CHAPTER 2

Fast matrix algebra for dense matrices with rank-deficient off-diagonal blocks

Chapter summary: *The chapter describes techniques for rapidly performing algebraic operations on dense matrices whose off diagonal blocks are of low (numerical) rank. The primary focus is on matrix inversion, but algorithms for matrix-vector and matrix-matrix multiplication are also described.*

2.1. Introduction

All direct solvers described in this text make frequent use of matrix operations such as matrix-matrix multiplications, matrix inversions, LU factorizations, etc. The matrices that are manipulated are almost all dense, but fortunately, they will be either of small size, or will have internal structure that allows the required operations to be performed rapidly even though the matrices are dense. To be precise, the "internal structure" that is exploited is that off-diagonal blocks of the matrices can be well approximated by matrices of low rank. The chapter will illustrate the key ideas by introducing a very simple set of "compressible" matrices, and then showing how to rapidly perform algebraic operations on such compressible matrices.

We note that there is in the literature a large amount of literature on "structured matrix computations." The type of structure we discuss in this text is closely related to the well-established $\mathcal{H}$ and $\mathcal{H}^2$ matrices of Hackbusch and co-workers [**1, 2, 3, 4**]. The format we discuss is much simpler than the $\mathcal{H}$-matrix framework. This means that it applies to fewer matrices, but it happens to be sufficiently general for our purposes. One reason for restricting attention to a simpler format is that it makes the text more accessible, but a much more important reason is that it leads (at least in this case) to faster algorithms.

The chapter starts by describing a very simple format in Sections 2.2 and 2.3 to illustrate the key ideas. Then in Sections 2.4 – 2.7, a more sophisticated format that leads to faster algorithms is introduced.

Note: The more complex format introduced in Sections 2.4 – 2.7 is only needed for the direct solvers for boundary integral equations described in Chapter 7. Sections 2.4 – 2.7 can safely be postponed (or skipped) by a reader interested only in direct solvers for sparse matrices.

2.2. Simply compressible matrices ($\mathcal{S}$-matrices)

Roughly speaking, we say that a matrix is "simply compressible", or an "$\mathcal{S}$-matrix," if it can be tessellated into submatrices in a pattern such as the one illustrated in Figure 2.1, and each off-diagonal block in the tessellation can be approximated by a low-rank matrix.

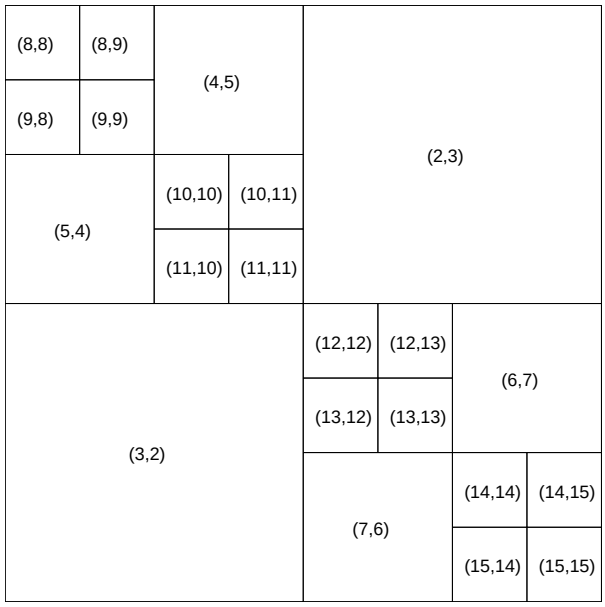

FIGURE 2.1. Matrix tessellation. The diagonal blocks are dense submatrices, while the off-diagonal blocks have rank at most p.

In order to give a precise definition of the term $\mathcal{S}$-matrix, we let p denote the maximal rank allowed for the off-diagonal blocks, and we then define the "compressibility" property recursively by saying that a square matrix $\mathbf{A}$ is an $\mathcal{S}$-matrix if, upon partitioning it into four pieces of equal size,

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix},$$

it is the case that $\mathbf{A}_{12}$ and $\mathbf{A}_{21}$ have rank at most p, and $\mathbf{A}_{11}$ and $\mathbf{A}_{22}$ are "compressible".

An $\mathcal{S}$-matrix of size $N \times N$ and (and compression rank p) can be stored using $O(p\,N\,\log N)$ real numbers, and a matrix-vector product involving a compressible matrix can be evaluated using $O(p\,N\,\log N)$ arithmetic operations, as shown in Algorithm I.

REMARK 2.1. *In practical applications, the off-diagonal blocks will almost never be exactly rank-deficient. Instead, it will be the case that they can to high accuracy be approximated by a matrix of low rank. To be precise, we say that a matrix* $\mathbf{B}$ *has* ε*-rank at most* p *is there exists a matrix* $\mathbf{C}$ *of rank exactly* p *such that* $||\mathbf{B} - \mathbf{C}|| \leq \varepsilon$*. Technically, this means that the* ε*-rank of* $\mathbf{B}$ *is the number of singular values of* $\mathbf{B}$ *larger than* ε*. As it happens, the off-diagonal blocks of the matrices under consideration in this text have singular values that decay exponentially fast, so the cut-off parameter* ε *can often be set to a very small number, say* $\varepsilon = 10^{-10}$*. This means that in practice, there is very little difference between a matrix of exact rank* k*, and of* ε*-rank* k*.*

REMARK 2.2. *For notational simplicity, we assume in this chapter that the ranks of all off-diagonal blocks are the same. It is a simple matter to use adaptively tuned ranks when implementing the algorithms.*

ALGORITHM I: MATRIX-VECTOR MULTIPLICATION FOR AN $\mathcal{S}$-MATRIX.

Given a vector $\mathbf{q}$ and an $\mathcal{S}$-matrix $\mathbf{A}$, form $\mathbf{u} = \mathbf{A}\mathbf{q}$.

(1) **function** $\mathbf{u} = \texttt{matvec}(\mathbf{A},\mathbf{q})$
(2) **if** ($\mathbf{A}$ is "small") **then**
(3) Evaluate by brute force: $\mathbf{u} = \mathbf{A}\mathbf{q}$
(4) **else**
(5) Split $\mathbf{A} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix}$.
(6) $\mathbf{u}_1 = \texttt{matvec}(\mathbf{A}_{11}, \mathbf{q}_1) + \mathbf{A}_{12}\mathbf{q}_2$
(7) $\mathbf{u}_2 = \texttt{matvec}(\mathbf{A}_{22}, \mathbf{q}_2) + \mathbf{A}_{21}\mathbf{q}_1$
(8) $\mathbf{u} = \begin{bmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \end{bmatrix}$
(9) **end if**
(10) **end function**

2.3. Inversion of compressible matrices

A recursive fast inversion scheme for compressible matrices can easily be derived from the following formula for the inverse of a 2×2 block matrix:

$$\begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix}^{-1} = \begin{bmatrix} \mathbf{X}_{11} & -\mathbf{X}_{11}\,\mathbf{A}_{12}\,\mathbf{A}_{22}^{-1} \\ -\mathbf{A}_{22}^{-1}\,\mathbf{A}_{21}\,\mathbf{X}_{11} & \mathbf{A}_{22}^{-1} + \mathbf{A}_{22}^{-1}\,\mathbf{A}_{21}\,\mathbf{X}_{11}\,\mathbf{A}_{12}\,\mathbf{A}_{22}^{-1} \end{bmatrix}, \tag{2.1}$$

where

$$\mathbf{X}_{11} = \left(\mathbf{A}_{11} - \mathbf{A}_{12}\,\mathbf{A}_{22}^{-1}\,\mathbf{A}_{21}\right)^{-1}.$$

From the formula (2.1), we immediately get the recursive inversion scheme for compressible matrices shown in Algorithm II. The efficiency of the algorithm is a consequence of the fact that the matrices $\mathbf{A}_{12}$ and $\mathbf{A}_{21}$ have low rank. As a result, the matrix-matrix multiplications that occur on lines (7) and (8) in fact consist simply of a small number of multiplications between compressible matrices and vectors. Moreover, the matrix additions in lines (7) and (8) are in fact low-rank updates to compressible matrices.

The simple algorithm described in Algorithm II has a potentially fatal flaw in that the ranks of the off-diagonal blocks could go from p to $2p$ in steps (7) and (8) since both of these involve adding a matrix of rank p to the off-diagonal blocks of a compressible matrix. Since the function `invert_matrix` is called recursively, this successive doubling of the rank quickly becomes untenable. What often saves us is that the matrices can often be re-compressed: every time we have performed an operation that could potentially increase the rank of the off-diagonal blocks, we recompress these blocks before proceeding.

When the recompression procedure succeeds in keeping the ranks of the off-diagonal blocks from exceeding some fixed number p as the computation proceeds, the computational complexity of the scheme described above is $O(p^2\,N\,\log^2 N)$. This is the performance typically observed in the context of direct solvers for elliptic PDEs.

In order to attain $O(N)$ complexity, we will first "roll out" the recursive definition of a "compressible" matrix in Section 2.4, and then introduce a more sophisticated way of representing the off-diagonal blocks in the tessellation in Section 2.5. (Recall that the discussion in the remainder of this chapter can be safely skipped for a while, since these techniques will not be used until we reach the Chapter 7 on direct solvers for integral equations.)

ALGORITHM II: INVERSION OF AN $\mathcal{S}$-MATRIX

Given an $\mathcal{S}$-matrix $\mathbf{A}$, compute its inverse $\mathbf{C}$ in $\mathcal{S}$-matrix format.

(1) **function** $\mathbf{C} = \texttt{invert\_matrix}(\mathbf{A})$
(2) **if** ($\mathbf{A}$ is "small") **then**
(3) Invert by brute force: $\mathbf{C} = \mathbf{A}^{-1}$
(4) **else**
(5) Split $\mathbf{A} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix}$.
(6) $\mathbf{X}_{22} = \texttt{invert\_matrix}(\mathbf{A}_{22})$
(7) $\mathbf{X}_{11} = \texttt{invert\_matrix}(\mathbf{A}_{11} - \mathbf{A}_{12}\,\mathbf{X}_{22}\,\mathbf{A}_{21})$
(8) $\mathbf{C} = \begin{bmatrix} \mathbf{X}_{11} & -\mathbf{X}_{11}\,\mathbf{A}_{12}\,\mathbf{X}_{22} \\ -\mathbf{X}_{22}\,\mathbf{A}_{21}\,\mathbf{X}_{11} & \mathbf{X}_{22} + \mathbf{X}_{22}\,\mathbf{A}_{21}\,\mathbf{X}_{11}\,\mathbf{A}_{12}\,\mathbf{X}_{22} \end{bmatrix}$.
(9) **end if**
(10) **end function**

Note: After any matrix addition, recompression back to rank p should be performed.

REMARK 2.3. *Other matrix operations can be derived in a similar fashion. For instance, suppose that we seek to compute an LU-factorization of a compressible matrix,*

$$\begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix} = \begin{bmatrix} \mathbf{L}_{11} & \mathbf{0} \\ \mathbf{L}_{21} & \mathbf{L}_{22} \end{bmatrix} \begin{bmatrix} \mathbf{U}_{11} & \mathbf{U}_{12} \\ \mathbf{0} & \mathbf{U}_{22} \end{bmatrix}.$$

The computation can be done via the following steps:

- $[\mathbf{L}_{11}, \mathbf{U}_{11}] = \texttt{lu\_structured}(\mathbf{A}_{11})$
- $\mathbf{L}_{21} = \mathbf{A}_{21}\mathbf{U}_{11}^{-1}$ *(execute via triangular solve on one of the factors of* $\mathbf{A}_{21}$*)*
- $\mathbf{U}_{12} = \mathbf{L}_{11}^{-1}\mathbf{A}_{12}$ *(execute via triangular solve on one of the factors of* $\mathbf{A}_{12}$*)*
- $[\mathbf{L}_{22}, \mathbf{U}_{22}] = \texttt{lu\_structured}(\mathbf{A}_{22} - \mathbf{L}_{21}\mathbf{U}_{12})$

Note that since $\mathbf{A}_{12}$ *and* $\mathbf{A}_{21}$ *are of rank p, the matrices* $\mathbf{L}_{21}$ *and* $\mathbf{U}_{12}$ *will automatically be of rank p as well. In the last step, recompression down to rank p is essential after forming the sum* $\mathbf{A}_{22} - \mathbf{L}_{21}\mathbf{U}_{12}$*. Next consider the multiplication of two compressible matrices*

$$\begin{bmatrix} \mathbf{C}_{11} & \mathbf{C}_{12} \\ \mathbf{C}_{21} & \mathbf{C}_{22} \end{bmatrix} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix} \begin{bmatrix} \mathbf{B}_{11} & \mathbf{B}_{12} \\ \mathbf{B}_{21} & \mathbf{B}_{22} \end{bmatrix} = \left[\begin{array}{c|c} \mathbf{A}_{11}\mathbf{B}_{11} + \mathbf{A}_{12}\mathbf{B}_{21} & \mathbf{A}_{11}\mathbf{B}_{12} + \mathbf{A}_{12}\mathbf{B}_{22} \\ \hline \mathbf{A}_{21}\mathbf{B}_{11} + \mathbf{A}_{22}\mathbf{B}_{21} & \mathbf{A}_{21}\mathbf{B}_{12} + \mathbf{A}_{22}\mathbf{B}_{22} \end{array}\right].$$

We see that all terms in the product involves at least one factor that is a low-rank matrix, except for the two terms $\mathbf{A}_{11}\mathbf{B}_{11}$ *and* $\mathbf{A}_{22}\mathbf{B}_{22}$*. Again the pattern repeats itself that the task of performing one operation on a matrix of size* $N \times N$ *gets reduced to the task of performing two operations on matrices of size* $N/2 \times N/2$*.*

2.4. Hierarchical partitions of the index vector

In Section 2.2, we the definition of an $\mathcal{S}$-matrix was recursive. To do away with this (for efficiency), we need to introduce a partition of the index vector $I = [1, 2, \ldots, N]$ into a binary tree structure. For simplicity, we limit attention to binary tree structures in which every level is fully populated. We let I form the root of the tree, and give it the index 1, $I_1 = I$. We next split the root into two roughly equi-sized vectors I_2 and I_3 so that $I_1 = I_2 \cup I_3$. The full tree is then formed by continuing to subdivide any interval that holds more than some preset fixed number n of indices. We use the integers $\ell = 0, 1, \ldots, L$ to label the different levels, with 0 denoting the coarsest level. A *leaf* is a node corresponding to a vector that never got split. For a non-leaf node τ, its *children* are the two boxes σ_1 and σ_2 such that $I_\tau = I_{\sigma_1} \cup I_{\sigma_2}$, and τ is then the *parent* of σ_1 and σ_2. Two boxes with the same parent are called *siblings*. These definitions are illustrated in Figure 2.2

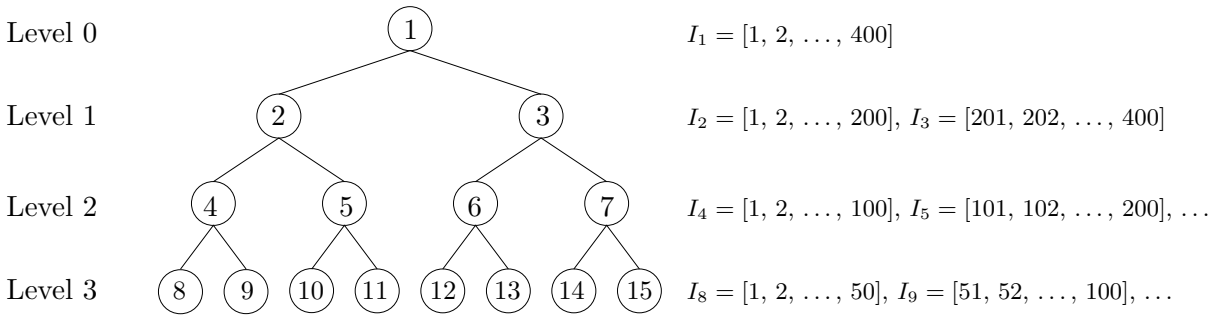

FIGURE 2.2. Numbering of nodes in a fully populated binary tree with $L = 3$ levels. The root is the original index vector $I = I_1 = [1, 2, \ldots, 400]$.

We now say that an $N \times N$ matrix $\mathbf{A}$ is an $\mathcal{S}$-matrix of rank p if for every sibling pair $\{\sigma, \tau\}$ of nodes in the tree the corresponding off-diagonal block of $\mathbf{A}$ admits a low-rank factorization

$$\begin{array}{cccccc} \mathbf{A}(I_\tau, I_\sigma) & = & \mathbf{X}_\tau & \tilde{\mathbf{A}}_{\tau,\sigma} & \mathbf{Y}_\sigma^*, \\ N_\tau \times N_\sigma & & N_\tau \times p & p \times p & p \times N_\sigma \end{array} \tag{2.2}$$

where N_τ and N_σ denote the number of elements in I_τ and I_σ, respectively.

2.5. Nested basis matrices

The $\mathcal{S}$-matrix format has the virtue of simplicity, but it is not very efficient. It requires $O(Np \log N)$ real numbers for storage, the matrix-vector product has complexity $O(Np \log N)$, and higher level operations such as matrix inversion or the matrix-matrix product involve higher powers of the $\log N$ factor. We will next develop a more efficient format that eliminates these $\log N$ factors. Our first step in this direction is to develop a more efficient way of storing the basis matrices (we must clearly get storage from $O(N \log N)$ down to $O(N)$ if there is to be any hope for $O(N)$ arithmetic!).

The general idea is to express the basis matrices for the off-diagonal blocks hierarchically. In other words, we will express the basis vectors used at one level in terms of the basis vectors used at the next finer level. Using the matrix illustrated in Figure 2.1 as an example, a basis for the column space of the block labelled $(4, 5)$ is constructed from the bases for the column spaces of the blocks $(8, 9)$ and $(9, 8)$. To formalize this notion, let τ be a node in the tree with sibling σ, and children β_1 and β_2. Then we require that the basis matrix $\mathbf{X}_\tau$ that spans the column space of $\mathbf{A}(I_\tau, I_\sigma)$ can be expressed in terms of the corresponding basis matrices $\mathbf{X}_{\beta_1}$ and $\mathbf{X}_{\beta_2}$ associated with its children,

$$\begin{array}{ccccc} \mathbf{X}_\tau & = & \begin{bmatrix} \mathbf{X}_{\beta_1} & \mathbf{0} \\ \mathbf{0} & \mathbf{X}_{\beta_2} \end{bmatrix} & \mathbf{U}_\tau \\ N_\tau \times k & & N_\tau \times 2k & 2k \times k \end{array} \tag{2.3}$$

Observe that when (2.3) holds, the matrix $\mathbf{X}_\tau$ (which is potentially very tall) need not be stored — it can when needed be constructed from $\mathbf{X}_{\beta_1}$ and $\mathbf{X}_{\beta_2}$ using the information in the small matrix $\mathbf{U}_\tau$. The process can then be continued down the tree to eliminate the need for storing the matrices $\mathbf{X}_{\beta_1}$ and $\mathbf{X}_{\beta_2}$. Let ν_1 and ν_2 denote the children of β_1 and let ν_3 and ν_4 be the children of β_2. Then we impose conditions, cf. (2.3),

$$\mathbf{X}_{\beta_1} = \begin{bmatrix} \mathbf{X}_{\nu_1} & \mathbf{0} \\ \mathbf{0} & \mathbf{X}_{\nu_2} \end{bmatrix} \mathbf{U}_{\beta_1} \qquad \text{and} \qquad \mathbf{X}_{\beta_2} = \begin{bmatrix} \mathbf{X}_{\nu_3} & \mathbf{0} \\ \mathbf{0} & \mathbf{X}_{\nu_4} \end{bmatrix} \mathbf{U}_{\beta_2} \tag{2.4}$$

Combining (2.3) and (2.4) we then find

$$\underset{N_\tau \times k}{\mathbf{X}_\tau} = \underset{N_\tau \times 4k}{\begin{bmatrix} \mathbf{X}_{\nu_1} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{X}_{\nu_2} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{X}_{\nu_3} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{X}_{\nu_4} \end{bmatrix}} \underset{4k \times 2k}{\begin{bmatrix} \mathbf{U}_{\beta_1} & \mathbf{0} \\ \mathbf{0} & \mathbf{U}_{\beta_2} \end{bmatrix}} \underset{2k \times k}{\mathbf{U}_\tau}.$$

This process can be continued all the way down to the leaves. The end result is that we only need to store the full basis matrices for the leaf nodes (which is fine since these are small), and the small matrices $\mathbf{U}_\tau$ for all parent nodes.

Of course, we also assume that the basis matrices $\{\mathbf{Y}_\tau\}_{\tau \in \mathcal{T}}$ for the row spaces satisfy an analogous hierarchical property. In other words, for any parent node τ with children β_1 and β_2, we require that there exist a $2k \times k$ matrix $\mathbf{V}_\tau$ such that

$$\underset{N_\tau \times k}{\mathbf{Y}_\tau} = \underset{N_\tau \times 2k}{\begin{bmatrix} \mathbf{Y}_{\beta_1} & \mathbf{0} \\ \mathbf{0} & \mathbf{Y}_{\beta_2} \end{bmatrix}} \underset{2k \times k}{\mathbf{V}_\tau} \tag{2.5}$$

REMARK 2.4. *It is in practice a simple matter to construct basis matrices that satisfy (2.3) and (2.5). What we need to do is to make sure that for every leaf τ, the columns of the basis matrix $\mathbf{X}_\tau$ span the column space of $\mathbf{A}(I_\tau, I_\tau^c)$ where I_τ^c is the complement of I_τ within the vector I. Analogously, the columns of $\mathbf{Y}_\tau$ must span the row space of $\mathbf{A}(I_\tau^c, I_\tau)$. Note that these are quite strict requirements since $\mathbf{A}(I_\tau, I_\tau^c)$ is a much larger matrix than the matrix $\mathbf{A}(I_\tau, I_\sigma)$ in (2.2) (since I_σ is a (potentially very small) subset of I_τ^c). Note also that while conceptually simple, the factorizations of matrices like $\mathbf{A}(I_\tau, I_\tau^c)$ would be very expensive is performed by brute force. In the direct solvers described in this text, there are usually short-cuts that make these computations very inexpensive, see, e.g., Sections 7.8.3 and 8.5*

2.6. The hierarchically block separable (HBS) sparse matrix format

We are now prepared to rigorously define what it means for an $N \times N$ matrix $\mathbf{A}$ to be *hierarchically block seperable (HBS)* with respect to a given binary tree $\mathcal{T}$ that partitions the index vector $I = [1, 2, \ldots, N]$. For simplicity, we suppose that the tree has L fully populated levels, and that for every leaf node τ, the index vector I_τ holds precisely n points, so that $N = n\,2^L$. Then $\mathbf{A}$ is HBS with block rank k if the following two conditions hold:

(1) Assumption on ranks of off-diagonal blocks at the finest level: For any two distinct leaf nodes τ and τ', define the $n \times n$ matrix

$$\mathbf{A}_{\tau,\tau'} = \mathbf{A}(I_\tau, I_{\tau'}). \tag{2.6}$$

Then there must exist matrices $\mathbf{U}_\tau$, $\mathbf{V}_{\tau'}$, and $\tilde{\mathbf{A}}_{\tau,\tau'}$ such that

$$\underset{n \times n}{\mathbf{A}_{\tau,\tau'}} = \underset{n \times k}{\mathbf{U}_\tau} \underset{k \times k}{\tilde{\mathbf{A}}_{\tau,\tau'}} \underset{k \times n}{\mathbf{V}_{\tau'}^*}. \tag{2.7}$$

(2) Assumption on ranks of off-diagonal blocks on level $\ell = L-1, L-2, \ldots, 1$: The rank assumption at level ℓ is defined in terms of the blocks constructed on the next finer level $\ell + 1$: For any distinct nodes τ and τ' on level ℓ with children σ_1, σ_2 and σ_1', σ_2', respectively, define

$$\mathbf{A}_{\tau,\tau'} = \begin{bmatrix} \tilde{\mathbf{A}}_{\sigma_1,\sigma_1'} & \tilde{\mathbf{A}}_{\sigma_1,\sigma_2'} \\ \tilde{\mathbf{A}}_{\sigma_2,\sigma_1'} & \tilde{\mathbf{A}}_{\sigma_2,\sigma_2'} \end{bmatrix}. \tag{2.8}$$

	Name:	Size:	Function:
For each leaf node τ:	$\mathbf{D}_\tau$	$n \times n$	The diagonal block $\mathbf{A}(I_\tau, I_\tau)$.
	$\mathbf{U}_\tau$	$n \times k$	Basis for the columns in the blocks in row τ.
	$\mathbf{V}_\tau$	$n \times k$	Basis for the rows in the blocks in column τ.
For each parent node τ:	$\mathbf{B}_\tau$	$2k \times 2k$	Interactions between the children of τ.
	$\mathbf{U}_\tau$	$2k \times k$	Basis for the columns in the (reduced) blocks in row τ.
	$\mathbf{V}_\tau$	$2k \times k$	Basis for the rows in the (reduced) blocks in column τ.

FIGURE 2.3. An HBS matrix $\mathbf{A}$ associated with a tree $\mathcal{T}$ is fully specified if the factors listed above are provided.

Then there must exist matrices $\mathbf{U}_\tau$, $\mathbf{V}_{\tau'}$, and $\tilde{\mathbf{A}}_{\tau,\tau'}$ such that

$$\underset{2k \times 2k}{\mathbf{A}_{\tau,\tau'}} = \underset{2k \times k}{\mathbf{U}_\tau} \quad \underset{k \times k}{\tilde{\mathbf{A}}_{\tau,\tau'}} \quad \underset{k \times 2k}{\mathbf{V}_{\tau'}^*}. \tag{2.9}$$

The two points above complete the definition. An HBS matrix is now fully described if the basis matrices $\mathbf{U}_\tau$ and $\mathbf{V}_\tau$ are provided for each node τ, and in addition, we are for each leaf τ given the $n \times n$ matrix

$$\mathbf{D}_\tau = \mathbf{A}(I_\tau, I_\tau), \tag{2.10}$$

and for each parent node τ with children σ_1 and σ_2 we are given the $2k \times 2k$ matrix

$$\mathbf{B}_\tau = \begin{bmatrix} \mathbf{0} & \tilde{\mathbf{A}}_{\sigma_1,\sigma_2} \\ \tilde{\mathbf{A}}_{\sigma_2,\sigma_1} & \mathbf{0} \end{bmatrix}. \tag{2.11}$$

Observe in particular that the matrices $\tilde{\mathbf{A}}_{\sigma_1,\sigma_2}$ are only required when $\{\sigma_1, \sigma_2\}$ forms a sibling pair. Figure 2.3 summarizes the required matrices, and Algorithm III shows how to evaluate the product $\mathbf{x} \mapsto \mathbf{A}\mathbf{x}$ using these factors.

REMARK 2.5. *The definition of the HBS property given in this section is flexible in the sense that we do not enforce any conditions on the factors $\mathbf{U}_\tau$, $\mathbf{V}_\tau$, and $\tilde{\mathbf{A}}_{\tau,\tau'}$ other than that (2.7) and (2.9) must hold. For purposes of numerical stability, further conditions are sometimes imposed. The perhaps strongest such condition is to require the matrices $\mathbf{U}_\tau$ and $\mathbf{V}_{\tau'}$ in (2.7) and (2.9) be orthonormal, see e.g.* [**5**] *(one can in this case require that the matrices $\tilde{\mathbf{A}}_{\tau,\tau'}$ be diagonal, so that (2.7) and (2.9) become singular value decompositions) . A choice that we have found highly convenient is to require (2.7) and (2.9) to be so called interpolatory decompositions (see Section 7.2.2). Then every $\mathbf{U}_\tau$ and $\mathbf{V}_{\tau'}$ contains a $k \times k$ identity matrix (which greatly accelerates computations), and each $\tilde{\mathbf{A}}_{\tau,\tau'}$ is a submatrix of the original matrix $\mathbf{A}$.*

2.7. Inversion of an HBS matrix

An important advantage of the HBS matrix format is that it allows a very easy and fast inversion algorithm. It is given here as Algorithm IV; it takes as input the matrices $\{\mathbf{D}_\tau, \mathbf{B}_\tau, \mathbf{U}_\tau, \mathbf{V}_\tau\}_{\tau \in \mathcal{T}}$ in an HBS representation of a matrix $\mathbf{A}$ and provides as output corresponding factors $\{\mathbf{G}_\tau, \mathbf{E}_\tau, \mathbf{F}_\tau\}_{\tau \in \mathcal{T}}$ in a representation of $\mathbf{A}^{-1}$. Once these factors are available, the product $\mathbf{x} \mapsto \mathbf{A}^{-1}\mathbf{x}$ can be constructed via Algorithm V.

The derivation of the factorization algorithm is given in Chapter 7, here we merely list the inversion algorithm to highlight the fact that the HBS format is not only faster than the $\mathcal{S}$-matrix format,

ALGORITHM III (HBS matrix-vector multiply)

Given a vector $\mathbf{q}$ *and a matrix* $\mathbf{A}$ *in HBS format, compute* $\mathbf{u} = \mathbf{A}\,\mathbf{q}$.

loop over all leaf boxes τ
    $\hat{\mathbf{q}}_\tau = \mathbf{V}_\tau^*\,\mathbf{q}(I_\tau)$.
end loop

loop over levels, finer to coarser, $\ell = L-1,\ L-2,\ \ldots,\ 1$
    loop over all parent boxes τ on level ℓ,
        Let σ_1 and σ_2 denote the children of τ.
        $\hat{\mathbf{q}}_\tau = \mathbf{V}_\tau^* \begin{bmatrix} \hat{\mathbf{q}}_{\sigma_1} \\ \hat{\mathbf{q}}_{\sigma_2} \end{bmatrix}$.
    end loop
end loop

$\hat{\mathbf{u}}_1 = 0$
loop over all levels, coarser to finer, $\ell = 1,\ 2,\ \ldots,\ L-1$
    loop over all parent boxes τ on level ℓ
        Let σ_1 and σ_2 denote the children of τ.
        $\begin{bmatrix} \hat{\mathbf{u}}_{\sigma_1} \\ \hat{\mathbf{u}}_{\sigma_2} \end{bmatrix} = \mathbf{U}_\tau\,\hat{\mathbf{u}}_\tau + \begin{bmatrix} 0 & \mathbf{B}_{\sigma_1,\sigma_2} \\ \mathbf{B}_{\sigma_2,\sigma_1} & 0 \end{bmatrix} \begin{bmatrix} \hat{\mathbf{q}}_{\sigma_1} \\ \hat{\mathbf{q}}_{\sigma_2} \end{bmatrix}$.
    end loop
end loop

loop over all leaf boxes τ
    $\mathbf{u}(I_\tau) = \mathbf{U}_\tau\,\hat{\mathbf{u}}_\tau + \mathbf{D}_\tau\,\mathbf{q}(I_\tau)$.
end loop

ALGORITHM IV (inversion of an HBS matrix)

loop over all levels, finer to coarser, $\ell = L,\ L-1,\ \ldots,\ 1$
    loop over all boxes τ on level ℓ,
        if τ is a leaf node
            $\tilde{\mathbf{D}}_\tau = \mathbf{D}_\tau$
        else
            Let σ_1 and σ_2 denote the children of τ.
            $\tilde{\mathbf{D}}_\tau = \begin{bmatrix} \hat{\mathbf{D}}_{\sigma_1} & \mathbf{B}_{\sigma_1,\sigma_2} \\ \mathbf{B}_{\sigma_2,\sigma_1} & \hat{\mathbf{D}}_{\sigma_2} \end{bmatrix}$
        end if
        $\hat{\mathbf{D}}_\tau = \left(\mathbf{V}_\tau^*\,\tilde{\mathbf{D}}_\tau^{-1}\,\mathbf{U}_\tau\right)^{-1}$.
        $\mathbf{E}_\tau = \tilde{\mathbf{D}}_\tau^{-1}\,\mathbf{U}_\tau\,\hat{\mathbf{D}}_\tau$.
        $\mathbf{F}_\tau^* = \hat{\mathbf{D}}_\tau\,\mathbf{V}_\tau^*\,\tilde{\mathbf{D}}_\tau^{-1}$.
        $\mathbf{G}_\tau = \hat{\mathbf{D}}_\tau - \tilde{\mathbf{D}}_\tau^{-1}\,\mathbf{U}_\tau\,\hat{\mathbf{D}}_\tau\,\mathbf{V}_\tau^*\,\tilde{\mathbf{D}}_\tau^{-1}$.
    end loop
end loop
$\mathbf{G}_1 = \begin{bmatrix} \hat{\mathbf{D}}_2 & \mathbf{B}_{2,3} \\ \mathbf{B}_{3,2} & \hat{\mathbf{D}}_3 \end{bmatrix}^{-1}$.

but is also in fact easier to use. Observe that since the inversion is exact, there is no need for re-compression in Algorithm III.

In practice, the off-diagonal blocks of a matrix $\mathbf{A}$ that we seek to invert are not exactly of rank p, but can to some finite precision be approximated by low-rank matrices. Suppose that these errors

ALGORITHM V (application of inverse)

Given a vector $\mathbf{u}$*, compute* $\mathbf{q} = \mathbf{A}^{-1}\,\mathbf{u}$ *using the compressed representation of* $\mathbf{A}^{-1}$ *resulting from Algorithm ?.*

loop over all leaf boxes τ
  $\hat{\mathbf{u}}_\tau = \mathbf{F}_\tau^*\,\mathbf{u}(I_\tau)$.
end loop

loop over all levels, finer to coarser, $\ell = L,\, L-1,\, \ldots, 1$
  loop over all parent boxes τ on level ℓ,
    Let σ_1 and σ_2 denote the children of τ.
    $\hat{\mathbf{u}}_\tau = \mathbf{F}_\tau^* \begin{bmatrix} \hat{\mathbf{u}}_{\sigma_1} \\ \hat{\mathbf{u}}_{\sigma_2} \end{bmatrix}$.
  end loop
end loop

$\begin{bmatrix} \hat{\mathbf{q}}_2 \\ \hat{\mathbf{q}}_3 \end{bmatrix} = \hat{\mathbf{G}}_1 \begin{bmatrix} \hat{\mathbf{u}}_2 \\ \hat{\mathbf{u}}_3 \end{bmatrix}$.

loop over all levels, coarser to finer, $\ell = 1,\, 2,\, \ldots,\, L-1$
  loop over all parent boxes τ on level ℓ
    Let σ_1 and σ_2 denote the children of τ.
    $\begin{bmatrix} \hat{\mathbf{q}}_{\sigma_1} \\ \hat{\mathbf{q}}_{\sigma_2} \end{bmatrix} = \mathbf{E}_\tau\,\hat{\mathbf{u}}_\tau + \mathbf{G}_\tau \begin{bmatrix} \hat{\mathbf{u}}_{\sigma_1} \\ \hat{\mathbf{u}}_{\sigma_2} \end{bmatrix}$.
  end loop
end loop

loop over all leaf boxes τ
  $\mathbf{q}(I_\tau) = \mathbf{E}_\tau\,\hat{\mathbf{q}}_\tau + \mathbf{G}_\tau\,\mathbf{u}(I_\tau)$.
end loop

are apportioned among the blocks in such a way that the HBS approximant $\mathbf{A}_\varepsilon$ satisfies

$$||\mathbf{A} - \mathbf{A}_\varepsilon|| \leq \varepsilon.$$

Then Algorithm IV computes an exact inverse $\mathbf{A}_\varepsilon^{-1}$. We find that

$$\mathbf{A}^{-1} - \mathbf{A}_\varepsilon^{-1} = \mathbf{A}^{-1}\big(\mathbf{A}_\varepsilon - \mathbf{A}\big)\mathbf{A}_\varepsilon^{-1}.$$

In consequence,

$$\frac{||\mathbf{A}^{-1} - \mathbf{A}_\varepsilon^{-1}||}{||\mathbf{A}^{-1}||} \leq ||\mathbf{A}_\varepsilon^{-1}||\,||\mathbf{A}_\varepsilon - \mathbf{A}|| \leq ||\mathbf{A}_\varepsilon^{-1}||\,\varepsilon.$$

We see that the error is magnified by a factor of $||\mathbf{A}_\varepsilon^{-1}||$, which is quite natural. Note that $||\mathbf{A}_\varepsilon^{-1}||$ can easily be estimated via a power iteration on $\big(\mathbf{A}_\varepsilon^{-1}\big)\big(\mathbf{A}_\varepsilon^{-1}\big)^*$ executed using Algorithm V (and its modified version for application of $\big(\mathbf{A}_\varepsilon^{-1}\big)^*$).

To be slightly more careful, let us next take into account rounding errors incurred when Algorithm IV is executed in floating point arithmetic. It would be hard to construct an à priori estimate of these errors, but that is not necessary. In practice, we rely instead on an à posteriori estimate as follows: Suppose that we seek to solve the equation

$$\mathbf{A}\mathbf{x} = \mathbf{f}.$$

We approximate $\mathbf{A}$ by an HBS matrix $\mathbf{A}_\varepsilon$ and then invert this matrix (in floating point arithmetic) to construct the approximate inverse $\mathbf{B}$. Finally, we form the approximate solution via

$$\mathbf{x}_{\text{approx}} = \mathbf{B}\mathbf{f}.$$

Then

$$||\mathbf{x}_{\text{approx}} - \mathbf{x}|| = ||\mathbf{Bf} - \mathbf{x}|| = ||\mathbf{BAx} - \mathbf{x}|| \leq ||\mathbf{BA} - \mathbf{I}||\,||\mathbf{x}||.$$

The relative error is therefore bounded by $||\mathbf{BA} - \mathbf{I}||$. In any situation where we can evaluate the product $\mathbf{x} \mapsto \mathbf{Ax}$ to very high accuracy (significantly smaller than ε, in other words), it is then an easy matter to estimate $||\mathbf{BA} - \mathbf{I}||$ via a power iteration on $\big(\mathbf{BA} - \mathbf{I}\big)\big(\mathbf{BA} - \mathbf{I}\big)^*$.

Bibliography

[1] Mario Bebendorf, *Hierarchical matrices*, Lecture Notes in Computational Science and Engineering, vol. 63, Springer-Verlag, Berlin, 2008, A means to efficiently solve elliptic boundary value problems. MR 2451321 (2009k:15001)

[2] Steffen Börm, *Efficient numerical methods for non-local operators*, EMS Tracts in Mathematics, vol. 14, European Mathematical Society (EMS), Zürich, 2010, $\mathcal{H}^2$-matrix compression, algorithms and analysis. MR 2767920

[3] Lars Grasedyck and Wolfgang Hackbusch, *Construction and arithmetics of $\mathcal{H}$-matrices*, Computing **70** (2003), no. 4, 295–334.

[4] Wolfgang Hackbusch, *A sparse matrix arithmetic based on H-matrices; Part I: Introduction to H-matrices*, Computing **62** (1999), 89–108.

[5] Zhifeng Sheng, Patrick Dewilde, and Shivkumar Chandrasekaran, *Algorithms to solve hierarchically semi-separable systems*, System theory, the Schur algorithm and multidimensional analysis, Oper. Theory Adv. Appl., vol. 176, Birkhäuser, Basel, 2007, pp. 255–294. MR MR2342902