the k largest eigenvalues of the matrix $YY^{\mathrm{T}} = U\Sigma^2 U^{\mathrm{T}}$. Since a basis V_τ will in general be used by more than one block, we add up all blocks of the form $\tau\times\sigma$ with $\sigma \in R_\tau$ (recall Fig. 3) to get

$$G_\tau := \sum_{\sigma\in R_\tau} L|_{\tau\times\sigma} L|_{\tau\times\sigma}^{\mathrm{T}}.$$

The eigenvectors corresponding to the k largest eigenvalues of G_τ form the optimal basis V_τ.

The case of $\mathscr{H}^2$-matrices is more complicated: since the bases V_τ are required to be *nested*, they cannot be computed independently. Since the choice of a basis V_τ restricts the possible solutions for all ancestors of τ in the tree $\mathscr{T}_I$, blocks corresponding to these ancestors have to be considered in the computation of V_τ. To this end, we introduce the extended set

$$R_\tau^+ := \{\sigma \in T_I : (\exists \tau_0 \in T_I : \sigma \in R_{\tau_0} \wedge \tau \subseteq \tau_0)\}$$

representing all blocks corresponding to an ancestor of τ (Fig. 4).

We want to find an orthogonal basis V_τ that minimises the Frobenius error in all blocks

$$\sum_{\sigma\in R_\tau^+} \|L|_{\tau\times\sigma} - V_\tau V_\tau^{\mathrm{T}} L|_{\tau\times\sigma}\|_{\mathrm{F}}^2 = \sum_{\sigma\in R_\tau^+} (\|L|_{\tau\times\sigma}\|_{\mathrm{F}}^2 - \|V_\tau V_\tau^{\mathrm{T}} L|_{\tau\times\sigma}\|_{\mathrm{F}}^2),$$

i.e. that maximises

$$\sum_{\sigma\in R_\tau^+} \|V_\tau^{\mathrm{T}} L|_{\tau\times\sigma}\|_{\mathrm{F}}^2 = \mathrm{tr}\left(V_\tau^{\mathrm{T}}\left(\sum_{\sigma\in R_\tau^+} L|_{\tau\times\sigma} L|_{\tau\times\sigma}^{\mathrm{T}}\right) V_\tau\right), \tag{19}$$

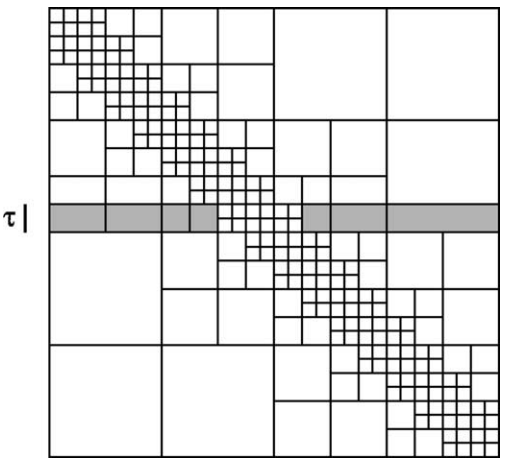


Fig. 4. All blocks contributing to a cluster τ.

where

$$\mathrm{tr}(M) := \sum_{i\in I} M_{ii}$$

denotes the trace of a matrix $M \in \mathbb{R}^{I\times I}$.

We can solve this problem by introducing the matrix

$$G_\tau := \sum_{\sigma\in R_\tau^+} L|_{\tau\times\sigma} L|_{\tau\times\sigma}^{\mathrm{T}},$$

computing its orthogonal diagonalisation (Schur decomposition)

$$Q_\tau D_\tau Q_\tau^{\mathrm{T}} = G_\tau$$

and forming V_τ by picking the first k columns of Q_τ.

If τ has sons $\{\tau_1, \ldots, \tau_s\} = S(\tau) \neq \emptyset$, we have to ensure that the cluster bases are nested. We define

$$\bar{V}_\tau := \begin{pmatrix} B_{\tau_1,\tau} \\ \vdots \\ B_{\tau_s,\tau} \end{pmatrix} \tag{20}$$

and rewrite Eq. (16) in the form

$$V_\tau = \begin{pmatrix} V_{\tau_1} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & V_{\tau_s} \end{pmatrix} \bar{V}_\tau.$$

Combining this reformulation of the nestedness condition (15) with the maximisation problem (19), we find a new problem: now we have to find an orthogonal matrix $\bar{V}_\tau$

that maximises

$$\operatorname{tr}\left(\bar{V}_\tau^{\mathrm{T}}\left(\sum_{\sigma\in R_\tau^+}\begin{pmatrix} V_{\tau_1}^{\mathrm{T}} & \cdots & 0\\ \vdots & \ddots & \vdots\\ 0 & \cdots & V_{\tau_s}^{\mathrm{T}}\end{pmatrix} L|_{\tau\times\sigma} L|_{\tau\times\sigma}^{\mathrm{T}} \begin{pmatrix} V_{\tau_1} & \cdots & 0\\ \vdots & \ddots & \vdots\\ 0 & \cdots & V_{\tau_s}\end{pmatrix}\right)\bar{V}_\tau\right)$$

$$= \operatorname{tr}\left(\bar{V}_\tau^{\mathrm{T}}\left(\sum_{\sigma\in R_\tau^+}\begin{pmatrix} V_{\tau_1}^{\mathrm{T}} L|_{\tau_1\times\sigma}\\ \vdots\\ V_{\tau_s}^{\mathrm{T}} L|_{\tau_s\times\sigma}\end{pmatrix}\begin{pmatrix} V_{\tau_1}^{\mathrm{T}} L|_{\tau_1\times\sigma}\\ \vdots\\ V_{\tau_s}^{\mathrm{T}} L|_{\tau_s\times\sigma}\end{pmatrix}^{\mathrm{T}}\right)\bar{V}_\tau\right)$$

$$= \operatorname{tr}\left(\bar{V}_\tau^{\mathrm{T}}\left(\sum_{\sigma\in R_\tau^+}\bar{L}_{\tau,\sigma}\bar{L}_{\tau,\sigma}^{\mathrm{T}}\right)\bar{V}_\tau\right)$$

with

$$\hat{L}_{\tau_i,\sigma} := V_{\tau_i}^{\mathrm{T}} L|_{\tau_i\times\sigma} \quad \text{for } i\in\{1,\ldots,s\} \text{ and } \bar{L}_{\tau,\sigma} := \begin{pmatrix}\hat{L}_{\tau_1,\sigma}\\ \vdots\\ \hat{L}_{\tau_s,\sigma}\end{pmatrix}. \tag{21}$$

This maximisation problem is again of the form discussed above and can therefore be solved by computing the orthogonal diagonalisation of

$$\bar{G}_\tau := \sum_{\sigma\in R_\tau^+}\bar{L}_{\tau,\sigma}\bar{L}_{\tau,\sigma}^{\mathrm{T}},$$

leading to the following algorithm:

```
procedure ComputeRowBasis(τ);
begin
  if S(τ) = ∅ then begin
    G_τ := 0;
    for σ ∈ R_τ^+ do G_τ := G_τ + L|_{τ×σ} L|_{τ×σ}^T;
    Compute the orthogonal diagonalisation
    Q_τ^T G_τ Q_τ = D;
    Form V_τ by copying the first k columns of Q_τ;
    for σ ∈ R_τ^+ do L_{τ,σ} := V_τ^T L|_{τ×σ}
  end
  else begin
    for τ' ∈ S(τ) do ComputeRowBasis(τ');
    Ḡ_τ := 0;
    for σ ∈ R_τ^+ do Ḡ_τ := Ḡ_τ + L̄_{τ,σ} L̄_{τ,σ}^T;
    Compute the orthogonal diagonalisation
    Q_τ^T Ḡ_τ Q_τ = D;
    Form V̄_τ by copying the first k columns of Q_τ;
    Split V̄_τ into (B_{τ',τ})_{τ'∈S(τ)} according to (20);
    for σ ∈ R_τ^+ do L̂_{τ,σ} := V̄_τ^T L̄_{τ,σ}
  end
end
```

Remark 5.10 (Complexity). For 'constant' rank, i.e. for $k_\tau = \min\{|\tau|, k_0\}$, the algorithm requires $\mathcal{O}(n^2 k_0)$ operations.
In the case of suitably varying rank, the algorithm reaches the optimal complexity $\mathcal{O}(n^2)$.

Remark 5.11 (Application to $\mathcal{H}$-matrices). The algorithm can, obviously, be applied to $\mathcal{H}$-matrices instead of general matrices. In this case, we can use the special structure in order to improve the complexity to $\mathcal{O}(nk_{\mathcal{H}}^2 \log(n))$, where $k_{\mathcal{H}}$ denotes the blockwise rank of the original $\mathcal{H}$-matrix.

6. Alternative matrix formats

Besides $\mathcal{H}$-matrices and $\mathcal{H}^2$-matrices there are some specialised other matrix formats for certain applications, e.g.

- wire-basket $\mathcal{H}$-matrices: if applications in the field of finite element matrices are considered, it is possible to combine $\mathcal{H}$-matrix ideas with local multi-grid solvers in order to reduce memory requirements [18];
- blended kernel approximation: for special geometries (e.g. surfaces of cylinders or bricks) the matrix of certain discretised integral operators takes the form of a tensor product of $\mathcal{H}$-matrices and circulant or Toeplitz matrices. This structure can be exploited by combining fast Fourier transforms and $\mathcal{H}$-matrix techniques to speed up the matrix vector multiplication [15];
- semi-explicit $\mathcal{H}$-matrices: the solution operator of a general elliptic problem can be expressed as a sum of the convolution operator corresponding to the fundamental solution and a pseudo-differential operator taking care of the boundary condition. The former operator can be treated by $\mathcal{H}$-matrix techniques while fast methods exist for the latter [21].

7. Applications

In this section, we will demonstrate how the $\mathcal{H}$-matrix-techniques described above can be applied to problems from the fields of integral equations, differential equations and control theory.

7.1. Integral equations

In order to demonstrate the advantage of the $\mathcal{H}$-matrix approach, we consider the simple example of the discretisation of the single layer potential on the unit circle in two space dimensions using a Galerkin method with piecewise constant basis functions.

Table 1
Approximation error for the single layer potential

n	$m=1$	$m=2$	$m=3$	$m=4$	$m=5$
1024	3.57×10^{-2}	2.16×10^{-3}	2.50×10^{-4}	7.88×10^{-6}	2.67×10^{-6}
2048	3.58×10^{-2}	2.19×10^{-3}	2.51×10^{-4}	7.86×10^{-6}	2.69×10^{-6}
4096	3.59×10^{-2}	2.20×10^{-3}	2.51×10^{-4}	7.87×10^{-6}	2.68×10^{-6}
8192	3.59×10^{-2}	2.20×10^{-3}	2.52×10^{-4}	7.76×10^{-6}	2.67×10^{-6}
16384	3.59×10^{-2}	2.21×10^{-3}	2.53×10^{-4}	7.87×10^{-6}	2.68×10^{-6}

7.1.1. $\mathcal{H}$-matrix approach

The logarithmic kernel function will be approximated by the interpolation approach given in Eq. (7) using tensor product Chebyshev points and corresponding Lagrange polynomials.

We report the relative error $\|L-\tilde{L}\|_2/\|L\|_2$ in Table 1. The Euclidean norms are approximated by performing 100 steps of the power iteration. The first column contains the number of degrees of freedom for each discretisation level, the following columns give the relative error. We observe that the error is bounded independently of the discretisation level and that it decreases very quickly when the interpolation order is increased.

The time required for matrix vector multiplications is given in Table 2. It was measured on a SUN Enterprise 6000 machine using UltraSPARC II processors running at 248 MHz by taking the time required for 100 matrix vector multiplications and dividing by 100. We can see that the complexity grows almost linearly with respect to the number of degrees of freedom and rather slowly with respect to the interpolation order. We let C_{leaf} be proportional to m^{d} in order to simplify the program. This accounts for the surprising improvement of the speed for $m=3$.

Finally, let us consider the time required for building the $\mathcal{H}$-matrix representation of the discretised integral operator. This is given in Table 3 and was measured on the same machine. The integral of the Lagrange polynomials was computed by using an exact Gauss quadrature formula, while the integral of the kernel function was computed analytically. Once more we observe an almost linear growth of the complexity with respect to the number of degrees of freedom and a quadratic growth with respect to the interpolation order.

Note that even on an old and quite slow processor like the 248 MHz UltraSPARC II, the boundary element matrix for more than half a million degrees of freedom can be approximated with an error less than 0.03% in less than half an hour.

7.1.2. $\mathcal{H}^2$-matrix approach

In order to illustrate the advantages of $\mathcal{H}^2$-matrices when considering integral operators, we will now approximate the same kernel function as before by the symmetric interpolation approach given in Eq. (9), once more using tensor product Chebyshev points and the corresponding Lagrange polynomials.

The relative approximation errors for the $\mathcal{H}^2$-matrix are reported in Table 4. As before in the $\mathcal{H}$-matrix case, the error is almost constant with respect to the number of degrees of freedom and decreases quickly when the order of the approximation is increased.

The results for the $\mathcal{H}^2$-matrix approximation are not as good as in the case of the $\mathcal{H}$-matrix for low ranks, but improve significantly as soon as the rank is increased.

In Table 5, we report the times required for the $\mathcal{H}^2$ matrix-vector multiplication. For the lowest rank, the $\mathcal{H}^2$-matrix technique requires more time than the $\mathcal{H}$-matrix approach, but even for order 2, the $\mathcal{H}^2$-matrix is significantly faster.

Table 6 contains the times required for building the $\mathcal{H}^2$-matrix approximation of the integral operator. Obviously, the $\mathcal{H}^2$-matrix approach is much faster than the $\mathcal{H}$-matrix approach in *all* experiments. Especially in the case of higher order expansions, the performance of the $\mathcal{H}^2$-matrix technique is much better than that of the $\mathcal{H}$-matrix method.

Table 2
Time (s) required for the matrix vector multiplication

n	$m=1$	$m=2$	$m=3$	$m=4$	$m=5$
1024	0.01	0.02	0.01	0.01	0.03
2048	0.02	0.04	0.03	0.05	0.07
4096	0.05	0.11	0.09	0.12	0.17
8192	0.12	0.24	0.19	0.26	0.39
16,384	0.27	0.53	0.41	0.56	0.83
32,768	0.57	1.15	0.90	1.23	1.90
65536	1.18	2.44	1.96	2.73	4.14
131,072	2.45	5.18	4.30	5.89	8.98
262,144	5.15	11.32	9.14	12.95	19.78
524,288	10.68	23.81	19.62	28.02	43.57

Table 3
Time (s) required for building the $\mathcal{H}$-matrix

n	$m=1$	$m=2$	$m=3$	$m=4$	$m=5$
1024	0.61	0.93	1.76	3.11	5.60
2048	1.25	2.03	3.85	7.04	12.94
4096	2.56	4.29	8.41	15.82	29.65
8192	5.25	9.16	18.10	35.31	66.27
16,384	10.75	19.30	39.32	77.47	146.65
32,768	22.15	40.83	85.16	169.16	324.36
65,536	45.79	87.32	185.85	368.46	702.63
131,072	92.64	180.73	387.63	788.06	1511.66
262,144	189.15	378.20	854.75	1775.85	3413.45
524,288	388.96	795.84	1743.66	3596.77	6950.55

Table 4
$\mathcal{H}^2$-approximation error for the single layer potential

n	$m=1$	$m=2$	$m=3$	$m=4$	$m=5$
1024	1.37×10^{-1}	8.51×10^{-3}	5.98×10^{-4}	4.27×10^{-5}	4.18×10^{-6}
2048	1.37×10^{-1}	8.56×10^{-3}	5.98×10^{-4}	4.29×10^{-5}	4.19×10^{-6}
4096	1.37×10^{-1}	8.59×10^{-3}	5.98×10^{-4}	4.30×10^{-5}	4.19×10^{-6}
8192	1.37×10^{-1}	8.60×10^{-3}	5.98×10^{-4}	4.31×10^{-5}	4.19×10^{-6}
16384	1.37×10^{-1}	8.61×10^{-3}	5.99×10^{-4}	4.31×10^{-5}	4.19×10^{-6}

Using $\mathcal{H}^2$-matrices, the boundary element matrix for more than half a million degrees of freedom can be approximated with an error less than 0.06% in less than 7 min, even on a slow processor like the UltraSPARC II.

7.2. Elliptic partial differential equations

The model problem of the Poisson equation on the unit sphere is studied numerically in Ref. [10]. Some theoretical results will be presented in a forthcoming technical report [2]. For the convenience of the reader, we will give a short summary of the numerical tests from Ref. [10].

7.2.1. Simple model problem

We consider the Poisson equation

$$-\partial_x^2 u(x,y) - \partial_y^2 u(x,y) = f(x,y), \qquad (x,y)\in\Omega \tag{22}$$

on the unit square $\Omega=[0,1]^2$ with essential boundary condition $u=0$ on the boundary $\Gamma:=\partial\Omega$.

For the discretisation of the variational formulation of Eq. (22) we choose continuous piecewise linear nodal (Lagrangian) basis functions ϕ_i defined by a regular triangulation of $[0,1]^2$. The stiffness matrix

$$A_{i,j} := \int_\Omega \langle\nabla\phi_i,\nabla\phi_j\rangle, \qquad i,j\in I=\{1,\ldots,n\} \tag{23}$$

and mass matrix

$$M_{i,j} := \int_\Omega \phi_i\phi_j, \qquad i,j\in I=\{1,\ldots,n\} \tag{24}$$

are both sparse. Our aim is to compute an approximate inverse $\widetilde{A^{-1}}$ to A as in Section 4.4. The cluster tree T_I and the block tree $\mathcal{T}_{I\times I}$ are constructed as in Example 2.1 and Section 2.3 with admissibility condition (6), $\eta=0.8$.

The time needed to compute the approximate inverse $\widetilde{A^{-1}}$ to A is given in Table 7 (the entry n.c. means that the entry was not computed due to limited storage capacity). If we fix the rank k then the required time grows almost linearly with respect to n ($9.3/1.9=4.9\approx 4=512^2/256^2$).

Once an approximate inverse is computed, one can solve an equation of the form $Ax=b$ by $\tilde{x}:=\widetilde{A^{-1}}b$. The time for one matrix-vector multiplication with the approximate inverse is reported in Table 8.

The relative error $\|A^{-1}-\widetilde{A^{-1}}\|/\|A^{-1}\|$ for the approximation of the inverse can be bounded by

$$\frac{\|A^{-1}-\widetilde{A^{-1}}\|}{\|A^{-1}\|} = \frac{\|(I-\widetilde{A^{-1}}A)A^{-1}\|}{\|A^{-1}\|} \le \|I-\widetilde{A^{-1}}A\|.$$

To measure $\|I-\widetilde{A^{-1}}A\|$ in the spectral norm, we perform 10 steps of the power iteration. The results are given in Table 9.

To compute a more accurate solution x of the equation $Ax=b$ one can use the linear iteration

$$x_0 := 0, \qquad x_{i+1} := x_i - \widetilde{A^{-1}}(Ax_i-b),$$

that has a convergence rate of $\|I-\widetilde{A^{-1}}A\|$. Therefore, it suffices to compute an approximate inverse $\widetilde{A^{-1}}$ such that $\|I-\widetilde{A^{-1}}A\|<1$. A solution of the linear equation $Ax=b$ can be gained up to any desired accuracy, but the discretisation error introduced by the choice of the subspace $\mathrm{span}\{\phi_i : i\in I\}$ is a lower bound for the error of the solution to the continuous problem.

Table 5
Time (s) required for the $\mathcal{H}^2$-matrix vector multiplication

n	$m=1$	$m=2$	$m=3$	$m=4$	$m=5$
1024	0.02	0.01	0.01	0.01	0.02
2048	0.04	0.03	0.03	0.03	0.07
4096	0.07	0.05	0.07	0.08	0.14
8192	0.15	0.10	0.17	0.16	0.29
16,384	0.32	0.26	0.33	0.31	0.57
32,768	0.66	0.49	0.66	0.60	1.12
65,536	1.32	1.00	1.34	1.19	2.25
131,072	2.68	2.00	2.75	2.50	4.77
262,144	5.29	4.30	5.61	5.18	9.50
524,288	10.72	8.26	10.91	9.99	18.57

Table 6
Time (s) required for building the $\mathcal{H}^2$-matrix

n	$m=1$	$m=2$	$m=3$	$m=4$	$m=5$
1024	0.49	0.62	0.78	1.05	1.57
2048	0.98	1.22	1.49	2.11	3.32
4096	1.93	2.31	2.97	4.18	6.35
8192	3.94	4.64	6.16	8.47	12.99
16,384	8.06	9.76	11.88	16.82	25.76
32,768	15.86	18.75	24.25	33.53	51.97
65,536	32.33	37.87	48.20	67.66	103.03
131,072	66.71	75.46	96.63	139.71	208.18
262,144	130.86	156.81	194.27	282.69	438.77
524,288	264.06	307.09	390.56	548.59	839.58

Table 7
Time (s) needed to compute the approximate inverse to A for increasing rank k

k	Number of degrees of freedom, n				
	32^2	64^2	128^2	256^2	512^2
1	$9.3\times10^{+0}$	$6.8\times10^{+1}$	$4.3\times10^{+2}$	$1.9\times10^{+3}$	$9.3\times10^{+3}$
2	$9.7\times10^{+0}$	$8.0\times10^{+1}$	$5.0\times10^{+2}$	$2.7\times10^{+3}$	$1.4\times10^{+4}$
3	$1.1\times10^{+1}$	$9.7\times10^{+1}$	$6.4\times10^{+2}$	$3.7\times10^{+3}$	$2.0\times10^{+4}$
4	$1.2\times10^{+1}$	$1.2\times10^{+2}$	$8.3\times10^{+2}$	$5.1\times10^{+3}$	$2.6\times10^{+4}$
5	$1.3\times10^{+1}$	$1.4\times10^{+2}$	$1.1\times10^{+3}$	$6.6\times10^{+3}$	$3.5\times10^{+4}$
10	$1.9\times10^{+1}$	$2.6\times10^{+2}$	$2.4\times10^{+3}$	$1.5\times10^{+4}$	n.c.
15	$2.1\times10^{+1}$	$3.2\times10^{+2}$	$3.0\times10^{+3}$	$2.1\times10^{+4}$	n.c.

Table 9
Relative error $\|I-\widetilde{A^{-1}}A\|_2$ for the approximation of A^{-1} with increasing rank k

k	Number of degrees of freedom, n				
	32^2	64^2	128^2	256^2	512^2
1	5.2×10^{-1}	2.1×10^{-0}	7.6×10^{-0}	$2.4\times10^{+1}$	$4.9\times10^{+1}$
2	3.5×10^{-2}	3.5×10^{-1}	2.0×10^{-0}	8.2×10^{-0}	$2.4\times10^{+1}$
3	4.9×10^{-3}	3.1×10^{-2}	2.0×10^{-1}	1.1×10^{-0}	5.1×10^{-0}
4	1.1×10^{-3}	9.1×10^{-3}	5.1×10^{-2}	2.7×10^{-1}	1.2×10^{-0}
5	2.2×10^{-4}	7.2×10^{-4}	4.5×10^{-3}	2.3×10^{-2}	1.0×10^{-1}
10	2.9×10^{-7}	4.1×10^{-6}	1.9×10^{-5}	8.0×10^{-5}	n.c.
15	8.4×10^{-13}	5.4×10^{-10}	2.2×10^{-9}	2.8×10^{-8}	n.c.

7.2.2. Simple model problem with 'jumping coefficients'

Changing the operator of Eq. (22) to

$$-\mathrm{div}(\sigma(x,y)\nabla u(x,y)) = f(x,y), \qquad (x,y)\in\Omega \tag{25}$$

for some non-constant function σ does, as far as numerical tests indicate, not destroy the approximability of the inverse by an $\mathscr{H}$-matrix [10]. This is a rather surprising result: for the Poisson equation the singularity function is asymptotically smooth while here neither the singularity function nor its behaviour is known. For a theoretical investigation of the $\mathscr{H}$-matrix approximation of this problem, see Ref. [2].

7.3. Matrix functions

7.3.1. Matrix exponential function

Matrix functions like the matrix exponential can be computed effectively by use of the Dunford–Cauchy representation

$$\exp(A) = \frac{1}{2\pi i}\int_\Gamma \exp(z)(zI-A)^{-1}\mathrm{d}z \tag{26}$$

of the operator. To do so, the matrices $zI-A$ have to be inverted. If the matrix A stems, e.g. from the discretisation of an elliptic partial differential equation, we can use the $\mathscr{H}$-matrix arithmetic to compute the inverse at certain points $z=z_j$ and compute the integral in Eq. (26) by some quadrature rule. This approach is investigated in Ref. [6], including numerical examples.

The Dunford–Cauchy representation can also be used for other operators [7].

An alternative way to compute the matrix exponential is using the truncated Taylor-series with scaling. Let A be a matrix and $s\in\mathbb{N}$ such that $\|A\|<2^{s-1}$. Then we find

$$\exp(A) = \exp(2^{-s}A)^{2^s} \approx \left(\sum_{i=0}^{k}\frac{1}{i!}(2^{-s}A)^i\right)^{2^s}$$

Therefore, one computes in a first step the matrix

$$B := \sum_{i=0}^{k}\frac{1}{i!}(2^{-s}A)^i$$

by

```
procedure expsmall(var B, A, k);
begin
  B := k^-1 A;
  for i := k - 1 downto 1 do B := i^-1 (A⊕A⊙B);
  B := B + I
end
```

with calling parameters $B, (2^{-s}A)$. Afterwards the matrix B^{2^s} is gained by squaring s times,

$$B_0 := B,\quad B_{i+1} := B_i\odot B_i, \qquad i=1,\ldots,s-1$$

```
procedure exp(var B, A, k);
begin
  s := ⌈log2(‖A‖)⌉;
  A := 2^-s A;
  expsmall(B, A, k);
  for i := 1 to s do B := B⊙B
end
```

The complexity for the computation is $\mathscr{O}(\log(\|A\|)n\log(n)^2k^2)$ since $\log_2(\|A\|)$ formatted multiplications and additions of $\mathscr{H}$-matrices are involved.

Table 8
Time (s) needed for one matrix-vector multiplication for increasing rank k

k	Number of degrees of freedom, n				
	32^2	64^2	128^2	256^2	512^2
1	2.7×10^{-2}	1.5×10^{-1}	6.9×10^{-1}	$3.0\times10^{+0}$	$1.4\times10^{+1}$
2	2.9×10^{-2}	1.7×10^{-1}	8.1×10^{-1}	$3.9\times10^{+0}$	$1.7\times10^{+1}$
3	3.2×10^{-2}	1.8×10^{-1}	9.7×10^{-1}	$4.7\times10^{+0}$	$2.1\times10^{+1}$
4	3.4×10^{-2}	2.1×10^{-1}	1.1×10^{-0}	$5.7\times10^{+0}$	$2.4\times10^{+1}$
5	3.5×10^{-2}	2.2×10^{-1}	1.2×10^{-0}	$6.4\times10^{+0}$	$3.4\times10^{+1}$
10	4.5×10^{-2}	3.3×10^{-1}	1.8×10^{-0}	$8.9\times10^{+0}$	n.c.
15	5.0×10^{-2}	3.9×10^{-1}	2.3×10^{-0}	$1.2\times10^{+1}$	n.c.

7.3.2. Matrix sign function

Let S be a matrix with spectrum $\sigma(S)$ that does not intersect the imaginary axis. The matrix sign function sign(S) is defined as the matrix function to sign: $\mathbb{C} \to \mathbb{C}$, $x + \mathrm{i}y \mapsto \mathrm{sign}(x)$. The sign of a number $x \neq 0$ is 1 if $x > 0$ and -1 if $x < 0$. This function plays an important role in the treatment of Riccati's equation.

A simple method to calculate the sign function of a matrix S is Newton's method applied to the equation $S^2 = I$, as it is described in Ref. [23].

Theorem 7.1 (Newton's method to calculate the matrix sign function). *Let $S \in \mathbb{R}^{n\times n}$ be a matrix whose spectrum does not intersect the imaginary axis. Then the iteration*

$$S_1 := S, \qquad S_{i+1} := \tfrac{1}{2}(S_i + S_i^{-1}) \tag{27}$$

converges quadratically to the sign of S.

Replacing the exact addition and inversion by their formatted counterparts $\tilde{S}_{i+1} := (1/2)(\tilde{S}_i \oplus \tilde{S}_i^{-1})$ yields an algorithm to compute the sign of an $\mathscr{H}$-matrix in the set of $\mathscr{H}$-matrices. To ensure a sufficient approximation of the matrix sign(S) by $\tilde{S}_\infty := \lim_{i\to\infty} \tilde{S}_i$, one needs more assumptions concerning the structure of S. A numerical test involving the computation of $\tilde{S}_\infty$ will follow in Section 7.4.

7.4. Lyapunov and Riccati equations

An algebraic matrix Riccati equation is an equation of the form

$$A^{\mathrm{T}}X + XA - XFX + G = 0 \tag{28}$$

where the matrices A, F, G are given and X is the unknown solution.

Strategies for solving algebraic matrix Riccati equations (for matrices of a certain structure) are varied. Basically, one can either try to solve the (nonlinear) equation (28) directly, or one can apply Newton's method to simplify the equation to a linear one. The latter results in a series of Lyapunov equations of the form

$$A^{\mathrm{T}}X + XA + G = 0 \tag{29}$$

where the matrices A, G are given and X is the unknown solution.

7.4.1. Low rank case

If the matrix G in Eq. (29) is of (global) low rank and the spectra of A and $-A$ are disjoint, then the singular values of the solution X decay exponentially [9,22]. This means that the rank k needed to approximate the solution X up to a relative error of ε is $k = \mathcal{O}(-\log(\varepsilon))$.

If also the matrix F is of low rank, this extends to the case of the Riccati equation.

Consequently, an appropriate representation for X is that of an $\mathbf{R}k$-matrix. Still one needs a method to compute X without losing the low rank structure.

Example 7.2 (Solution by use of the matrix exponential). Let A be negative definite. Then the solution X to (29) can be represented in the form

$$X = \int_0^\infty \exp(tA^{\mathrm{T}})G\exp(tA)\mathrm{d}t.$$

The integral can be discretised and one has to compute for several $t_i \in (0,\infty)$ the matrix $\exp(t_iA^{\mathrm{T}})G\exp(t_iA)$. This involves the computation of $\exp(t_iA)$ as in Section 7.3.1 and two products. If the matrix G is of low rank k_G then the products can be computed as in Section 3.5.
To solve the Riccati equation (28) we apply Newton's method and have to determine the iterates

$$X_{i+1} = \int_0^\infty \exp(t(A - FX_i)^{\mathrm{T}})(X_iFX_i + G)\exp(t(A - FX_i))\mathrm{d}t. \tag{30}$$

for a suitable initial guess X_0.

Example 7.3 (Solution by use of the matrix sign). An algorithm to solve certain Riccati equations by use of the matrix sign function is presented in Ref. [23]: let $A \in \mathbb{R}^{n\times n}$ be negative definite and $F, G \in \mathbb{R}^{n\times n}$ symmetric positive semidefinite of low rank. Then the stabilising solution X of (28) satisfies

$$\begin{bmatrix} N_{11} \\ N_{21} \end{bmatrix} X = -\begin{bmatrix} N_{12} \\ N_{22} \end{bmatrix}, \tag{31}$$

where the matrices $N_{11}, N_{12}, N_{21}, N_{22} \in \mathbb{R}^{n\times n}$ are

$$\begin{bmatrix} N_{11} & N_{12} \\ N_{21} & N_{22} \end{bmatrix} := \mathrm{sign}\left(\begin{bmatrix} A^{\mathrm{T}} & G \\ F & -A \end{bmatrix}\right) - \begin{bmatrix} I & 0 \\ 0 & I \end{bmatrix}. \tag{32}$$

In the Lyapunov case $F = 0$ this simplifies to $X = N_{12}/2$. In Ref. [12] it is proven that the matrix sign $\left[\begin{smallmatrix} A^{\mathrm{T}} & G \\ F & -A \end{smallmatrix}\right]$ consists of low rank structures plus the identity. The computation of sign$(\cdots)$ is done as in Section 7.3.2. The solution of Eq. (31) simplifies essentially if N_{11} is regular: $X = -N_{11}^{-1}N_{12}$. The exact inversion and multiplication are replaced by the formatted $\mathscr{H}$-matrix arithmetics.

To test the approximability of all the matrices appearing in Eq. (27) we solve the autonomous linear quadratic optimal control problem for the heat equation in one dimension. This has already been investigated in Ref. [24] but there the low rank structure of the solution was not exploited.

Example 7.4 (Control of the heat equation). We consider the linear quadratic optimal control problem of

Table 10
The last but one row shows the number of Newton steps to compute sign and the last row contains the time (s) needed to compute the rank $k = 2$ solution on a Sun Quasar with 450 MHz

Rank	Number of degrees of freedom, n				
	256	1024	4096	16,384	65,536
$k = 1$	1.5×10^{-1}	1.3×10^{-1}	2.5×10^{-0}	Divergent	Divergent
$k = 2$	2.6×10^{-4}	4.2×10^{-4}	1.2×10^{-3}	5.6×10^{-4}	6.7×10^{-4}
$k = 3$	1.2×10^{-5}	1.3×10^{-5}	1.5×10^{-5}	2.3×10^{-5}	3.9×10^{-5}
$k = 4$	9.1×10^{-8}	1.1×10^{-7}	$1.0 \times 10 - 6$	1.8×10^{-6}	6.2×10^{-7}
$k = 5$	4.6×10^{-9}	1.1×10^{-8}	1.5×10^{-8}	3.0×10^{-8}	3.1×10^{-8}
$k = 6$	3.7×10^{-10}	2.4×10^{-10}	4.9×10^{-10}	5.9×10^{-10}	1.7×10^{-9}
Iterat.	14	17	20	23	26
Time ($k = 2$)	8.5	67	462	3033	18263

the one-dimensional heat flow: the goal is to minimise

$$J(u) := \int_0^\infty (y(t)^2 + u(t)^2)\mathrm{d}t \tag{33}$$

for $u \in L_2(0,\infty)$ where y is defined via the differential equation

$$\begin{aligned}
&\frac{\partial}{\partial t}x(t,\xi) = \frac{\partial^2}{\partial \xi^2}x(t,\xi) + b(\xi)u(t), && \xi \in (0,1),\ t \in (0,\infty),\\
&x(t,\xi) = 0, && \xi \in \{0,1\},\ t \in (0,\infty),\\
&x(0,\xi) = x_0(\xi), && \xi \in (0,1),\\
&y(t) = \int_{0.2}^{0.3} x(t,\xi)\mathrm{d}\xi, && t \in (0,\infty).
\end{aligned}$$

The starting value $x_0 \in L_2(0,1)$ is given (not important here) and b is defined as

$$b(\xi) := \begin{cases} l & \xi \in (0.2, 0.3) \\ 0 & \text{otherwise} \end{cases}.$$

The differential equation is discretised by finite differences on a uniform mesh of $(0,1)$ with n inner grid-points and mesh width h. If we define the matrices

$$A_{ij} := \begin{cases} 2h^{-2} & i = j \\ -h^{-2} & |i-j| = 1, \\ 0 & \text{otherwise} \end{cases} \quad B_{i1} := \begin{cases} 1 & ih \in [0.2, 0.3] \\ 0 & \text{otherwise} \end{cases},$$

$$C_{1j} := \int_{0.2}^{0.3} \phi_j(x)\mathrm{d}x, \qquad i,j \in \{1,\ldots,n\},$$

where ϕ_i denotes the ith Lagrange basis function for the interpolation, then the minimising discrete control u is

$$u(t) = -B^{\mathrm{T}}Xx(t), \qquad t \in (0,\infty),$$

where $X \in \mathbb{R}^{n\times n}$ is the unique, nonnegative symmetric solution of the algebraic matrix Riccati equation

$$A^{\mathrm{T}}X + XA - XFX + G = 0$$

for the matrices $F := BB^{\mathrm{T}}$ and $G := C^{\mathrm{T}}C$.

The matrices F, G are of rank one. Using the algorithm described in Example 7.3, we can compute an approximate solution $\tilde{X}$ to Eq. (28). The results can be seen in Table 10, where we present the relative error $\varepsilon := \|\tilde{X} - X\|_2/\|X\|_2$ for increasing rank k and n degrees of freedom.

7.4.2. $\mathscr{H}$-matrix case

If the matrix G in Eq. (29) is an $\mathscr{H}$-matrix with blockwise rank k_G, one can prove under moderate assumptions (see Ref. [12] for details and a proof) that the solution X can be approximated in the $\mathscr{H}$-matrix format with slightly increased blockwise rank k_X.

If also the matrix F is of low rank then this extends to the case of the Riccati equation.

Consequently, an appropriate representation for X is that of an $\mathscr{H}$-matrix. An approximate solution $\tilde{X}$ can be computed as in Example 7.2 or 7.3.

References

[1] Bebendorf M. Approximation of boundary element matrices. Numer Math 2000;86:565–89.

[2] Bebendorf M, Hackbusch W. Existence of $\mathscr{H}$-matrix approximants to the inverse FE-matrix of elliptic operators with L^∞-coefficients. Technical report. Leipzig, Germany: Max Planck Institute for Mathematics in the Sciences; 2002 (submitted for publication).

[3] Bebendorf M, Rjasanov S. Adaptive low-rank approximation of collocation matrices. Technical report 39, Universität Saarbrücken; 2001.

[4] Börm S, Hackbusch W. Data-sparse approximation by adaptive $\mathscr{H}^2$-matrices. Technical report 86, Leipzig, Germany: Max Planck Institute for Mathematics in the Sciences; 2001.

[5] Börm S, Hackbusch W. $\mathscr{H}^2$-matrix approximation of integral operators by interpolation. Technical report 104, Leipzig, Germany: Max Planck Institute for Mathematics in the Sciences; 2001.

[6] Gavrilyuk I, Hackbusch W, Khoromskij B. $\mathscr{H}$-matrix approximation for the operator exponential with applications. Technical report 42, Leipzig, Germany: Max Planck Institute for Mathematics in the Sciences; 2000.

[7] Gavrilyuk I, Hackbusch W, Khoromskij B. $\mathscr{H}$-matrix approximation for elliptic solution operators on cylindric domains, east–west. J Numer Math 2001;9:25–58.

[8] Giebermann K. Multilevel approximation of boundary integral operators. Computing 2001;67:183–207.
[9] Grasedyck L. Singular value bounds for the Cauchy matrix and solutions to Sylvester equations. Technical report 13, Germany: University Kiel; 2001.
[10] Grasedyck L. Theorie und Anwendungen Hierarchischer Matrizen. PhD Thesis. Universität Kiel; 2001.
[11] Grasedyck L, Hackbusch W. Construction and arithmetics of $\mathscr{H}$-matrices. Technical report. Leipzig, Germany: Max Planck Institute for Mathematics in the Sciences; 2002 (submitted for publication).
[12] Grasedyck L, Hackbusch W, Khoromskij B. Application of $\mathscr{H}$-matrices in control theory. Technical report. Leipzig, Germany: Max Planck Institute for Mathematics in the Sciences; 2002 (submitted for publication).
[13] Grasedyck L, Hackbusch W, LeBorne S. Adaptive refinement and clustering of $\mathscr{H}$-matrices. Technical report 106, Leipzig, Germany: Max Planck Institute of Mathematics in the Sciences; 2001.
[14] Hackbusch W. A sparse matrix arithmetic based on $\mathscr{H}$-matrices. Part I. Introduction to $\mathscr{H}$-matrices. Computing 1999;62:89–108.
[15] Hackbusch W, Khoromskij B. Blended kernel approximation in the $\mathscr{H}$-matrix techniques. Technical report 66, Leipzig, Germany: Max Planck Institute for Mathematics in the Sciences; 2000.
[16] Hackbusch W, Khoromskij B. $\mathscr{H}$-matrix approximation on graded meshes. In: Whiteman JR, editor. The mathematics of finite elements and applications X. Amsterdam: Elsevier; 2000. p. 307–16.
[17] Hackbusch W, Khoromskij B. A sparse matrix arithmetic based on $\mathscr{H}$-matrices. Part II. Application to multi-dimensional problems. Computing 2000;64:21–47.
[18] Hackbusch W, Khoromskij B. Towards $\mathscr{H}$-matrix approximation of linear complexity. In: Elschner IGJ, Silbermann B, editors. Problems and methods in mathematical physics. The Siegfried Prössdorf memorial volume, Birkhäuser; 2001. p. 194–220.
[19] Hackbusch W, Khoromskij B, Sauter S. On $\mathscr{H}^2$-matrices. In: Bungartz H, Hoppe R, Zenger C, editors. Lectures on applied mathematics. ; springer-volag, Berlin; 2000. p. 9–29.
[20] Hackbusch W, Nowak ZP. On the fast matrix multiplication in the boundary element method by panel clustering. Numerische Mathematik 1989;54:463–91.
[21] Khoromskij B. Data-sparse approximate inverse in elliptic problems: Green's function approach. Technical report 79, Leipzig, Germany: Max Planck Institute for Mathematics in the Sciences; 2001.
[22] Penzl T. Eigenvalue decay bounds for solutions of Lyapunov equations: the symmetric case. Syst Control Lett 2000;40: 139–44.
[23] Roberts JD. Linear model reduction and solution of the algebraic Riccati equation by use of the sign function. Int J Control 1980;32: 677–87.
[24] Rosen JIG, Wang C. A multilevel technique for the approximate solution of operator Lyapunov and algebraic Riccati equations. Siam J Numer Anal 1995;32:514–41.
[25] Sauter S. Variable order panel clustering. Computing 2000;64: 223–61.
[26] Tyrtyshnikov E. Mosaic-skeleton approximation. Calcolo 1996;33: 47–57.