Robust Treecode Approximation for Kernel Machines

William B. March
Institute for Computational Engineering and Sciences
University of Texas
Austin, TX, USA
march@ices.utexas.edu

Bo Xiao
Institute for Computational Engineering and Sciences
University of Texas
Austin, TX, USA
bo@ices.utexas.edu

Sameer Tharakan
Institute for Computational Engineering and Sciences
University of Texas
Austin, TX, USA
sameer@ices.utexas.edu

Chenhan D. Yu
Department of Computer Science
University of Texas
Austin, TX, USA
chenhan@cs.utexas.edu

George Biros
Institute for Computational Engineering and Sciences
University of Texas
Austin, TX, USA
gbiros@acm.org

ABSTRACT

Since exact evaluation of a kernel matrix requires $\mathcal{O}(N^2)$ work, scalable learning algorithms using kernels must approximate the kernel matrix. This approximation must be robust to the kernel parameters, for example the bandwidth for the Gaussian kernel.

We consider two approximation methods: Nystrom and an algebraic treecode developed in our group. Nystrom methods construct a global low-rank approximation of the kernel matrix. Treecodes approximate just the off-diagonal blocks, typically using a hierarchical decomposition.

We present a theoretical error analysis of our treecode and relate it to the error of Nystrom methods. Our analysis reveals how the block-rank structure of the kernel matrix controls the performance of the treecode. We evaluate our treecode by comparing it to the classical Nystrom method and a state-of-the-art fast approximate Nystrom method. We test the kernel matrix approximation accuracy for several different bandwidths and datasets. On the MNIST2M dataset (2M points in 784 dimensions) for a Gaussian kernel with bandwidth $h = 1$, the Nystrom methods' error is over 90% whereas our treecode delivers error less than 1%. We also test the performance of the three methods on binary classification using two models: a Bayes classifier and kernel ridge regression. Our evaluation reveals the existence of bandwidth values that should be examined in cross-validation but whose corresponding kernel matrices cannot be approximated well by Nystrom methods. In contrast, the treecode scheme performs much better for these values.

1. INTRODUCTION

Given a kernel function $\mathcal{K}$, a set of $\mathcal{X}$ of points $\{x_j\}_{j=1}^N \in \mathbb{R}^d$, and weights $w_j \in \mathbb{R}$, the $N \times N$ kernel matrix K is

$$K_{ij} = \mathcal{K}(x_i, x_j), \; i, j = 1, \ldots, N. \tag{1}$$

Here, we focus on the Gaussian, a commonly used kernel, which is parameterized by a *bandwidth* h. Many kernel-based learning methods require matrix-vector products Kw that cost $\mathcal{O}(N^2)$ since K is typically dense. For large N, Kw cannot be computed exactly and an approximation scheme is necessary. The bandwidth h is crucial to the properties of K, which in turn affect the performance of approximation methods. For example, for very small h, K approaches the identity matrix whereas for large h, K approaches the rank-one constant matrix. Typically, cross-validation is used to select h; this requires that the kernel approximation method is accurate (*i.e.* robust) for different values of h.

Motivation. We need a kernel matrix approximation scheme that scales well with both the problem size N and with the dimension d and is also robust to a range of values of h. As an example, we consider the following simple learning task. We study three binary classification problems (for more details, see §4). We use a Bayes classifier where the individual class densities are estimated via Gaussian kernel density estimation [27]. For a 1,000 point test set, we show the resulting classification accuracy, α_c (16), for a range of values of the bandwidth h.

COVTYPE		SUSY		MNIST2M	
h	α_c	h	α_c	h	α_c
0.35	71.6	0.50	65.7	4	95.0
0.22	74.0	0.15	72.1	2	97.4
0.14	79.8	0.09	75.0	1	100
0.02	95.4	0.05	76.7	0.1	99.5
0.001	6.4	0.01	64.3	0.05	13.6

Our point here is this: regardless of the true, optimal bandwidth for a given task, *any* Gaussian kernel-based learning method must be able to accurately train models for bandwidths in this range in order to effectively perform cross validation.[1] We want to emphasize that kernel methods exhibit interesting behavior across a fairly wide range of bandwidths. So, what are our options regarding kernel approximation schemes?

The *Nystrom method* and its variants are the most popular matrix approximation schemes [28]. Nystrom methods construct a low-rank approximation of the entire matrix K

[1]The example is used to motivate our work, not to make any claims regarding the classification method. For this example we use exact kernel evaluation and test only 1000 points (disjoint from the training set)—not enough to assess the classifier.

KDD'15, August 10-13, 2015, Sydney, NSW, Australia.

DOI: http://dx.doi.org/10.1145/2783258.2783272.

by subsampling it (see §2.1). Nystrom scales as $\mathcal{O}(Nr+r^3)$, where r is the target numerical rank of the matrix. If K is nearly low rank, Nystrom methods are fast and accurate. However, in many applications—especially for large-scale data with millions of points—there are choices of h which are relevant to machine learning applications and which do not result in a globally low-rank matrix K.

Treecodes are an alternative to Nystrom approximations. Treecodes construct a representation of the matrix in which only *off-diagonal* subblocks are approximated by low-rank factorization. These submatrices are identified using a hierarchical clustering of the points. Although treecodes *do not require* the entire matrix to be low rank, they have other restrictions. Most treecodes construct low rank factorizations using series expansions that are kernel dependent and do not scale well with the ambient dimension d. In a series of recent papers, we have introduced an algebraic treecode algorithm that circumvents these difficulties. In [20], we studied the approximation scheme of off-diagonal blocks, in [21] we introduced ASKIT[2], the new treecode scheme whose scalability depends only on the intrinsic dimensionality of the dataset, and in [22], we introduced the parallel algorithms that enable scalability of ASKIT on distributed and shared memory systems. In our previous work, an important missing piece was a precise analysis that shed light on the effect of the different parameters on the overall error of the method and the comparison of ASKIT with Nystrom methods.

Contributions. We prove a new result for the convergence of ASKIT. This result connects the parameters of ASKIT to the kernel matrix approximation error. Also, it allows a theoretical comparison between ASKIT and Nystrom methods. We introduce a key parameter that can be approximately measured and allows us to determine which approach is preferable for a given task (see §3).

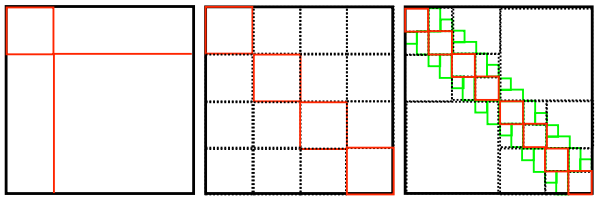
We compare ASKIT with two Nystrom variants: the classical Nystrom and MEKA, an efficient block-Nystrom method [26]. In this work, MEKA was shown to outperform several state-of-the-art Nystrom methods. We apply standard Nystrom, MEKA, and ASKIT on three datasets. We found that for several bandwidths that give good classification accuracy (using Bayes and ridge regression), Nystrom and MEKA cannot approximate the matrix well and lead to erroneous conclusions regarding the optimal bandwidth, while ASKIT performs well for all values of h (*e.g.* Table 6).

In addition, we introduce two modifications to ASKIT: level restriction and adaptive rank selection. The former can increase accuracy, and the latter can reduce the cost and the need to select manually the ranks for the off-diagonal blocks. However, due to space limitations we do not discuss their effect in any detail. We outline ASKIT in §2.2.

Related work. There is a significant amount of literature for both Nystrom and treecodes. In computational physics (typically in two and three dimensions), treecodes have been scaled to trillions of points [10, 11, 16]. Important works in high dimensional treecodes include [9, 12, 17, 25, 29]. Their common feature is that the low rank factorizations of the off-diagonal blocks are constructed using analytic expansions (e.g., Taylor expansions). Such expansions do not scale with the ambient dimension [21].

For Nystrom methods we refer the reader to [7, 14] for a thorough literature review of the main theoretical results on generalized Nystrom methods and randomized matrix approximations. One of the fastest Nystrom algorithms, MEKA, was introduced in [26]. It is a block approach that constructs a low-rank factorization of K. We discuss this method in more detail in §2. In [26], the authors compare MEKA to several other state-of-the art Nystrom methods. Like us, they also identify the lack of scalability of Nystrom methods with respect to the rank and the problem size (as a function of the kernel width). Some of the datasets and parameters we use in our experiments partially follow [26].

[2] *A*pproximate *S*keletonization *K*ernel *I*ndependent *T*reecode.



(a) Nystrom. (b) MEKA. (c) ASKIT.

Figure 1: *We illustrate the basic approaches used in Nystrom methods and* ASKIT. *We show the partitioning of the kernel matrix used in each method. The blocks in red are either computed exactly (in the Nystrom and ASKIT methods) or are approximated (in* MEKA*). In Figure 1(c), we show the evaluation pattern in* ASKIT. *In addition to the exact evaluations in red,* ASKIT *computes exact interactions with the κ nearest neighbors in green and approximates the off-diagonal blocks. We illustrate the simple (one-dimensional) case where neighbor interactions are near the diagonal. In general, neighbor interactions appear throughout the matrix.*

Limitations. ASKIT is only a treecode; a dual-tree or fast-multipole-type algorithm will be faster. Our implementation of ASKIT only supports training. This is just an implementation issue but it prevents testing on large numbers of points. The construction of low rank blocks (skeletonization phase) is not optimized for single core performance, so the timings for ASKIT are not representative. The evaluation phase has been highly optimized, however.

Notation. Here we summarize the main notation we use in the paper: r is the approximation rank used in the Nystrom method, s is the number of skeleton points per node (the approximation rank for off-diagonal blocks in ASKIT), and $\tilde{K}$ is an approximation of K. For subsets of points S and T, we use $K(S,T)$ to refer to the submatrix of K with rows indexed by points in S and columns by points in T. We refer to the vector in a kernel matrix-vector multiplication as the *weights*, denoted w, and the result of the multiplication as the *potentials*, denoted u. For the treecode, we use α to refer to a tree node and $\mathtt{l}(\alpha)$ and $\mathtt{r}(\alpha)$ for its children. We refer to a point for which we compute a potential as a *target*, and points contributing to its potential as *sources*.

2. METHODOLOGY

We now discuss the two main methods used in this paper: *Nystrom methods* and *ASKIT*.

2.1 Nystrom Methods

The Nystrom method [28] uses subsampling to construct an approximation. We sample $r \leq \ell \ll N$ columns (which is

equivalent to sampling ℓ data points). We construct the $\ell \times \ell$ matrix K_ℓ of interactions between the sampled points. Then we approximate K by $\tilde{K} = K_{N,\ell} K_\ell^\dagger K_{\ell,N}$, where $K_{N,\ell}$ is the $N \times \ell$ matrix of kernel interactions between all points and the subsampled points, and $K_\ell^\dagger$ is the approximate pseudo-inverse of K_ℓ computed using the eigendecomposition of K_ℓ. For out datasets, we tested several values for ℓ. We found that $\ell = r$ works best. Larger ℓ (e.g., $\ell = 2r$) result in negligible accuracy improvement at much higher cost.

The approximation error of the Nystrom method has been bounded for many sampling distributions [15]. In the uniform case, with high probability the error is bounded in terms of the $r+1^{\text{st}}$ singular value σ_{r+1} by [20]

$$\boxed{\epsilon_N = \left\| Kw - \tilde{K}w \right\| \leq \left(1 + 6\frac{N}{\ell}\right)^{\frac{1}{2}} \sigma_{r+1}(K)\|w\|.} \quad (2)$$

MEKA. Like Nystrom methods, MEKA uses matrix factorization to approximate kernel interactions. It partitions the data into $\mathcal{C}$ clusters using k-means. The full kernel matrix K is partitioned into $\mathcal{C}^2$ blocks according to cluster memberships. The algorithm then computes a compact representation of each of these blocks. Let $K^{(i,j)}$ be a block of the partitioned kernel matrix. For an on-diagonal block, MEKA uses the standard Nystrom approximation: $K^{(i,i)} \approx W^{(i)} W^{(i)\,T}$, where $W^{(i)}$ is the matrix of eigenvectors of block i (scaled by their eigenvalues). For an off-diagonal block $K^{(i,j)}$, MEKA subsamples sets of points v_i and v_j from blocks i and j. It then computes a matrix $M^{(i,j)}$ by solving the least-squares problem $K(v_i, v_j) = W_{v_i}^{(i)} M^{(i,j)} W_{v_j}^{(j)\,T}$. Taking these approximations together, the entire MEKA approximation is $\tilde{K} = WMW^T$, where W is the direct sum of $W^{(1)}, \ldots, W^{(\mathcal{C})}$ and M is formed from the blocks $M^{(i,j)}$. By construction, MEKA's error is worse than Nystrom (for the same r), but MEKA is much faster and requires less storage.

2.2 ASKIT

ASKIT, described in detail in [21], is a *treecode*. Rather than directly approximating the entire matrix K, as in Nystrom methods, treecodes partition the matrix then approximate some off-diagonal blocks. More concretely, consider an even split of the points into two groups. This corresponds to a matrix partitioning

$$K = \begin{bmatrix} K_1 & G_1 \\ G_2 & K_2 \end{bmatrix}. \quad (3)$$

Treecodes recursively split each group of points, which corresponds to further partitioning the on-diagonal blocks K_1 and K_2. This splitting is typically accomplished with a space-partitioning tree. In ASKIT, we use a ball tree. We then determine which blocks to evaluate exactly and which to approximate using a pruning rule, described for ASKIT below.

The key to ASKIT is a novel method for creating low-rank approximations of the off-diagonal matrix blocks G_i. Fix a node α, and let q be the number of points in α. Consider the $N - q$ points which are not in α. Let $G \in \mathbb{R}^{(N-q)\times q}$ be the submatrix of K consisting of all interactions between points in α and all other points.

ASKIT uses the *interpolative decomposition* (ID) [23] as a factorization of G. The ID is a rank s decomposition

$$G \approx G_{\text{col}} P \quad (4)$$

where G_{col} consists of s columns of G and $P \in \mathbb{R}^{s\times q}$. In other words, the ID represents a matrix as a linear combination of a subset of its columns.

We refer to the columns selected in G_{col} as the *skeleton*, which we denote $\mathcal{S}_\alpha$. We compute s *effective weights*

$$\tilde{w}_\alpha = P w_\alpha \quad (5)$$

obtained from the original weights of points in α. Given the skeleton and effective weights, we can compute the approximate contribution of the points in α at a target x as $\mathcal{K}(x, \mathcal{S}_\alpha)\tilde{w}_\alpha$ in $\mathcal{O}(s)$ time and $\mathcal{O}(s)$ storage.

Algorithm 1 ASKIT(Point set $\mathcal{X}$, Tree α_{root})

```
Skeletonize(α_root)
for all x ∈ X
    u(x) = Evaluate(x, α_root)
```

Algorithm 2 Skeletonize(Node α)

```
if α is not a leaf
    Skeletonize(l(α)); Skeletonize(r(α))
    X_α = S_l(α) ∪ S_r(α)
    N_α = (N_l(α) ∪ N_r(α))/X_α
else
    X_α = points in α
    N_α = (∪_{x∈α} N_x)/X_α
Collect the first ℓ points in N_α ∪ U into T_α
Form G = K(T_α, X_α) and compute ID G ≈ G_col P
Store skeleton S_α and skeleton weights w̃_α
```

Algorithm 3 Evaluate(Point x, Node α)

```
if N_x ∩ X_α is empty
    u(x) += K(x, S_α) w̃_α
else if α is a leaf
    u(x) += K(x, α) w_α
else
    u(x) += Evaluate(x, l(α)) + Evaluate(x, r(α))
```

Sampling targets. The ID can be computed using a rank-revealing, pivoted QR factorization of G. However, if done directly, this requires $O(Nq^2)$ work. This is more expensive than computing all of the interactions directly in $O(Nq)$ time. Therefore, we must efficiently approximate the ID of G. We do this by subsampling the rows of G to form an $\ell \times q$ matrix G'. This is equivalent to sampling a subset of the target points.

There are two considerations to take into account when sampling targets. First, the samples need to accurately capture the row space of the entire matrix G. Second, the samples need to be chosen efficiently enough to form a part of a fast matrix-vector multiplication algorithm. Randomized approximation of matrices is a rich research topic [14, 19]. Importance sampling distributions based on column norms and statistical leverage scores can be shown to accurately capture the row space. On the other hand, these distributions require the entire matrix in advance.

As a heuristic approximation, we employ nearest-neighbor information for importance sampling. The nearest-neighbors of all points can be computed exactly or approximately using randomized projection methods [1, 5, 9, 24]. We let $\mathcal{N}_x$

denote the neighbor list of a point x and $\mathcal{N}_\alpha$ the union of the neighbor lists of points in node α. We choose target points first from points in $\mathcal{N}_\alpha$ which are not in α, then sample additional points $\mathcal{U}$ uniformly at random as needed.

Skeletonization. ASKIT first performs a "skeletonization" step. Starting with the smallest blocks G (corresponding to the leaf level of the tree), it constructs skeletons for each block using the sampling method described above. Then, it constructs representations of off-diagonal blocks at higher levels by combining the skeletons of the node's children and performing a new ID (see Alg. 2).

Evaluation. Given the skeletons for each off-diagonal block, ASKIT then performs an evaluation step to compute $\tilde{K}w$. As in all treecodes, some of the evaluations are performed directly, and some are approximated. ASKIT uses a novel combinatorial pruning condition to determine this split. For a target point i, the contribution of any leaf node which owns a nearest neighbor of i is evaluated exactly. This includes the on-diagonal blocks and some off-diagonal contributions (see Figure 1(c)). Any remaining nodes are evaluated approximately using the skeleton; we refer to this partitioning as *pruning* (see Alg. 3). We show these steps combined in the entire algorithm in Alg. 1.

Adaptive rank selection. We use a variant of ASKIT which chooses the rank s of approximation adaptively. As above, we compute the ID of a given matrix G, whose columns correspond to the columns to be compressed and the rows are some sampled target points. We compute a pivoted QR factorization to obtain $G\Pi = QR$ for some permutation Π and upper-triangular matrix R with diagonal entries R_i. Then, given a user-specified approximation tolerance τ and a maximum possible rank $s_{\max}$, we select the approximation rank s by the minimum of $s_{\max}$ and $s = \arg\min_j \left(|R_{j+1}/R_1| < \tau\right)$.

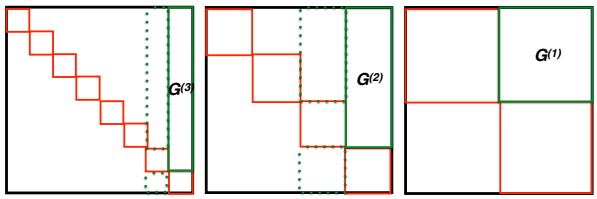
Complexity. The complexity of ASKIT has been derived in [21]. We assume the sample size ℓ and the maximum leaf node size m are $\mathcal{O}(s)$ and that $N \gg s$. We also consider a variant of ASKIT where we do not construct skeletons (and thus never approximate) above a given tree level L, and so we never prune in Alg. 3 above level L. In this case, (neglecting the dN storage for the data themselves and assuming kernel evaluations require $\mathcal{O}(d)$ time), we have the following big-$\mathcal{O}$ complexity bounds, with Nystrom and MEKA included for comparison:

	RAM	approx	eval
Nystrom	$Nr + r^2$	$r^3 + Nrd$	dNr
MEKA	$N\mathcal{C}r + \mathcal{C}^2r^2$	$\mathcal{C}r^3 + \frac{N}{\mathcal{C}}rd$	$dN\mathcal{C}r$
ASKIT	$\kappa N + s^2\frac{N}{m}$	$\frac{N}{m}(s^3 + s^2d)$	$dNs\kappa\left(\log(\frac{N}{m}) - L + 2^L\right)$

where "RAM" is the storage requirement, "approx" is the time to compute the approximate representation (skeletonization in ASKIT), and "eval" is the time to compute an approximate matrix-vector product.

3. THEORY

We now derive a new error bound for ASKIT. Both ASKIT and Nystrom methods can incur error due to 1) the use of low-rank approximations for matrix blocks and from 2) randomly sampling points to compute these approximations. In addition, ASKIT employs the interpolative decomposition, which can incur additional error beyond the SVD. We take all three sources of error into account in our bound.



(a) **Level 3.** (b) **Level 2.** (c) **Level 1.**

Figure 2: *We illustrate the matrix blocks approximated in the skeletonization step of ASKIT. We show on-diagonal blocks in red and off-diagonal in green. We highlight an off-diagonal block $G^{(i)}$ at each level, along with the block corresponding to its sibling (dashed line), which extends above and below the diagonal. At the leaf level (Fig. 2(a)), we compute an approximate ID by sampling rows of $G^{(3)}$ to obtain skeleton points. Moving up the tree, (Fig. 2(b)), we skeletonize $G^{(2)}$ by merging the skeleton points of block $G^{(3)}$ and its sibling, then computing another approximate ID. We continue (Fig. 2(c)) this process all the way up the tree.*

Throughout, we consider the 2-norm of the approximation error in ASKIT:

$$\epsilon_A = \|Kw - \tilde{K}w\| \leq \|K - \tilde{K}\|\|w\| \tag{6}$$

where K is the exact kernel matrix and $\tilde{K}$ is the approximation computed by ASKIT.

Matrix blocks. We introduce some notation for on- and off-diagonal matrix blocks and their singular values. Unless noted otherwise, we consider a fixed approximation rank and a tree traversal with $\kappa = 1$. We also consider a balanced binary tree.

Let K denote the full $N \times N$ kernel matrix. The first split in the tree creates four submatrices, given in (3). We also partition the weight vector w accordingly. The tree recursively splits collections of points, which corresponds to splitting the on-diagonal blocks K_1 and K_2. We refer to an on-diagonal block at level i as $K^{(i)}$ – *i.e.* a block corresponding to the interactions of points in a node with themselves. For the same node, we use $G^{(i)}$ to refer to the interactions between points in the node as sources and all other points. These are the blocks approximated in ASKIT. We illustrate these blocks in Fig. 2.

Error due to ID. ASKIT uses the ID instead of the SVD, which introduces additional error. For a rank s ID of a matrix G with q columns, we have that [13]

$$\|G - G_{\text{col}}P\| \leq (1 + qs(q-s))\,\sigma_{s+1}(G). \tag{7}$$

We also prove a bound on the error due to combining ID's in the tree below. We state this bound as a part of our final ASKIT error bound.

Error due to uniform sampling. We now consider the error due to subsampling targets to form an approximate ID. We begin by considering a uniform sampling distribution. The accuracy of this approach can be bounded in terms of the *matrix coherence* using a theorem from [20].

THEOREM 1. *Let G be $n \times q$. Sample ℓ rows of G to form G'. Let ζ be the coherence of G with respect to a given rank s, defined as $\zeta = \max_j \|U(j, 1{:}s)\|^2$, where U is the matrix of left singular vectors of G and we use MATLAB notation.*

Let $\ell \geq 10q\zeta \log(2s/\delta)$ and let Π be a projection onto the sampled rows. Then, with probability at least $(1-\delta)$

$$\|G(I-\Pi)\| \leq \left(1+6\frac{n}{\ell}\right)^{\frac{1}{2}} \sigma_{s+1}(G). \tag{8}$$

Note that the quantity ζ is bounded between s/q and 1. In the case that ζ is small, the number of samples needed is proportional to $s \log s$.

Error from better sampling distributions. Note that better sampling distributions can lead to tighter error bounds than in (8). For instance, sampling from an importance distribution based on leverage scores can lead to $(1+\epsilon)$ error [18, 19]. Distributions based on Euclidean distances [6] and nearest neighbor information [20] can be effective as well.

Measuring singular value decay. Consider any tree node at level i corresponding to on-diagonal block $K^{(i)} \in \mathbb{R}^{q\times q}$ and off-diagonal block $G^{(i)} \in \mathbb{R}^{(N-q)\times q}$. Define the matrix $L^{(i)} \in \mathbb{R}^{N\times q}$ as

$$L^{(i)} = \begin{bmatrix} G^{(i)} \\ K^{(i)} \end{bmatrix} \tag{9}$$

up to some permutation of rows.

We then define $\gamma(s)$ to be the ratio of the s^{th} singular value of $G^{(i)}$ to $L^{(i)}$:

$$\gamma^{(i)}(s) = \frac{\sigma_s(G^{(i)})}{\sigma_s(L^{(i)})} \quad \text{and} \quad \gamma(s) = \max_i \gamma^{(i)}(s), \tag{10}$$

i.e. γ is the maximum of $\gamma^{(i)}$ over all levels and nodes. Note that $\gamma \leq 1$ by the interlacing property of singular values [8]. With this observation, we have

$$\sigma_s(G^{(i)}) \leq \gamma(s)\sigma_s(L^{(i)}) \leq \gamma(s)\sigma_s(K). \tag{11}$$

ASKIT error bound. Using this definition together with a bound on the error due to combining IDs in the tree (Theorem 3, proved in §6), we can state our new result on the error in ASKIT.

THEOREM 2. *Let γ be defined as in (10). Compute an approximation with* ASKIT *using $\kappa = 1$. Then:*

$$\boxed{\epsilon_A \leq \{c_{samp} + c_{id}\} \log(N/m)\gamma(s+1)\sigma_{s+1}(K)\|w\|} \tag{12}$$

with the sampling error and ID error terms defined as:

$$c_{samp} = 2 + \left(1 + 6\frac{N-m}{\ell}\right)^{\frac{1}{2}} \tag{13}$$

$$c_{id} = (1 + ms(m-s))^{\frac{1}{2}} + \log(N/m)\left(1+2s^3\right)^{\frac{1}{2}}. \tag{14}$$

Here m is the number of points per leaf, s the approximation rank (fixed), and ℓ the far field sample size.

We provide the proof in §6. The term c_{samp} is from sampling targets uniformly at random, c_{id} is the error of an interpolative decomposition, and the factor $\log(N/m)$ is from the accumulation of this error up the levels of the tree.

This bound in practice. We observe errors better than predicted by this bound. The sampling term c_{samp} is for the case of uniform sampling. We use samples based on nearest neighbor distances. In [20], we show that this sampling can approximate well the optimal sampling based on leverage scores.

The term c_{id} is also a worst-case bound. While matrices exist for which (7) is not loose, empirical work on kernel matrices shows that the ID is nearly as accurate as the SVD in practice [20].

Therefore, we observe in practice that the key term is $\log(N/m)\gamma(s+1)\sigma_{s+1}(K)$—the decay of the spectrum of the entire matrix combined with the decay of off-diagonal blocks.

Effect of increasing κ. In practice, we generally set the number of neighbors in ASKIT to be larger than one. In terms of the error bounds discussed above, this has several effects. First, we use nearest neighbor information to bias our sampling distribution. While it is difficult to bound the quality of this sampling *a priori* without knowing the distribution of nearest neighbors, we do see that this method is more effective in practice. This has the effect of sharply reducing the term c_{samp} in (12).

Second, increasing κ means that we prune fewer nodes and perform more direct evaluations. We can alter our definition of γ to account for this. For a fixed set of columns, the on-diagonal block $K^{(i)}$ corresponds to the set of all rows which compute direct evaluations with these columns. Similarly, $G^{(i)}$ will consist of fewer rows, which correspond to points which are even farther from these columns. Therefore, the term γ decreases as we increase κ.

We measure the behavior of γ and $\sigma_s(G^{(i)})/\sigma_s(K^{(0)})$ in Table 1 for two values of κ. We see that for the selected subset of the COVTYPE set, γ is already significantly less than one. Also, we see that increasing κ dramatically decreases γ.

Tradeoff between Nystrom and ASKIT. We make several simplifying assumptions in keeping with our use of these methods in practice. We set $m = s$ and $\ell = 2s$. We assume N is large, so that small additive constants can be neglected. Further, we assume that the error due to the ID is some small constant ($c_{\text{id}} \approx 2$). For comparison, we set the approximation ranks to be equal ($r = s$).

In this case, we have that $\epsilon_N = \sqrt{3N/s}\,\sigma_{s+1}(K)$ and that

$$\epsilon_A = \left(\sqrt{3N/s} + c_{\text{id}}\right)\log(N/s)\gamma(s+1)\sigma_{s+1}(K).$$

When comparing the predicted error of the two methods, we consider the ratio

$$\frac{\epsilon_A}{\epsilon_N} \approx \left(\log\left(\frac{N}{s}\right) + c_{\text{id}}\log\left(\frac{N}{s}\right)\sqrt{\frac{s}{N}}\right)\gamma(s+1). \tag{15}$$

We see that when γ is small compared to $\log^{-1}(N/s)$, we expect better error for ASKIT than that for Nystrom for a fixed approximation rank $s = r$. Or, from another perspective, to achieve the same error with both methods will require r to be much larger than s when γ is small, with a resulting increase in the complexity of the Nystrom method.

As the kernel bandwidth h decreases, γ will decrease as well, since K will become larger on its diagonal. In the next section, we explore the behavior of Nystrom and ASKIT on kernel matrices. We show that our predicted error behavior holds in practice. In addition, we show that there are significant (in the sense of being applicable to real learning tasks) regimes of h for which γ is very small and thus for which ASKIT outperforms Nystrom.

Level	$\gamma^{(i)}(s)$ (10)	$\frac{\sigma_s(G^{(i)})}{\sigma_s(K^{(0)})}$	s
$\kappa = 1$			
4	0.3194	0.0252	344
5	0.4645	0.0129	270
6	0.6040	0.0058	228
7	0.7719	0.0038	160
$\kappa = 64$			
4	0.0598	0.0057	336
5	0.2908	0.0074	250
6	0.3994	0.0065	210
7	0.4976	0.0022	176

Table 1: *Experimental measures of γ for the **COVTYPE** data set. We subsample 32,000 points and set $h = 0.22$. We construct a tree on these points, and perform the adaptive rank ASKIT algorithm with $\tau = 1E\text{-}3$. We then empirically measure the maximum of $\gamma^{(i)}$ and $\sigma_s(G^{(i)})/\sigma_s(K^{(0)})$ over the indicated level for the value of s chosen by τ. The first set of results is for $\kappa = 1$ and the second is for $\kappa = 64$.*

Set	N	d
COVTYPE	$500,000$	54
SUSY18D	4.5×10^6	18
SUSY8D	4.5×10^6	8
MNIST2M	2×10^6	784

Table 3: *The sizes of the data sets used in our experiments. N is the training set size and d is the dimension. The SUSY8D set is obtained from the full SUSY set by taking the first 8 features.*

4. EMPIRICAL RESULTS

We now turn to numerical examples which illustrate the robustness of ASKIT with respect to bandwidth for three relatively large datasets, described in Table 3.[3] Using these datasets, we discuss two key questions: How do the approximation accuracies of ASKIT and the two Nystrom methods vary with h? Further, how does this matrix approximation affect the classification results obtained in cross-validation studies?

Setup: All tests took place on the Maverick and Stampede clusters at the Texas Advanced Computing Center. Maverick nodes have two Intel Xeon E5-2680 v2 (2.8GHz) processors and 256GB RAM. The Stampede nodes have two Intel Xeon E5-2680 (2.7GHz) processors and 32GB RAM. All tests were done in double-precision arithmetic.

ASKIT is written in `C++`. It uses OpenMP to parallelize across cores, the Intel MKL for linear algebra routines, custom highly-optimized kernel evaluations, and MPI for distributed memory parallelism [22]. The skeletonization phase for ASKIT is parallelized but it is harder to get high performance because it requires pivoted QR factorization. The evaluation phase is highly optimized and achieves near peak performance. The nearest neighbors required for ASKIT have been precomputed and stored. We must perform the skeletonization step in ASKIT for each new bandwidth h. However, multiple evaluations (for different regularization values or in our iterative solver), only require us to store the matrix P from (4). Given this, we can update the skeleton weights and perform the new evaluation step extremely quickly. All experiments on one node (no MPI) are labeled as ASKIT. Multi-node runs are labeled as P-ASKIT.

Nystrom is our own MATLAB implementation. MEKA is also written in MATLAB.[4] All expensive calculations for Nystrom and MEKA use calls to highly optimized and multithreaded vendor LAPACK libraries. Therefore, comparing the one node timings across methods is justified.[5]

	#	Param	$h = 0.05$			$h = 0.15$		
			ϵ_2	T	T_E	ϵ_2	T	T_E
Nystrom	13	$r = 1024$	>9E-1	84	<1	7E-1	83	<1
	14	$r = 2048$	>9E-1	218	<1	5E-1	497	<1
	15	$r = 4096$	mem	–	–	mem	–	–
MEKA	16	$r = 1024$	>9E-1	207	<1	>9E-1	201	<1
	17	$r = 2048$	>9E-1	425	<1	7E-1	432	<1
	18	$r = 4096$	>9E-1	944	<1	5E-1	960	<1
	19	$r = 8192$	>9E-1	2353	<1	4E-1	2367	<1
ASKIT	20	$\tau = 1E\text{-}1$	2E-3	7792	193	3E-1	1209	57
	21	$\tau = 5E\text{-}1$	3E-3	885	67	4E-1	731	41
	22	$\kappa = 2048$	1E-4	21K	274	2E-1	4757	101
P-ASKIT	23	$p = 32$ (M)	1E-4	1117	12	8E-2	1033	12
	24	$p = 256$ (S)	9E-5	455	4	5E-2	449	4

Table 4: *Results for **SUSY 18D** data and different values of h. The column "#" labels rows for reference in the text and "**Param**" gives the parameter varied for each algorithm. We report the error ϵ_2 (see Eqn. 17), the total time T and the evaluation time T_E. The entry "mem" indicates that our experiment ran out of memory. For MEKA, we fix the number of clusters $C = 10$. For ASKIT, we use $\kappa = 512$, $s_{max} = 2048$, $m = 2048$, and $L = 5$. Run 22 uses $\tau = 1E\text{-}1$. P-ASKIT refers to ASKIT in parallel with $\kappa = 2048$ and $\tau = 1E\text{-}5$. The parameter p refers to the number of compute nodes on (M)averick or (S)tampede.*

Bandwidth selection. The selection of the bandwidth was guided by a simple Bayes classifier that uses direct kernel evaluations. For a given test point x_i, we assign the label $y_i = \text{sign}\left(\frac{1}{N_1}\sum_j K(x_i, x_j) - \frac{1}{N_{-1}}\sum_j K(x_i, x_j)\right)$ where the summations run over sets of training points with labels $+1$ and -1 (of sizes N_1 and N_{-1}, respectively). For the COVTYPE set, we perform "one-vs-rest" classification. We evaluate the model on the test data using exact kernel evaluations and report the classification accuracy

$$\alpha_c = \frac{\sum_{i=1}^{n} \mathbf{1}\left(\tilde{y}_i = y_i\right)}{n} \tag{16}$$

where $\mathbf{1}$ is an indicator function. The results are reported in §1. For the COVTYPE and MNIST2M tests, some of the wider bandwidths coincide with the values reported in [26].

Parameter selection. For the basic Nystrom method, we explore a range of values of r up to the maximum allowed

[3] The URLs of the three datasets are SUSY and COVTYPE from UCI [2] and MNIST from [4]. Lower dimensions and different point counts were created by processing the original sets. For all datasets, all features were normalized to be in the range $[0, 1]$.

[4] http://www.cs.utexas.edu/~ssi/meka/

[5] Overall MEKA produces similar accuracy to Nystrom. The timings for MEKA are a bit pessimistic because we used a tight threshold for zeroing off-diagonal blocks and a large number of clusters since we wanted as much accuracy as possible. We experimented with the number of clusters and often MEKA can run much faster with very slight deterioration of the error. The same is the case for ASKIT, in which we can adjust the tolerance to get significantly faster for the same accuracy. We have not performed any such tuning and we report in general the most accurate results for both MEKA and ASKIT.

	#	Params	$h = 0.02$			$h = 0.16$			$h = 0.22$			$h = 0.35$		
			ϵ_2	T	T_E	ϵ_2	T	T_E	ϵ_2	T	T_E	ϵ_2	T	T_E
Nystrom	1	$r = 1024$	>9E-1	14	<1	4E-1	10	<1	2E-1	20	<1	4E-2	21	<1
	2	$r = 2048$	>9E-1	42	<1	3E-1	24	<1	9E-2	56	<1	2E-2	50	<1
	3	$r = 4096$	>9E-1	151	<1	2E-1	77	<1	3E-2	225	<1	8E-3	229	<1
	4	$r = 8192$	>9E-1	612	<1	1E-1	311	<1	2E-2	1223	<1	5E-3	1221	<1
	5	$r = 16384$	>9E-1	2875	<1	5E-2	1523	<1	1E-2	1539	<1	4E-3	1534	<1
MEKA	6	$r = 1024$	>9E-1	44	<1	4E-1	35	<1	2E-1	35	<1	6E-2	34	<1
	7	$r = 2048$	>9E-1	118	<1	2E-1	74	<1	9E-2	73	<1	3E-2	74	<1
	8	$r = 8192$	>9E-1	1514	<1	6E-2	696	<1	2E-2	698	<1	4E-3	701	<1
	9	$r = 16384$	>9E-1	7209	<1	3E-2	3828	<1	9E-3	3817	<1	2E-3	3806	<1
ASKIT	10	$\tau =$ 1E-1	9E-13	254	12	7E-2	238	7	9E-2	237	7	1E-1	239	6
	11	$\tau =$ 1E-5	9E-13	831	20	2E-3	1043	21	2E-3	668	19	4E-3	336	14
	12	$\tau =$ 1E-12	9E-13	1197	21	2E-3	1689	22	1E-3	1686	22	2E-3	1814	22

Table 2: *Results for **COVTYPE** data and different values of h. The column "#" labels rows for reference in the text and "**Params**" gives the parameter varied for each algorithm. We report the error ϵ_2 (see Eqn. 17), the total time T and the evaluation time T_E. For* MEKA*, we fix the number of clusters $\mathcal{C} = 20$. For* ASKIT*, we use $\kappa = 2048$, $s_{max} = 2048$, $m = 2048$, and $L = 5$.*

by our memory resources. For MEKA, we also vary r and $\mathcal{C}$. We set the percentage of off-diagonal blocks $\eta = 0.05$. For ASKIT, we vary the number of neighbors and the approximation tolerance as given in the tables.

	#	Param	$h = 0.5$			$h = 1$		
			ϵ_2	T	T_E	ϵ_2	T	T_E
Nystrom	25	$r = 1024$	>9E-1	63	<1	>9E-1	63	<1
	26	$r = 2048$	>9E-1	122	<1	>9E-1	120	<1
	27	$r = 4096$	>9E-1	299	<1	>9E-1	301	<1
	28	$r = 8192$	mem	–	–	mem	–	–
MEKA	29	$r = 2048$	>9E-1	471	<1	>9E-1	750	<1
	30	$r = 8192$	>9E-1	1570	<1	>9E-1	1581	<1
ASKIT	31	$\kappa = 256$	9E-5	3139	1418	3E-2	2585	1330
P-ASKIT	32	$\kappa = 256$	1E-4	226	32	3E-2	154	31
	33	$\kappa = 512$	3E-5	243	39	2E-2	181	38
	34	$\kappa = 1024$	5E-6	306	50	2E-2	239	47
	35	$\kappa = 2048$	9E-7	410	65	8E-3	370	62

Table 5: *Results for **MNIST** data and different values of h. The column "#" labels rows for reference in the text and "**Param**" gives the parameter varied for each algorithm. We report the error ϵ_2 (see Eqn. 17), the total time T and the evaluation time T_E. The entry "**mem**" indicates that our experiment ran out of memory. For MEKA, we fix the number of clusters $\mathcal{C} = 10$. For ASKIT, we use $s_{max} = 2048$, $m = 2048$, $L = 5$, and $\tau =$ 1E-1. P-ASKIT uses $p = 32$ nodes of Maverick.*

Matrix error experiments. For a given vector of charges w, we compute the approximate matrix vector product $\tilde{u} = \tilde{K}w$, where $\tilde{K}$ depends on the approximation method used. We are interested in the relative error of the approximate product:

$$\epsilon_2 = \|Kw - \tilde{K}w\|/\|Kw\|. \quad (17)$$

Since the exact product Kw is prohibitively expensive to compute, we instead sample 1,000 entries of the vectors $u = Kw$ and $\tilde{u} = \tilde{K}w$. For each dataset, we draw entries of w independently from a standard normal distribution. We average results for ϵ_2 over 10 independent samples of w and choose the same set of test points in all of our experiments. For each dataset, we use identical error evaluation points for all three methods. We report results in Tables 2, 4, and 5.

Classification experiments. For selected bandwidths, we also perform a simple classification experiment. We train a linear classifier in kernel space; we use ridge regression,

#	Method	h	λ	Param	ϵ_2	α_c
36	ASKIT	0.07	0.01	$L = 4$	2E-3	96.7%
37	Nystrom	0.07	0.01	$r = 16384$	5E-1	89.3%
38	MEKA	0.07	0.01	$r = 16384$	5E-1	90.7%
39	ASKIT	0.22	1.43	$L = 4$	9E-2	90%
40	Nystrom	0.22	1.43	$r = 8192$	2E-2	90%
41	MEKA	0.22	1.43	$r = 8192$	3E-2	87.8%

Table 6: *Regression results for **COVTYPE** data. The column "#" labels rows for reference in the text, h is the kernel bandwidth, and λ is the regularization parameter, and "**Param**" gives the parameter varied for each algorithm. We report the error of our kernel approximation ϵ_2 (Eqn. 17) and the classification accuracy α_c (Eqn. 16). ASKIT uses $m = 2048$, $\kappa = 2048$, $s_{max} = 2048$, and $\tau =$ 1E-7. MEKA uses $\mathcal{C} = 10$ clusters. While separate cross validation for each method will result in slightly different values of λ, we fix one λ for comparison.*

#	Method	h	λ	Param	ϵ_2	α_c
42	ASKIT	0.07	10	$L = 4$	6E-2	77.2%
43	Nystrom	0.07	0.005	$r = 2048$	8E-1	67.0%
44	MEKA	0.07	0.005	$r = 2048$	>9E-1	53.7%
45	ASKIT	0.15	181	$L = 4$	3E-2	76.8%
46	Nystrom	0.15	181	$r = 2048$	1E-1	74.1%
47	MEKA	0.15	181	$r = 2048$	2E-1	75.1%

Table 7: *Regression results for **SUSY 8D** data. The column "#" labels rows for reference in the text, h is the kernel bandwidth, and λ is the regularization parameter, and "**Param**" gives the parameter varied for each algorithm. We report the error of our kernel approximation ϵ_2 (Eqn. 17) and the classification accuracy α_c (Eqn. 16). ASKIT uses $m = 2048$, $\kappa = 2048$, $s_{max} = 2048$, and $\tau =$ 1E-3. MEKA uses $\mathcal{C} = 10$ clusters. While separate cross validation for each method will result in slightly different values of λ, we fix one λ for comparison.*

then choose the sign of the prediction as our estimated class. This is equivalent to solving the linear system $(\tilde{K}+\lambda I)w = y$ with the approximate kernel matrix $\tilde{K}$ constructed from the training data, y the vector of training labels (± 1), and a regularization parameter λ.

For Nystrom and MEKA, we can simply invert the approximate representation to solve for the regression weights.

We performed cross-validation to choose the regularization parameter using an estimate of the spectrum of K to bound the range of potential values of λ.

For ASKIT, we solve the linear system iteratively using a Krylov method (GMRES) using the PETSc library [3]. We choose the regularization values with guidance from the Nystrom methods plus trial and error, since treecodes do not produce a global factorization.

For all methods, once we find the regression weights, we use them to directly evaluate $y_t = Kw$ at selected test points (not used in the training set). We output the label as $\text{sign}(y_t)$. The classification error from this method is given in (16). We report results in Tables 6 and 7.

4.1 Discussion

In the following, we use the run ID (denoted by #) to identify the row of the experiment we refer to. We discuss the results separately for each dataset.

COVTYPE: In Table 2, we notice that all three methods give good accuracy for $h = 0.22$ and $h = 0.35$. The convergence slows down for Nystrom methods for $h = 0.16$: Nystrom in #4 gives 10% error in 311 seconds, whereas ASKIT in #10 gives 7% error in 238 seconds. Nystrom in #5 gives 5% error in 1523 seconds whereas ASKIT in #11 gives 0.2% error, an order of magnitude better, in 1043 seconds. Further doubling the rank of Nystrom is impractical since the time increases eightfold. Notice that ASKIT uses much smaller off-diagonal block ranks (2,048 vs 16,384), yet is more accurate. For ASKIT the error stagnates at 0.2%. Further decreasing it would require increasing $s_{\max}$ and κ. For the smallest bandwidth $h = 0.02$, which was the best value for the Bayes classifier, K essentially has only nearest neighbor interactions, and ASKIT delivers 12 digits of accuracy. Of course, Nystrom methods cannot be used for such narrow bandwidths.

More interestingly, consider the ridge regression results in Table 6. For $h = 0.22$ and the same approximation accuracy and regularization λ, the three methods perform similarly.[6] However, the value $h = 0.07$ gives better results—provided we can approximate K accurately. ASKIT delivers 0.2% error whereas both Nystrom methods have over 50% error and thus erroneously report poorer classification.

SUSY: We check the approximation error for two bandwidths, $h = 0.05$ and $h = 0.15$. This is a much harder dataset than COVTYPE despite its lower dimensionality. For ASKIT the memory requirements are also significant. Just storing the neighbor information requires over 100GB.[7]

We would like to remark that both bandwidths we check are not narrow. If we just use the nearest neighbor solution from ASKIT (we tested both 512 and 2048 neighbors per point), we get huge errors (greater than 100%). For $h = 0.15$, both Nystrom methods deliver 40% error at best (#19) whereas ASKIT does a little better with 20% (#22). Increasing the ranks for Nystrom exhausts the memory. To reduce the runtime for ASKIT, we run the remaining steps of convergence on multiple nodes. We can push the error to 5% on 256 nodes on Stampede (#24). For $h = 0.05$ we are able to reach 0.2% accuracy on a single node (#20) and four digits of accuracy with parallel ASKIT (#24). Both Nystrom methods give over 90% error (#14, 19). Looking at the ridge regression results (in which we run the 8D dataset to have a lower memory footprint) for $h = 0.07$ (#43, 44) we observe that the two Nystrom methods produce highly inaccurate classification results due to their large approximation error, whereas for $h = 0.15$ they perform well (#46, 47). ASKIT performs well in both cases (#42, 45).

MNIST: For this dataset we only run the matrix approximation test. For both the bandwidths we test ($h = 0.5$ and $h = 1$), Nystrom methods cannot produce any reliable approximation and run out of memory whereas ASKIT converges in a satisfactory way.

Finally, one could question whether ASKIT uses too many direct evaluations—perhaps close to computing Kw exactly. The number of evaluations is significantly higher than than in Nystrom methods but nowhere near an exact evaluation. In Table 8, we report the average number of kernel evaluations per point (during the evaluation phase) as a percentage of the number needed for exact computation.

Run	10	11	24	24	32
h	0.16	0.35	0.05	0.15	0.5
ϵ_2	7E-2	4E-3	5E-2	9E-5	1E-4
$\%K$	6.7%	13%	8.1%	8.0%	6.7%

Table 8: ASKIT kernel evaluations. *We show the number of total kernel interactions performed by ASKIT for selected runs from Tables 2, 4, and 5. "**Run**" refers to rows of the other tables, $\boldsymbol{h}$ is the kernel bandwidth, $\boldsymbol{\epsilon_2}$ is the error, and $\boldsymbol{\%K}$ is the percentage of kernel interactions computed out of the number needed for exact evaluation.*

5. CONCLUSIONS

Nystrom methods construct a global low-rank approximation. Treecodes construct hierarchical, low-rank approximations of off-diagonal blocks. Our main point is that the latter methods are more robust as they can approximate kernels in both narrow and wide bandwidth regimes. ASKIT is not always the fastest choice and is not as general since it requires nearest-neighbor information (which is also expensive to compute and store). Nystrom methods have many advantages: they are simple to implement and can be easily factorized so that both the forward and inverse kernel matrices can be applied quickly. But, we find that the performance of Nystrom methods is sensitive to the data and the choice of kernel bandwidth. As the bandwidth decreases (even slightly), Nystrom methods can incur severe error. Although the optimal bandwidth for a learning application may lie in the regime where Nystrom methods are effective, we point out that cross-validation still requires computations on a range of bandwidths, including values for which the errors of Nystrom methods are prohibitive.

For small datasets, it is unlikely that anything but a very wide bandwidth would produce good classification, and Nystrom methods should be the method of choice. But as we consider datasets with millions and billions of points, we cannot expect that the kernel matrix has a global low rank structure. ASKIT combines geometric information of the dataset with algebraic approximation methods to provide a kernel-independent hierarchical scheme that works well across all bandwidths. We used two elementary kernel classification schemes to demonstrate this observation. Ongoing work includes extending ASKIT to support different sources and targets and coupling it to more sophisticated classifiers.

[6] $h = 0.22$ was reported as optimal for ridge regression in [26].

[7] The timings for $h = 0.05$ (#22 and #20) highlight the performance loss due to our unoptimized implementation of the skeletonization step.

6. APPENDIX

We now give the proof of Theorem 2. ASKIT consists of two phases: the skeletonization phase, where the low-rank factorizations are constructed, and the evaluation phase, where the approximate potentials are computed. The proof of the error bound follows these phases. In Theorem 3, we bound the error in the matrix approximations due to combining IDs up the tree in the skeletonization phase, resulting in the log factor in c_{id}. Then, in Theorem 4, we prove a bound on the additional error incurred due to subsampling these IDs. We then bound the error incurred in the evaluation phase as a sum over all the matrices we approximate. This sum also goes over all levels of the tree, resulting in the overall log factor.

THEOREM 3. *Let G be the off-diagonal block corresponding to any tree node. Hierarchically form an approximate ID from the skeletons of the children of G (as in* ASKIT*, but without sampling rows). Then:*

$$\begin{aligned}\|G-\tilde{G}\|_2 \leq & [1+ms(m-s)]^{\frac{1}{2}}\left(\sigma_{s+1}(\hat{G})\right) \\ & + \log\left(\frac{N}{m}\right)\left(1+2s^3\right)^{\frac{1}{2}}\left(\sigma_{s+1}(\bar{G})\right)\end{aligned} \tag{18}$$

where $\hat{G}$ is the leaf and $\bar{G}$ is the internal node with maximum $s+1^{st}$ singular value among descendants of G.

PROOF. Consider leaf node 1 and leaf node 2, which are siblings. We approximate the $(N-m)\times m$ matrices G_1 and G_2 with rank s interpolative decompositions $\tilde{G}_1$ and $\tilde{G}_2$, each of which have error relative to G_i given by (7).

Let G be the matrix with $2m$ columns and $N-2m$ rows formed by merging the points in each of the leaf nodes. We form an ID for G by combining the ID's of G_1 and G_2, then computing an ID of the resulting matrix. In other words, we compute an ID of the $(N-2m)\times 2s$ matrix $[\tilde{G}_1,\tilde{G}_2]$.

In this case, we can write

$$\|G-\tilde{G}\| \leq \|G-[\tilde{G}_1,\tilde{G}_2]\| + \|[\tilde{G}_1,\tilde{G}_2]-\tilde{G}\|. \tag{19}$$

Without loss of generality, assume that $\sigma_{s+1}(G_1)\geq\sigma_{s+1}(G_2)$ and let w be the unit vector for which $\|G-\tilde{G}\|$ is maximized. Then, the first term is maximized when w applies only to $\tilde{G}_1$. Therefore:

$$\begin{aligned}\|G-\tilde{G}\| \leq & [1+ms(m-s)]^{\frac{1}{2}}\,\sigma_{s+1}(G_1) \\ & + \left(1+2s^3\right)^{\frac{1}{2}}\sigma_{s+1}([\tilde{G}_1,\tilde{G}_2]).\end{aligned} \tag{20}$$

Continuing up the tree, we repeat this bound for each node, inserting the bounds for its children and repeatedly applying the triangle inequality. Then,

$$\begin{aligned}\|G-\tilde{G}\| \leq & [1+ms(m-s)]^{\frac{1}{2}}\,\sigma_{s+1}(G_i) \\ & + \left(1+2s^3\right)^{\frac{1}{2}}\left(\sum_j \sigma_{s+1}(\tilde{G}_j)\right)\end{aligned} \tag{21}$$

where G_i corresponds to the leaf node descendant of G with largest $s+1$ singular value and the summation is over all internal nodes which are descendants of G. The summation contains at most $\log(N/m)$ terms, and we can bound each term with the worst case over all internal nodes. Doing so gives us the bound. □

We now turn to a bound which combines the error due to the ID throughout the tree with the error from sampling. Applying this bound to the entire evaluation phase completes the proof of Theorem 2.

THEOREM 4. *Assume that at each node, the number of sampled rows satisfies the condition in Theorem 1 and that the result of the theorem holds. Then,*

$$\begin{aligned}\epsilon_A \leq & \left\{\left[2+\left(1+6\frac{N-m}{\ell}\right)^{\frac{1}{2}}\right]\right. \\ & \left.+\,(1+ms(m-s))^{\frac{1}{2}}+\log\left(\frac{N}{m}\right)\left(1+2s^3\right)^{\frac{1}{2}}\right\} \\ & \times\log(N/m)\max\sigma_{s+1}(G)\|w\|\end{aligned} \tag{22}$$

where the maximum is over all subblocks G.

PROOF. We begin by bounding the error of the approximation of any node G. Let G' be the submatrix of G obtained by sampling rows. Then,

$$\|G-\tilde{G}\| \leq \|G-G'_{(s)}\| + \|G'_{(s)}-\tilde{G}\| \tag{23}$$

where $G'_{(s)}$ is the best rank s approximation of G'.

For the first term, we combine Theorem 9.3 of [14] with Theorem 1 to obtain

$$\begin{aligned}\|G-\tilde{G}\| &\leq \sigma_{s+1}(G)+\|(I-\Pi)G\| \\ &\leq \sigma_{s+1}(G)+\left(1+6\frac{N-m}{\ell}\right)^{\frac{1}{2}}\sigma_{s+1}(G).\end{aligned}$$

For the second term, we again apply the triangle inequality $\|G'_{(s)}-\tilde{G}\|\leq\|G'_{(s)}-G'\|+\|G'-\tilde{G}\|$. Using (18) for the second term, we have

$$\begin{aligned}\|G'_{(s)}-\tilde{G}\| \leq & \sigma_{s+1}(G')+[1+ms(m-s)]^{\frac{1}{2}}\left(\sigma_{s+1}(\hat{G})\right) \\ & +\log\left(\frac{N}{m}\right)\left(1+2s^3\right)^{\frac{1}{2}}\left(\sigma_{s+1}(\bar{G})\right).\end{aligned}$$

Here we make use of the fact that since G' is a submatrix of G, then $\sigma_{s+1}(G')\leq\sigma_{s+1}(G)$. We can also bound the singular values of $\hat{G}$ and $\bar{G}$ by the value of the block G with overall maximum singular value.

Since $\kappa=1$, the approximation computed by ASKIT is

$$\tilde{K}w=\sum_{i=0}^{\mathcal{D}}\tilde{G}^{(i)}w^{(i)}+K^{(\mathcal{D})}w^{(\mathcal{D})} \tag{24}$$

where the summation runs over all levels of the tree and the matrices $G^{(i)}$ and $K^{(i)}$ are interpreted as direct sums over the subblocks corresponding to individual matrix nodes. Since the term $K^{(\mathcal{D})}w^{(\mathcal{D})}$ is exact (corresponding to direct evaluations at the leaves), we only need to bound the error of each term $\tilde{G}^{(i)}$.

The error of each term in the summation over levels is maximized in the event that the entire weight vector appears on the $\tilde{G}^{(i)}$ with the worst error. Since there are $\log(N/m)$ terms in this sum, the total error is bounded by

$$\begin{aligned}\|G-\tilde{G}\| \leq & \log(N/m)\,\{2\sigma_{s+1}(G) \\ & +(1+6(N-m)/\ell)^{\frac{1}{2}}\,\sigma_{s+1}(G) \\ & +[1+ms(m-s)]^{\frac{1}{2}}\left(\sigma_{s+1}(\hat{G})\right) \\ & +\log(N/m)\left(1+2s^3\right)^{\frac{1}{2}}\left(\sigma_{s+1}(\bar{G})\right)\Big\}.\end{aligned} \tag{25}$$

Replacing each singular value with the maximum over all nodes achieves the result. □

Acknowledgements: This material is based upon work supported by AFOSR grants FA9550-12-10484 and FA9550-11-10339; by NSF grants CCF-1337393, OCI-1029022; by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Applied Mathematics program under Award Numbers DE-SC0010518, DE-SC0009286, and DE- FG02-08ER2585; by NIH grant 10042242; and by the Technische Universität München - Institute for Advanced Study, funded by the German Excellence Initiative (and the European Union Seventh Framework Programme under grant agreement 291763). Any opinions, findings, and conclusions or recommendations expressed herein are those of the authors and do not necessarily reflect the views of the AFOSR or the NSF. Computing time on the Texas Advanced Computing Centers Stampede system was provided by an allocation from TACC and the NSF.

7. REFERENCES

[1] A. Andoni and P. Indyk, *Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions*, Comm. of the ACM, 51 (2008), p. 117.

[2] K. Bache and M. Lichman, *UCI machine learning repository*, 2013.

[3] S. Balay et al., *Efficient management of parallelism in object oriented numerical software libraries*, in Modern Software Tools in Scientific Computing, E. Arge, A. M. Bruaset, and H. P. Langtangen, eds., Birkhäuser Press, 1997, pp. 163–202.

[4] C.-C. Chang and C.-J. Lin, *LIBSVM: A library for support vector machines*, ACM Transactions on Intelligent Systems and Technology, 2 (2011), pp. 27:1–27:27.

[5] S. Dasgupta and Y. Freund, *Random projection trees and low dimensional manifolds*, in Proceedings of the 40th annual ACM symposium on Theory of computing, ACM, 2008, pp. 537–546.

[6] A. Deshpande and S. Vempala, *Adaptive sampling and fast low-rank matrix approximation*, in Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, Springer, 2006, pp. 292–303.

[7] A. Gittens and M. Mahoney, *Revisiting the Nystrom method for improved large-scale machine learning*, in Proceedings of the 30th International Conference on Machine Learning, 2013, pp. 567–575.

[8] G. H. Golub and C. H. V. Loan, *Matrix Computations*, Johns Hopkins, fourth ed., 1996.

[9] A. Gray and A. Moore, *N-body problems in statistical learning*, Advances in neural information processing systems, (2001), pp. 521–527.

[10] L. Greengard and J. Strain, *The fast Gauss transform*, SIAM Journal on Scientific and Statistical Computing, 12 (1991), pp. 79–94.

[11] Greengard, L., *Fast Algorithms For Classical Physics*, Science, 265 (1994), pp. 909–914.

[12] M. Griebel and D. Wissel, *Fast approximation of the discrete Gauss transform in higher dimensions*, Journal of Scientific Computing, 55 (2013), pp. 149–172.

[13] M. Gu and S. C. Eisenstat, *Efficient algorithms for computing a strong rank-revealing qr factorization*, SIAM Journal on Scientific Computing, 17 (1996), pp. 848–869.

[14] N. Halko, P. Martinsson, and J. Tropp, *Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions*, SIAM Review, 53 (2011), p. 217.

[15] S. Kumar, M. Mohri, and A. Talwalkar, *Sampling methods for the Nyström method*, Journal of Machine Learning Research, 13 (2012), pp. 981–1006.

[16] I. Lashuk et al., *A massively parallel adaptive fast multipole method on heterogeneous architectures*, Communications of the ACM, 55 (2012), pp. 101–109.

[17] D. Lee, A. Gray, and A. Moore, *Dual-tree fast gauss transforms*, Advances in Neural Information Processing Systems, 18 (2006), p. 747.

[18] M. Mahoney and P. Drineas, *Cur matrix decompositions for improved data analysis*, Proceedings of the National Academy of Sciences, 106 (2009), p. 697.

[19] M. W. Mahoney, *Randomized algorithms for matrices and data*, Foundations and Trends in Machine Learning, 3 (2011), pp. 123–224.

[20] W. B. March and G. Biros, *Far-field compression for fast kernel summation methods in high dimensions*, Submitted for publication, (2014), pp. 1–43. arxiv.org/abs/1409.2802v1.

[21] W. B. March, B. Xiao, and G. Biros, *ASKIT: Approximate skeletonization kernel-independent treecode in high dimensions*, SIAM Journal on Scientific Computing, 37 (2015), pp. A1089–A1110.

[22] W. B. March, B. Xiao, C. Yu, and G. Biros, *An algebraic parallel treecode in arbitrary dimensions*, 29th IEEE International Parallel and Distributed Processing Symposium, May 2015.

[23] P.-G. Martinsson, V. Rokhlin, and M. Tygert, *A randomized algorithm for the decomposition of matrices*, Applied and Computational Harmonic Analysis, 30 (2011), pp. 47–68.

[24] L. Moon et al., *Parallel algorithms for clustering and nearest neighbor search problems in high dimensions*, in 2011 ACM/IEEE conference on Supercomputing, Poster Session, Piscataway, NJ, USA, 2011, IEEE Press.

[25] V. I. Morariu, B. V. Srinivasan, V. C. Raykar, R. Duraiswami, and L. S. Davis, *Automatic online tuning for fast Gaussian summation.*, in NIPS, 2008, pp. 1113–1120.

[26] S. Si, C.-J. Hsieh, and I. Dhillon, *Memory efficient kernel approximation*, in Proceedings of The 31st International Conference on Machine Learning, 2014, pp. 701–709.

[27] L. Wasserman, *All of Statistics: A Concise Course in Statistical Inference*, Springer, 2004.

[28] C. Williams and M. Seeger, *Using the nyström method to speed up kernel machines*, in Proceedings of the 14th Annual Conference on Neural Information Processing Systems, 2001, pp. 682–688.

[29] C. Yang et al., *Improved fast gauss transform and efficient kernel density estimation*, in Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on, IEEE, 2003, pp. 664–671.