SIAM J. SCI. COMPUT.
Vol. 41, No. 5, pp. S61–S85

ROBUST AND ACCURATE STOPPING CRITERIA FOR ADAPTIVE RANDOMIZED SAMPLING IN MATRIX-FREE HIERARCHICALLY SEMISEPARABLE CONSTRUCTION*

CHRISTOPHER GORMAN†, GUSTAVO CHÁVEZ‡, PIETER GHYSELS‡, THÉO MARY§, FRANÇOIS-HENRY ROUET¶, AND XIAOYE SHERRY LI‡

Abstract. We present new algorithms for randomized construction of hierarchically semiseparable (HSS) matrices, addressing several practical issues. The HSS construction algorithms use a partially matrix-free, adaptive randomized projection scheme to determine the maximum off-diagonal block rank. We develop both relative and absolute stopping criteria to determine the minimum dimension of the random projection matrix that is sufficient for desired accuracy. Two strategies are discussed to adaptively enlarge the random sample matrix: repeated doubling of the number of random vectors and iteratively incrementing the number of random vectors by a fixed number. The relative and absolute stopping criteria are based on probabilistic bounds for the Frobenius norm of the random projection of the Hankel blocks of the input matrix. We discuss parallel implementation and computation and communication cost of both variants. Parallel numerical results for a range of applications, including boundary element method matrices and quantum chemistry Toeplitz matrices, show the effectiveness, scalability, and numerical robustness of the proposed algorithms.

Key words. HSS matrix, randomized sampling, adaptivity

AMS subject classification. 65Fxx

DOI. 10.1137/18M1194961

1. Introduction.

Randomization schemes have proven to be a powerful tool for computing a low-rank approximation of a dense matrix or, as we call it in this work, *compressing* it. The main advantage of randomization is that these methods usually require fewer computations and communication than their traditional deterministic counterparts, resulting in large savings in terms of memory and floating point operations.

For classes of dense matrices that have off-diagonal blocks that can be approximated as low-rank submatrices, randomized methods are particularly advantageous. These matrices are referred to as *structured*, and there are many types of matrix formats that can take advantage of this structure; these include, to name a few, hierarchically semiseparable (HSS) matrices [3], $\mathcal{H}$ and $\mathcal{H}^2$ matrices [11, 10]. This work focuses on HSS representations and, more specifically, efficient HSS compression. HSS compression is the central component of the HSS framework; once a matrix

*Received by the editors June 18, 2018; accepted for publication (in revised form) February 13, 2019; published electronically October 29, 2019.
https://doi.org/10.1137/18M1194961
Funding: This work was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration, and in part by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Scientific Discovery through Advanced Computing (SciDAC) program. This work was also supported by the National Energy Research Scientific Computing Center (NERSC), a U.S. Department of Energy Office of Science User Facility operated under Contract DE-AC02-05CH11231.

†University of California Santa Barbara, Santa Barbara, CA 93106 (gorman@math.ucsb.edu).
‡Computational Research, Lawrence Berkeley National Laboratory, Berkeley, CA 94720 (gichavez@lbl.gov, pghysels@lbl.gov, xsli@lbl.gov).
§University of Manchester, Manchester, M13 9PL UK (theo.mary@manchester.ac.uk).
¶Livermore Software Technology Corporation, Livermore, CA 94551 (fhrouet@lstc.com).

is compressed into its HSS form, one can take advantage of fast algorithms for multiplication, factorization, etc.

One way to speed up HSS compression involves using randomization [16, 12]. Randomization involves generating *samples* of size at least the maximum rank of the HSS representation. Since the exact rank of low-rank matrices is usually not known in practice, adaptive algorithms are needed in order to generate sufficient, yet not too many, random samples until the range is well approximated and the matrix is compressed to a desired tolerance. This ensures robustness and high performance of the overall algorithm.

This paper builds on our previous work [18], which gives an explicit adaptive algorithm. One of the highlights of this work is the development of a new stopping criterion for adaptation that considers both relative and absolute error. We demonstrate the effectiveness of this novel approach, and others, in a set of numerical experiments that showcase the scalability and robustness of the new algorithms on a variety of matrices from different applications.

The paper is organized as follows. In section 2, we discuss the HSS randomized construction algorithm, while in section 3 we present two adaptive sampling strategies, the new stopping criterion for terminating adaptation, and the related probability theory. The parallel algorithms are presented and analyzed in section 4, followed by numerical experiments in section 5.

2. HSS matrices and randomized construction. In this section we briefly introduce the HSS representation and the randomized construction algorithm that we use. The following notation is used: * denotes complex transposition, $\#I_\phi$ is the number of elements in index set $I_\phi = \{i_1, i_2, \ldots, i_n\}$, $R_\phi = R(I_\phi, :)$ is the matrix consisting of only the rows I_ϕ of matrix R using all columns.

Consider a square matrix $A \in \mathbb{C}^{N\times N}$, an index set $I_A = \{1, \ldots, N\}$, and a binary tree $\mathcal{T}$, ordered level by level, starting with zero at the root node. Each node ϕ of the tree is associated with a contiguous subset $I_\phi \subset I_A$. For two siblings ν_1 and ν_2, children of ϕ, it holds that $I_{\nu_1} \cup I_{\nu_2} = I_\phi$ and $I_{\nu_1} \cap I_{\nu_2} = \emptyset$. It follows that $\cup_{\phi\in\text{leaves}(\mathcal{T})} I_\phi = I_{\text{root}(\mathcal{T})} = I_A$. The same tree $\mathcal{T}$ is used for the rows and the columns of A, and only diagonal blocks are partitioned. An example of the resulting matrix partitioning is given in Figure 1 and the corresponding tree is shown in Figure 2.

The diagonal blocks of A, denoted D_ϕ, are stored as dense matrices in the leaves ϕ of the tree $\mathcal{T}$: $D_\phi = A(I_\phi, I_\phi)$. The off-diagonal blocks $A_{\nu_1,\nu_2} = A(I_{\nu_1}, I_{\nu_2})$ are factored (approximately) as

$$A_{\nu_1,\nu_2} \approx U^{\text{big}}_{\nu_1} B_{\nu_1,\nu_2} \left(V^{\text{big}}_{\nu_2}\right)^* , \tag{2.1}$$

where $U^{\text{big}}_{\nu_1}$ is $\#I_{\nu_1} \times r^r_{\nu_1}$ and $V^{\text{big}}_{\nu_2}$ is $\#I_{\nu_2} \times r^c_{\nu_2}$.[1] The HSS-rank r is defined as the maximum of r^r_ϕ and r^c_ϕ over all off-diagonal blocks, where typically $r \ll N$. B_{ν_1,ν_2} and B_{ν_2,ν_1} are stored at the parent node. For a node ϕ with children ν_1 and ν_2, U^{big}_ϕ and V^{big}_ϕ are represented hierarchically as

$$U^{\text{big}}_\phi = \begin{bmatrix} U^{\text{big}}_{\nu_1} & 0 \\ 0 & U^{\text{big}}_{\nu_2} \end{bmatrix} U_\phi \quad \text{and} \quad V^{\text{big}}_\phi = \begin{bmatrix} V^{\text{big}}_{\nu_1} & 0 \\ 0 & V^{\text{big}}_{\nu_2} \end{bmatrix} V_\phi \,. \tag{2.2}$$

[1] Superscripts r and c are used to denote that $U^{\text{big}}/V^{\text{big}}$ are column/row bases for the row/column Hankel blocks of A.

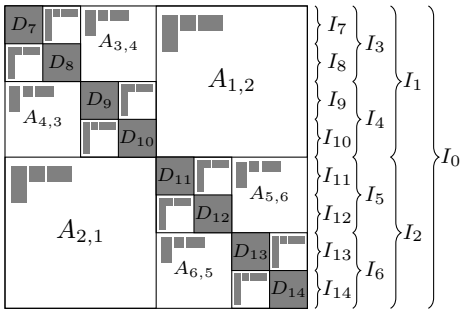

FIG. 1. *Illustration of an HSS matrix using* 4 *levels. Diagonal blocks are partitioned recursively. Gray blocks denote the U, B, V, and D matrices.*

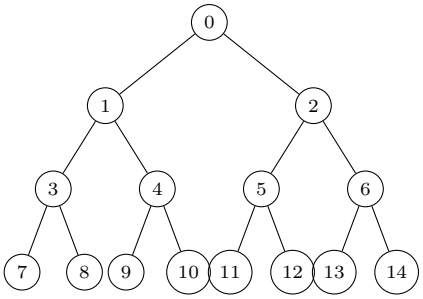

FIG. 2. *Tree for Figure* 1*, using level-by-level top-down numbering. All nodes except the root store U_μ and V_μ. Leaves store D_μ, nonleaves B_{ν_1,ν_2}, B_{ν_2,ν_1}.*

Note that for a leaf node $U_\phi^{\text{big}} = U_\phi$ and $V_\phi^{\text{big}} = V_\phi$. Hence, every node ϕ, except the root, keeps matrices U_ϕ and V_ϕ. The top two levels of the example shown in Figure 1 can be written out explicitly as

$$A = \begin{bmatrix} D_3 & U_3 B_{3,4} V_4^* & \begin{bmatrix} U_3 & 0 \\ 0 & U_4 \end{bmatrix} U_1 B_{1,2} V_2^* \begin{bmatrix} V_5^* & 0 \\ 0 & V_6^* \end{bmatrix} \\ U_4 B_{4,3} V_3^* & D_4 & \\ \begin{bmatrix} U_5 & 0 \\ 0 & U_6 \end{bmatrix} U_2 B_{2,1} V_1^* \begin{bmatrix} V_3^* & 0 \\ 0 & V_4^* \end{bmatrix} & D_5 & U_5 B_{5,6} V_6^* \\ & U_6 B_{6,5} V_5^* & D_6 \end{bmatrix}. \tag{2.3}$$

An HSS matrix requires $\mathcal{O}(rN)$ storage and can perform matrix-vector multiplication in $\mathcal{O}(rN)$ operations, compared to $\mathcal{O}(N^2)$ for classical dense matrix-vector multiplication. We refer the reader to the standard HSS references [3, 28] for more details.

HSS matrix construction (or *compression*) based on randomized sampling techniques has attracted a lot of attention in recent years. Compared to standard HSS construction techniques [27, 22] that assume that an explicit matrix is given in input, randomized techniques allow the design of *matrix-free* construction algorithms. A *fully matrix-free* construction algorithm relies solely on the availability of a matrix-vector product function. A *partially matrix-free* algorithm also relies on a matrix-vector product function but additionally requires access to individual entries of the matrix. For certain applications, for example, Toeplitz systems, where fast (e.g., linear time) matrix-vector products exist, a randomized algorithm typically has linear or log-linear complexity instead of quadratic complexity for standard construction algorithms.

There are two main algorithms for randomized HSS construction. The algorithm by Lin, Lu, and Ying [13] is fully matrix-free. It follows a top-down traversal of the HSS tree. It has been succesfully used to design linear solvers and eigensolvers for different applications [25, 14]. The randomized HSS construction algorithm by Martinsson [16] is partially matrix-free. It requires less random projections (by a factor of $\mathcal{O}(\log N)$) but requires access to $\mathcal{O}(rN)$ matrix elements; it follows a bottom-up traversal of the HSS tree. This algorithm has been used to build different dense and sparse solvers [29, 26] and is the basis for our work. Our distributed-memory randomized HSS construction implemented in STRUMPACK [18] is based on this algorithm, and the rest of this section recalls the main ideas.

Let R be an $N \times d$ random matrix, where $d = r + p$ with p a small oversampling parameter. The matrix U_ϕ^{big} is a basis for the column space of $A(I_\phi, I_A \setminus I_\phi)$, which is called a Hankel block. Likewise, V_ϕ^{big} is a basis for the row space of $A(I_A \setminus I_\phi, I_\phi)^*$.

To compute the random projection of these Hankel blocks, we compute $S^r = AR$ and $S^c = A^*R$ such that $S^r(I_\phi, :)$ gives the random projection of $A(I_\phi, :)$. Let D_ϕ for nonleaf nodes be defined (recursively) as $D_\phi = \left[\begin{smallmatrix} D_{\nu_1} & A_{\nu_1,\nu_2} \\ A_{\nu_2,\nu_1} & D_{\nu_2} \end{smallmatrix}\right]$, and with nodes ϕ on level ℓ of the HSS tree, we define the block diagonal matrix $D^{(\ell)} = \operatorname{diag}\left(D_{\phi_i}, \ldots, D_{\phi_j}\right)$. At each level of the HSS tree we can compute the samples of the Hankel blocks as $S^{r,(\ell)} = \left(A - D^{(\ell)}\right) R = S^r - D^{(\ell)} R$ and $S^{c,(\ell)} = S^c - D^{(\ell)*} R$. Working from the leaves up to the root this can be performed efficiently since the off-diagonal blocks of D_ϕ are already compressed. Consider a leaf node ϕ, and let $S^r_\phi = S^r(I_\phi, :) - D_\phi R(I_\phi, :)$. To compute U_ϕ from this, an interpolative decomposition (ID) is used, which approximates S^r_ϕ as a linear combination of a set J^r_ϕ of its rows: $S^r_\phi = U_\phi S^r_\phi(J^r_\phi, :) + \mathcal{O}(\varepsilon)$, where $\#J^r_\phi$ is the ε-rank of S^r_ϕ. This decomposition can be computed using a rank-revealing QR (RRQR) factorization, or QR with column pivoting (QRCP), applied to $(S^r_\phi)^*$ as follows:

(2.4)
$$(S^r_\phi)^* \approx Q \begin{bmatrix} R_1 & R_2 \end{bmatrix} \Pi^T = (QR_1) \begin{bmatrix} I & R_1^{-1} R_2 \end{bmatrix} \Pi^T = (S^r_\phi(J^r_\phi, :))^* \begin{bmatrix} I & R_1^{-1} R_2 \end{bmatrix} \Pi^T,$$

with Π a permutation matrix, Q orthogonal, and R_1 upper triangular. We stop and truncate the RRQR factorization when $|R_{k,k}| \leq \varepsilon_{\text{abs}}$ or $|R_{k,k}| / |R_{1,1}| \leq \varepsilon_{\text{rel}}$. From (2.4), U_ϕ can be defined as $U_\phi = \Pi \left[\begin{smallmatrix} I \\ \left(R_1^{-1} R_2\right)^* \end{smallmatrix}\right] = \Pi^r_\phi \left[\begin{smallmatrix} I \\ E^r_\phi \end{smallmatrix}\right]$; and likewise, $V_\phi = \Pi^c_\phi \left[\begin{smallmatrix} I \\ E^c_\phi \end{smallmatrix}\right]$. From these definitions of U_ϕ and V_ϕ, it follows that $B_{\nu_1,\nu_2} = A_{\nu_1,\nu_2}(J^r_{\nu_1}, J^c_{\nu_2})$.

For a nonleaf node ϕ with (leaf) children ν_1 and ν_2 we need to guarantee the nested basis property, (2.2). We have

$$S^{r,(\ell)}_\phi = \left(A(I_\phi, :) - D_\phi\right) R = A(I_\phi, :) R - \begin{bmatrix} D_{\nu_1} & A_{\nu_1,\nu_2} \\ A_{\nu_2,\nu_1} & D_{\nu_2} \end{bmatrix} \begin{bmatrix} R(I_{\nu_1}, :) \\ R(I_{\nu_2}, :) \end{bmatrix}, \tag{2.5}$$

$$= \begin{bmatrix} S^r_{\nu_1} \\ S^r_{\nu_2} \end{bmatrix} - \begin{bmatrix} & A_{\nu_1,\nu_2} \\ A_{\nu_2,\nu_1} & \end{bmatrix} \begin{bmatrix} R(I_{\nu_1}, :) \\ R(I_{\nu_2}, :) \end{bmatrix}, \tag{2.6}$$

$$\approx \begin{bmatrix} U_{\nu_1} & \\ & U_{\nu_2} \end{bmatrix} \begin{bmatrix} S^r_{\nu_1}(J^r_{\nu_1}, :) - B_{\nu_1,\nu_2} V^*_{\nu_2} R(I_{\nu_2}, :) \\ S^r_{\nu_2}(J^r_{\nu_2}, :) - B_{\nu_2,\nu_1} V^*_{\nu_1} R(I_{\nu_1}, :) \end{bmatrix}. \tag{2.7}$$

We let

$$\phi.R^r \leftarrow V^*_\phi R(I_\phi, :), \tag{2.8}$$

$$\phi.S^r \leftarrow \begin{bmatrix} S^r_{\nu_1}(J^r_{\nu_1}, :) - B_{\nu_1,\nu_2} \nu_2.R \\ S^r_{\nu_2}(J^r_{\nu_2}, :) - B_{\nu_2,\nu_1} \nu_1.R \end{bmatrix}, \tag{2.9}$$

so we can apply $\text{ID}((\phi.S^r)^*)$ in order to compute U_ϕ, since the U_{ν_1} and U_{ν_2} bases have been factored out in (2.7). This can be applied recursively.

3. Adaptive randomized sampling. In most practical problems, we do not know the maximum HSS rank a priori; hence, the number of samples needs to be chosen adaptively.

3.1. Previous algorithms. In [18], we developed the first adaptive sampling scheme, which is illustrated in Algorithm 1, with some helper functions listed in Table 1. The adaptation works as follows. Each HSS node has a state which can be `UNTOUCHED`, `COMPRESSED`, or `PARTIALLY_COMPRESSED`. Each node starts in the `UNTOUCHED` state. The compression proceeds bottom-up from the leaf nodes, with

Algorithm 1. Adaptive HSS compression of $A \in \mathbb{C}^{N\times N}$ using cluster tree $\mathcal{T}$ with relative and absolute tolerances ε_{rel} and ε_{abs}, respectively.

```
1  function H = HSSCompressAdaptive(A, T)
2      d ← d_0;   Δd ← 0;   R ← randn(N, d);   S^r ← AR;   S^c ← A*R
3      foreach τ ∈ T do τ.state ← UNTOUCHED
4      while root(T).state ≠ COMPRESSED and d < d_max do
5          CompressNodeAdaptive(A, R, S^r, S^c, root(T), d, Δd)
6          Δd ← d or constant                    // double or fixed increment
7          R̄ ← randn(N, Δd);   R ← [R  R̄]
8          S^r ← [S^r  AR̄];   S^c ← [S^r  A*R̄]
9          d ← d + Δd
10     end
11     return T
12 function CompressNodeAdaptive(A, R, S^r, S^c, τ, d, Δd)
13     if isleaf(τ) then
14         if τ.state = UNTOUCHED then τ.D ← A(τ.I, τ.I)
15     else
16         ν_1, ν_2 ← children(τ)
17         CompressNodeAdaptive(A, R, S^r, S^c, ν_1, d, Δd)
18         CompressNodeAdaptive(A, R, S^r, S^c, ν_2, d, Δd)
19         if ν_1.state ≠ COMPRESSED or ν_2.state ≠ COMPRESSED then return
20         if τ.state = UNTOUCHED then
21             τ.B_12 ← A(ν_1.I^r, ν_2.I^c);   τ.B_21 ← A(ν_2.I^r, ν_1.I^c)
22     end
23     if isroot(τ) then τ.state ← COMPRESSED;   return
24     if τ.state = UNTOUCHED then
25         ComputeLocalSamples(R, S^r, S^c, τ, 1 : d + Δd)
26     else ComputeLocalSamples(R, S^r, S^c, τ, d + 1 : d + Δd)
27     if τ.state ≠ COMPRESSED then
28         try
29             {(τ.U)*, τ.J^r} ← ID((τ.S^r)*, ε_rel/level(τ), ε_abs/level(τ))
30             {(τ.V)*, τ.J^c} ← ID((τ.S^c)*, ε_rel/level(τ), ε_abs/level(τ))
                                              // tolerances scaled by level
31             τ.state ← COMPRESSED
32             ReduceLocalSamples(R, τ, 1 : d + Δd)
33         catch      // RRQR/ID failed to reach tolerance ε_rel or ε_abs
34             τ.state ← PARTIALLY_COMPRESSED
35             return
36     else ReduceLocalSamples(R, τ, d, d + 1 : d + Δd)
```

d_0 initial number of random vectors. At each (nonroot) node ϕ, coefficients U_ϕ and V_ϕ are computed using ID, which might fail if it is detected that d_0 random vectors are not sufficient to accurately capture the range of the corresponding Hankel block with a prescribed relative or absolute tolerance, ε_{rel} or ε_{abs}, respectively. (How this can be detected will be discussed in detail in the following subsections.) If this happens, node ϕ is marked as `PARTIALLY_COMPRESSED` and the compression algorithm returns to the outer loop. Note that in a parallel run, processing of other HSS nodes

TABLE 1
List of helper functions.

`randn`(m, n)	an $m \times n$ matrix with independently and identically distributed (i.i.d.) $\mathcal{N}(0, 1)$ elements
`rows`(A)/`cols`(A)	number of rows/columns in matrix A
`isleaf`(ϕ)	`true` if ϕ is a leaf node, `false` otherwise
`isroot`(ϕ)	`true` if ϕ is a root node, `false` otherwise
`children`(ϕ)	a list with the children of node ϕ, always zero or two
`levels`$(\mathcal{T})$	number of levels in tree $\mathcal{T}$
`level`(ϕ)	level of node ϕ, starting from 0 at the root
$\{U, J\} \leftarrow$ `ID`$(S, \varepsilon_{\text{rel}}, \varepsilon_{\text{abs}})$	interpolative decomposition with relative and absolute tolerances, returning coefficient matrix and index set
`ComputeLocalSamples` $(R, S^r, S^c, \phi, \iota)$	See (2.9) (updating only columns ι)
`ReduceLocalSamples` (R, ϕ, ι)	See (2.8) (updating only columns ι)

in independent subtrees can still continue. The number of random vectors is increased by Δd and the recursive HSS compression is called again. At this point, there will be at least one node in the HSS tree which is in the `PARTIALLY_COMPRESSED` state. All the descendants of this node are in the `COMPRESSED` state (or compression of that `PARTIALLY_COMPRESSED` node could not have been started), and all ancestors are in the `UNTOUCHED` state. The compression algorithm will again start at the leaf nodes. For the already `COMPRESSED` nodes, the U and V coefficients do not need to be recomputed, but these nodes still need to be visited in order to add Δd extra columns to $\phi.R^r/\phi.R^c$ and $\phi.S^r/\phi.S^c$. For a graphical representation of the adaptive compression procedure see Figure 3.

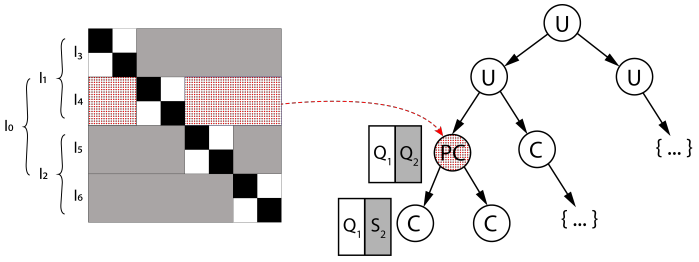

FIG. 3. *Illustration of the tree traversal during adaptive HSS compression. Each node in the tree can be in either* `UNTOUCHED` *(U),* `PARTIALLY_COMPRESSED` *(PC), or* `COMPRESSED` *(C) state. For the meaning of the Q and S matrices, see section* 3.2.

Theorem 4.2 in [21] shows that for a matrix A and its HSS approximation H, with a total of L levels, and with each off-diagonal block compressed with relative tolerance ε_{rel}, it holds that $\|A - H\|_F \leq C(N, L)\varepsilon_{\text{rel}}\|A\|_F$. Inspired by this result, in the HSS compression Algorithm 1, the user-desired absolute and relative tolerances ε_{abs} and ε_{rel} are scaled with the current HSS level for the RRQR factorization. Hence, it is expected that the final HSS compression satisfies $\|A - H\|_F \leq \varepsilon_{\text{abs}}$ or $\|A - H\|_F/\|A\|_F \leq \varepsilon_{\text{rel}}$; we have no proof, but see section 5 for evidence supporting this statement.

For the above adaptive sampling to work reliably, we need to address two main questions:

Algorithm 2. Adaptive computation of Q, an approximate basis for the range of the Hankel block A, using the DOUBLING strategy with an oversampling parameter p.

```
function Q = RS-DOUBLING(A, d_0, p, ε_rel, ε_abs)
    k ← 1;   m ← rows(A);   r ← ∞
    R_1 ← randn(m, d_0 + p)
    S_1 ← A R_1
    while (r > 2^{k-1} d_0) do
        {Q, r} ← RRQR_HMT(S_k, ε_rel, ε_abs)
        Δd ← 2^{k-1} d_0 + p                    // double sample size
        R_{k+1} ← randn(m, Δd)
        S_{k+1} ← [S_k  A R_{k+1}]
        k ← k + 1
    end
    return Q
```

(1) *How to compute Δd to increase the sample size?*
(2) *When to terminate the adaptation?*

In fact, the stopping criteria in question (2) depend on how Δd is computed in question (1). In the following, we will present two different strategies for computing Δd, one is the commonly used DOUBLING method, another is the INCREMENTING method which incurs less communication and hence is preferrable on large parallel machines.

3.1.1. Doubling strategy with oversampling. Algorithm 2 computes an (orthogonal) approximate basis for the range of an $m \times n$ matrix A, with an oversampling parameter p, up to a given relative or absolute tolerance ε_{rel} or ε_{abs}. Initially, a random projection of the input matrix A with a tall and skinny random matrix R, with $d_0 + p$ columns, is computed as $S = AR$. Let Q be an orthonormal basis for S. From [12, Theorem 1.1], we have

$$\mathbb{E}\|(I - QQ^*)A\|_2 \leq \left[1 + 4\frac{\sqrt{d_0 + p}}{p - 1} \cdot \sqrt{\min\{m, n\}}\right] \sigma_{d_0+1}\,, \tag{3.1}$$

where σ_{d_0+1} is the $(d_0 + 1)$th singular value of A. The factor in brackets in (3.1), the deviation from the optimal error σ_{d_0+1}, decreases with increasing oversampling p. In order to guarantee a relative or absolute error bound on $\|A - QQ^*A\|$, we apply a modified RRQR factorization (a column pivoted QR) to S, which includes this factor. The modified RRQR (`RRQR_HMT` in Algorithm 2, named after the authors of [12]) returns a rank $r = k$ as soon as

$$R_{k,k} \leq \frac{\varepsilon_{\text{abs}}}{1 + 4\frac{\sqrt{d_0+p}}{d_0+p-k-1} \cdot \sqrt{\min\{m, n\}}} \quad \text{or} \quad \frac{R_{k,k}}{R_{1,1}} \leq \frac{\varepsilon_{\text{rel}}}{1 + 4\frac{\sqrt{d_0+p}}{d_0+p-k-1} \cdot \sqrt{\min\{m, n\}}}\,. \tag{3.2}$$

Here R refers to the upper triangular QR factor, not to be confused with the random matrix R. This modification is to take into account that at step k of the `RRQR_HMT` algorithm, the amount of oversampling is $d_0 + p - k - 1$. If the rank r returned by the modified RRQR is $r < d_0$, i.e., at least p oversampling vectors are used;

then the orthonormal basis Q returned by RRQR is accepted. However, if RRQR does not achieve the required tolerance, or if it achieves the required tolerance but $d_0 \leq r \leq d_0 + p$, then the resulting Q basis is rejected. The number of random vectors (excluding the p oversampling columns) is doubled and the random projection with these new vectors is added to S. A RRQR factorization is applied to the entire new random projection matrix S. Since in this scenario RRQR is always redone from scratch, the number of adaptation steps should be minimized in order to minimize the work in RRQR. Doubling the number of random vectors in each step ensures one only needs $\mathcal{O}(\log(r))$ adaptation steps,[2] but in the worst-case scenario, i.e., when $r = d_0 - 1$, the amount of oversampling is $r + p - 2$.

This is the approach previously used in [18, 7, 6], although in those references, the scaling factor from (3.1) was not included. Including this scaling factor slightly increases the rank but yields a more accurate compression. When used in the larger HSS compression algorithm (Algorithm 1), the matrix sizes m and n from (3.2) are the sizes of the original Hankel block.

Remark. Algorithms 2 (and 3 later) corresponds to the successive (rank-revealing) QR factorizations at the `PARTIALLY_COMPRESSED` node and is part of the bigger Algorithm 1. The successive random sampling steps are actually performed in Algorithm 1 but are included here for convenience. Here, Algorithm 2 (and 3 later) is presented as only returning the orthonormal basis of the range of A, but when used in the overall HSS compression (Algorithm 1), the upper triangular factor from the final RRQR factorization is also required.

There are several drawbacks with the DOUBLING method:

1) It often leads to unnecessary amount of oversampling, and hence the cost associated with it;
2) The rank-revealing factorization has to be performed from scratch in each step, which leads to additional computational cost and communication. In particular, the communication involved in column pivoting in the repeated RRQR factorizations can be a serious bottleneck in a distributed memory code.

This leads to the new design of the INCREMENTING approach described below, including the new theory of a stochastic matrix norm estimation, which is the basis of both absolute and relative stopping criteria.

3.2. Blocked incrementing strategy. To mitigate the problems with the DOUBLING strategy, we developed a new strategy that builds Q incrementally; hence, we call this INCREMENTING strategy. Before presenting our ultimate algorithm, we first review the earlier work along this line.

In [12], Algorithm 4.2 ("Adaptive Randomized Range Finder") is presented for the adaptive computation of an orthonormal basis Q for the range of a matrix A, up to an absolute tolerance ε_{abs}. This algorithm computes an approximate orthonormal basis for the range of A one vector at a time and then determines how well this basis approximates the range. Adding one vector at a time to the basis amounts to performing a sequence of multiple matrix-vector products (BLAS-2), which are memory-bound and less efficient on modern hardware. In [14], Algorithm 2 ("Parallel Adaptive Randomized Orthogonalization") uses a similar approach, but implements a blocked version, relying on BLAS-3 operations which can achieve much higher performance. This algorithm also relies only on an absolute tolerance.

[2] Logarithm is in base 2 throughout the paper.

Comparing to the previous algorithms, our method also implements a blocked version. Moreover, we use both an absolute and a relative stopping criterion. The stopping criterion is based on a stochastic error bound we develop in section 3.4 and a block Gram–Schmidt orthogonalization [19, 17].

Let Q be the current orthonormal approximate basis for the range of an $m \times n$ matrix A. Given a random Gaussian matrix $R \in \mathbb{R}^{n\times d}$, compute first the random samples $S = AR$ followed by the projection $\widehat{S} = (I - QQ^*)\, S$. This last operation is one step of block Gram–Schmidt. In practice, to ensure orthogonality under roundoff errors, we apply this step twice as $\widehat{S} = (I - QQ^*)^2\, S$ [19]. Thus, $\widehat{S}$ contains information about the range of A that is not included in Q. A small $||\widehat{S}||_F$ means that either Q was already a good basis for the range of A, or S was in the range of Q, which is unlikely. If $||\widehat{S}||_F$ is not small enough, $\widehat{S}$ is used to expand Q. Since $\widehat{S}$ is already orthogonal to Q, we only need to do internal orthogonalization of $\widehat{S}$ using a QR factorization, $[\overline{Q}, R] = \mathrm{QR}(\widehat{S})$, and then augment $Q \leftarrow [Q\ \overline{Q}]$.

3.3. Need for new stopping criteria for incrementing adaptation. The main issue is to determine whether the sample matrices S^r and S^c capture well the column space of the corresponding Hankel blocks. As we shall see later, this decision relies on a good estimate of the matrix norm and hence the error estimation of the residual matrix (see (3.26)), which in turn is critical in devising the stopping criteria of adaptation. The goal is to ensure sufficient samples are used to guarantee the desired accuracy while not performing too much oversampling, as this degrades performance. We first review a well-known result from the literature and explain how it falls short when used as a stopping criterion. Then in section 3.4 we propose our new approach.

Recall the following result from [12].

LEMMA 3.1 (Lemma 4.1 in [12]). *Let B be a real $m \times n$ matrix. Fix a positive integer p and a real number $\alpha > 1$. Draw an independent family $\{\omega^i : i = 1, \ldots, p\}$ of standard Gaussian vectors. Then*

$$||B||_2 \le \alpha\sqrt{\frac{2}{\pi}} \max_{i=1,\ldots,p} \left|\left|B\omega^i\right|\right|_2 \tag{3.3}$$

except with probability α^{-p}.

The value of α represents a trade-off—larger α makes the bound less strict and more probable to achieve. For $\alpha = 10$ and $p = 10$, the bound in (3.3) holds with very high probability $1 - 10^{-10}$. In [14], relation (3.3) is used as a stopping criterion in an adaptive HSS construction algorithm, with $B = (I - QQ^*)F$, as

$$||(I - QQ^*)\, F||_2 \le \alpha\sqrt{\frac{2}{\pi}} \max_{i=1,\ldots,p} \left|\left|(I - QQ^*)\, F\omega^i\right|\right|_2 \le \varepsilon, \tag{3.4}$$

where F is a Hankel block of the input matrix A, Q is a matrix with orthonormal columns which approximates the range of F, and ε is the tolerance. The column dimension of Q is increased until the second inequality in (3.4) is satisfied, which guarantees that $\|(I - QQ^*)F\|_2 \le \varepsilon$.

There are several drawbacks with this criterion.

1) If different F blocks vary greatly in size, different blocks will be compressed to different *relative* tolerances, even though they are compressed to the same *absolute* tolerance. The error bound in Lemma 3.1 is not conducive to relative error bounds because it only gives an upper bound on the error and not necessarily an accurate approximation of it; see section 3.6 for examples showing this. Thus, it is not possible

to use this bound to compute a relative stopping criterion. In section 3.4, we derive an accurate relative bound based on a stochastic estimation of the matrix norm.
2) The error estimate in (3.4) assumes that we have access to the entire Hankel block F and the associated random samples. However, in the matrix-free HSS construction as described in section 2, only the local random samples $\phi.S^r$ and $\phi.S^c$ are available. In (2.5)–(2.7), the random sample matrix S^r_ϕ of the Hankel block corresponding to HSS node ϕ, i.e., $A(I_\phi, I_A \setminus I_\phi)$, is expressed in terms of the coefficients U_{ν_1} and U_{ν_2} of its children; see (2.7). The random sample S^r_ϕ of the Hankel block is constructed by subtracting the random sample $D_\phi R(I_\phi, :)$ of the diagonal block (see (2.5)), using the already compressed D_ϕ (D_ϕ is exact only at the leaves). Hence, this introduces an approximation error.
3) Another issue arises due to the HSS nested basis property; see (2.2). In order to maintain the HSS nested basis property, the Hankel block is not compressed directly, but only its coefficient in the $[U_{\nu_1}\ 0\,;\ 0\ U_{\nu_2}]$ basis is compressed. This coefficient is defined in (2.9) as $\phi.S^r$. The difficulty arises from the use of the interpolative decomposition to compress the off-diagonal blocks (see (2.4)), since it leads to nonorthonormal U_ϕ. Hence, the bound from Lemma 3.1 should really be applied to

$$\left|\left|\begin{bmatrix} U_{\nu_1} & 0 \\ 0 & U_{\nu_2} \end{bmatrix} \left(I - Q_\phi Q_\phi^*\right) \phi.S^r \right|\right|_2 \tag{3.5}$$

to derive a correct stopping criterion for adaptive HSS compression. As long as the U_ϕ are orthonormal, there is no problem, since $||\cdot||_2$ is unitarily invariant.

In practice, a (strong) RRQR factorization [8] will cause the U matrices to be well-conditioned, and its elements to be bounded [12], so the absolute tolerance should not be affected much. It seems plausible for the nonorthonormal factors to essentially cancel out when using the relative stopping criterion, making it a more reliable estimate. The hierarchical nature of the U and V matrices makes it possible to efficiently compute the products in (3.5) should this be desired.

3.4. Mathematical theory for the new error bound. Ideally, we would like to bound our errors with respect to A: $\|(I - QQ^*)A\|_F/\|A\|_F$ and $\|(I - QQ^*)A\|_F$, where A is a Hankel block of the original input matrix. However, since A might not be readily available (or expensive to compute), we instead use the random samples S. We now establish a stochastic F-norm relationship between $\|A\|_F$ and $\|S\|_F$ and show that this estimate is accurate to high probability.

Let $A \in \mathbb{R}^{m\times n}$ and $x \in \mathbb{R}^n$ with $x_i \sim \mathcal{N}(0,1)$. Furthermore, let

$$A = U\Sigma V^* = \begin{bmatrix} U_1 & U_2 \end{bmatrix} \begin{bmatrix} \Sigma_r & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} V_1^* \\ V_2^* \end{bmatrix} \tag{3.6}$$

be the singular value decomposition (SVD) of A, and $\xi = V^*x$. Since x is a Gaussian random vector, so is ξ. By the rotational invariance of $||\cdot||_2$, it follows that

$$||Ax||_2^2 = ||\Sigma\xi||_2^2 = \sigma_1^2\xi_1^2 + \cdots + \sigma_r^2\xi_r^2\,, \tag{3.7}$$

with $\sigma_1 \geq \cdots \geq \sigma_r > 0$ the positive singular values. Hence,

$$\mathbb{E}\left(||Ax||_2^2\right) = \sigma_1^2 + \cdots + \sigma_r^2 = ||A||_F^2\ . \tag{3.8}$$

From what we just showed, if $R \in \mathbb{R}^{n\times d}$ with $R_{jk} \sim \mathcal{N}(0,1)$, it is clear

$$||AR||_F^2 = ||AR_1||_2^2 + \cdots + ||AR_d||_2^2 \tag{3.9}$$

with R_i a Gaussian random vector, so that

$$\frac{1}{\sqrt{d}}\sqrt{\mathbb{E}\left(||AR||_F^2\right)} = ||A||_F \,. \tag{3.10}$$

Theorem 3.3 shows that particular realizations will, with high probability, be close to the expected value. In particular, we can approximate the difference between our approximation and the actual matrix subblock, allowing us to compute both the absolute and relative error in contrast to Lemma 3.1. We call (3.10) the Gaussian error bound (GEB) because we use Gaussian random vectors to accurately estimate the matrix F-norm, allowing us to approximate $\|(I - QQ^*)A\|_F$. Future work investigating a random variable whose expectation value is $||A||_2$ (or some power) would be beneficial. At this point, we settle for using the Frobenius norm because we can accurately approximate it.

In order to facilitate the analysis, we define the random variable

$$X \sim \sigma_1^2\xi_1^2 + \cdots + \sigma_r^2\xi_r^2\,, \tag{3.11}$$

where $\xi_i \sim \mathcal{N}(0,1)$, and thus $\mathbb{E}(X) = ||A||_F^2$. From (3.7), we see that $||Ax||_2^2$ and X have the same probability distribution, so we focus on understanding X. Consider

$$\overline{X}_d \sim \frac{1}{d}\left(X_1 + \cdots + X_d\right), \tag{3.12}$$

where X_i are independent realizations of X. It is easy to see $\mathbb{E}\left(\overline{X}_d\right) = ||A||_F^2$. From [2], we have the following theorem.

THEOREM 3.2 (Chernoff's inequality). *Given a random variable X, we have*

$$\mathbb{P}\left[X \geq a\right] \leq \min_{t>0} e^{-ta}\,\mathbb{E}\left(e^{tX}\right)\,. \tag{3.13}$$

Here, $\mathbb{E}\left(e^{tX}\right)$ is the moment generating function of a random variable X. A slight modification of Theorem 3.2 gives

$$\mathbb{P}\left[X \leq a\right] \leq \min_{t>0} e^{ta}\,\mathbb{E}\left(e^{-tX}\right)\,. \tag{3.14}$$

We can now prove the probability tails of X and $\overline{X}_d$ decay exponentially away from $||A||_F^2$. Taken together, we have the following theorem.

THEOREM 3.3 (probabilistic error bounds). *Given $\overline{X}_d$ as defined in* (3.12) *with $r \geq 2$, the following bounds on the tail probabilities hold:*

$$\begin{aligned}
&\mathbb{P}\left[\overline{X}_d \geq ||A||_F^2\,\mu\right] \leq \exp\left(-\frac{d\mu}{2}\right)||A||_F^{dr}\prod_{k=1}^{r}\left(A_k'\right)^{-d}, \qquad \mu > 1,\\
&\mathbb{P}\left[\overline{X}_d \leq ||A||_F^2\,\mu\right] \leq \exp\left(\frac{d\mu}{2}\right)||A||_F^{dr}\prod_{k=1}^{r}\left(A_k''\right)^{-d}, \qquad \mu \in [0,1).
\end{aligned} \tag{3.15}$$

Here,

$$\begin{aligned}
||A||_F^2 &= \sigma_1^2 + \cdots + \sigma_r^2,\\
\left(A_k'\right)^2 &= ||A||_F^2 - \sigma_k^2,\\
\left(A_k''\right)^2 &= ||A||_F^2 + \sigma_k^2.
\end{aligned} \tag{3.16}$$

We know $\mathbb{E}\left(\overline{X}_d\right) = ||A||_F^2$*, so* μ *controls multiplicative deviation above or below the expectation value. Furthermore, these tail probabilities decay exponentially away from the expectation value under mild conditions.*

Proof. From its definition, X is a linear combination of chi-squared distributions, so by using properties of the moment generating function we see

$$M_{\overline{X}_d}(t) = \prod_{k=1}^{r} \left(1 - \frac{2\sigma_k^2}{d} t\right)^{-\frac{d}{2}}. \tag{3.17}$$

By setting $\bar{t} = \frac{d}{2||A||_F^2}$ and $a = ||A||_F^2 \mu$ with $\mu > 1$, Theorem 3.2 implies

$$\mathbb{P}\left[\overline{X}_d \geq ||A||_F^2 \mu\right] \leq \exp\left(-\frac{d\mu}{2}\right) ||A||_F^{dr} \prod_{k=1}^{r} \left(A_k'\right)^{-d}, \qquad \mu > 1, \tag{3.18}$$

where A_k' is defined in (3.16). In a similar manner, we can obtain the upper bound

$$\mathbb{P}\left[\overline{X}_d \leq ||A||_F^2 \mu\right] \leq \exp\left(\frac{d\mu}{2}\right) ||A||_F^{dr} \prod_{k=1}^{r} \left(A_k''\right)^{-d}, \qquad \mu \in [0, 1). \tag{3.19}$$

It is clear that these upper bounds are not optimal, and more rigorous analysis would produce tighter bounds, but we will not investigate this further. The bounds we just obtained are sufficient for our purposes.

We now determine when $\overline{X}_d$ has exponentially decaying tail probabilities. For $\mu > 1$, we have

$$\begin{aligned} \mathbb{P}\left[\overline{X}_d \geq ||A||_F^2 \mu\right] &\leq \exp\left(-\frac{d\mu}{2}\right) ||A||_F^{dr} \prod_{k=1}^{r} \left(A_k'\right)^{-d} \\ &= \exp\left[\frac{d}{2}\left\{(\nu_1 + \cdots + \nu_r) - \mu\right\}\right], \end{aligned} \tag{3.20}$$

where

$$\nu_k = \ln\left[\frac{1}{1 - \frac{\sigma_k^2}{||A||_F^2}}\right]. \tag{3.21}$$

To have exponential decay in probability, we require

$$\nu_1 + \cdots + \nu_r < \mu. \tag{3.22}$$

Because $-\ln x$ is convex on $(0, \infty)$, we see that for $\alpha \in (0, 1)$, we have

$$\ln\left(\frac{1}{1-x}\right) \leq \frac{x}{\alpha} \ln\left(\frac{1}{1-\alpha}\right), \qquad x \in [0, \alpha]. \tag{3.23}$$

Now, we know $\sigma_k \leq ||A||_2$, so we set $\alpha = \frac{||A||_2^2}{||A||_F^2}$. Combining this with the fact that $\ln(1 + x) \leq x$, we see

$$\begin{aligned} \nu_1 + \cdots + \nu_r &\leq \frac{\sigma_1^2}{||A||_F^2} \frac{1}{\alpha} \ln\left(\frac{1}{1-\alpha}\right) + \cdots + \frac{\sigma_r^2}{||A||_F^2} \frac{1}{\alpha} \ln\left(\frac{1}{1-\alpha}\right) \\ &\leq \frac{1}{1-\alpha}. \end{aligned} \tag{3.24}$$

Thus, we require

$$\mu > 1 + \frac{||A||_2^2}{||A||_F^2 - ||A||_2^2} \tag{3.25}$$

in order to have exponentially decaying tail probabilities in d.

The case when $\mu \in [0, 1)$ is similar, and we obtain exponentially decaying tail probabilities in d when $\mu < \ln 2$. ∎

3.5. New adaptation algorithm with new stopping criteria. Algorithm 3 presents a new approach to adaptive rank determination. In contrast to Algorithm 2, in Algorithm 3, the rank-revealing factorization is only performed when the number of sample vectors is guaranteed to be sufficient. This decision criterion is based on (3.10) and uses an adaptive blocked implementation.

We have been unable to find an explicit reference to a *relative* stopping criterion. From [23, 24] we know that using a relative tolerance for the compression of off-diagonal blocks leads to a relative error in Frobenius norm for the final computed HSS approximation. Since this is frequently desired, we use both relative and absolute stopping criteria explicitly (see line 7 in Algorithm 3). Relative tolerances are especially useful if the magnitude of different matrix subblocks differ significantly.

Algorithm 3. Adaptive computation of Q, an approximate basis for the range of the Hankel block A, using the INCREMENTING strategy.

```
1  function Q = RS-INCREMENTING(A, d_0, Δd, ε_rel, ε_abs)
2      k ← 1;    m ← rows(A)
3      R_1 ← randn(m, d_0)
4      S_1 ← A R_1
5      {Q, R̄_1} ← QR(S_1)
6      while true do
7          k ← k + 1
8          R_k ← randn(m, Δd)
9          S_k ← A R_k                                   // new samples
10         Ŝ_k ← (I − QQ*)^2 S_k              // iterated block Gram-Schmidt
11         if (||Ŝ_k||_F/||S_k||_F < ε_rel or ||Ŝ_k||_F/√Δd < ε_abs) then break
12         {Q_k, R̄_k} ← QR(Ŝ_k)
13         if min(diag(|R̄_k|)) < ε_abs or min(diag(|R̄_k|)) < ε_rel|(R̄_1)_11| then
           break
14         Q ← [Q Q_k]
15     end
16     {Q, r} ← RRQR([S_1 ... S_k], ε_rel, ε_abs)
17     return Q
```

Additional difficulties arise in the blocked version that do not appear in the single vector case. Adding single vectors to the basis allows one to always ensure that each basis vector adds new information. In the blocked case however, $\widehat{S}$ can become (numerically) rank-deficient. Therefore, in Algorithm 3, line 13, we look at the diagonal elements of $\overline{R}$ in order to determine if $\widehat{S}$ is rank-deficient. If the diagonal elements of $\overline{R}$ are less than a specified tolerance (relative or absolute), then we have complete knowledge of the range space (up to the specified tolerance), and we can compress

the HSS node. Ideally, if we continue sampling the range at this point, the error will only decrease and while new information is gained, enough information is already known to compute the ID. For stringent compression tolerances (such as $\varepsilon_{\text{rel}} \approx 10^{-14}$ in double precision), continuing to compute more random samples would potentially lead to forming Q which is *not* orthonormal so that the stopping criterion is never reached until the maximum allowable samples are computed. When this happens, Q is no longer an approximation of the range of the Hankel block. This can also happen for small ε_{abs} tolerances as well; see section 5.

In summary, the new stopping criteria is based on the four conditions below in (3.26), i.e., when $||\widehat{S}||_F$ is small or $\overline{R}$ is rank-deficient. As soon as any of them are satisfied, we will compute the interpolative decomposition of the random samples at that node.

$$\frac{||\widehat{S}||_F}{||S||_F} < \varepsilon_{\text{rel}}\,, \quad \frac{1}{\sqrt{d}}||\widehat{S}||_F < \varepsilon_{\text{abs}}\,, \quad \frac{\texttt{min}_i \left|\overline{R}_{ii}\right|}{\rho} < \varepsilon_{\text{rel}}\,, \quad \texttt{min}_i \left|\overline{R}_{ii}\right| < \varepsilon_{\text{abs}}\,. \tag{3.26}$$

Here, ρ estimates the 2-norm of the random samples in order to quantify when the $\overline{R}$ matrices are determined to be rank-deficient. Because the actual computation of this would be expensive, a few possible choices are $|(\overline{R_1})_{11}|$, $\max_j |(\overline{R_1})_{jj}|$, $\max_{k,j} |(\overline{R_k})_{jj}|$, or an approximation of $||A||_F$. All of these values give an estimate of how large the diagonal values of $\overline{R}$ are, although the first two give reasonable estimates with minimizing communication by only needing to check the diagonal elements of $\overline{R}$ once.

3.6. Comparison between GEB and Halko–Martinsson–Tropp (HMT). In this section we present a small set of examples comparing HMT error bound and GEB applied to low-rank QB approximation. If Q is approximate orthonormal basis for A, then a QB factorization is $A \approx Q(Q^*A) = QB$ [15]. We will compare HMT and GEB when attempting to solve the low-rank fixed-precision approximation of A with a QB factorization using block randomized sampling with one pivoted QR factorization at the end to accurately compute an orthonormal basis for the range space; this happens during the compression of each matrix subblock. We are not the first to recognize the shortcomings of the HMT error bound; [30] presents an algorithm for solving the fixed-precision low-rank approximation problem. One downside to that method is that a good approximation of $||A||_F$ must be known a priori and the method only allows for approximation down to relative error $O(\sqrt{\varepsilon_{\text{mach}}})$. From the examples presented here and in section 5, our method does not appear to have this limitation.

The test matrices we use in this section are similar to those in [30]. The matrices all have the SVD $A = U\Sigma V^*$, where U and V are matrices with orthonormal columns, and Σ will vary depending on the matrix. The tests are run using double precision with $\varepsilon_{\text{mach}} \approx 10^{-16}$. All of these matrices will have rank $r = 100$ with $A \in \mathbb{R}^{1,000\times 1,000}$. We use the following nonzero singular values:

- Matrix 1: $\sigma_k = k^{-2}$. This matrix is chosen to have singular values that decay slowly.
- Matrix 2: $\sigma_k = 2^{-53(k-1)/100}$. This matrix is chosen to have singular values which decay quickly.
- Matrix 3: $\sigma_k = 100\varepsilon_{\text{mach}} + (1 + 2^{k-26})^{-1}$. This matrix is chosen to have singular values with S-decay: they initially hover around 1, quickly decay, and then hover around $100\varepsilon_{\text{mach}} \approx 10^{-14}$.

All of these matrices are chosen with $\sigma_1 \approx 1$. A plot of the singular values are shown in Figure 4.

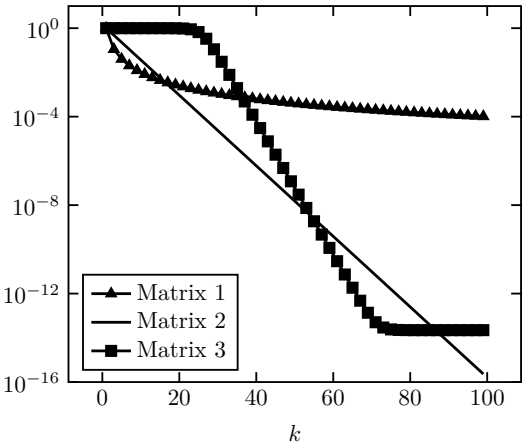

FIG. 4. *Here is a plot of the singular values of Matrices* 1–3 *in section* 3.6*: Matrix* 1*, slow decay; Matrix* 2*, fast decay; Matrix* 3*, S-shaped decay.*

We will perform two sets of tests. In the first, we will look at how well we can approximate or bound the 2-norm or Frobenius norm of the three matrices. In the second, we will attempt compute a QB factorization by building up an orthonormal basis for the range in block form until the stopping criterion determines enough samples have been computed. At this point, a RRQR factorization (QRCP) will be used to compute the QB factorization.

From Lemma 3.1, we can vary α and p. We let $\alpha \in \{2, 5, 10\}$ and chose p smallest so that $\alpha^{-p} \leq 10^{-\ell}$. To be sure to get a range of probability failures, we looked at $\alpha^{-p} \in \{10^{-9}, 10^{-12}, 10^{-15}\}$. From the results in Table 2, we see in *every* case the average value for the 2-norm upper bound is always at least 3x larger than the actual value ($||A||_2 \approx 1$), and frequently is 10x larger. If one is wanting a good *estimate* of the error, and not an *overestimate*, this will not be useful. We contrast this with the accurate results seen in Table 3 when we attempt to compute the F-norm of Matrices 1–3. Using eight random samples gives a close F-norm estimate, although using more samples results in a better approximation. In each case, we computed the average over 10,000 trials to compute the mean and standard deviation.

TABLE 2

Upper bound of $||A||_2$ *for Matrices* 1–3 *based on Lemma* 3.1 *(HMT) for different failure probabilities (columns) and* α *(rows).* p *is the smallest integer for which* $\alpha^{-p} \leq 10^{-\ell}$*. We performed* 10,000 *trials and computed the mean (M) and standard deviation (S). The correct values are* $||A||_2 \approx 1$*.*

		1E-9			**1E-12**			**1E-15**		
		p	M	S	p	M	S	p	M	S
Matrix 1	**2**	30	3.73	0.71	40	3.90	0.70	50	4.04	0.68
	5	13	8.02	1.96	18	8.55	1.89	22	8.87	1.85
	10	9	14.9	4.1	12	15.8	3.9	15	16.5	3.9
Matrix 2	**2**	30	4.11	0.66	40	4.26	0.64	50	4.38	0.63
	5	13	9.12	1.77	18	9.58	1.72	22	9.85	1.69
	10	9	17.2	3.6	12	18.0	3.6	15	18.6	3.5
Matrix 3	**2**	30	10.07	0.60	40	10.22	0.58	50	10.33	0.56
	5	13	24.1	1.7	18	24.5	1.6	22	24.8	1.6
	10	9	47.1	3.5	12	47.9	3.4	15	48.6	3.3

For the second test, we performed the adaptive QB compression algorithm 1,000 times and averaged the final 2-norm error of the approximation and the total samples used. We built up the approximation in blocks of 16 random samples and stopped

TABLE 3

Estimated $||A||_F$ for Matrices 1–3 using Eq. (3.10) (GEB). We performed 10,000 trials and computed the mean (M) and standard deviation (S) for samples $d \in \{8, 16, 32, 64\}$. The true F-norm values are listed.

		8		**16**		**32**		**64**	
	True	M	S	M	S	M	S	M	S
Matrix 1	1.040	1.012	0.233	1.029	0.167	1.036	0.119	1.038	0.085
Matrix 2	1.386	1.367	0.200	1.378	0.142	1.383	0.101	1.384	0.073
Matrix 3	4.905	4.903	0.244	4.901	0.172	4.903	0.122	4.904	0.087

based on the HMT stopping criterion (we chose $\alpha = 5$ and used 13 samples, so the failure probability was less than 10^{-9}) or (3.26) (we set $\varepsilon_{\text{abs}} = \varepsilon_{\text{rel}}$ because $||A||_2 \approx 1$). The results are shown in Table 4 for Matrices 1–3; the different matrices were compressed to various tolerances. In *every* instance, more random samples are used in HMT than the new stopping criterion, while the 2-norm error is below the desired tolerance in each case (both HMT and GEB).

TABLE 4

QB approximation results for Matrices 1–3. For each absolute error tolerance, we averaged 1,000 trials to determine the average error (Err) and average samples used (Samp) in order to compute a QB approximation once we used either the HMT stopping criterion or new stopping criterion to determine when we had approximated the range. Random samples were computed in blocks of 16. In the case when we used 200+ samples, we were not able to meet the HMT stopping criterion and used the maximum of 200 random samples.

	1E-1		**1E-2**		**1E-3**		**1E-4**	
Matrix 1	Err	Samp	Err	Samp	Err	Samp	Err	Samp
HMT	4E-2	42	3E-3	92	3E-4	128	2E-15	128
GEB	4E-2	32	5E-3	32	4E-4	80	2E-15	112
	1E-3		**1E-6**		**1E-9**		**1E-12**	
Matrix 2	Err	Samp	Err	Samp	Err	Samp	Err	Samp
HMT	4E-4	48	4E-7	76	4E-10	96	4E-13	112
GEB	6E-4	32	7E-7	48	1E-9	65	7E-13	94
	1E-3		**1E-6**		**1E-9**		**1E-12**	
Matrix 3	Err	Samp	Err	Samp	Err	Samp	Err	Samp
HMT	4E-4	64	3E-7	80	4E-10	80	3E-13	200+
GEB	5E-4	48	9E-7	59	6E-10	64	6E-13	80

While these two tests were performed on only a few matrices, it lends support to the fact that the new stopping criterion is useful for adaptive compression algorithms. Numerical results for HSS compression can be found in section 5, where we also see favorable results.

3.7. Flop counts: DOUBLING versus INCREMENTING. First, note that the full QR factorization $Q = \text{QR}(A)$ for $A \in \mathbb{R}^{m \times n}$ (with $m \gg n$) performs $2mn^2$ floating point operations (flops).[3] Assuming the numerical rank of A is r, then the RRQR factorization $Q = \text{RRQR}(A)$ requires $2mnr$ flops. Given $S \in \mathbb{R}^{m \times r}$ and $X \in \mathbb{R}^{m \times d}$, the projection $S \leftarrow (I - XX^*)S$ requires $4mdr$ flops (ignoring a lower order mr term), and twice that for the iterated ($2\times$) block Gram–Schmidt step.

In the DOUBLING strategy, Algorithm 2, at step k, $2^{k-1}d_0 + p$ random vectors have already been used, where p is a small oversampling parameter (e.g., $p = 10$). Then, $2^{k-1}d_0$ new random samples are added to the sample matrix S_k, leading to

[3]The flop count and communication count throughout the paper are all in big-$\mathcal{O}$ sense. For brevity, we omit the $\mathcal{O}$ symbol in those expressions.

$2^k d_0 + p$ columns for S_k. Except for the final step, the operation $Q_k \leftarrow \mathrm{RRQR}(S_k)$ costs $\sim 2m(2^k d_0)^2 = 2m4^k d_0^2$ flops. The total number of steps needed to reach the final rank r is $N \sim \log(r/d_0)$. Summing the costs of N steps: $2md_0^2 \sum_{k=1}^{N} 4^k$, we obtain the total flop count $\sim \frac{8}{3}mr^2$. In the final step, the RRQR terminates early when the tolerance is satisfied at rank r, but the number of sample vectors can be up to $\sim 2r$ in the worst case, with the RRQR cost $\sim 2m(2r)r = 4mr^2$. Adding together, the total flop count is $\sim \frac{20}{3}mr^2$.

The INCREMENTING strategy, Algorithm 3, starts with d_0 random samples and adds Δd new random vectors at each step. At step k we have the sample matrix $S = [S_1 \ \cdots \ S_k]$, and the orthogonal matrix $Q = [Q_1 \ \cdots \ Q_{k-1}]$. We first compute the orthogonal projection $\hat{S}_k \leftarrow (I - QQ^*)^2 S_k$, which costs $8m(k-1)\Delta d^2$ flops, followed by $Q_k \leftarrow \mathrm{QR}(\hat{S}_k)$, which costs $2m\Delta d^2$, and then append Q_k to Q. The total number of steps needed to reach the final rank r is $N \sim (r - d_0)/\Delta d \sim r/\Delta d$. Summing the cost of N steps: $8m\Delta d^2 \sum_{k=1}^{N}(k-1)$, gives the overall cost $\sim 4mr^2$. The additional cost of the final step RRQR is $\sim 2mr^2$. Adding together, the total flop count is $\sim 6mr^2$.

From the above analysis, we see that the DOUBLING scheme may require more flops mainly due to the potentially large sampling size in the final step. DOUBLING also involves a larger amount of data movement due to the column pivoting needed in each step of RRQR. This manifests itself in the communication cost of the parallel algorithm, which will be analyzed in section 4.3.

4. Parallel algorithm.

The parallel algorithm uses the same parallelization framework as described in [18, section 3]. The data partitioning and layout is based on the HSS tree, following a proportional mapping of subtrees to subsets of processes in a top-down traversal. The HSS tree can be specified by the user. The tree should be binary but can be imbalanced and does not need to be complete. For HSS nodes that are mapped to multiple processors, the matrices stored at those nodes are distributed in two dimensional block-cyclic (ScaLAPACK style) layout.

4.1. Partially matrix-free interface.

The randomized HSS compression algorithm is so-called partially matrix-free. This means it does not require every single element of the input matrix A. What is required is a routine to perform the random sampling $S = AR$, as well as a way to extract subblocks from A. Recall from section 2 that at the leafs, $D_\phi = A(I_\phi, I_\phi)$. Furthermore, due to the use of interpolative decompositions, $B_{\nu_1,\nu_2} = A(J^r_{\nu_1}, J^c_{\nu_2})$, is a subblock of A.

If the input matrix A is given as an explicit dense matrix, the random sampling $S = AR$ is performed in parallel using the parallel basic linear algebra subprograms (PBLAS) routine `PDGEMM` with a two dimensional block-cyclic data layout for A, S, and R. In this case, the input matrix A is also redistributed—with a single collective message passing interface (MPI) call—from the two dimensional block-cyclic layout to a layout corresponding to the HSS tree, such that extraction of subblocks for D_ϕ and B_{ν_1,ν_2} does not require communication between otherwise independent HSS subtrees.

Instead of forming an explicit dense matrix A, the user can also specify multiplication and element extraction routines. The multiplication routine computes, for a given random matrix R, the random sample matrices $S^r = AR$ and $S^c = A^*R$. This is the more interesting use case, since for certain classes of structured matrices a fast multiplication algorithm is available; consider, for instance, sparse matrices, low-rank and hierarchical matrices, combinations of sparse and low-rank matrices, or operators that can be applied using the fast Fourier transform or similar techniques.

The element extraction routine should be able to return matrix subblocks $A(I, J)$, defined by row and column index sets I and J, respectively. Depending on how the user data is distributed, computing matrix elements might involve communication between all processes. In this case, the HSS compression traverses the tree level by level (from the leafs to the root), with synchronization at each level, and element extraction for all blocks D_ϕ and B_{ν_1,ν_2} on the same level, performed simultaneously in order to aggregate communication messages and minimize communication latency. If no communication is required for element extraction, then independent subtrees can be compressed concurrently.

4.2. Parallel restart. During factorization, nodes can be in either `UNTOUCHED` (`U`), `PARTIALLY_COMPRESSED` (`PC`), or `COMPRESSED` (`C`) state. A node cannot start compression until both its children are in the `C` state. If during HSS compression, a process encounters an internal HSS node with children that are not both in the `C` state, this process stops the HSS tree traversal. Independent subtrees proceed with compression if they can be compressed successfully with the current number of random samples. This can lead to load imbalances if the HSS tree, or the off-diagonal block ranks, are imbalanced. Eventually, all processes synchronize to perform the random sampling in parallel. Hence there is some overhead associated with restarting the HSS compression algorithm to add more random samples. In addition, random sampling is more efficient, in terms of floating point throughput, when performed with more random vectors at once.

4.3. Communication cost in parallel adaptation. In [18], we analyzed the communication cost of the entire parallel HSS algorithm, assuming no adaptivity. In this section, we will focus only on the cost of adaptivity, using either DOUBLING or INCREMENTING strategy.

Consider the current node of the HSS tree that requires adaptation (called "PC" node). Assume the final rank is r, the row dimension of the sample matrix is m, and P processes work on this node in parallel. We use the pair [#messages, #words] to denote the *communication cost* which counts the number of messages and the number of words transferred for a given operation, typically along the critical path. A broadcast of W words among P processes has the cost $[\log P,\ W \log P]$. This assumes that broadcast follows a tree-based implementation: there are $\log P$ steps on the critical path (any branch of the tree) and W words are transferred at each step, yielding $\log P$ messages and $W \log P$ words.

Both DOUBLING and INCREMENTING strategies need a sampling phase. In the worst case, the DOUBLING scheme may use up to $2r$ sample vectors while r vectors are sufficient for the INCREMENTING scheme. We use the `PxGEMM` routine from PBLAS to compute the matrix-matrix product. The implementation uses scalable universal matrix multiplication algorithm (SUMMA) [20] and can be modeled, asymptotically, as $[n \log p, \frac{mn}{\sqrt{P}}]$ for the product matrix S of dimension $m \times n$. This relies on the fact that, when computing a product $S = AR$, the `PxGEMM` routine selects an algorithm that reduces communication based on the size of the operands A, S, R. In our case, matrix A is the largest operand, so `PxGEMM` chooses an algorithm that communicates only S and R. The selection strategy is described in [9].

For a rectangular matrix of dimension $m \times n$, assuming $m/P \geq n$, the communication cost for nonpivoted QR factorization (`PDGEQRF` in ScaLAPACK) is $[2n \log P, \frac{mn}{\sqrt{P}} \log P]$ [1, 5]. For QR factorization with column pivoting, i.e., `PDGEQPF` in ScaLAPACK, additional communication is needed at each step to compute the column norm

and permutation of the column with the maximum norm to the leading position. Computing the maximum column norm needs two reductions along row and column dimensions, costing $2\log\sqrt{P} = \log P$ messages. The additional communication volume is of lower order term. In total, `PDGEQPF` has communication cost $[3n\log P, \frac{mn}{\sqrt{P}}\log P]$.

In the DOUBLING strategy (Algorithm 2), we need $s = \log\frac{r}{d_0}$ steps of augmentations to reach the final rank r. At the kth step, we perform RRQR (ID) for S_k of dimension $m \times d_0 2^k$, using `PDGEQPF`. The total communication cost of s steps sums up to

$$\sum_{k=0}^{s}\left[3d_0 2^k\log P,\ \frac{m(d_0 2^k)}{\sqrt{P}}\log P\right] = \left[3d_0\log P\sum_{k=0}^{s}2^k,\ \frac{md_0}{\sqrt{P}}\log P\sum_{k=0}^{s}2^k\right] \tag{4.1}$$
$$= \left[3d_0 2^{s+1}\log P,\ \frac{md_0 2^{s+1}}{\sqrt{P}}\log P\right] = \left[6r\log P,\ \frac{2mr}{\sqrt{P}}\log P\right].$$

In the final step, we may need to perform a RRQR on a sample matrix as large as $m \times (2r)$, which has the communication cost $[3(2r)\log P, \frac{m(2r)}{\sqrt{P}}\log P]$. Adding together, the total communication cost is $[12r\log P, \frac{4mr}{\sqrt{P}}\log P]$.

In the INCREMENTING strategy (Algorithm 3), we need $s = \frac{r}{\Delta d}$ steps of increments to reach the final rank r. At the kth step, two costly operations are the block Gram–Schmidt and block QR (lines 10 and 12, respectively, in Algorithm 3).

Each Gram–Schmidt orthogonalization step requires two matrix multiplications; we use `PDGEMM` in PBLAS, which uses a pipelined SUMMA algorithm [20]. The Q matrix is of dimension $m \times d$, where $d = \Delta d(k-1)$. The S_k matrix is of dimension $m \times \Delta d$. The communication cost for $Q^* \cdot S_k$ is $[\Delta d, \frac{\Delta d(m+d)}{\sqrt{P}}]$. The cost for another multiplication $Q \cdot (Q^* \cdot S_k)$ is the same. Since we do Gram–Schmidt twice, the total communication cost of s steps sums up to

$$2\cdot 2\sum_{k=1}^{s}\left[\Delta d, \frac{\Delta d(m+d)}{\sqrt{P}}\right] = 4\cdot\left[r, \sum_{k=1}^{s}\frac{m\Delta d + \Delta d^2\cdot k}{\sqrt{P}}\right] = \left[4r, 4\frac{mr + r^2/2}{\sqrt{P}}\right]. \tag{4.2}$$

Next, we perform QR for $\hat{S}_k$ of dimension $m \times \Delta d$, using `PDGEQRF`. The total communication cost of s steps sums up to

$$\sum_{k=1}^{s}\left[2\,\Delta d\log P,\ \frac{m\,\Delta d}{\sqrt{P}}\log P\right] = \left[2r\log P,\ \frac{mr}{\sqrt{P}}\log P\right]. \tag{4.3}$$

Comparing (4.2) and (4.3), we see that the communication in Gram–Schmidt is a lower order term compared to the QR factorizations, therefore we ignore it. In a final step (line 16 in Algorithm 3) we perform a large RRQR, with communication cost $[3r\log P, \frac{mr}{\sqrt{P}}\log P]$. This is added into (4.3). The leading cost of the INCREMENTING strategy is $[5r\log P, \frac{2mr}{\sqrt{P}}\log P]$. Compared to the DOUBLING strategy, the INCREMENTING strategy requires both fewer messages and smaller communication volume.

Table 5 summarizes the computational costs of the old DOUBLING algorithm and the new INCREMENTING algorithm. As can be seen, the INCREMENTING algorithm is superior to the DOUBLING algorithm in both flop count and communication count. Within each algorithm, the sampling stage (matrix-matrix multiplication) incurs fewer messages and asymptotically lower communication volume than the RRQR stage.

TABLE 5

Algorithmic characteristics of DOUBLING *and* INCREMENTING *strategies. Here, we only list the costs of the two most expensive phases of the algorithms: sampling and RRQR.*

	Flop count	[#messages, #words]	Stopping criteria
DOUBLING			
→ sampling	$4mnr$	$\left[2r\log P,\ \frac{2mr}{\sqrt{P}}\right]$	
→ RRQR	$\frac{20}{3}mr^2$	$\left[12r\log P,\ \frac{4mr}{\sqrt{P}}\log P\right]$	(3.2)
INCREMENTING (NEW)			
→ sampling	$2mnr$	$\left[r\log P,\ \frac{mr}{\sqrt{P}}\right]$	
→ RRQR	$6mr^2$	$[5r\log P,\ \frac{2mr}{\sqrt{P}}\log P]$	(3.26)

5. Numerical experiments.

5.1. Experiments setup. Numerical experiments were conducted using the Cori supercomputer at National Energy Research Scientific Computing Center. Each node has two sockets, each socket is populated with a 16-core Intel Xeon Processor E5-2698 v3("Haswell") at 2.3 GHz and 128 GB of RAM memory. We used STRUMPACK v2.2 linked with Intel MKL and compiled with the Intel compilers version 17.0.1.132 in Release mode.

5.2. Test problems. The first set of numerical experiments depicts six dense linear systems from a variety of applications in double precision (dp) and single-precision (sp). A complete description of the problems under consideration can be found in [18], with the exception of the first experiment at Table 6. The first experiment is the parameterized example $\alpha I + \beta UDV^*$. U and V are orthogonal matrices of rank r and are distributed in order to allow for fast element extraction and scalable tests, and D is either the identity matrix or a diagonal matrix with decaying entries, giving us a matrix that has off-diagonal blocks with either constant or decaying singular values. In this particular case we set $D_{k,k} = 2^{-53(k-1)/\ell}$ such that $\ell = 500$ is the double precision numerical rank. Each experiment is performed at three increasingly tighter approximation tolerances, while we report on the three main stages involved in the solution of the linear system: approximation—compression—of the dense matrix, factorization, and solve. The metrics of interest at each stage are memory consumption, flops, and max wall-clock time from all MPI ranks.

Numerical experiments (Table 6) show that compression takes the most flops and time, followed by a ULV factorization and solve [4], which are much cheaper in comparison. Nonetheless, as we increase the accuracy of the approximation, we notice a moderate increase in the wall-clock time in all three stages.

5.3. Performance breakdown. The second set of numerical experiments illustrates the new INCREMENTING adaptive technique in comparison with the traditional DOUBLING adaptive technique that relies on RRQR with column pivoting. Table 7 shows the detailed breakdown of the flops and time in different stages of the two algorithms for the boundary element method (BEM) Sphere linear system with $d_0 = 128$ and $\Delta d = 256$ and using $p = 1,024$ cores.

The first observation is that in both INCREMENTING and DOUBLING nearly 90% of the flops are in the initial sampling step. On the other hand, since the sampling step involves highly efficient matrix-matrix multiplication (i.e., at a higher compute rate and less communcation than RRQR), the percentage time spent at this step is under 50%. This result is consistent with our derived analytical performance formula in Table 5.

TABLE 6

Solving linear systems from different applications. The largest experiments ($N = 500,000$) used $p = 1,024$ cores, whereas the rest of the experiments used $p = 64$ cores. The absolute tolerance ε_{abs} is kept constant at $1E-08$. All tests use the INCREMENTING *adaptive strategy.*

Matrix	**N**	ε_{rel}	**HSS compression**				ULV^* **factorization**			**Solve**	
			HSS rank	Mem (MB)	Flops $\times 10^{12}$	Time (s)	Mem (MB)	Flops $\times 10^{12}$	Time (s)	Flops $\times 10^{9}$	Time (s)
$\alpha I + \beta UDV^*$ **(dp)**	500k	1E-02	28	356	0.01	4.22	559	0.01	0.17	0.19	0.10
		1E-06	59	671	0.03	4.71	1343	0.03	0.18	0.45	0.14
		1E-10	73	868	0.06	4.93	1927	0.06	0.24	0.64	0.15
Toeplitz (dp)	500k	1E-02	2	40	0.70	1.50	64	0.001	0.11	0.02	0.06
		1E-06	2	40	0.70	1.52	64	0.001	0.11	0.02	0.08
		1E-10	2	40	0.70	1.62	64	0.001	0.11	0.02	0.09
Quantum Chem. (dp)	500k	1E-02	12	235	34.60	9.26	383	0.01	0.11	0.12	0.09
		1E-06	75	308	34.61	9.51	486	0.01	0.22	0.16	0.09
		1E-10	113	377	34.62	9.66	615	0.01	0.28	0.21	0.09
BEM Acoustic (dp)	10k	1E-02	723	363	2.49	6.10	770	0.40	0.91	0.54	0.09
		1E-06	1249	607	5.12	12.72	1304	1.12	1.91	0.91	0.17
		1E-10	1332	631	5.20	12.90	1366	1.23	1.97	0.95	0.27
BEM Sphere (sp)	27k	1E-01	500	159	9.77	11.16	364	0.13	0.55	0.47	0.09
		1E-03	1491	433	26.17	29.85	1089	1.20	1.91	1.39	0.16
		1E-05	2159	795	46.97	60.07	1908	3.57	4.63	2.51	0.20
Schur100 (sp)	10k	1E-01	283	41	0.05	1.37	93	0.02	0.20	0.12	0.053
		1E-03	429	82	0.20	2.26	195	0.08	0.32	0.26	0.055
		1E-05	490	109	0.34	2.49	255	0.14	0.41	0.34	0.056

TABLE 7

Performance breakdown. "Compute Samples" and "Reduce Samples" are defined in Table 1.

	Flops $\times 10^{12}$ **(% Flops)**		**Time (% Time)**	
	INCREMENTING	DOUBLING	INCREMENTING	DOUBLING
Compression	29.75	56.07	5.50	8.24
HSS Rank	1480	1945		
Relative error	2.40E-03	2.68E-04		
→ Random samp.	26.61 (89.5%)	50.22 (89.6%)	2.40 (43.5%)	3.93 (47.8%)
→ ID	0.85 (2.9%)	2.73 (4.9%)	1.54 (27.9%)	3.04 (36.9%)
→ QR	0.51 (1.7%)	-	0.52 (9.5%)	-
→ Orthogonalize	0.61 (2.1%)	-	0.18 (3.3%)	-
→ Comp. samples	1.08 (3.6%)	2.89 (5.2%)	0.55 (10.0%)	0.86 (10.4%)
→ Red. samples	0.09 (0.3%)	0.23 (0.4%)	0.32 (5.8%)	0.41 (5.0%)
Factorization	1.29	2.67	1.98	2.83
Solve	0.001	0.002	0.14	0.15

The second most costly step is ID, i.e., RRQR. It has less than 5% of the flops but takes 28% and 37% time, respectively. This shows that the data movement associated with column pivoting is expensive.

Using the same tolerance $1E-3$, the INCREMENTING strategy achieves sufficient accuracy, while the DOUBLING strategy does more adaptations, leading to higher rank and the accuracy level more than needed and taking longer time.

5.4. Adaptivity performance.

5.4.1. Evaluation of new stopping criterion. This section considers the parametrized problem $\alpha I + \beta UDV^*$ with $N = 20,000$, $\alpha = 1$, $\beta = 1$, and $D_{k,k} = 2^{-53(k-1)/\ell}$, with $\ell = 200$, using $p = 256$ cores. We provide a comparison with the

classical stopping criterion proposed in HMT [12], as shown in Table 8. In our strategy, we vary both ε_{rel} and ε_{abs}. HMT only works with absolute tolerance, so we put it in the last row of the table.

TABLE 8
New stopping criterion and HMT criterion [12]. *Each entry has two numbers: one is the relative error of the HSS approximation given by* $||A - HSS \cdot I||_F/||A||_F$, *another is the HSS rank.*

		ε_{abs}			
		1E-02	**1E-06**	**1E-10**	**1E-14**
ε_{rel}	**1E-02**	1.05E-02/43	1.05E-02/43	1.05E-02/43	1.05E-02/43
	1E-06	1.82E-05/77	1.82E-05/77	1.82E-05/77	1.82E-05/77
	1E-10	4.91E-06/87	5.18E-09/127	5.18E-09/127	5.18E-09/127
	1E-14	4.91E-06/87	6.68E-10/138	6.58E-13/187	6.58E-13/187
	HMT	3.14E-06/87	3.50E-10/139	3.61E-14/192	5.53E-15/5,000+

It can be seen that with our new criterion, when we set ε_{rel} and ε_{abs} to be the same, the approximation accuracy is at the same level of the requested tolerance (see the diagonal of Table 8).

With the HMT criterion, however, since it uses an upper bound as a termination metric, this in practice can be pessimistic, delivering an approximation error that is usually smaller than requested, at the expense of large ranks. Larger ranks in turn lead to increased memory requirements, which often form the bottleneck for large scale scientific computing applications. With our new strategy the accuracy and memory usage can be controlled more precisely by the user.

As a result, when the given tolerance is close to machine precision, HMT criterion requires many steps, as depicted in the bottom right corner of Table 8, where the algorithm terminated not due to achieving ε_{abs}, but because it reached the maximum allowable rank (5,000 in this case). Thus, there will be some absolute tolerances that cannot be satisfied, yet it is not clear how small this tolerance is or how to determine it *before* attempting compression. Furthermore, it is not clear how to determine when this happens during compression, either. A relative tolerance is much easier to set and frequently of most practical interest.

5.4.2. Evaluation of adaptivity cost. In order to assess the cost associated with our adaptivity strategy, we performed the following experiments with two matrices for which we know the HSS ranks. Thus, we can choose a precise number of random vectors, so that there is no need for adaptation. This should be the fastest possible case, and we denote this as "known-rank." Suppose we do not have an adaptive implementation, the best a user can do is to restart compression from scratch (unable to reuse the partial compression results) when the final residual is large, manually increasing the number of random vectors. We denote this as "hard-restart." In between these two modes is our adaptive strategy INCREMENTING.

The first matrix is $I + UDV^*$, where $D_{k,k} = 2^{-53(k-1)/\ell}, \ell = 1200, N = 60,000$. The second matrix is the BEM acoustic problem, with $N = 10,000$. Table 9 shows the compression times with different configurations of d_0 and Δd. It is clear that our adaptive strategy is nearly as fast as the fastest "known-rank" case and is up to 2.7x faster than the "hard-restart."

5.5. Scalability. The last numerical experiment depicts two strong scaling studies, as shown in Figure 5. The first test problem depicts the quantum chemistry dense linear system of size $N = 300,000$ and HSS relative approximation error of $1E-2$ and HSS leaf size of 128. This matrix is amenable to efficient rank compression, resulting

TABLE 9

Evaluation of adaptivity cost. For all the tests, we use $\varepsilon_{abs} = \varepsilon_{rel} = 1E-14$, We use $1,024$ cores for the first problem and 64 cores for the second one. With each strategy, we report the compression time (Compr. time), HSS rank, and the number of adaptation steps needed (# adapt.)

"Problem"	"Phase"	"Known-rank"	INCREMENTING	"Hard-restart"
$I + UDV^*$	Compr. time	36.5	37.2	100.3
$d_0 = 128$	HSS-rank	1162	1267	1165
$\Delta d = 64$	Num. adapt.	0	17	4
$I + UDV^*$	Compr. time	35.7	36.6	54.6
$d_0 = 512$	HSS-rank	1162	1194	1161
$\Delta d = 128$	# adapt.	0	5	2
BEM acoustics	Compr. time	11.15	12.5	18.3
$d_0 = 128$	HSS-rank	1264	1334	1348
$\Delta d = 64$	# Adapt.	0	24	4
BEM sphere	Compr. time	11.9	10.5	18.9
$d_0 = 512$	HSS-rank	1276	1352	1362
$\Delta d = 128$	# Adapt.	0	9	2

in an HSS rank of 12. In contrast, for a problem with larger numerical rank and tighter numerical accuracy, we show the scalability from the parametrized test case $\alpha I + \beta UDV^*$ with $N = 500,000$, $\alpha = 1$, $\beta = 1$, and $D_{k,k} = 2^{-53(k-1)/\ell}$, with $\ell = 500$. The resulting HSS rank is $\ell = 480$ at an HSS relative approximation tolerance of $1E-14$ with HSS leaf size of 128. The first example scales better since its HSS rank is small and there is no need for adaptation, whereas the second example requires a few steps for rank adaptation, given that its HSS rank is quite large.

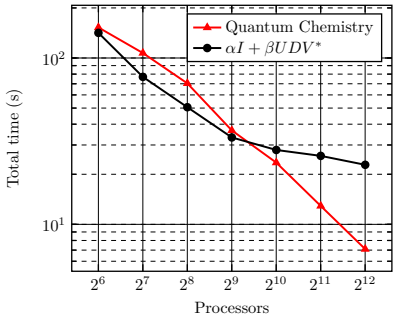

FIG. 5. *Strong scaling experiment up to $p = 4,096$ cores of the Cori supercomputer. The quantum chemistry problem has $N = 300,000$ and HSS rank of 12, and the parametrized problem has $N = 500,000$ and HSS rank of 480.*

6. Conclusion. We presented two new stopping criteria that allow to accurately predict the quality of low-rank approximations computed using randomized sampling. This helps reduce the total number of random samples as well as reduce the communication cost. We apply these adaptive randomized sampling schemes for the construction of HSS matrices. Compared to previous adaptive randomized HSS compression approaches, our new methods are more rigorous and include both absolute and relative stopping criteria. The numerical examples show faster compression time for the new incremental adaptive strategy compared to previous methods. Randomized numerical linear algebra methods are very interesting from a theoretical standpoint. However, writing robust and efficient, parallel software is far from trivial. In this paper we

have focused on a number of practical issues regarding the adaptive compression of HSS matrices, leading to faster compression, with guaranteed accuracy. The methods shown here should carry over to other structured matrix representations.

Acknowledgments. We thank Guillaume Sylvand (Airbus) for providing us with the BEM test problems. We thank Daniel Haxton (LBNL) and Jeremiah Jones (Arizona State University) for providing us with the quantum chemistry test problem.

We thank Yang Liu (LBNL), Wissam Sid Lakhdar (LBNL), and Liza Rebrova (UCLA) for the insightful discussions throughout this research.

REFERENCES

[1] L. S. Blackford, J. Choi, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley, *ScaLAPACK Users' Guide*, SIAM, Philadelphia, 1997.

[2] P. Brémaud, *Discrete Probability Models and Methods: Probability on Graphs and Trees, Markov Chains and Random Fields, Entropy and Coding*, Probability Theory and Stochastic Modelling, Springer New York, 2017.

[3] S. Chandrasekaran, M. Gu, and W. Lyons, *A fast adaptive solver for hierarchically semiseparable representations*, Calcolo, 42 (2005), pp. 171–185.

[4] S. Chandrasekaran, M. Gu, and T. Pals, *A fast ULV decomposition solver for hierarchically semiseparable representations*, SIAM J. Matrix Anal. Appl., 28 (2006), pp. 603–622.

[5] J. Demmel, L. Grigori, M. Hoemmen, and J. Langou, *Communication-optimal Parallel and Sequential QR and LU Factorizations*, SIAM J. Sci. Comput., 34 (2012), pp. A206–A239.

[6] P. Ghysels, X. S. Li, C. Gorman, and F.-H. Rouet, *A robust parallel preconditioner for indefinite systems using hierarchical matrices and randomized sampling*, in Proceedings of the Parallel and Distributed Processing Symposium (IPDPS), IEEE, 2017, pp. 897–906.

[7] P. Ghysels, X. S. Li, F.-H. Rouet, S. Williams, and A. Napov, *An efficient multi-core implementation of a novel HSS-structured multifrontal solver using randomized sampling*, SIAM J. Sci. Comput., 38 (2016), pp. 358–384.

[8] M. Gu and S. C. Eisenstat, *Efficient algorithms for computing a strong rank-revealing QR factorization*, SIAM J. Sci. Comput., 17 (1996), pp. 848–869.

[9] J. Gunnels, C. Lin, G. Morrow, and R. Van De Geijn, *A flexible class of parallel matrix multiplication algorithms*, in Proceedings of the First Merged International Parallel Processing Symposium and Symposium on Parallel and Distributed Processing, IEEE, 1998, pp. 110–116.

[10] W. Hackbusch and S. Börm, *Data-sparse Approximation by Adaptive $\mathcal{H}^2$-Matrices*, Computing, 69 (2002), pp. 1–35.

[11] W. Hackbusch and B. N. Khoromskij, *A Sparse $\mathcal{H}$-Matrix Arithmetic*, Computing, 64 (2000), pp. 21–47.

[12] N. Halko, P. G. Martinsson, and J. A. Tropp, *Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions*, SIAM Rev., 53 (2011), pp. 217–288.

[13] L. Lin, J. Lu, and L. Ying, *Fast construction of hierarchical matrix representation from matrix–vector multiplication*, J. Comput. Phys., 230 (2011), pp. 4071–4087.

[14] X. Liu, J. Xia, and M. V. de Hoop, *Parallel Randomized and Matrix-Free Direct Solvers for Large Structured Dense Linear Systems*, SIAM J. Sci. Comput., 38 (2016), pp. S508–S538.

[15] P. Martinsson and S. Voronin, *A randomized blocked algorithm for efficiently computing rank-revealing factorizations of matrices*, SIAM J. Sci. Comput., 38 (2016), pp. S485–S507, https://doi.org/10.1137/15M1026080.

[16] P.-G. Martinsson, *A fast randomized algorithm for computing a hierarchically semiseparable representation of a matrix*, SIAM J. Matrix Anal. Appl., 32 (2011), pp. 1251–1274.

[17] Å. Björck, *Numerics of Gram-Schmidt orthogonalization*, Linear Algebra Appl., 197-198 (1994), pp. 297–316.

[18] F.-H. Rouet, X. S. Li, P. Ghysels, and A. Napov, *A distributed-memory package for dense Hierarchically Semi-Separable matrix computations using randomization*, ACM Trans. Math. Softw., 42 (2016).

[19] G. Stewart, *Block Gram-Schmidt Orthogonalization*, SIAM J. Sci. Comput., 31 (2008), pp. 761–775.

[20] R. A. van de Geijn and J. Watts, *SUMMA: Scalable universal matrix multiplication algorithm*, Concurrency Pract. Ex., 9 (1997), pp. 255–274.
[21] J. Vogel, J. Xia, S. Cauley, and V. Balakrishnan, *Superfast divide-and-conquer method and perturbation analysis for structured eigenvalue solutions*, SIAM J. Sci. Comput., 38 (2016), pp. A1358–A1382.
[22] S. Wang, X. S. Li, J. Xia, Y. Situ, and M. V. De Hoop, *Efficient scalable algorithms for solving dense linear systems with hierarchically semiseparable structures*, SIAM J. Sci. Comput., 35 (2013), pp. C519–C544.
[23] Y. Xi and J. Xia, *On the stability of some hierarchical rank structured matrix algorithms*, SIAM J. Matrix Anal. Appl., 37 (2016), pp. 1279–1303.
[24] Y. Xi, J. Xia, S. Cauley, and V. Balakrishnan, *Superfast and stable structured solvers for Toeplitz least squares via randomized sampling*, SIAM J. Matrix Anal. Appl., 35 (2014), pp. 44–72.
[25] Y. Xi, J. Xia, and R. Chan, *A fast randomized eigensolver with structured LDL factorization update*, SIAM J. Matrix Anal. Appl., 35 (2014), pp. 974–996.
[26] J. Xia, *Randomized sparse direct solvers*, SIAM J. Matrix Anal. Appl., 34 (2013), pp. 197–227.
[27] J. Xia, S. Chandrasekaran, M. Gu, and X. S. Li, *Fast algorithms for hierarchically semiseparable matrices*, Numer. Linear Algebra Appl., 17 (2010), pp. 953–976.
[28] J. Xia, S. Chandrasekaran, M. Gu, and X. S. Li, *Superfast multifrontal method for large structured linear systems of equations*, SIAM J. Matrix Anal. Appl., 31 (2010), pp. 1382–1411.
[29] J. Xia, Y. Xi, and M. Gu, *A superfast structured solver for Toeplitz linear systems via randomized sampling*, SIAM J. Matrix Anal. Appl., 33 (2012), pp. 837–858.
[30] W. Yu, Y. Gu, and Y. Li, *Efficient randomized algorithms for the fixed-precision low-rank matrix approximation*, SIAM J. Matrix Anal. Appl., 39 (2018), pp. 1339–1359, https://doi.org/10.1137/17M1141977.