Received: 3 November 2021 | Revised: 18 March 2022 | Accepted: 7 April 2022

DOI: 10.1002/nla.2448

RESEARCH ARTICLE

WILEY

Hierarchical adaptive low-rank format with applications to discretized partial differential equations

Stefano Massei[1] | Leonardo Robol[1] | Daniel Kressner[2]

[1]Department of Mathematics, University of Pisa, Pisa, Italy

[2]EPF Lausanne, Lausanne, Switzerland

Correspondence
Stefano Massei, Department of Mathematics, University of Pisa, Pisa, Italy.
Email: stefano.massei@unipi.it

Funding information
The work of Stefano Massei has been partially supported by the SNSF research project *Fast algorithms from low-rank updates*, grant number: 200020_178806.

Abstract
A novel framework for hierarchical low-rank matrices is proposed that combines an adaptive hierarchical partitioning of the matrix with low-rank approximation. One typical application is the approximation of discretized functions on rectangular domains; the flexibility of the format makes it possible to deal with functions that feature singularities in small, localized regions. To deal with time evolution and relocation of singularities, the partitioning can be dynamically adjusted based on features of the underlying data. Our format can be leveraged to efficiently solve linear systems with Kronecker product structure, as they arise from discretized partial differential equations (PDEs). For this purpose, these linear systems are rephrased as linear matrix equations and a recursive solver is derived from low-rank updates of such equations. We demonstrate the effectiveness of our framework for stationary and time-dependent, linear and nonlinear PDEs, including the Burgers' and Allen–Cahn equations.

KEYWORDS
hierarchical matrices, low-rank approximation, Sylvester equations, time-dependent structures

1 | INTRODUCTION

Low-rank based data compression can sometimes lead to a dramatic acceleration of numerical simulations. A striking example is the solution of two-dimensional elliptic partial differential equations (PDEs) on rectangular domains with smooth source terms. In this case, the (structured) discretization of the source term and the solution lead to matrices that allow for excellent low-rank approximations. Under suitable assumptions on the differential operator, one can recast the corresponding discretized PDE as a matrix equation.[1,2] In turn, this yields the possibility to facilitate efficient algorithms for matrix equations with low-rank right-hand side.[3,4] However, in many situations of interest, the smoothness property is not present in the whole domain. A typical instance is solutions that feature singularities along curves, while being highly regular elsewhere. This renders a global low-rank approximation ineffective. Adaptive discretization schemes, such as the adaptive finite element method, are one way to handle such situations. In this work, we will focus on a purely algebraic approach.

During the last decades, there has been significant effort in developing hierarchical low-rank formats that apply low-rank approximation only locally. These formats recursively partition the matrix into blocks that are either represented as a low-rank matrix or are sufficiently small to be stored as a dense matrix. Examples of such formats include block low-rank matrices,[5] $\mathcal{H}$-matrices,[6] $\mathcal{H}^2$-matrices,[7] HODLR matrices,[8] HSS matrices,[9] as well as SMASH.[10,11] These techniques are often applied in the context of matrices that arise from the discretization of operators and are known a priori to feature low-rank off-diagonal blocks. The use of these formats for representing the solution of a linear system, instead of the operator, has also been proposed in a number of works, such as References 12 and 13. Many of these techniques work best if the location of low-rank blocks is known. For example, when the matrix arises from the discretization of an integral operator with a singular kernel, the location of the singularities can be used to define a so called admissibility criterion[6-8] to decide a priori which blocks admit a good low-rank approximation. This can make these formats too inflexible to treat time-dependent problems for which the region of non-smoothness evolves over time. In the context of tensors, it has been recently proposed a bottom-up approach to identify a partitioning of the domain and perform a piecewise compression of a target tensor by means of local high-order singular value decompositions.[14] A very different and promising approach proceeds by forming high-dimensional tensors from a quantization of the function and applying the so called QTT compression format; see Reference 15 and the references therein.

In this article, we propose a new framework of structured matrices that automatically adapts the choice of the hierarchical partitioning and the location of the low-rank blocks without requiring the use of an admissibility criterion. While the representation format itself is derived from a minor variation of $\mathcal{H}$-matrices, the admissibility is decided on the fly by the success or failure of low-rank approximation techniques. To emphasize this adaptive choice, we use *hierarchical adaptive low-rank* (HALR) matrices to refer to our framework. Note that we keep the ordering of the matrix A fixed, that is, a suitable ordering of its indices needs to be performed a priori.

This work focuses on the application of HALR matrices to the following class of time-dependent PDEs:

$$\begin{cases} \frac{\partial u}{\partial t} = Lu + f(t, u, \nabla u), & (x, y) \in \Omega,\ t \in [0, T_{\max}], \\ u(x, y, 0) = u_0(x, y), \end{cases} \tag{1}$$

where $\Omega \subset \mathbb{R}^2$ is a rectangular domain, L is a linear differential operator, f is nonlinear, and (1) is coupled with appropriate boundary conditions in space. Discretizing (1) in time with the IMEX Euler method[16] and in space with, for example, finite differences leads to

$$(I - \Delta t L_n)\mathbf{u}_{n,\ell+1} = \mathbf{u}_{n,\ell} + \Delta t(\mathbf{f}_{n,\ell} + \mathbf{b}_{n,\ell}), \tag{2}$$

where L_n represents the discretization of the operator L, $\mathbf{u}_{n,\ell}$ and $\mathbf{f}_{n,\ell}$ are the discrete counterparts of u and f at time $t_\ell := \ell\Delta t$, $\ell \in \mathbb{N}$, and $\mathbf{b}_{n,\ell}$ accounts for the boundary conditions. When using finite differences on a tensor grid, it is natural to reshape the vectors $\mathbf{u}_{n,\ell}, \mathbf{f}_{n,\ell}, \mathbf{b}_{n,\ell}$ into matrices $U_{n,\ell}, F_{n,\ell}, B_{n,\ell}$. In our examples, the matrix L_n will often take the form $L_n = I \otimes A_{1,n} + A_{2,n} \otimes I$, a structure that is sometimes referred as having splitting-rank 2[2] and which allows to rephrase the linear system (2) as a linear matrix equation.

As a more specific guiding example, let us consider the two-dimensional Burgers' equation over the unit square:

$$\frac{\partial u}{\partial t} = K\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) - u \cdot \left(\frac{\partial u}{\partial x} + \frac{\partial u}{\partial y}\right), \quad (x, y) \in \Omega := (0, 1)^2, \tag{3}$$

with $K > 0$. Under suitably chosen boundary conditions, the solution of (3) is given by $u(x, y, t) = \left[1 + \exp\left(\frac{x+y-t}{2K}\right)\right]^{-1}$; see Reference 17, Example 3. For a fixed time t, the snapshot $u_t := u(\cdot, \cdot, t)$ describes a transition between two levels across the line $x + y = t$. For a small coefficient K, the transition becomes quite sharp, see Figure 1.

Let $U^{\mathrm{sol}}_{n,\ell}$ be the matrix collecting the samples of u_{t_ℓ} on an equispaced 2D lattice; $U^{\mathrm{sol}}_{n,\ell}$ has a time dependent rank structure. More specifically, the submatrices of $U^{\mathrm{sol}}_{n,\ell}$ corresponding to subdomains which are far away from $x + y = t_\ell$ are numerically low-rank because they contain samples of a smooth function over a rectangular domain; see the lower part of Figure 1. Therefore, an efficient representation strategy for the solution of (3) needs to adapt the block low-rank structure of $U^{\mathrm{sol}}_{n,\ell}$ according to ℓ.

In this work, we develop techniques for:

(i) Computing a HALR representation for the discretization of a function explicitly given in terms of a black-box evaluation function.
(ii) Solving the linear system (2) by exploiting the HALR structure in the right-hand side and the decomposition $L_n = I \otimes A_{1,n} + A_{2,n} \otimes I$.

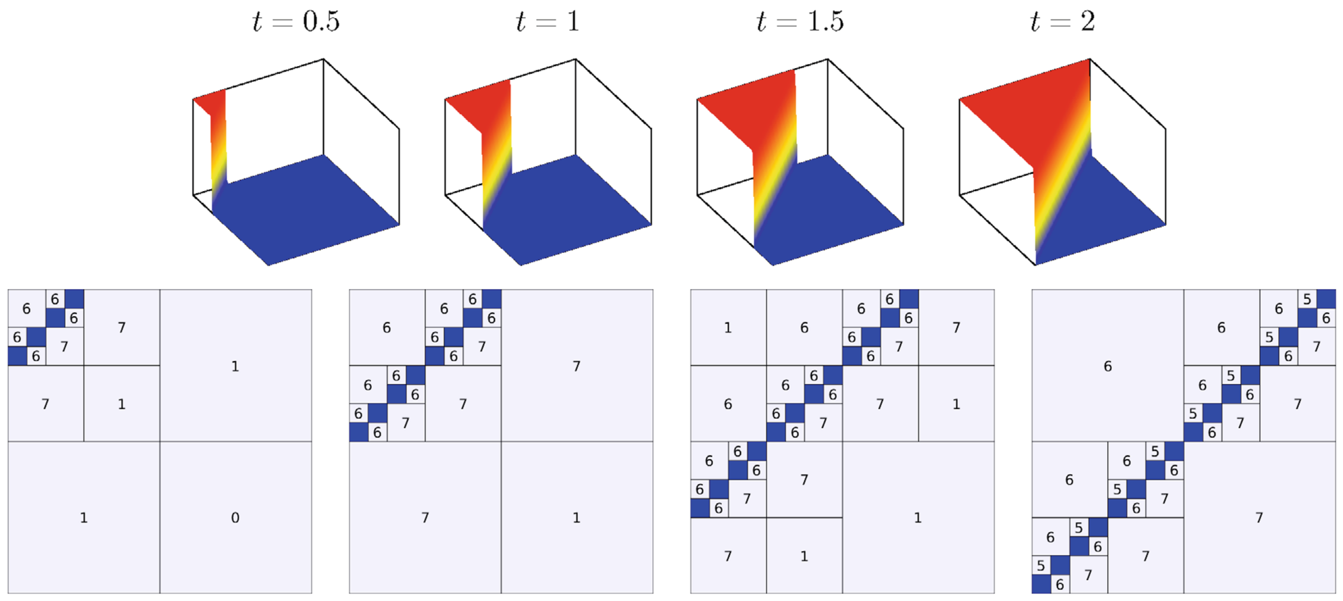


FIGURE 1 Top: Snapshots of $u(x, y, t) = \left[1 + \exp\left(\frac{x+y-t}{2K}\right)\right]^{-1}$ for $t = 0.5, 1, 1.5, 2$ and $K = 0.001$. Bottom: Corresponding block low-rank structure of $U^{\text{sol}}_{n,\ell}$ for $n = 4096$; the numbers indicate the rank of the corresponding block while full rank blocks are colored in blue.

Task (i) yields structured representations for the initial condition $\mathbf{u}_{n,0}$ and the source term $\mathbf{f}_{n,\ell}$. Taken together, tasks (i) and (ii) allow to efficiently compute the matricized solution $U_{n,\ell+1}$ of (2). The assumption on the discretized operator in (ii) is satisfied for $L = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$ and enables us to rephrase (2) as the matrix equation

$$\left(\frac{1}{2}I - \Delta t A_{1,n}\right) U_{n,\ell+1} + U_{n,\ell+1}\left(\frac{1}{2}I - \Delta t A_{2,n}^T\right) = \Delta t F_{n,\ell} + U_{n,\ell} + B_{n,\ell}. \tag{4}$$

The article is organized as follows; in Section 2, we introduce HALR matrices and discuss their arithmetic. Section 2.4 focuses on solving matrix equations of the form (4) where the right-hand side is represented in the HALR format. There, we propose a divide-and-conquer method whose cost scales comparably to the memory resources used for storing the right-hand side. In Section 3, we address the problems of constructing and adapting HALR representations. In particular, Section 3.3 considers the following scenario: given a parameter maxrank, determine the partitioning that provides the biggest reduction of the storage cost and uses low-rank blocks of rank bounded by maxrank. In Section 4, we incorporate HALR matrices into integration schemes for PDEs, and we perform numerical tests that demonstrate the computational benefits of our approach. Conclusions are drawn in Section 5.

1.1 | Notation

To simplify the statements of some definitions, we introduce the following compact notation for intervals of consecutive integers:

$$[\![i_l, i_r]\!] := \{i_l, i_l + 1, \dots, i_r\} \subseteq \mathbb{N}, \quad \text{for } 0 < i_l \leq i_r.$$

In addition, we write $[\![i_l, i_r]\!] < [\![i'_l, i'_r]\!]$ if $i_r < i'_l$ and we use the symbol $\sqcup$ to indicate the union of disjointed sets.

2 | HALR

We are concerned with matrix partitioning described by quad-trees, that is, trees with four branches at each node. More explicitly, given a matrix A we consider the block partitioning

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}, \quad (5)$$

where the blocks A_{ij} can be either dense blocks, low-rank matrices, or recursively partitioned. The cases of interest are those where large portions of the matrix are in low-rank form. This is in the spirit of well-established hierarchical low-rank formats such as $\mathcal{H}$-matrices[6] and $\mathcal{H}^2$-matrices.[7] To formalize our deliberations, we first provide the definition of a quad-tree cluster.

Definition 1. Let $m, n \in \mathbb{N}$. A tree $\mathcal{T}$ is called quad-tree cluster for $[\![1, m]\!] \times [\![1, n]\!]$ if

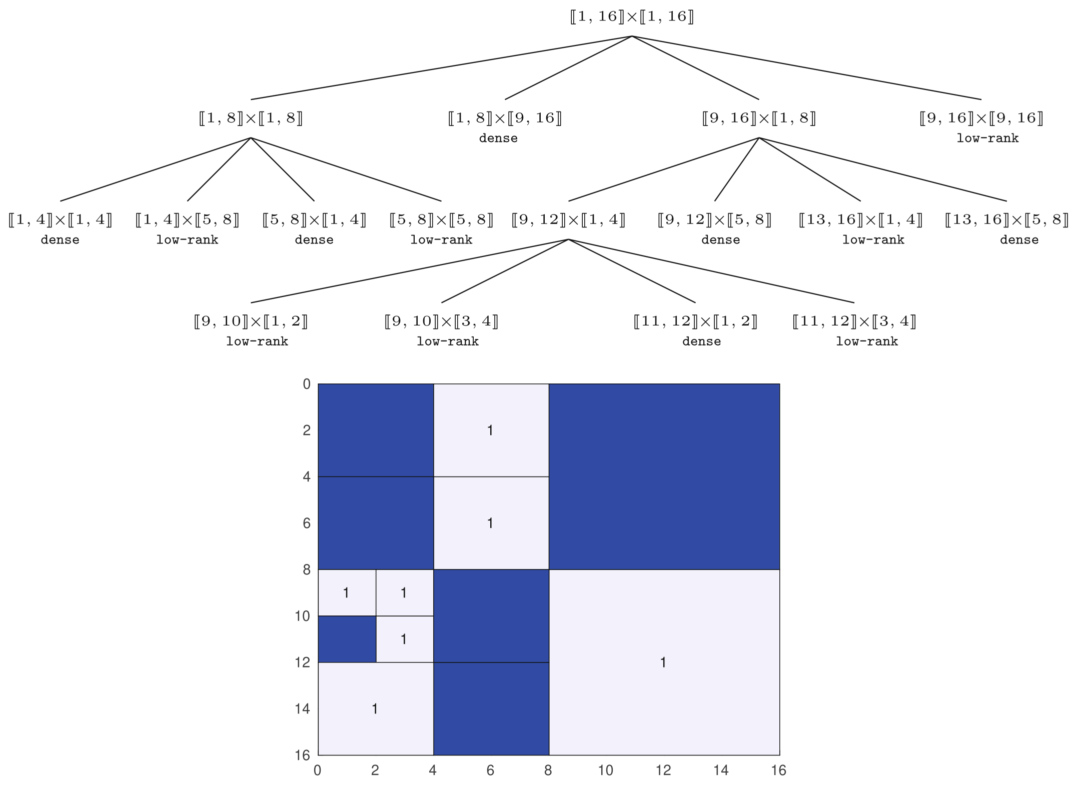
- The root node is $[\![1, m]\!] \times [\![1, n]\!]$.
- Each node I is a subset of $[\![1, m]\!] \times [\![1n]\!]$ of the form

$$I = I_r \times I_c := [\![m_1, m_2]\!] \times [\![n_1, n_2]\!].$$

- Each non-leaf node $I = I_r \times I_c$ has 4 children $I_{11}, I_{12}, I_{21}, I_{22}$, that are of the form $I_{ij} = I_{r_i} \times I_{c_j}$ such that $I_r = I_{r_1} \sqcup I_{r_2}$, $I_c = I_{c_1} \sqcup I_{c_2}$, and $I_{r_1} < I_{r_2}$, $I_{c_1} < I_{c_2}$.
- Each leaf node is labeled either as `dense` or `low-rank`.

The *depth* of $\mathcal{T}$ is the maximum distance of a node from the root; throughout this work we indicate depths with the lower case letter p with an appropriate subscript, if required by the context.

An example of a quad-tree cluster of depth 4 is given in Figure 2; this induces the block structure of a 16×16 matrix shown in the bottom part of the figure. This block structure is formalized in the following definition.



[[1, 16]]×[[1, 16]]
[[1, 8]]×[[1, 8]]
[[1, 8]]×[[9, 16]] dense
[[9, 16]]×[[1, 8]]
[[9, 16]]×[[9, 16]] low-rank
[[1, 4]]×[[1, 4]] dense
[[1, 4]]×[[5, 8]] low-rank
[[5, 8]]×[[1, 4]] dense
[[5, 8]]×[[5, 8]] low-rank
[[9, 12]]×[[1, 4]]
[[9, 12]]×[[5, 8]] dense
[[13, 16]]×[[1, 4]] low-rank
[[13, 16]]×[[5, 8]] dense
[[9, 10]]×[[1, 2]] low-rank
[[9, 10]]×[[3, 4]] low-rank
[[11, 12]]×[[1, 2]] dense
[[11, 12]]×[[3, 4]] low-rank
0 2 4 6 8 10 12 14 16
1

FIGURE 2 Example of a quad-tree cluster of depth 4 and the induced partitioning on the matrix. The leaf nodes labeled as `dense` correspond to dense blocks colored in blue. The leaf nodes labeled as `low-rank` are taken of rank 1 and correspond to the blocks colored in gray.

Definition 2. Let $A \in \mathbb{C}^{m \times n}$ and $\mathcal{T}$ be a quad-tree cluster for $[\![1, m]\!] \times [\![1, n]\!]$.

1. Given $k \in \mathbb{N}$, A is said to be a $(\mathcal{T}, k)$ HALR matrix, in short $(\mathcal{T}, k)$-HALR, if for every leaf node $I_r \times I_c$ of $\mathcal{T}$ labeled as `low-rank`, the submatrix $A(I_r, I_c)$ has rank at most k.
2. The smallest integer k for which A is $(\mathcal{T}, k)$-HALR is called the $\mathcal{T}$-HALR rank of A.

There are close connections between $(\mathcal{T}, k)$-HALR matrices and $\mathcal{H}$-matrices.[6] More precisely, any $\mathcal{H}$-matrix with low-rank blocks of rank at most k and with binary row and column cluster trees is a $(\mathcal{T}, k)$-HALR matrix. In this case, the quad-tree cluster is obtained from the Cartesian product of the row and column cluster trees. The HODLR format[8] is a special case discussed in more detail in Section 2.3. Compared to $\mathcal{H}$-matrices, Definition 1 is—in principle—more flexible by allowing for quad-tree clusters that cannot be written as subsets of any Cartesian product of a row and a column cluster trees. Note, however, that we will make little use of this flexibility in the following. In particular, the constructions described in Section 3 always lead to HALR matrices that are also $\mathcal{H}$-matrices.

In the next sections, we will describe operations involving HALR matrices and we will tacitly assume to have access to their structured representations, that is, the quad-tree clusters and the low-rank factors of the `low-rank` leaves. How to retrieve the HALR representation of a given matrix will be discussed in Section 3.

2.1 | Matrix-vector product

In complete analogy with the $\mathcal{H}$-matrix arithmetic, the HALR structure allows to perform the matrix-vector product efficiently by relying on the block-recursive procedure described in Algorithm 1.

Algorithm 1. Matrix-vector product with an HALR matrix A

```
1: procedure HALR_MatVec($A$, $v$)
2:     if $A$ is a leaf node then
3:         return $Av$                                  ▷ Exploiting low-rank structure if present
4:     else
5:         Partition $v = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$
6:         return $\begin{bmatrix} \text{HALR\_MatVec}(A_{11}, v_1) + \text{HALR\_MatVec}(A_{12}, v_2) \\ \text{HALR\_MatVec}(A_{21}, v_1) + \text{HALR\_MatVec}(A_{22}, v_2) \end{bmatrix}$
7:     end if
8: end procedure
```

In the particular case when the cluster only contains the root, A itself is either `low-rank` (of rank k) or `dense` and Algorithm 1 requires $\mathcal{O}((m+n)k)$ and $\mathcal{O}(mn)$ flops, respectively. We remark that this cost corresponds to the memory required for storing A. This statement holds in more generality.

Lemma 1. *Let $A \in \mathbb{C}^{m \times n}$ be a $(\mathcal{T}, k)$-HALR, and $v \in \mathbb{C}^n$ a vector. Computing Av by Algorithm 1 requires $\mathcal{O}(S)$ flops, where S is the memory required to store A.*

Proof. The result is shown by induction on the depth of $\mathcal{T}$. By the discussion above, the claim is true when $\mathcal{T}$ consists of a single node. If the result holds for trees of depth up to d, and $\mathcal{T}$ has depth $d+1$, the cost for Av is dominated by the cost of the 4 recursive calls to HALR_MatVec. Using the induction assumption, it follows that the cost for these calls sums up to $\mathcal{O}(S)$. ■

2.2 | Arithmetic operations

We proceed by analyzing the interplay between the quad-tree cluster partitioning and the usual matrix operations. If A is a given $(\mathcal{T}, k)$-HALR we can define the transpose of $\mathcal{T}$ as the natural cluster tree for A^T.

Definition 3. Let $m, n \in \mathbb{N}$, and $\mathcal{T}$ a quad-tree cluster for $[\![1, m]\!] \times [\![1, n]\!]$. The transposed quad-tree cluster $\mathcal{T}^T$ is defined as the quad-tree cluster on $[\![1, n]\!] \times [\![1, m]\!]$ obtained from $\mathcal{T}$ by:

(i) replacing each node $I_r \times I_c$ with $I_c \times I_r$,
(ii) swapping the subtrees at I_{12} and I_{21} for every non-leaf node.

Clearly, A is $(\mathcal{T}, k)$-HALR if and only if A^T is $(\mathcal{T}^T, k)$-HALR.

Remark 1. In the following, we want to regard a subtree of $\mathcal{T}$ again as a quad-tree cluster. For such a subtree to satisfy Definition 1, we tacitly shift its root $[\![m_1, m_2]\!] \times [\![n_1, n_2]\!]$ to $[\![1, m_2 - m_1]\!] \times [\![1, n_2 - n_1]\!]$ and, analogously, all other nodes in the subtree. In the opposite direction, when connecting a tree to a leaf of $\mathcal{T}$, we shift the root (and the other nodes) of the tree such that it matches the index set of the leaf.

Now, let us focus on arithmetic operations between HALR matrices. When dealing with binary operations, we need to ensure some compatibility between the sizes of the hierarchical partitioning in order to unambiguously define the partitioning of the result. To this aim, we introduce the notions of row and column compatibility, which will be used in the next section for characterizing matrix products and additions.

Definition 4. Given $m_A, n_A, m_B, n_B \in \mathbb{N}$, let $\mathcal{T}_A, \mathcal{T}_B$ be quad-tree clusters for $[\![1, m_A]\!] \times [\![1, n_A]\!]$ and $[\![1, m_B]\!] \times [\![1, n_B]\!]$, respectively.

- $\mathcal{T}_A$ and $\mathcal{T}_B$, with roots I_A and I_B, are said to be row-compatible if one of the following two conditions are satisfied:
  (i) $\mathcal{T}_A$ or $\mathcal{T}_B$ only contains the root, and $m_A = m_B$.
  (ii) For every $i, j = 1, 2$ the subtrees at $(I_A)_{ij}$ and $(I_B)_{ij}$ are row-compatible.
- $\mathcal{T}_A$ and $\mathcal{T}_B$ are said column-compatible if $\mathcal{T}_A^T$ and $\mathcal{T}_B^T$ are row-compatible.
- $\mathcal{T}_A$ and $\mathcal{T}_B$ are said *compatible* if they are both row- and column-compatible.

Intuitively, two quad-trees $\mathcal{T}_A$ and $\mathcal{T}_B$ are row (resp. column) compatible if taking the same path in $\mathcal{T}_A$ and $\mathcal{T}_B$ yields index sets with the same number of row (resp. column) indices.

According to Definition 4, compatibility does not depend on the labeling of the leaf nodes. Moreover, two clusters can be compatible even if they have different depths (or contain subtrees of different depths). The following definition introduces a partial ordering among compatible trees. This will be used to define the intersection between quad-tree clusters, which in turn allows us to characterize the natural partitioning of binary matrix operations involving A and B.

Definition 5. Let $\mathcal{T}_A, \mathcal{T}_B$ be compatible quad-tree clusters for $[\![1, m]\!] \times [\![1, n]\!]$. We write $\mathcal{T}_A \leq \mathcal{T}_B$ if one of the following conditions is satisfied

(i) $\mathcal{T}_A$ only contains the root labeled as `low-rank`.
(ii) $\mathcal{T}_B$ only contains the root labeled as `dense`.
(iii) For every $i, j = 1, 2$ the subtrees $(\mathcal{T}_A)_{ij}$ and $(\mathcal{T}_B)_{ij}$ at $(I_A)_{ij}$ and $(I_B)_{ij}$, respectively, verify $(\mathcal{T}_A)_{ij} \leq (\mathcal{T}_B)_{ij}$.

The idea behind Definition 5 is that $\mathcal{T}_A \leq T_B$ implies that a $(\mathcal{T}_A, k)$-HALR matrix has a stronger structure than an $(\mathcal{T}_B, k)$-HALR one. In fact, any $(\mathcal{T}_A, k)$-HALR is also a $(\mathcal{T}_B, k)$-HALR for all $\mathcal{T}_B \geq \mathcal{T}_A$. A low-rank matrix itself corresponds to the format with the strongest structure. Based on this, we define the intersection between $\mathcal{T}_A$ and $\mathcal{T}_B$ as the strongest structure among the ones which are weaker than both $\mathcal{T}_A$ and $\mathcal{T}_B$.

Definition 6. Let $\mathcal{T}_A, \mathcal{T}_B$ be compatible quad-tree clusters for $[\![1, m]\!] \times [\![1, n]\!]$. Their intersection $\mathcal{T} := \mathcal{T}_A \cap \mathcal{T}_B$ is defined recursively as follows:

(i) If $\mathcal{T}_A$ (resp. $\mathcal{T}_B$) only contain the root labeled as `low-rank` then $\mathcal{T}_A \cap \mathcal{T}_B = \mathcal{T}_B$ (resp. $\mathcal{T}_A \cap \mathcal{T}_B = \mathcal{T}_A$).
(ii) If $\mathcal{T}_A$ or $\mathcal{T}_B$ only contain the root labeled as `dense` then $\mathcal{T}_A \cap \mathcal{T}_B$ is a tree that only contains the root labeled as `dense`.
(iii) If $\mathcal{T}_A$ and $\mathcal{T}_B$ contain more than one node then their intersection is constructed by connecting the subtrees $\mathcal{T}_{ij} = (\mathcal{T}_A)_{ij} \cap (\mathcal{T}_B)_{ij}$, $i, j = 1, 2$, to the root $I = I_A = I_B$.

Remark 2. The neutral element for the intersection is given by the quad-tree $\mathcal{T}$ only containing the root labeled as `low-rank`, that is, a low-rank matrix.

We now make use of the notions defined above to infer the structure of $A + B$ from the ones of A and B.

Lemma 2. *Let $A, B \in \mathbb{C}^{m\times n}$ be $(\mathcal{T}_A, k_A)$-HALR and $(\mathcal{T}_B, k_B)$-HALR, respectively. If $\mathcal{T}_A, \mathcal{T}_B$ are compatible then $A + B$ is $(\mathcal{T}_A \cap \mathcal{T}_B, k_A + k_B)$-HALR.*

Proof. We recall that the sum of two matrices of rank at most k_A and k_B, respectively, has rank at most $k_A + k_B$. The statement follows from traversing the tree $T_A \cap \mathcal{T}_B$; for every leaf in the tree for which both submatrices of A and B are low rank, the resulting submatrix in $A + B$ will have rank at most $k_A + k_B$. ■

It is instructive to consider two special cases. First, if $\mathcal{T}_A = \mathcal{T}_B$, then $A + B$ shares the same quad-tree cluster (with higher rank). Second, in view of Remark 2, if A is low rank then $A + B$ has the same structure as B, with a rank increase by (at most) the rank of A.

The proof of Lemma 2 suggests a recursive procedure that is summarized in Algorithm 2. An inductive argument analogous to the one used for Lemma 1 shows that the complexity of Algorithm 2 is bounded by two times the cost of storing a $(\mathcal{T}_A \cap \mathcal{T}_B, k_A + k_B)$-HALR matrix. Note that this estimate can be reduced by exploiting the fact that Line 5 is executed at no cost by simply appending the low-rank factors of A, B. For example, when A is a rank-k_A matrix the cost reduces to k_A times the number of entries in the dense blocks of B, which equals the storage needed for a $(\mathcal{T}_B, 0)$-HALR matrix

Algorithm 2. Sum of HALR matrices

```
1:  procedure HALR_SUM($A$, $B$)
2:      if $A$ and/or $B$ are leaf nodes labeled as dense then
3:          return the dense matrix $A + B$
4:      else if $A$ and $B$ are leaf nodes labeled as low-rank then
5:          return a low-rank factorization of $A + B$
6:      else
7:          If $A$ (resp. $B$) is a low-rank leaf, partition it according to $B$ (resp. $A$)
8:          return $\begin{bmatrix} \text{HALR\_SUM}(A_{11}, B_{11}) & \text{HALR\_SUM}(A_{12}, B_{12}) \\ \text{HALR\_SUM}(A_{21}, B_{21}) & \text{HALR\_SUM}(A_{22}, B_{22}) \end{bmatrix}$
9:      end if
10: end procedure
```

For a matrix product $A \cdot B$ of HALR matrices, it is natural to assume that A^T and B are row compatible. Assuming that $\mathcal{T}_A$ and $\mathcal{T}_B$ denote, as usual, the quad-tree clusters of A and B, the matrix product $A \cdot B$ stored in the HALR format is computed with the following procedure:

(i) If $\mathcal{T}_A$ (resp. $\mathcal{T}_B$) only contains the root labeled as `low-rank`, then the resulting tree only contains the root labeled as `low-rank` and its factorization is obtained efficiently by applying B on the right low-rank factor of A (resp. the analogous procedure for B); otherwise

(ii) If $\mathcal{T}_A$ (resp. $\mathcal{T}_B$) only contains the root labeled as `dense`, then $A \cdot B$ is computed by applying B on each row of A, and the resulting tree only contains the root labeled as `dense`; otherwise

(iii) We partition

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}, \quad B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}, \quad C = AB = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix},$$

determine recursively $A_{ik}B_{kj}$ along with their clusters $\mathcal{T}_{ijk}$ for $i, j, k = 1, 2$, and set $C_{ij} = A_{i1}B_{1j} + A_{i2}B_{2j}$ with cluster $\mathcal{T}_{ij1} \cap \mathcal{T}_{ij2}$.

We note that it is difficult to predict a priori the quad-tree cluster of AB because even if A and B contain many low-rank blocks, the structure may be completely lost in AB; see the example reported in Figure 3. Also computing $A \cdot B$ may cost significantly more than the storage cost of the outcome, for example, when A and B are dense matrices. On the other

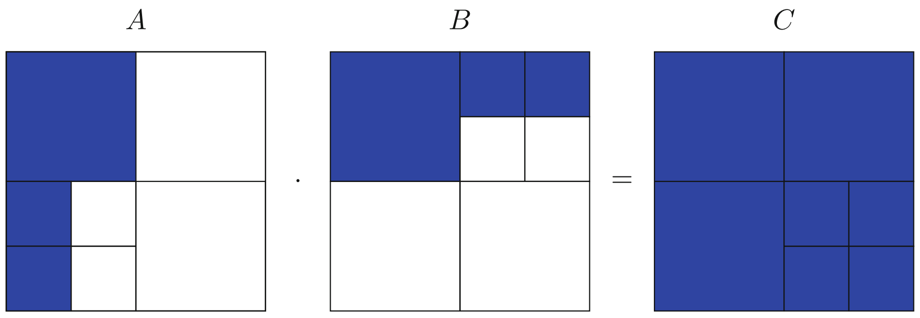


FIGURE 3 Example of loss of structure when computing the matrix–matrix multiplication. The blue region correspond to nodes labeled as `dense`, and the empty regions to nodes labeled as `low-rank`.

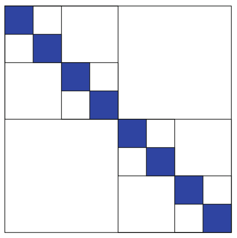


FIGURE 4 Example of partitioning induced by a HODLR cluster tree of depth 3

hand, in Section 2.3, we will show that if one of the two factors happens to be a HODLR matrix then the cost of computing $A \cdot B$ and its quad-tree structure are predictable.

2.3 | HODLR matrices

HODLR matrices are special cases of HALR matrices; all the off-diagonal blocks have low rank. To formalize this notion, we adopt the definition given in Reference 18, rephrased in the formalism of quad-tree clusters.

Definition 7. A quad-tree cluster $\mathcal{T}_p^{(H)}$ of depth p is said to be a HODLR cluster tree if either $p = 1$ and $\mathcal{T}_p^{(H)}$ only contains the root labeled as `dense`, or if the children I_{ij} at the root of $\mathcal{T}_p^{(H)}$ satisfy:

- I_{12} and I_{21} are leaf nodes labeled as `low-rank`.
- The subtrees at I_{11} and I_{22} are HODLR cluster trees of depth $p-1$.

We say that a matrix is $(\mathcal{T}_p^{(H)}, k)$-HODLR if it is $(\mathcal{T}_p^{(H)}, k)$-HALR. The smallest integer k for which a matrix A is $(\mathcal{T}_p^{(H)}, k)$-HODLR is called the HODLR rank of A.

An example of a HODLR cluster tree is reported in Figure 4. A crucial property of HODLR matrices is that they are block diagonal up to a low-rank correction. This allows to predict the structure of a product of HALR matrices whenever one of the factors is, in fact, a HODLR matrix.

Lemma 3. *Let $A \in \mathbb{C}^{m\times m}$ be a $(\mathcal{T}_{p_A}^{(H)}, k_A)$-HODLR matrix and $B \in \mathbb{C}^{m\times n}$ be a $(\mathcal{T}, k_B)$-HALR matrix of depth p_B. If A^T and B are row compatible and $p_A \geq p_B$, then $A \cdot B$ is a $(\mathcal{T}, k_B + (p_A - 1) \cdot k_A)$-HALR matrix. Similarly, if C is an $n \times n$ $(\mathcal{T}_{p_C}^{(H)}, k_C)$-HODLR matrix and $p_C \geq p_B$ then $B \cdot C$ is a $(\mathcal{T}, k_B + (p_C - 1) \cdot k_C)$-HALR matrix.*

Proof. We prove only the first statement, the second can be obtained by transposition. We proceed by induction on p_A; if $p_A = 1$, then A is composed of a single dense block. Since $p_B \leq p_A$, B is also composed of a single block, either labeled as `low-rank` or `dense`. Both structures are preserved when multiplying with A.

Suppose that the claim is valid for $p_A - 1 \geq 1$. If $\mathcal{T}$ is composed of a single node, the claim is valid. Otherwise, by decomposing A in its diagonal and off-diagonal parts, we may write

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \underbrace{\begin{bmatrix} A_{11}B_{11} & A_{11}B_{12} \\ A_{22}B_{21} & A_{22}B_{22} \end{bmatrix}}_{M_D} + \underbrace{\begin{bmatrix} A_{12}B_{21} & A_{12}B_{22} \\ A_{21}B_{11} & A_{21}B_{12} \end{bmatrix}}_{M_O}. \tag{6}$$

In view of the induction step, each block of M_D is a $(\mathcal{T}_{ij}, k_B + (p_A - 2)k_A)$-HALR matrix, for $i,j = 1,2$, where $\mathcal{T}_{ij}$ are the quad-tree clusters associated with the first level of $\mathcal{T}$. In particular, M_D is $(\mathcal{T}, k_B + (p_A - 2)k_A)$-HALR. Finally, note that all the blocks of M_O have rank bounded by k_A, and therefore M_O is $(\mathcal{T}, k_A)$-HALR. We conclude that $AB = M_D + M_O$ is $(\mathcal{T}, k_B + (p_A - 1)k_A)$-HALR. ■

The complexity of multiplying by a HODLR matrix can be bounded in terms of the storage of the other factor.

Lemma 4. *Under the assumptions of Lemma 3 the cost of computing $A \cdot B$ is $\mathcal{O}(S(n_{\min} + k_A(p_A - 1)))$ where S is the storage cost of B and $n_{\min}$ is an upper bound on the size of the dense diagonal blocks of A.*

Proof. For $p_A = 1$, A is a square matrix of size at most $n_{\min}$ while B has low rank or is dense. In both cases, it directly follows that the cost of multiplication is $\mathcal{O}(Sn_{\min})$.

For the induction step, we recall the splitting (6) of $A \cdot B$ into two terms M_D and M_O. The term M_O is a product between B and a matrix of rank (at most) $2k_A$, which, according to Lemma 1, requires $c_v S k_A$ operations for some constant c_v. The term M_D consists of four products $A_{ii}B_{ij}$, where A_{ij} is a HODLR matrix of depth $p_A - 1$ and B_{ij} is a HALR matrix. By induction, there is a constant $c \geq c_v + 2$ such that the cost for each of these four multiplications is bounded by $cS_{ij}(n_{\min} + k_A(p_A - 2))$ operations, where S_{ij} denotes the storage cost of B_{ij}. Adding the corresponding rank-k_A submatrix of M_O requires at most $2S_{ij}k_A$ operations, as discussed after Lemma 2. Therefore, the cost for computing the block (i,j) of the product $A \cdot B$ is bounded by $cS_{ij}(n_{\min} + k_A(p_A - 1))$. Summing over i,j concludes the proof. ■

Remark 3. When performing arithmetic operations between HALR matrices, or HODLR and HALR matrices, it is often observed that the numerical rank of the blocks in the outcome is significantly less than the worst case scenario depicted in Lemmas 2 and 4. Hence, it is advisable to perform a recompression stage, see Reference 6, Algorithm 2.17, p. 33, when expanding low-rank factorizations, such as in line 5 of Algorithm 2.

2.4 | Solving Sylvester equations with HODLR coefficients and HALR right-hand side

As pointed out in Section 1, when dealing with PDEs defined on a rectangular two-dimensional domain, one frequently encounters linear matrix equations of the form

$$AX + XB = C, \tag{7}$$

with square matrices A, B and a right-hand side C of matching size. To simplify the discussion, we will assume that A and B are of equal size n. As A, B stem from the discretization of a 1D differential operator, they are typically $(\mathcal{T}_p^{(H)}, k)$-HODLR for some small k. In contrast to our previous work,[18] where we assumed C to be HODLR as well, we now consider the more general setting when C is $(\mathcal{T}, k_C)$-HALR. In the following, we require that $\mathcal{T}_p^{(H)}$ is compatible with $\mathcal{T}$ and $p \geq p_C$, where p_C denotes the depth of $\mathcal{T}$.

The particular case when C is a dense matrix will be discussed in further detail in Section 2.4.1. For the moment, we let DENSERHS_SYLV denote the algorithm chosen for this case. If, instead, C is low-rank, well-studied low-rank solvers are available, such as Krylov subspace methods and ADI (see Reference 19 for a survey). Under suitable conditions on the spectra of A and B and given a low-rank factorization of C, these solvers return an approximation to the solution X in factorized low-rank format. Since the specific choice of the low-rank solver is not crucial for the following discussion, we refer to this routine as LOWRANKRHS_SYLV.

Equation (7) can be solved recursively using an extension of our divide-and-conquer approach[18] for HODLR matrices C. If $\mathcal{T}$ only contains the root and, hence, C is composed of a single block, we use either DENSERHS_SYLV (if C is `dense`) or LOWRANKRHS_SYLV (if C is `low-rank`). Otherwise, we partition (7) according to the four children of the root of $\mathcal{T}$:

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} X_{11} & X_{12} \\ X_{21} & X_{22} \end{bmatrix} + \begin{bmatrix} X_{11} & X_{12} \\ X_{21} & X_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}.$$

In the spirit of Reference 18, we first solve the equation associated with the diagonal blocks of A and B:

$$\begin{bmatrix} A_{11} & 0 \\ 0 & A_{22} \end{bmatrix} \begin{bmatrix} \tilde{X}_{11} & \tilde{X}_{12} \\ \tilde{X}_{21} & \tilde{X}_{22} \end{bmatrix} + \begin{bmatrix} \tilde{X}_{11} & \tilde{X}_{12} \\ \tilde{X}_{21} & \tilde{X}_{22} \end{bmatrix} \begin{bmatrix} B_{11} & 0 \\ 0 & B_{22} \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}, \tag{8}$$

which is equivalent to solving the four decoupled equations

$$A_{ii}\tilde{X}_{ij} + \tilde{X}_{ij}B_{jj} = C_{ij}, \quad i,j = 1,2, \quad \tilde{X} := \begin{bmatrix} \tilde{X}_{11} & \tilde{X}_{12} \\ \tilde{X}_{21} & \tilde{X}_{22} \end{bmatrix}, \tag{9}$$

where, by recursion, $\tilde{X}$ can be represented in the $\mathcal{T}$-HALR format. Letting $\delta X := X - \tilde{X}$ and subtracting (8) from (7), we obtain

$$A\delta X + \delta XB = -\begin{bmatrix} 0 & A_{12} \\ A_{21} & 0 \end{bmatrix} \tilde{X} - \tilde{X} \begin{bmatrix} 0 & B_{12} \\ B_{21} & 0 \end{bmatrix},$$

which is a Sylvester equation with right-hand side of rank at most $4k$. In turn, δX is computed using LowRankRHS_Sylv, and $X = \tilde{X} + \delta X$ is retrieved performing a low-rank update. Note that the Sylvester equations in (9) have again HODLR coefficients and HALR right-hand side, with the depth decreased by one. Applying this step recursively yields the divide-and-conquer scheme reported in Algorithm 3. Note that, the approximate solution returned by Algorithm 3 retains the HALR format, with the quad-tree cluster $\mathcal{T}$ inherited from C.

Algorithm 3. Divide-and-conquer approach for solving $AX + XB = C$

```
1: procedure D&C_Sylv($A, B, C$)
2:     if $p_C = 1$ then
3:         if $C$ is low-rank then
4:             return LowRankRHS_Sylv($A, B, C$)
5:         else
6:             return DenseRHS_Sylv($A, B, C$)
7:         end if
8:     else
9:         for $i, j = 1, 2$ do
10:            $\tilde{X}_{ij} \leftarrow$ D&C_Sylv($A_{ii}, B_{jj}, C_{ij}$)
11:        end for
12:        $\tilde{C} \leftarrow -\begin{bmatrix} 0 & A_{12} \\ A_{21} & 0 \end{bmatrix} \tilde{X} - \tilde{X} \begin{bmatrix} 0 & B_{12} \\ B_{21} & 0 \end{bmatrix}$
13:        $\delta X \leftarrow$ LowRankRHS_Sylv($A, B, \tilde{C}$)
14:        return $\begin{bmatrix} \tilde{X}_{11} & \tilde{X}_{12} \\ \tilde{X}_{21} & \tilde{X}_{22} \end{bmatrix} + \delta X$
15:    end if
16: end procedure
```

In practice, LowRankRHS_Sylv in lines 4 and 13 uses low-rank factorizations of the matrices C and $\tilde{C}$, and returns the solutions in factorized form. The low-rank factors of C at line 4 are given as C is a leaf node of an HALR matrix. At line 4 they are easily retrieved using the low-rank factorizations of the off-diagonal blocks of A and B that are stored in their HODLR representations; see Reference 18, Section 3.1 for more details. When X is assembled by its blocks in line 14, an HALR structure with the appropriate tree is created.

2.4.1 | Sylvester equation with dense right-hand side

We now consider the solution of a Sylvester equation (7) with dense C and HODLR coefficients A, B. This is needed in Line 6 of Algorithm 3, but it may also be of independent interest.

For small n (say, $n \leq 200$), it is most efficient to convert A and B to dense matrices, and use a standard dense solver, such as the Bartels–Stewart method or RECSY,[20] requiring $\mathcal{O}(n^3)$ operations.

For large n, we will see that it is more efficient to use a recursive approach instead of a dense solver. For this purpose, we partition C into a block matrix in accordance with the row partition of A and the column partition of B. More specifically, if the size of the minimal blocks in the partitioning of A and B is $n_{\min}$ and $n = 2^p n_{\min}$, we represent C as a $\frac{n}{n_{\min}} \times \frac{n}{n_{\min}}$ block matrix, that is, a $(\mathcal{T}, 0)$-HALR of depth p with all leaf nodes labeled as `dense`. Then (7) is solved recursively in analogy to Algorithm 3. The resulting procedure is summarized in Algorithm 4, where DENSESOLVER_SYLV indicates the standard dense solver.

2.4.2 | Complexity analysis of the D&C Sylvester solvers

In order to perform a complexity analysis we need to make a simplifying assumption on the convergence of the low-rank Sylvester solver, which usually depends on several features of the problem, such as the spectrum of A and B.

Algorithm 4. Solver of $AX + XB = C$ for a dense matrix C

```
1: procedure DENSERHS_SYLV($A, B, C$)
2:     if $p_A = p_B = 1$ then
3:         return DENSESOLVER_SYLV($A, B, C$)
4:     else
5:         Partition $C$ according to the partitioning of $A, B$:
6:         $C \leftarrow \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$    $C_{ij}$ dense for all $i, j$
7:         for $i, j = 1, 2$ do
8:             $\tilde{X}_{ij} \leftarrow$ DENSERHS_SYLV($A_{ii}, B_{jj}, C_{ij}$)
9:         end for
10:        $\tilde{C} \leftarrow -\begin{bmatrix} 0 & A_{12} \\ A_{21} & 0 \end{bmatrix} \tilde{X} - \tilde{X} \begin{bmatrix} 0 & B_{12} \\ B_{21} & 0 \end{bmatrix}$
11:        $\delta X \leftarrow$ LOWRANKRHS_SYLV($A, B, \tilde{C}$)
12:        return $\begin{bmatrix} \tilde{X}_{11} & \tilde{X}_{12} \\ \tilde{X}_{21} & \tilde{X}_{22} \end{bmatrix} + \delta X$
13:    end if
14: end procedure
```

Assumption 1. The computational cost of LOWRANKRHS_SYLV for $AX + XB = C$ is $\mathcal{O}(k_C kn \log n + k^2 n \log^2 n)$, where n is the size of A, B, and k their HODLR rank, and k_C is the rank of C. The rank of X is $O(k_C)$.

Assumption 1 is satisfied, for example, if the extended Krylov subspace method[21] converges to fixed (high) accuracy in $\mathcal{O}(1)$ iterations and the LU factors of A and B are HODLR matrices of HODLR rank $\mathcal{O}(k)$.[1] This requires the solution of a linear system with A and B in each iteration, via precomputing accurate approximations of the LU decompositions of A, B at the beginning with cost $\mathcal{O}(k^2 n \log^2 n)$. In other situations, for example, when the number of steps and/or the number of linear systems per step depend logarithmically on n in order to reach a fixed accuracy, Assumption 1 and the following discussion can be easily adjusted by adding $\log n$ factors.

Before analyzing the more general Algorithm 3, it is instructive to first focus on Algorithm 4. We note that Algorithm 4 solves $(\frac{n}{n_{\min}})^2$ dense Sylvester equations of size $n_{\min}$ and, at each level $j = 0, \ldots, p-1$, as well as 4^j Sylvester equations of size $\frac{n}{2^j}$ and with right-hand sides of rank at most $4k$. In addition, computing the low-rank factorization at line 10 requires $\mathcal{O}(\frac{n^2}{4^j}k)$ operations, amounting to a total cost of $\mathcal{O}(n^2)$. Under Assumption 1, LowRankRHS_Sylv solves the equations at level j with a cost bounded by $\mathcal{O}(\frac{k^2 n \log^2 n}{2^j})$. Hence, the total computational cost is $\mathcal{O}(n^2(n_{\min} + k^2\log^2 n))$. For large n and moderate k, we can therefore expect that Algorithm 4 is faster than a dense solver of complexity $\mathcal{O}(n^3)$.

The following lemma estimates the cost of Algorithm 3 for a general HALR matrix C, which reduces to our previous estimates in the two extreme cases: $\mathcal{O}(k_C kn \log n + k^2 n\log^2 n)$ if C is `low-rank` and $\mathcal{O}(n^2 n_{\min} + k^2 n^2 \log^2 n)$ algorithm if C is `dense`.

Lemma 5. *Consider the Sylvester equation $AX + XB = C$ with $(\mathcal{T}_p^{(H)}, k)$-HODLR matrices $A, B \in \mathbb{C}^{n\times n}$ and a $(\mathcal{T}, k_C)$-HALR matrix $C \in \mathbb{C}^{n\times n}$, with a quad-tree cluster $\mathcal{T}$ that is compatible with $\mathcal{T}_p^{(H)}$ and has depth $p_C \le p$. Suppose that $p \sim \log(n)$, and let $n_{\min}$ denote the size of minimal blocks in A, B. If Assumption 1 holds, then the cost of Algorithm 3 for computing the solution X is $\mathcal{O}(S(n_{\min} + k^2\log^2 n))$, where S is the storage required for C.*

Proof. We prove the result by induction on p_C. For $p_C = 1$, the result holds following the discussion above, because $S = n^2$ if C is `dense` and $S = 2k_C n$ if C is `low-rank`.

As the induction step is similar to the proof of Lemma 4, we will keep it briefer. When $p_C > 1$, Algorithm 3 consists of four stages:

1. Solution of $A_{ii}\tilde{X}_{ij} + \tilde{X}_{ij}B_{jj} = C_{ij}$, for $i, j = 1, 2$.
   By the induction hypothesis, each solve is $\mathcal{O}(S_{ij}(n_{\min} + k^2\log^2 n))$, where S_{ij} denotes the storage of C_{ij} and, hence, the total cost is $\mathcal{O}(S(n_{\min} + k^2\log^2 n))$.
2. Computation of the right-hand side in line 12.
   This computation involves $4k$ matrix-vector products with $\tilde{X}$. After $p-1$ recursive steps, the storage for $\tilde{X}$ is at most the one for C plus the one for storing the $p-1$ low-rank updates, which amounts to $\mathcal{O}(S + kpn)$ according to Assumption 1. Hence, by Lemma 1, the cost of this step is $\mathcal{O}(Sk + k^2 pn)$.
3. Solution of Sylvester equation in line 13.
   Because the rank of the right-hand side is bounded by $4k$, the cost of this step step is $\mathcal{O}(k^2 n\log^2 n)$.
4. Update of $\tilde{X}$ in line 14.
   The cost of this step is $\mathcal{O}(Sk + k^2 pn)$, the storage of $\tilde{X}$ times the rank of δX.

The total cost is dominated by the cost of Step 1, because one can easily prove by induction that pn is bounded by $\mathcal{O}(S)$. This completes the proof. ■

Note that it is not the HALR rank k_C but the storage cost S of the right-hand side C that appears explicitly in the complexity bound of Lemma 5. The advantage of using S instead of an upper bound induced by k_C is that it allows us to better explain why isolated relatively high ranks can still be treated efficiently.

We remark that when A and B are banded, for example, when they arise from the discretization of 1D differential operators, Algorithm 3 can be executed without computing the HODLR representations of A and B. Indeed, the low-rank factorizations of the off-diagonal blocks at line 12 are easily retrieved on the fly and one can implement a solver of Sylvester equations that exploits the sparse structure of A, B, in LowRankRhs_Sylv.

2.5 | HALR matrices in hm-toolbox

The hm-toolbox[22] available at https://github.com/numpi/hm-toolbox is a MATLAB toolbox for working conveniently with HODLR and HSS matrices via the classes hodlr and hss, respectively. We have added functionality for HALR matrices to the toolbox. A new class halr has been introduced, which stores a $(\mathcal{T}, k)$-HALR matrix A as an object with the following properties:

- `sz` is a 1×2 vector with the number of rows and columns of the matrix A.
- `F` contains a dense representation of A if it corresponds to a leaf node labeled as `dense`.
- `U`, `V` contain the low-rank factors of A if it corresponds to a leaf node labeled as `low-rank`.
- `admissible` is a Boolean flag that is set to true for leaf nodes labeled as `low-rank`.
- `A11`, `A21`, `A12`, `A22` contain 4 halr objects corresponding to the children of A.

The toolbox implements operations between halr objects, such as Algorithms 1 and 2 and matrix multiplication, as well as the Sylvester equation solver described in Section 2.4. Similar to hodlr and hss, when arithmetic operations are performed recompression (e.g., low-rank approximation) is applied in order to limit storage while ensuring a relative accuracy. More specifically, an estimate of the norm of the result $C = A \text{ op } B$ is computed beforehand, and recompressions are performed using a tolerance $\epsilon \cdot \|C\|$, where ϵ is a global tolerance.

3 | CONSTRUCTION VIA ADAPTIVE DETECTION OF LOW-RANK BLOCKS

In this section, we deal with task (i) described in Section 1, that is: given a function handle $f : [\![1, m]\!] \times [\![1, n]\!] \to \mathbb{C}$ construct an HALR representation of $A = (a_{ij})_{i,j} \in \mathbb{C}^{m \times n}$ such that $a_{ij} = f(i, j)$.

3.1 | Low-rank approximation

We start by considering a simpler problem, the (global) approximation of A with a low-rank matrix. Several methods have been proposed for this problem,[23-26] which target different scenarios. In the following sections, we will often need to determine if a matrix is *sufficiently low-rank* in the sense that it can be approximated, within a certain accuracy, with a matrix of rank bounded by $\mathsf{maxrank}$. For this purpose, we assume the availability of a procedure $(U, V, \text{flag}) = \text{LRA}(A, \mathsf{maxrank}, \epsilon)$ that returns a low-rank factorization $A \approx UV^T$, of rank at most $\mathsf{maxrank}$. The returned flag indicates whether the approximation verifies $\|A - UV^T\| \lesssim \epsilon$.

In our implementation, we will rely on the *adaptive cross approximation* (ACA) algorithm with partial pivoting,[24] which only requires the evaluation of a few matrix rows and columns selected by the algorithm. The parameter ϵ is used in the heuristic stopping criterion of the method, which in practice usually ensures the requirement on the absolute error stated above. More specifically, the first pivot is chosen as the entry of maximum magnitude over a random sample of $\mathcal{O}(n)$ entries from A, and the corresponding row and column are used to construct a rank-1 approximation to A. Then, at step j ACA chooses a new pivot from the residual matrix $A - U_j V_j^T$ with partial pivoting, where $U_j V_j^T$ is the current rank-j approximation, and continues the process by improving the approximation with a rank-1 update. The process is continued until the norm of the update becomes smaller than ϵ; then, as a safeguard condition, we inspect the residual matrix in $\mathcal{O}(n)$ randomly sampled entries. If the rank-1 update corresponding to the maximum entry has norm bounded by ϵ, then we stop. If this is not the case, the process is continued until the condition is met.

The parameter ϵ is used in the heuristic stopping criterion of the method, which in practice usually ensures the requirement on the absolute error stated above. When aiming at a relative accuracy ϵ_{rel}, we need to set $\epsilon = \epsilon_{\mathsf{rel}}\|A\|$; if $\|A\|$ is not available, it is estimated during the first ACA steps. The cost of ACA for returning an approximation of rank k is $\mathcal{O}((k^2 + kc_A)(m + n))$ where c_A is the cost of evaluating one entry of A. The approximation is returned in factorized form as a product of $m \times k$ and $k \times n$ matrices and therefore the storage cost is $\mathcal{O}(k(m + n))$.

Depending on the features of A, other choices for the procedure LRA might be attractive. For instance, if the matrix-vector product by A and A^T can be performed efficiently (for instance when A is sparse), then a basis for the column range of A can be well-approximated by taking matrix-vector products with a small number of random vectors, and this can be used to construct an approximate low-rank factorization as described in Reference 23. The methodology described in the following sections can be adapted to this context, by replacing the procedure LRA.

3.2 | HALR approximation with prescribed cluster

Letting $\mathcal{T}$ denote a prescribed quad-tree cluster on $[\![1, m]\!] \times [\![1, n]\!]$, we consider the problem of approximating A within a certain tolerance ϵ, with a $(\mathcal{T}, k)$-HALR $\sim A$ for some, hopefully small k. A straightforward strategy for building $\sim A$ is to perform the following operations on its blocks:

(i) For a leaf node labeled `low-rank`, run LRA (without limitation on the rank) to approximate the block in factored form.
(ii) For a leaf node labeled `dense`, assemble and explicitly store the whole block.
(iii) For a non-leaf node, proceed recursively with its children.

To avoid an overestimation of the ranks for blocks of relatively small norm, we first approximate the norm of the entire matrix with the norm of a rough approximation of A obtained by running LRA for a small value of maxrank.

3.3 | HALR approximation with prescribed maximum rank

We now discuss the problem at the heart of HALR: Given an integer maxrank determine a quad-tree cluster $\mathcal{T}$ and an $(\mathcal{T}, \widetilde{k})$-HALR matrix $\widetilde{A}$ such that $\widetilde{k} \le \text{maxrank}$ and $\|A - \widetilde{A}\| \le \epsilon$. Moreover, we ideally want $\mathcal{T}$ to be minimal in the sense that if $\hat{A}$ is another $(\hat{\mathcal{T}}, \hat{k})$-HALR approximating A within the tolerance ϵ, and $\hat{k} \le \text{maxrank}$, then $\hat{\mathcal{T}} \not< \mathcal{T}$. In this context, we consider all the trees (or subtrees) that contain only dense leaves to be equivalent to a single dense node.

We propose to compute $\mathcal{T}$ and $\widetilde{A}$ with the following greedy algorithm:

(i) We apply LRA limited by maxrank to the matrix A. If this is successful, as indicated by the returned flag, then $\mathcal{T}$ is set to a tree with a single node that is labeled `low-rank` and contains the approximation returned by LRA.
(ii) If LRA fails and the size of A is smaller than a fixed parameter $n_{\min}$ then $\mathcal{T}$ is set to a tree with a single node labeled as `dense` and the matrix is formed explicitly.
(iii) Otherwise we split A in 4 blocks of nearly equal sizes and we proceed recursively on each block. Then:

- If the 4 blocks are all leaves labeled as `dense`, then we merge them into a single dense block.
- Otherwise, we attach to the root of $\mathcal{T}$ the four subtrees resulting from the recursive calls.

The whole procedure is summarized in Algorithm 5. Under reasonable assumptions, the cost of Algorithm 5 is proportional to the one for storing its outcome.

Lemma 6. *Let $A \in \mathbb{C}^{n\times n}$, $n = 2^p n_{\min}$, and assume that for fixed $\epsilon > 0$ and $\text{maxrank} \in \mathbb{N}$ the cost of LRA$(B, \text{maxrank}, \epsilon)$ is $\mathcal{O}(t)$ for any $t \times t$ matrix B. Denote by $\widehat{A}$ the HALR matrix returned by Algorithm 5 with input parameters A, maxrank, ϵ, $n_{\min}$. If evaluating the entries of A costs $\mathcal{O}(1)$, then the cost of Algorithm 5 is $\mathcal{O}(S)$, where S is the maximum between the storage for $\widehat{A}$ and n.*

Proof. We proceed by induction on the depth p of the quad-tree cluster of $\widehat{A}$. If $p = 1$ and $\widehat{A}$ is a low-rank matrix it means that Algorithm 5 has run one (successful) call of LRA on A; the latter costs $\mathcal{O}(n) = \mathcal{O}(S)$. If $\widehat{A}$ is a dense matrix, then Algorithm 5 has run (without success) 4^j calls of LRA on $n/2^j \times n/2^j$ matrices for each level $j = 0, \dots, p$, and has evaluated all the entries of A at the end of the recursion. The overall cost of these operations is $\mathcal{O}(n^2) = \mathcal{O}(S)$. When $p > 1$ we have that the cost of Algorithm 5 is given by the first call of LRA, that is, $\mathcal{O}(n)$, plus the costs of the four recursive calls on the blocks of size $n/2 \times n/2$. Applying the induction step and the fact that $\mathcal{O}(S) \ge \mathcal{O}(n)$ we get the claim. ■

3.4 | Refining an existing partitioning

As operations are performed on a $(\mathcal{T}, k_A)$-HALR matrix A, its low-rank properties may evolve and it can be beneficial to readjust the tree $\mathcal{T}$ accordingly by making use of Algorithm 5. More specifically, we refine $\mathcal{T}$ by performing the following steps from bottom to top:

Algorithm 5. Approximation of a matrix A using the greedy construction of the quad-tree cluster $\mathcal{T}$. The (absolute) approximation accuracy is determined by ϵ

```
1:  procedure HALR_ADAPTIVE($A$, maxrank, $\epsilon$, $n_{\min}$)
2:      $(m, n) \leftarrow$ SIZE($A$)
3:      $(U, V, \text{flag}) \leftarrow$ LRA($A$, maxrank, $\epsilon$)
4:      if LRA succeeds then
5:          $H.U \leftarrow U, H.V \leftarrow V, H.\text{admissible} = 1$
6:      else
7:          if $\min\{m, n\} \leq n_{\min}$ then
8:              $H.F \leftarrow A, H.\text{admissible} = 0$
9:          else
10:             $H.\text{admissible} = 0, m_1 \leftarrow \lceil \frac{m}{2} \rceil, n_1 \leftarrow \lceil \frac{n}{2} \rceil$
11:             $H.A_{11}$ = HALR_ADAPTIVE($A(1 : m_1, 1 : n_1)$, maxrank, $\epsilon$)
12:             $H.A_{21}$ = HALR_ADAPTIVE($A(m_1 + 1 : m, 1 : n_1)$, maxrank, $\epsilon$)
13:             $H.A_{12}$ = HALR_ADAPTIVE($A(1 : m_1, n_1 + 1 : n)$, maxrank, $\epsilon$)
14:             $H.A_{22}$ = HALR_ADAPTIVE($A(m_1 + 1 : m, n_1 + 1 : n)$, maxrank, $\epsilon$)
15:             if $H.A_{ij}$ are labeled as dense for $i, j = 1, 2$ then
16:                 $H.F \leftarrow \begin{bmatrix} H.A_{11}.F & H.A_{12}.F \\ H.A_{21}.F & H.A_{22}.F \end{bmatrix}$
17:                 $H.A_{ij} \leftarrow [\,]$                                ▷ Remove the successors
18:             end if
19:         end if
20:     end if
21: end procedure
```

(i) Algorithm 5 with maximum rank maxrank is applied to each leaf node and the leaf node is replaced with the outcome.
(ii) A node with 4 children that are `dense` leaf nodes is merged into a single `dense` leaf node.
(iii) For a node with 4 children that are `low-rank` leaf nodes, we form the low-rank matrix obtained by merging them. If its numerical rank is bounded by maxrank, we replace the node with a `low-rank` block. Otherwise, the node remains unchanged.

The procedure is summarized in Algorithm 6; to decide whether to merge four low-rank blocks in (iii), we make use of the method COMPRESSFACTORS that computes a reduced truncated singular value decomposition of UV^T; this requires $\mathcal{O}(k^2(m + n) + k^3)$ flops, where k is the number of columns of U, V, see Reference 6, Algorithm 2.17, p. 33.

In the next sections, Algorithm 6 is regularly used to deal with situations where a matrix B is obtained from operating with ℓ HALR matrices $A_1, \dots, A_\ell$ and its tree is initially set to the intersection of the cluster trees of $A_1, \dots, A_\ell$. A relevant special case is the one where only A_1 is a general HALR matrix and all the other matrices are low-rank; in this case the initial tree for B is the one of A_1.

4 | NUMERICAL EXAMPLES

In Sections 2 and 3, we have developed all the tools needed to implement an efficient implicit time integration scheme for the reaction diffusion equation (1), provided that the discretization of the operator L has the Kronecker sum structure $I \otimes A_n + B_n \otimes I$. In the following, we describe in detail how to put all pieces together for the representative case of the Burgers' equation. Then we provide numerical tests for other problems that can be treated similarly. The experiments have been run on a server with two Intel(R) Xeon(R) E5-2650v4 CPU with 12 cores and 24 threads each, running at 2.20 GHz, using MATLAB R2017a with the Intel(R) Math Kernel Library Version 11.3.1. In all case studies, the relative truncation threshold has been set to $\epsilon_{\text{rel}} = 10^{-8}$.

Algorithm 6. Refine the quad-tree cluster of an HALR matrix

```
procedure REFINECLUSTER($A$, maxrank, $\epsilon$)
    if A is leaf node then
        $A \leftarrow$ HALR_ADAPTIVE($A$, maxrank, $\epsilon$)
    else
        $A.A_{ij} \leftarrow$ REFINECLUSTER($A.A_{ij}$, maxrank, $\epsilon$) for $i,j = 1,2$.
        if $A.A_{ij}$ are dense leaf nodes for $i,j = 1,2$ then
            $A.F \leftarrow \begin{bmatrix} A.A_{11}.F & A.A_{12}.F \\ A.A_{21}.F & A.A_{22}.F \end{bmatrix}$
        end if
        if $A.A_{ij}$ are low-rank leaf nodes for $i,j = 1,2$ then
            $U \leftarrow \begin{bmatrix} A.A_{11}.U & A.A_{12}.U & & \\ & & A.A_{21}.U & A.A_{22}.U \end{bmatrix}$
            $V \leftarrow \begin{bmatrix} A.A_{11}.V & & A.A_{21.V} & \\ & A.A_{12}.V & & A.A_{22}.V \end{bmatrix}$
            $(U, V) \leftarrow$ COMPRESSFACTORS($U, V, \epsilon$)        ▷ Reference 6, Algorithm 2.17, p. 33
            if rank($UV^T$) $\leq$ maxrank then
                $(A.U, A.V) \leftarrow (U, V)$
                $A$.admissible $\leftarrow 1$
                $A.A_{ij} \leftarrow [\,]$ for $i,j = 1,2$
            end if
        end if
    end if
end procedure
```

4.1 | Burgers' equation

We consider the following Burgers' equation[17(Example3)] with Dirichlet boundary conditions:

$$\begin{cases} \frac{\partial u}{\partial t} = K\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) - u \cdot \left(\frac{\partial u}{\partial x} + \frac{\partial u}{\partial y}\right) = 0 & (x,y) \in \Omega = (0,2) \times (0,2), \\ u(x,y,t) = \frac{1}{1+\exp((x+y-t)/2K)} & \text{for } t = 0 \text{ or } (x,y) \in \partial\Omega, \end{cases}$$

for $K = 0.001$. We make use of a uniform finite differences discretization in space, with step $h = \frac{2}{n-1}$, combined with a Euler IMEX method for the discretization in time with step $\Delta t = 5 \cdot 10^{-4}$. This yields

$$\left(\frac{1}{2}I - \Delta t A_n\right) U_{n,\ell+1} + U_{n,\ell+1}\left(\frac{1}{2}I - \Delta t A_n\right) = U_{n,\ell} + \Delta t(F_{n,\ell} + B_{n,\ell}), \tag{10}$$

where denoting with $\circ$ the Hadamard (component-wise) product, we have set

$$\begin{aligned} F_{n,\ell} &:= U_{n,\ell} \circ \left[D_{n,\ell} U_{n,\ell} + U_{n,\ell} D_{n,\ell}^T + (e_n v_{n,\ell}^T + v_{n,\ell} e_n^T)/h\right], \quad (v_{n,\ell})_j = u(jh, 2, \ell\Delta t), \\ B_{n,\ell} &:= (e_1 w_{n,\ell+1}^T + w_{n,\ell+1} e_1^T + e_n v_{n,\ell+1}^T + v_{n,\ell+1} e_n^T)/h^2, \quad (w_{n,\ell})_j = u(jh, 0, \ell\Delta t), \end{aligned}$$

and

$$A_n = \frac{1}{h^2}\begin{bmatrix} -2 & 1 & & \\ 1 & -2 & \ddots & \\ & \ddots & \ddots & 1 \\ & & 1 & -2 \end{bmatrix}, \quad D_{n,\ell} = \frac{1}{h}\begin{bmatrix} -1 & 1 & & \\ & \ddots & \ddots & \\ & & \ddots & 1 \\ & & & -1 \end{bmatrix}.$$

Algorithm 7. Time stepping procedure for the Burgers' equation

```
1: procedure BURGERS_IMEX($n$, $\Delta t$, $T_{\max}$)
2:     $h \leftarrow \frac{2}{n-1}$
3:     $\left(U_{n,0}\right)_{ij} \leftarrow u(ih, jh, 0)$
4:     $t \leftarrow 0, \ell \leftarrow 0$
5:     while $t \leq T_{\max}$ do
6:         $F_{n,\ell} \leftarrow U_{n,\ell} \circ \left[D_{n,\ell} U_{n,\ell} + U_{n,\ell} D_{n,\ell}^T + (e_n v_{n,\ell}^T + v_{n,\ell} e_n^T)/h\right]$
7:         $B_{n,\ell} \leftarrow (e_1 w_{n,\ell+1}^T + w_{n,\ell+1} e_1^T + e_n v_{n,\ell+1}^T + v_{n,\ell+1} e_n^T)/h^2$
8:         $R \leftarrow U_{n,\ell} + \Delta t(F_{n,\ell} + B_{n,\ell})$
9:         $U_{n,\ell+1} \leftarrow$ LYAP($\frac{1}{2}I - \Delta t A_n, R$)        ▷ Solve the Lyapunov equation (10)
10:        $t \leftarrow t + \Delta t, \quad \ell \leftarrow \ell + 1$
11:    end while
12: end procedure
```

Note that $\text{rank}(B_{n,\ell}) \leq 4$. The time stepping procedure is reported in Algorithm 7.

If Algorithm 7 is executed with standard dense numerical linear algebra each time step requires $\mathcal{O}(n^3)$ flops and $\mathcal{O}(n^2)$ storage. In order to exploit the additional structure observed in Figure 1, we propose to maintain the HALR representations of the matrices $F_{n,\ell}$ and $U_{n,\ell}$. In particular:

(i) At line 3, we employ Algorithm 5 to retrieve a quad-tree cluster tree $\mathcal{T}$ and a $(\mathcal{T}, \widetilde{k})$-HALR representation of $U_{n,0}$. The rank $\widetilde{k}$ satisfies $\widetilde{k} \leq \mathsf{maxrank}$.
(ii) In place of lines 6–8, we compute a $(\mathcal{T}, k_R)$-HALR representation for R using the algorithm described in Section 3.2. More specifically, we force the quad-tree cluster to be the one of $U_{n,\ell}$. We remark that an efficient handle function for evaluating the entries of R is obtained by leveraging the HALR structure of $F_{n,\ell}$, $U_{n,\ell}$ and the low-rank structure of $B_{n,\ell}$.
(iii) We refine the cluster tree of R using Algorithm 6. During this process, the truncation is performed according to a relative threshold $\epsilon_{\mathsf{rel}} = 10^{-5}$, comparable with the accuracy of the time integration method. This avoids taking into account the increase of the rank caused by the accumulation of the errors.
(iv) Since the Lyapunov equation (10) has HODLR coefficients and HALR right hand-side we employ Algorithm 3 for its solution at line 9. Consequently, the matrix $U_{n,\ell+1}$ inherits the same quad-tree cluster of R.

Note that the refinement of the cluster at step (iii) is the only operation that can modify the quad-tree cluster used to represent the solution. The test has been repeated for $\mathsf{maxrank} = 25, 50, 75, 100$. In Table 1, we report the total computational time (labeled as T_{tot}), and the maximum memory consumption for storing the solution in each run, measured in MB. We also report the total time spent solving Lyapunov equations (phase (iv), labeled as T_{lyap}) and approximating the right-hand side and adapting the HALR structure (phases (ii) and (iii), labeled as T_{adapt}). These two phases accounts for most of the computational cost (between 85% and 90%); the solution of the Lyapunov equation is the most expensive operation. The ratio $T_{\text{lyap}}/T_{\text{adapt}}$ seems to grow with n, and is around 2 at $n = 16{,}384$.

Figure 5 describes in detail the case $\mathsf{maxrank} = 50$. The solution at time $t = 0$ (iteration 0) has a low-rank structure; the region where the shock happens is confined to the origin in $[0, 2]^2$. After some iterations, the shock moves causing an increase in the rank required to approximate the solution, and the method switches to the HALR structure. When the time approaches $t = 3.75$ (iteration 7500), the solution becomes numerically low-rank again, because the shock moves close to top-right corner of $[0, 2]^2$. This progression is reported in the top part of Figure 5, which shows the time required for each iteration, and the structure adopted by the method. We remark that since the 1D Laplacian can be diagonalized via the sine transform, Algorithm 7 can be efficiently implemented also without exploiting the local low-rank structure. In particular, the iteration cost becomes $\mathcal{O}(n^2 \log(n))$. In the left part of Table 3 we have reported the times required by a dense version of Algorithm 7 for integrating (3) where any hierarchical structure is ignored, and the Lyapunov equations are solved by diagonalizing the Laplace operator using the FFT; for this case, we have also reported the average time for

TABLE 1 Time and storage required for integrating the Burgers' equation from $t = 0$ to $t = 4$, for different values of n and of maxrank

n	T_{tot} (s)	T_{lyap} (s)	T_{adapt} (s)	Mem.	T_{tot} (s)	T_{lyap} (s)	T_{adapt} (s)	Mem.
	maxrank = 25				maxrank = 50			
4096	**20,057.6**	9742.6	7256.7	13.1	22,334.0	10,604.3	7767.4	**10.3**
8192	**54,659**	29,231.1	17,104.5	17.8	57,096.9	32,116.9	17,346.2	**16.4**
16,384	132,238.3	80,762.6	36,539.2	**25.3**	**119,130.4**	76,431.5	31,011.5	35.3
	maxrank = 75				maxrank = 100			
4096	26,727.0	12,915.1	8923.3	10.8	29,383.2	14,174.7	10,362.5	12.1
8192	59,340.9	33,756.1	18,825.8	22.4	63,150.1	34,108.4	22,163.0	24.8
16,384	119,602.0	71,187.1	35,398.9	46.5	125,688.6	71,050.3	40,701.3	50.4

Note: The best times and memory usage for a given n are reported in bold. The reported memory is measured in megabytes (MB) and is the maximum memory consumption for storing the solution during the iterations. The reported timings are the cumulative ones for 8000 time steps.

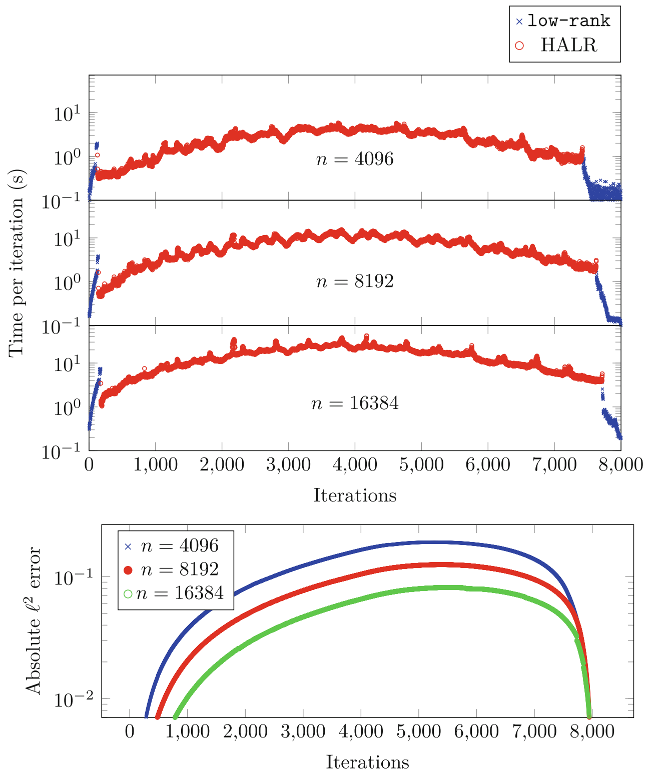


FIGURE 5 Top figure: Time required for solving the Lyapunov equation and the adaptive approximation at each iteration; the iterations marked as `low-rank` correspond to the case where the matrix has the trivial partitioning with only one block labeled as `low-rank`; bottom figure: approximation error during the iteration, obtained computing the l^2-norm of the difference with the exact solution. The reported timings are for maxrank = 50, and $n = 4096$, $n = 8192$, and $n = 16,384$. The reported errors are absolute; for comparison, note that the ℓ^2 norm of the solution grows monotonically from 0 to 1 in the time interval [0, 4], as the solution converges pointwise to 1.

solving the Lyapunov equation via fast diagonalization; we see that leveraging the HALR structure makes the algorithm faster since dimension 8192.

The bottom plot of Figure 5 shows the absolute approximation error in the discrete l^2-norm, computed comparing the numerical solution with the true solution $u(x, y, t) = \left[1 + \exp\left(\frac{x+y-t}{2K}\right)\right]^{-1}$. The error curve associated with the implementation of Algorithm 7 in dense arithmetic matches the one reported in Figure 5 confirming that the low-rank truncations have negligible effects on the computed solution. We remark that the displayed errors come from the discretization,

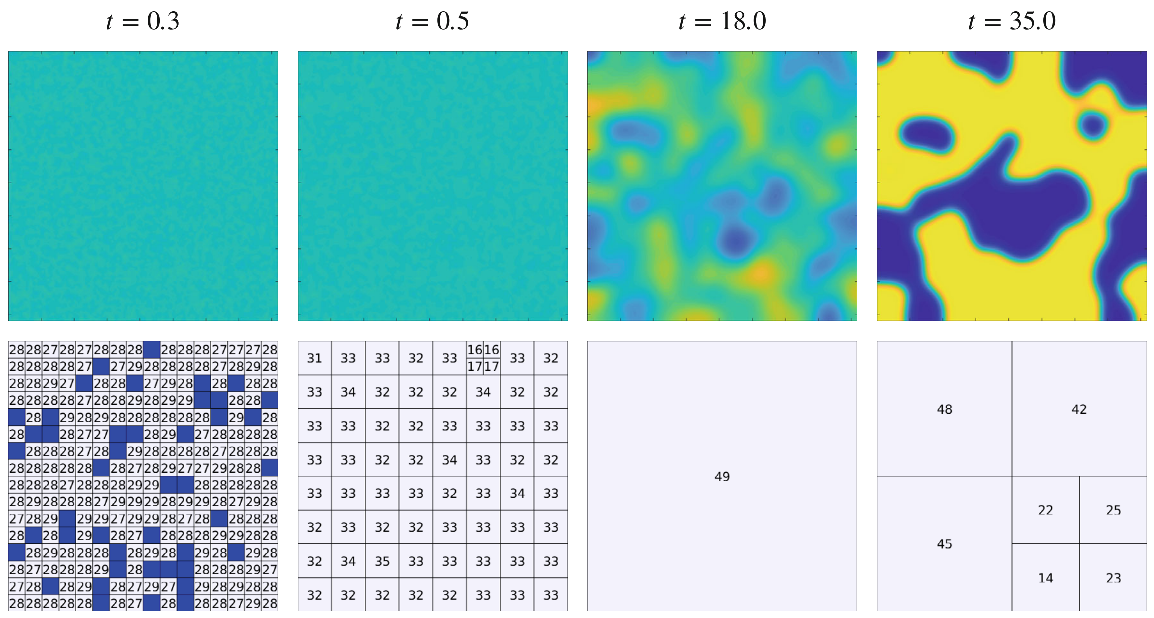


FIGURE 6 Evolution of the structure and the solution at different time steps

and are not introduced by the low-rank approximations in the blocks: we have verified the computations using dense unstructured matrices, obtaining the same results.

4.2 | Allen–Cahn equation

The Allen–Cahn equation is a reaction–diffusion equation which describes a phase separation process. It takes the following form:

$$\begin{cases} \frac{\partial u}{\partial t} + \nu(-\Delta)u = g(u) & \text{in } \Omega, \\ \frac{\partial u}{\partial \vec{n}} = 0 & \text{on } \partial\Omega, \\ u(x,y,0) = u_0(x,y), \end{cases} \tag{11}$$

where $\nu = 5 \cdot 10^{-5}$ is the mobility, $\Omega = [0,1]^2$, and the source term is the cubic function $g(u) := u(u - 0.5)(1 - u)$. This test problem is described in Reference [27]. For a fixed choice of (x, y), the solution $u(x, y, t)$ converges either to 1 or 0 for most points inside the domain as $t \to \infty$.

We discretize the problem with the IMEX implicit Euler method in time and centered finite differences in space, exactly as in the Burger's equation example. The only difference is that in this problem we are considering Neumann boundary conditions instead of Dirichlet.

In this example, we choose the initial (discrete) solution randomly, distributed as $u(x_i, y_j, 0) \sim N(\frac{1}{2}, 1)$, with every grid point independent of the others. Integrating the system yields a model for spinodal decompositions.[27] We remark that with this choice the matrix $U_{n,0}$ has no low-rank structure, and will be treated as a dense matrix. On the other hand, during the time evolution, the smoothing effect of the Laplacian makes the solution $U_{n,\ell}$ well-approximable by low-rank matrices, at least locally (see Figure 6). For even larger ℓ, the solution converges to either 0 or 1, giving rise to several "flat regions," which can be approximated by low-rank blocks, and the structure $U_{n,\ell}$ can be efficiently memorized using a $(\mathcal{T}, k)$-HALR representation.

We have integrated the solution for $t \in [0, 40]$, using $\Delta t = 0.1$, and grid sizes from 1024 up to 16,384. The simulation has been run for maxrank $= 25, 50, 75, 100$. The time and storage used for the integration has been reported in Table 2, analogously to the Burgers' equation case. Note that here the maximum memory consumption is always attained at $t = 0$, where the solution is stored as a dense matrix.

TABLE 2 Time and storage required for integrating the Allen–Cahn equation from 0 to 40, for different values of n and of maxrank

n	T_{tot} (s)	T_{lyap} (s)	T_{adapt} (s)	Mem.	T_{tot} (s)	T_{lyap} (s)	T_{adapt} (s)	Mem.
	maxrank = 25				maxrank = 50			
1024	277.1	153.0	124.1	8.0	317.6	197.1	120.5	8.0
2048	1080.6	733.7	346.9	32.0	713.9	561.0	152.9	32.0
4096	1754.2	1479.8	274.4	128.0	900.7	701.5	199.2	128.0
8192	3702.4	3202.8	499.5	512.0	1823.7	1428.9	394.7	512.0
16,384	7704.5	6392.1	1312.4	2048.0	3688.1	2765.7	922.5	2048.0
	maxrank = 75				maxrank = 100			
1024	229.6	151.1	78.4	8.0	**187.9**	118.9	69.0	8.0
2048	430.9	338.2	92.7	32.0	**346.9**	252.0	94.9	32.0
4096	619.7	444.2	175.4	128.0	**505.2**	325.6	179.6	128.0
8192	1424.5	1036.0	388.4	512.0	**1147.4**	731.0	416.3	512.0
16,384	2899.1	1982.0	917.1	2048.0	**2336.8**	1331.8	1005.0	2048.0

Note: The best times for a given N are reported in bold. The reported memory is measured in megabytes (MB) and is the maximum memory consumption for storing the solution during the iterations.

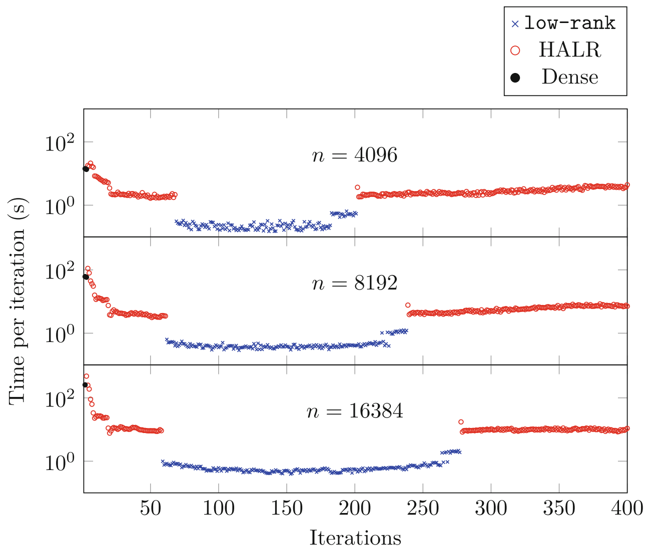


FIGURE 7 Execution time per iteration for the Allen–Cahn problem. The reported timings are for maxrank = 50, and $n = 4096$, $n = 8192$, and $n = 16{,}384$.

Figures 6 and 7 focus on the case $n = 4096$ and maxrank = 50. The evolution in time of the solution and of the corresponding HALR structure are reported in Figure 6. The initial structure is a tree with a single node labeled as `dense`, and the HALR representation can be used already at time 0.3. Then, the solution becomes (numerically) low-rank at time 6.7 (with rank approximately 50); the third image in Figure 6 shows the low-rank structure at time $t = 18$. Later, as the phase separation happens, the representation becomes again HALR, and stabilizes at the format shown in the fourth figure. The time required for each iteration, and the structure adopted is reported in Figure 7. Analogously to the Burgers' example, the 1D Laplacian with Neumann boundary conditions can diagonalized via the cosine transform providing a dense method with iteration cost $\mathcal{O}(n^2 \log(n))$. In the right part of Table 3, we have reported the times required by the dense method for integrating (11) and the average time for solving the Lyapunov equation via fast diagonalization; we see that exploiting the structure makes the HALR approach faster from dimension 16,384.

TABLE 3 Time required for integrating the Burgers' equation and the Allen–Cahn equation, for different values of n, relying on sine and cosine transforms

	FFT-based algorithms					
	Burgers			Allen–Cahn		
n	T_{tot} (s)	Avg. T_{lyap} (s)	T_{HALR} (s)	T_{tot} (s)	Avg. T_{lyap} (s)	T_{HALR} (s)
4096	18,094	2.26	20,057.6	174.97	0.44	505.2
8192	70,541	8.82	54,659	847.3	2.12	1147.4
16,384	295,507	36.94	119,130.4	2967	7.42	2336.8

Note: The time step is chosen as in the experiments using the HALR structure. The time required by the best performing version of the HALR algorithm is reported for reference in the column T_{HALR}.

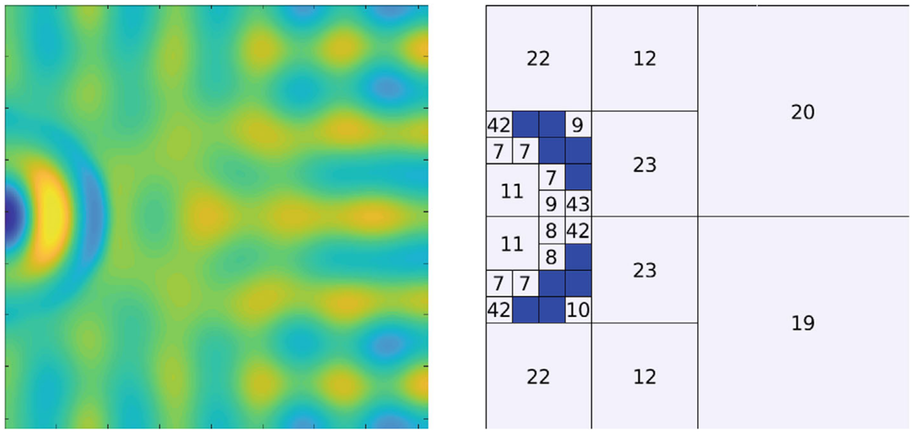


FIGURE 8 Solution of the Helmholtz equation (12) discretized on a 4096 × 4096 grid (left) and its representation in the HALR format (right), with maxrank = 50.

4.3 | Inhomogeneous Helmholtz equation

Let us consider the following Helmholtz equation with Neumann boundary conditions on the square $\Omega := [-1,1]^2$:

$$\begin{cases} \Delta u + ku + f = 0, \\ \frac{\partial u}{\partial \vec{n}} = 0 \quad \text{on } \partial\Omega, \end{cases} \qquad \begin{cases} k(x,y) := 2500 \cdot e^{-50\left|x^2+(y+1)^2-\frac{1}{4}\right|}, \\ f(x,y) := \frac{e^{-x^2-y^2}}{100}. \end{cases} \tag{12}$$

The chosen coefficient $k(x,y)$ is negligible outside of a semi annular region centered in $(0,-1)$; the source term f is concentrated around the origin. The numerical solution of (12), reported in Figure 8, is well approximated in the HALR format which refines the block low rank structure in the region where k takes the larger values.

The usual finite difference discretization of (12) provides the $n^2 \times n^2$ linear system

$$(A \otimes I + I \otimes A + D_k)\,\text{vec}(X) + \text{vec}(F) = 0, \tag{13}$$

where A is the 1D Laplacian matrix with Neumann boundary conditions, D_k is the diagonal matrix containing the evaluations of $k(x,y)$ at the grid points and F contains the analogous evaluations of the source term. Note that, omitting the matrix D_k, (13) can be solved as a Lyapunov equation. In the spirit of numerical methods for generalized matrix equations,[28] we propose to solve (13) with a structured GMRES iteration using the Lyapunov solver as preconditioner; more specifically we store all the (matricized) vectors generated by the GMRES in the HALR format. If necessary (when the rank grows) we adjust the partitioning of the latter via Algorithm 6. The inner product between vectors are computed using the block

Algorithm 8. Structured and preconditioned GMRES iteration for (12)

```
procedure PGMRES($A$, $K$, $F$, tol, maxrank)                ▷ $\mathrm{diag}(\mathrm{vec}(K)) = D_k$
    $B \leftarrow$ LYAP($A, F$)
    $U_1 \leftarrow B/\|B\|_F$
    for $j = 1, 2, \dots$ do
        $R = AU_j + U_jA^T + K \circ U_j$
        $R \leftarrow$ REPARTITION($R$, maxrank)
        $W \leftarrow$ LYAP($A, R$)
        for $s = 1, \dots, j$ do
            $H_{s,j} \leftarrow$ DOT($W, U_s$)                    ▷ Trace of $W^TU_s$, see Algorithm 9
            $W \leftarrow W - H_{s,j} \cdot U_s$
        end for
        $H_{j+1,j} \leftarrow \|W\|_F$, $U_{j+1} \leftarrow W/\|W\|_F$
        $y \leftarrow \|B\|_F H^\dagger e_1$
        if $\|Hy - \|B\|_F e_1\| <$ tol $\cdot \|B\|_F$ then
            break
        end if
    end for
    return $\sum_j y_j U_j$
end procedure
```

TABLE 4 Time and storage required for solving the inhomogeneous Helmholtz equation (12), for different values of n and maxrank = 50

	maxrank = **50**				
n	T_{tot} **(s)**	T_{lyap} **(s)**	T_{adapt} **(s)**	**It.**	**Mem.**
1024	72.83	42.82	11.39	25	1.72
2048	231.92	161.35	24.25	26	3.73
4096	603.03	362.22	75.74	26	8.18
8192	1773.6	982.37	244.54	26	16.68
16,384	5884	3065	1044.4	28	33.3

Note: The reported memory is measured in megabytes (MB) and refers to the storage of the solution.

Algorithm 9. Trace inner product for two HALR matrices

```
procedure DOT($A$, $B$)
    if $A$ and $B$ are leaf nodes or at least one is low-rank then
        return Trace($A^TB$)                ▷ Exploiting the low-rank structure of $A$ or $B$, if any
    else
        if $A$ is dense or $B$ is dense then
            Set the partitioning of $A$ and $B$ equal to the finest of the two.
        end if
        return DOT($A_{11}, B_{11}$) + DOT($A_{12}, B_{12}$)+DOT($A_{21}, B_{21}$)+DOT($A_{22}, B_{22}$)
    end if
end procedure
```

recursive procedure described in Algorithm 9, which returns the trace of A^TB for two HALR matrices A and B. Finally, the solution is constructed as a linear combination of HALR matrices. The whole procedure is reported in Algorithm 8.

Equation (12) has been solved for different grid sizes with $\mathsf{maxrank} = 50$. The time and memory consumption are reported in Table 4. The storage needed for the solution scales linearly with n. In addition, the table contains the number of iterations needed by the preconditioned GMRES to reach the relative tolerance $\mathsf{tol} = 10^{-4}$. We note that the number of iterations grows very slowly as the grid size increases. The time required depends on many factors, such as the distribution of the ranks and the complexity of the structure in the basis generated by the GMRES; we just remark that it grows subquadratically for this example. In future work, we plan to explore the use of restarting mechanisms and other truncation strategies in order to optimize the approach.

5 | CONCLUSIONS

In this work, we have proposed a new format for storing matrices arising from 2D discretization of functions which are smooth almost everywhere, with localized singularities. Low-rank decompositions, which are effective for globally smooth functions, become ineffective in this case. The proposed structure automatically adapts to the matrix, and requires no prior information on the location of the singular region. The storage and complexity interpolates between dense and low-rank matrices, based on the structure, with these two cases as extrema.

We demonstrated techniques for the efficient adaptation of the structure in case of moving singularities, with the aim of tracking time-evolution of 2D PDEs; the examples show that the proposed techniques can effectively detect changes in the structure, and ensure the desired level of accuracy. We developed efficient Lyapunov and Sylvester solvers for matrix equations with HALR right-hand side and HODLR coefficients. This case is of particular interest, as it often arises in discretized PDEs. Several numerical experiments demonstrate both the effectiveness and the flexibility of the approach.

The proposed format may be generalized to the discretization of functions on 3D box domains by moving from matrices to tensors, swapping quadtrees with octrees, making the necessary adjustments, and choosing a suitable low-rank format for subtensors. Similar ideas to the ones presented in this work may be used to detect the hierarchical structure in an adaptive way, and to adjust the structure in time. However, devising an efficient Sylvester solver remains challenging. Despite the existence of low-rank solvers for linear systems with Kronecker structure in the Tucker format[29] exploiting the hierarchical structure introduces major difficulties, which we plan to investigate in future work.

CONFLICT OF INTEREST

This study does not have any conflicts to disclose.

DATA AVAILABILITY STATEMENT

Data sharing is not applicable to this article as no new data were created or analyzed in this study.

ENDNOTE

[1]This is the case when A and B are endowed with stronger structures like hierarchical semiseparability (HSS).[9]

ORCID

Leonardo Robol https://orcid.org/0000-0002-6545-1748
Daniel Kressner https://orcid.org/0000-0003-3369-2958

REFERENCES

1. Palitta D, Simoncini V. Matrix-equation-based strategies for convection–Diffusion equations. BIT Numer Math. 2016;56(2):751–76.
2. Townsend A, Olver S. The automatic solution of partial differential equations using a global spectral method. J Comput Phys. 2015;299:106–23.
3. Druskin V, Knizhnerman L, Simoncini V. Analysis of the rational Krylov subspace and ADI methods for solving the Lyapunov equation. SIAM J Numer Anal. 2011;49(5):1875–98.
4. Benner P, Li RC, Truhar N. On the ADI method for Sylvester equations. J Comput Appl Math. 2009;233(4):1035–45.
5. Amestoy P, Ashcraft C, Boiteau O, Buttari A, L'Excellent JY, Weisbecker C. Improving multifrontal methods by means of block low-rank representations. SIAM J Sci Comput. 2015;37(3):A1451–74. https://doi.org/10.1137/120903476
6. Hackbusch W. Hierarchical matrices: algorithms and analysis. Springer Series in Computational Mathematics. Vol 49. Heidelberg: Springer; 2015.

7. Börm S. Data-sparse approximation of non-local operator by $\mathscr{H}^2$-matrices. Linear Algebra Appl. 2007;422(2-3):380–403.
8. Hackbusch W, Khoromskij BN, Kriemann R. Hierarchical matrices based on a weak admissibility criterion. Computing. 2004;73(3):207–43.
9. Xia J, Chandrasekaran S, Gu M, Li XS. Fast algorithms for hierarchically semiseparable matrices. Numer Linear Algebra Appl. 2010;17(6):953–76.
10. Cai D, Chow E, Erlandson L, Saad Y, Xi Y. SMASH: structured matrix approximation by separation and hierarchy. Numer Linear Algebra Appl. 2018;25(6):e2204. https://doi.org/10.1002/nla.2204
11. Erlandson L, Cai D, Xi Y, and Chow E. Accelerating parallel hierarchical matrix-vector products via data-driven sampling. Proceedings of the 2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS); 2020:749-758; IEEE.
12. Grasedyck L. Existence of a low rank or $\mathscr{H}$-matrix approximant to the solution of a Sylvester equation. Numer Linear Algebra Appl. 2004;11(4):371–89.
13. Massei S, Palitta D, Robol L. Solving rank-structured Sylvester and Lyapunov equations. SIAM J Matrix Anal Appl. 2018;39(4):1564–90.
14. Ehrlacher V, Grigori L, Lombardi D, Song H. Adaptive hierarchical subtensor partitioning for tensor compression. SIAM J Sci Comput. 2021;43(1):A139–63.
15. Kazeev V, Schwab C. Quantized tensor-structured finite elements for second-order elliptic PDEs in two dimensions. Numer Math. 2018;138(1):133–90. https://doi.org/10.1007/s00211-017-0899-1
16. Ascher UM, Ruuth SJ, Spiteri RJ. Implicit-explicit Runge-Kutta methods for time-dependent partial differential equations. Appl Numer Math. 1997;25(2-3):151–67.
17. Liao W. A fourth-order finite-difference method for solving the system of two-dimensional Burgers' equations. Int J Numer Methods Fluids. 2010;64(5):565–90.
18. Kressner D, Massei S, Robol L. Low-rank updates and a divide-and-conquer method for linear matrix equations. SIAM J Sci Comput. 2019;41(2):A848–76.
19. Simoncini V. Computational methods for linear matrix equations. SIAM Rev. 2016;58(3):377–441.
20. Jonsson I, Kågström B. Recursive blocked algorithm for solving triangular systems. I. One-sided and coupled Sylvester-type matrix equations. ACM Trans Math Softw. 2002;28(4):392–415.
21. Simoncini V. A new iterative method for solving large-scale Lyapunov matrix equations. SIAM J Sci Comput. 2007;29(3):1268–88.
22. Massei S, Robol L, Kressner D. hm-toolbox: MATLAB software for HODLR and HSS matrices. SIAM J Sci Comput. 2020;42(2):C43–68.
23. Halko N, Martinsson PG, Tropp JA. Finding structure with randomness: probabilistic algorithms for constructing approximate matrix decompositions. SIAM Rev. 2011;53(2):217–88.
24. Bebendorf M. Approximation of boundary element matrices. Numer Math. 2000;86(4):565–89.
25. Tyrtyshnikov E. Incomplete cross approximation in the mosaic-skeleton method. Computing. 2000;64(4):367–80.
26. Simon HD, Zha H. Low-rank matrix approximation using the Lanczos bidiagonalization process with applications. SIAM J Sci Comput. 2000;21(6):2257–74.
27. Burrage K, Hale N, Kay D. An efficient implicit FEM scheme for fractional-in-space reaction-diffusion equations. SIAM J Sci Comput. 2012;34(4):A2145–72.
28. Benner P, Breiten T. Low rank methods for a class of generalized Lyapunov equations and related issues. Numer Math. 2013;124(3):441–70.
29. Kressner D, Tobler C. Low-rank tensor Krylov subspace methods for parametrized linear systems. SIAM J Matrix Anal Appl. 2011;32(4):1288–316.

How to cite this article: Massei S, Robol L, Kressner D. Hierarchical adaptive low-rank format with applications to discretized partial differential equations. Numer Linear Algebra Appl. 2022;29(6):e2448. https://doi.org/10.1002/nla.2448