# Multiresolution Matrix Compression

**Nedelina Teneva**
Department of Computer Science
The University of Chicago
nteneva@uchicago.edu

**Pramod K Mudrakarta**
Department of Computer Science
The University of Chicago
pramodkm@uchicago.edu

**Risi Kondor**
Department of Computer Science
Department of Statistics
The University of Chicago
risi@uchicago.edu

## Abstract

Multiresolution Matrix Factorization (MMF) is a recently introduced method for finding multiscale structure and defining wavelets on graphs and matrices. MMF can also be used for matrix compression (sketching). However, the original MMF algorithm of (Kondor et al., 2014) scales with $n^3$ or worse (where $n$ is the number of rows/columns in the matrix to be factorized) making it infeasible for large scale problems. In this paper we describe pMMF, a fast parallel MMF algorithm, which can scale to $n$ in the range of millions. Our experimental results show that when used for matrix compression, pMMF often achieves much lower error than competing algorithms (especially on network data), yet for sparse matrices its running time scales close to linearly with $n$.

## 1 INTRODUCTION

The massive size of modern datasets often requires matrices arising in learning problems to be reduced in size. Distance matrices or Gram (kernel) matrices, in particular, are often a computational bottleneck, because they are large and dense. Assuming that the number of data points is $n$, the space complexity of these matrices is $n^2$, while the time complexity of the linear algebra operations involved in many learning algorithms, such as eigendecomposition, matrix inversion, or solving least squares problems, usually scales with $n^3$. Without efficient matrix compression (or, as they are sometimes called, sketching) methods, many popular machine learning algorithms are simply inapplicable to today's datasets, where $n$ is often on the order of millions.

The most classical method for compressing a symmetric matrix $A \in \mathbb{R}^{n\times n}$ is Principle Component Analysis (PCA), which projects $A$ onto the subspace spanned by its $k$ leading eigenvectors (Eckart and Young, 1936). PCA is optimal in the sense that the projection minimizes the reconstruction error of $A$ measured in terms of any of the three most common matrix norms: Frobenius, operator, or nuclear. The main drawback of PCA is its computational complexity — calculating the $k$ leading eigenvectors of $A$, unless $A$ is very sparse, costs $O(kn^2)$, which is often prohibitive.

To address this issue, in recent years randomized sketching methods have become very popular. These algorithms come in three main flavors (see (Woodruff, 2014) for a recent review):

1. **Dense projection methods** project $A$ to a random subspace spanned by a small number of dense random vectors and use Johnson–Lindenstrauss type arguments to show that this preserves most of the structure (Halko et al., 2011);
2. **Row/column sampling methods** (when $A$ is positive semi-definite, usually called Nyström methods) construct an approximation to $A$ from a small subset of its rows/columns sampled from a carefully chosen probability distribution (Smola and Schökopf, 2000; Fowlkes et al., 2004; Drineas and Mahoney, 2005; Deshpande et al., 2006; Kumar et al., 2009; Zhang and Kwok, 2010; Wang and Zhang, 2013; Gittens and Mahoney, 2013);
3. **Structured sparsity based methods** combine the advantages of (i) and (ii) by sampling a small number of rows/columns, but after applying a dense but efficiently computable transformation, such as an FFT (Ailon and Chazelle, 2010; Martinsson, 2008).

What all of the above randomized algorithms have in common is the underlying assumption that $A$ has low rank, or at least that it can be well modeled by a low rank surrogate. Recently, however, a number of authors have proposed *multiresolution structure* as an alternative to the low rank paradigm, especially in the context of graph/network data (Ravasz and Barabási, 2003; Coifman and Maggioni, 2006; Savas et al., 2011). Multiresolution Matrix Fac-

Appearing in Proceedings of the $19^{th}$ International Conference on Artificial Intelligence and Statistics (AISTATS) 2016, Cadiz, Spain. JMLR: W&CP volume 51. 

torization (MMF) applies these ideas to matrices directly (Kondor et al., 2014), and has been proposed as an alternative to 1–3 for matrix sketching. However, the actual algorithm described in (Kondor et al., 2014) for computing MMF factorizations has time complexity $O(n^3)$ (or an even higher power $n$ for higher order rotations), ruling out its use on large datasets.

The present paper describes the first practical MMF–based approach to matrix sketching, bypassing the limitations of (Kondor et al., 2014) by introducing a new, fast parallel MMF algorithm called pMMF. Our experiments show that

1. When $A$ is sparse, pMMF can easily scale to $n$ in the range of millions even on modest hardware.
2. For many matrices of interest, the wall clock running time of pMMF is close to linear in $n$.
3. The compression error of the resulting MMF-based sketching scheme is often much lower than that of the other sketching methods.

The code for pMMF has been made publicly available in the form of a C++ software library distributed under the GNU Public License v.3.0. (Kondor et al., 2015).

## 2 MATRIX COMPRESSION WITH MMF

In the following, $[n]$ will denote the set $\{1, 2, \ldots, n\}$. Given a matrix $A \in \mathbb{R}^{n\times n}$ and two (ordered) sets $S_1, S_2 \subseteq [n]$, $A_{S_1,S_2}$ will denote the $|S_1|\times|S_2|$ dimensional submatrix of $A$ cut out by the rows indexed by $S_1$ and the columns indexed by $S_2$. $\overline{S}$ will denote the complement of $S$, in $[n]$, i.e., $[n]\setminus S$. $B_1 \cup B_2 \cup \ldots \cup B_m = [n]$ denotes that the ordered sets $B_1, \ldots, B_m$ form a partition of $[n]$. $A_{:,i}$ or $[A]_{:,i}$ denotes the $i$'th column of $A$.

### 2.1 The MMF Sketching Model

The Multiresolution Matrix Factorization (MMF) of a symmetric matrix $A \in \mathbb{R}^{n\times n}$ is a multi-level factorization of the form

$$A \approx Q_1^\top \ldots Q_{L-1}^\top Q_L^\top H\, Q_L\, Q_{L-1} \ldots Q_1, \qquad (1)$$

where $Q_1, \ldots, Q_L$ is a sequence of carefully chosen orthogonal matrices (rotations) obeying the following constraints:

MMF1. Each $Q_\ell$ is chosen from some subclass $\mathcal{Q}$ of highly sparse orthogonal matrices. In the simplest case, $\mathcal{Q}$ is the class of **Givens rotations**, i.e., orthogonal matrices that only differ from the identity matrix in four matrix elements

$$[Q_\ell]_{i,i} = \cos\theta, \qquad [Q_\ell]_{i,j} = -\sin\theta,$$
$$[Q_\ell]_{j,i} = \sin\theta, \qquad [Q_\ell]_{j,j} = \cos\theta,$$

for some pair of indices $(i, j)$ and rotation angle $\theta$. Slightly more generally, $\mathcal{Q}$ can be the class of so-called **k–point rotations**, which rotate not just two, but $k$ coordinates, $(i_1, \ldots, i_k)$. Note that the Givens rotation case corresponds to $k=2$.

MMF2. The effective size of the rotations decreases according to a set schedule $n = \delta_0 \geq \delta_1 \geq \ldots \geq \delta_L$, i.e., there is a nested sequence of sets $[n] = S_0 \supseteq S_1 \supseteq \ldots \supseteq S_L$ with $|S_\ell| = \delta_\ell$ such that $[Q_\ell]_{\overline{S_{\ell-1}},\overline{S_{\ell-1}}}$ is the $n-\delta_{\ell-1}$ dimensional identity. $S_\ell$ is called the **active set** at level $\ell$. In the simplest case, exactly one row/column is removed from the active set after each rotation.

MMF3. $H$ is $\mathbf{S_L}$**–core-diagonal**, which means that all its entries are zero, except for (a) the submatrix $[H]_{S_L,S_L}$ called the core, and (b) the rest of the diagonal.

Moving the rotations in (1) over to the left hand side, the structure implied by the above conditions can be represented graphically as in Figure 1.

In general, MMF is only an approximate factorization, because there is no guarantee that using a set number of rotations, $A$ can be brought into perfectly core-diagonal form. Instead, MMF factorization algorithms try to find a combination of $Q_1, \ldots, Q_L$ and $H$ that minimize some notion of factorization error, in the simplest case, the squared Frobenius norm of the difference between the l.h.s and r.h.s of (1), called the **residual**.

The original motivation for introducing multiresolution matrix factorizations was to mimic the structure of fast orthogonal wavelet transforms. For example, when $A$ is the Laplacian matrix of a graph $\mathcal{G}$ and $U = Q_L \ldots Q_2 Q_1$, the rows of $U$ can be interpreted as wavelets on the vertices of $\mathcal{G}$. However, the sequence of transformations

$$A = A_0 \mapsto A_1 \mapsto A_2 \mapsto \ldots \mapsto A_L \mapsto H, \qquad (2)$$

where $A_\ell = Q_\ell \ldots, Q_1 A Q_1^\top \ldots Q_\ell^\top$, can also be seen as compressing $A$ to size $\delta_1\times\delta_1$, then $\delta_2\times\delta_2$, etc., all the way down to $\delta_L\times\delta_L$. To contrast the resulting MMF-based matrix sketching scheme with the randomized sketching methods described in the Introduction, note the following:

1. Since $U$ is dense, computing $A \mapsto H = UAU^\top$ in one shot would have a cost of $O(n^3)$, which is usually prohibitive. However, thanks to the extreme sparsity of the $Q_\ell$ rotations, doing the same by applying $Q_1, Q_2, \ldots, Q_L$ in sequence reduces the complexity to $O(n^2)$, or potentially even $O(n)$ if $A$ is sparse and has multiresolution structure (as some of our experiments confirm).
2. In contrast to other sketching methods, MMF sketching preserves not just the $\delta_L \times \delta_L$ dimensional core of $H$, but also the $n - \delta_L$ entries on the rest of its diagonal. However, the effective compression ratio is still $\delta_L : n$, since from the point of view of typical downstream computations, dealing with the diagonal entries is trivial. Two cases in point are computing $A^{-1}$ and

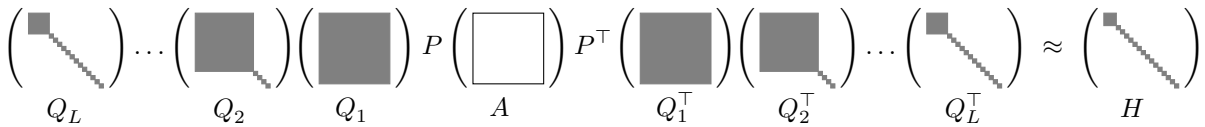


Figure 1: A graphical representation of the structure of Multiresolution Matrix Factorization. Here, $P$ is a permutation matrix which ensures that $S_\ell = \{1, \ldots, \delta_\ell\}$ for each $\ell$. Note that $P$ is introduced only for the sake of visualization. An actual MMF would not contain such an explicit permutation.

computing $\det(A)$, both of which appear in, e.g., Gaussian Process inference: for the former one merely needs to invert the entries on the diagonal, while for the latter one multiplies them together.

3. Nyström sketching and MMF sketching make fundamentally different assumptions about the nature of the matrix $A$. The former exploits low rank structure, which one might expect when, for example, $A$ is the Gram matrix of the linear kernel between data points lying near a low dimensional subspace, or when $A$ is some kind of preference matrix governed by a small number of underlying factors. In contrast (exactly because it preserves the diagonal entries of $H$), MMF does not need $A$ to be low rank, but rather exploits multiresolution structure, akin to a loose hierarchical "clusters of clusters" type organization, which has indeed been observed in large datasets. For other recent work on the connection between multiresolution analysis and hierarchical structures in graphs and matrices, see (Coifman and Maggioni, 2004; Lee et al., 2008; Shuman et al., 2013), and the structured matrix literature, e.g., (Chandrasekaran et al., 2005; Martinsson, 2008). For an application of the structured matrix idea to the learning setting, see (Wang et al., 2015).

### 2.2 The complexity of existing MMF algorithms

The main obstacle to using MMF for matrix sketching has been the cost of computing the factorization in the first place. The algorithms presented in (Kondor et al., 2014) construct the sequence of transformations (2) in a greedy way, at each level choosing $Q_\ell$ so as to allow eliminating a certain number of rows/columns from the active set while incurring the least possible contribution to the final error. Assuming the simplest case of each $Q_\ell$ being a Givens rotation, this involves (a) finding the pair of indices $(i, j)$ involved in $Q_\ell$, and (b) finding the rotation angle $\theta$. In general, the latter is easy. However, finding the optimal $(i, j)$ (or, in the case of $k$–point rotations, $(i_1, \ldots, i_k)$) is a combinatorial problem that scales poorly with $n$. Specifically,

1. The optimization is based on inner products between columns, so it requires computing the Gram matrix $G_\ell = A_{\ell-1}^\top A_{\ell-1}$ at a complexity of $O(n^3)$. Note that this needs to be done only once: once we have $G_1 = A^\top A$, each subsequent $G_\ell$ can be computed via the recursion $G_{\ell+1} = Q_\ell G_\ell Q_\ell^\top$.
2. In the simplest MMF algorithm of (Kondor et al., 2014), GREEDYJACOBI, $Q_\ell$ is chosen to be the $k$–point rotation which allows removing a single row/column from the active set with the least contribution to the final error. The complexity of finding the indices involved in $Q_\ell$ is $O(n^k)$. Given that there are $O(n)$ rotations in total, in the $k = 2$ case the cost of finding all of them is $O(n^3)$.
3. The drawback of GREEDYJACOBI is that it tends to lead to cascades, where a single row/column is repeatedly rotated against a series of others. This motivated proposing an alternative MMF algorithm, GREEDYPARALLEL, in which each $Q_\ell$ is the direct sum of $\lfloor |S_{\ell-1}| / k \rfloor$ separate non-overlapping $k$–point rotations. GREEDYPARALLEL avoids cascades, but finding $Q_\ell$ involves solving a non-trivial matching problem between the active rows/columns of $A_{\ell-1}$. In the $k = 2$ case, the matching can be done in $O(n^3)$ time by the so-called Blossom Algorithm (Edmonds, 1965). For $k > 2$ it becomes forbiddingly expensive, even for small $n$.

## 3 PARALLEL MMF (pMMF)

pMMF, our new MMF algorithm for large scale matrix compression, gets around the limitations of earlier MMF algorithms by employing a two level factorization strategy. The high level picture is that $A$ is factored in the form

$$A \approx \overline{Q}_1^\top \overline{Q}_2^\top \ldots \overline{Q}_P^\top H \, \overline{Q}_P \ldots \overline{Q}_2 \overline{Q}_1, \tag{3}$$

where each $\overline{Q}_p$, called the $p$'th **stage** of the factorization, is a compound rotation, which is block diagonal in the following generalized sense.

**Definition 1** *Let $M \in \mathbb{R}^{n \times n}$, and $B_1 \cup B_2 \cup \ldots \cup B_m$ be a partition of $[n]$. We define the $(u, v)$ **block** of $M$ (with respect to the partition $(B_1, \ldots, B_m)$) as the submatrix $[\![M]\!]_{u,v} := M_{B_u, B_v}$. We say that $M$ is $(B_1, \ldots, B_m)$–**block diagonal** if $[\![M]\!]_{u,v} = 0$ unless $u = v$.*

At the lower level, each stage $\overline{Q}_p$ itself factors in the form

$$\overline{Q}_p = Q_{\ell_p} \ldots Q_{\ell_{p-1}+2} Q_{\ell_{p-1}+1}$$

into a product of (typically, a large number of) elementary rotations obeying the same constraints MMF1 and MMF2 that we described in Section 2.1. Thus, expanding each stage, the overall form of (3) is identical to

(1), except for the additional constraint that the elementary rotations $Q_1, \ldots, Q_L$ are forced into contiguous runs $Q_{\ell_{p-1}+1}, \ldots, Q_{\ell_p}$ conforming to the same block structure.

The block diagonal structure of each $\overline{Q}_p$ naturally lends itself to parallelization. When applying an existing factorization to a vector $v$ (e.g., to compute $K^{-1}v$ in ridge regression or Gaussian process inference), if $v$ is blocked the same way, $[\![v]\!]_u$ need only be multiplied by $[\![\overline{Q}_p]\!]_{u,u}$, making it possible to distribute the computation over $m$ different processors.

However, what is more critical for our purposes is that it makes it possible to distribute computing the factorization in the first place. In fact, blocking accelarates the computation in three distinct ways:

1. The rotations constituting each diagonal block $[\![\overline{Q}_p]\!]_{u,u}$ can be computed completely independently on separate processors.
2. Instead of computing a single $n$ dimensional Gram matrix at a cost of $O(n^3)$, each stage requires $m$ separate $O(n/m)$ dimensional Gram matrices, which may be computed in parallel (assuming $m$–fold parallelism) in time $O(n^3/m^2)$.
3. The complexity of the per-rotation index search problem in the GREEDYJACOBI subroutine is reduced from $O(n^k)$ to $O((n/m)^k)$.

### 3.1 Blocking and reblocking

The key issue in pMMF is determining how to block each stage so that the block diagonal constraints introduced by (3) will have as little impact on the quality of the factorization as possible (measured in terms of the residual or some other notion of error). On the one hand, it is integral to the idea of multiresolution analysis that at each stage the transform must be *local*. In the case of MMF this manifests in the fact that a given row/column of $A$ will tend to only interact with other rows/columns that have high normalized inner product with it. This suggests blocking $A$ by clustering its rows/columns by normalized inner product. For speed, pMMF employs a randomized iterative clustering strategy with additional constraints on the cluster sizes to ensure that the resulting block structure is close to balanced.

On the other hand, enforcing a single block structure on all the stages would introduce artificial boundaries between the different parts of $A$ and prevent the algorithm from discovering the true multiresolution structure of the matrix. Therefore, pMMF reclusters the rows/columns of $A_\ell$ before each stage, and reblocks the matrix accordingly. It is critical that the reblocking process be done efficiently, maintaining parallelism, without ever having to push the entire matrix through a single processor (or a single machine, assuming a cluster architecture). $m$–fold parallelism is maintained by first applying the reblocking map row-wise in parallel for each column of blocks in the original matrix, then applying it column-wise in parallel for each row of blocks in the resulting matrix. One of the main reasons that we decided to implement bespoke dense/sparse blocked matrix classes in the pMMF software library was that we were not aware of any existing matrix library with this functionality. The top level pseudocode of pMMF is presented in Algorithm 1.

**Algorithm 1 pMMF** (top level of the pMMF algorithm)

**Input:** a symmetric matrix $A \in \mathbb{R}^{n \times n}$
**Set** $A_0 \leftarrow A$
**for** $(p = 1$ to $P)$ {
  **cluster** the active columns of $A_{p-1}$ to $(B_1^p, \ldots, B_m^p)$
  **reblock** $A_{p-1}$ according to $(B_1^p, \ldots, B_m^p)$
  **for** $(u = 1$ **to** $m)$ $[\![\overline{Q}_p]\!]_{u,u} \leftarrow$ **FindRotInCluster**$(p, u)$
  **for** $(u = 1$ **to** $m)$ {
    **for** $(v = 1$ **to** $m)$ {
      **set** $[\![A_p]\!]_{u,v} \leftarrow [\![\overline{Q}_p]\!]_{u,u} [\![A_{p-1}]\!]_{u,v} [\![\overline{Q}_p]\!]_{v,v}^\top$
}}}
$H \leftarrow$ the core of $A_L$ plus its diagonal
**Output:** $(H, \overline{Q}_1, \ldots, \overline{Q}_p)$

**Algorithm 2 FindRotInCluster**$(p, u)$ — assuming $k = 2$ and that $\eta$ is the compression ratio

**Input:** a matrix $\mathcal{A} = [A_p]_{:,B_u} \in \mathbb{R}^{n \times c}$ made up of the $c$ columns of $A_{p-1}$ forming cluster $u$ in $A_p$
**compute** the Gram matrix $G = \mathcal{A}^\top \mathcal{A}$
**set** $I = [c]$ (the active set)
**for** $(s = 1$ **to** $\lfloor \eta c \rfloor)$ {
  **select** $i \in I$ uniformly at random
  **find** $j = \text{argmax}_{I \setminus \{i\}} |\langle \mathcal{A}_{:,i}, \mathcal{A}_{:,j} \rangle| / \|\mathcal{A}_{:,j}\|$
  **find** the Givens rotation $q_s$ of columns $(i, j)$
  **set** $\mathcal{A} \leftarrow q_s \mathcal{A} q_s^\top$
  **set** $G \leftarrow q_s G q_s^\top$
  **if** $\|\mathcal{A}_{i,:}\|_{\text{off-diag}} < \|\mathcal{A}_{j,:}\|_{\text{off-diag}}$ **set** $I \leftarrow I \setminus \{i\}$
  **else set** $I \leftarrow I \setminus \{j\}$
}
**Output:** $[\![\overline{Q}_p]\!]_{u,u} = q_{\lfloor \eta c \rfloor} \cdots q_2 q_1$

### 3.2 Randomized Greedy Search for Rotations

Even with blocking, assuming that the characteristic cluster size is $c$, the $O(c^k)$ complexity of finding the indices involved in each rotation by the GREEDYJACOBI strategy is a bottleneck. To address this problem, pMMF uses randomization. First a single row/column $i_1$ is chosen from the active set (within a given cluster) uniformly at random, and then $k-1$ further rows/columns $i_2, \ldots, i_k$ are selected from the same cluster according to some separable objective function $\phi(i_2, \ldots, i_k)$ related to minimizing the con-

| | serial MMF | | pMMF operations | | pMMF time | | $N_{\text{proc}}$ |
|---|---|---|---|---|---|---|---|
| | dense | sparse | dense | sparse | dense | sparse | |
| Computing Gram matrices | $n^3$ | $\gamma n^3$ | $Pcn^2$ | $\gamma Pcn^2$ | $Pc^3$ | $\gamma Pc^3$ | $m^2$ |
| Finding Rotations | $n^3$ | $n^3$ | $cn$ | $cn$ | $c^2$ | $c$ | $m$ |
| Updating Gram matrices | $n^3$ | $\gamma^2 n^3$ | $c^2 n$ | $\gamma^2 c^2 n$ | $c^3$ | $\gamma^2 c^3$ | $m$ |
| Applying rotations | $kn^2$ | $\gamma kn^2$ | $kn^2$ | $\gamma kn^2$ | $kc^2$ | $\gamma kc^2$ | $m^2$ |
| Clustering | | | $pmn^2$ | $\gamma pmn^2$ | $pcn$ | $\gamma pcn$ | $m^2$ |
| Reblocking | | | $pn^2$ | $\gamma pn^2$ | $pcn$ | $\gamma pcn$ | $m$ |
| Factorization total | $n^3$ | $n^3$ | $Pcn^2$ | $\gamma Pcn^2$ | $Pc^3$ | $\gamma Pc^3$ | $m^2$ |

Table 1: The rough order of complexity of different subtasks in pMMF vs. the original serial MMF algorithm of (Kondor et al., 2014). Here $n$ is the dimensionality of the original matrix, $A$, $k$ is the order of the rotations, and $\gamma$ is the fraction of non-zero entries in $A$, when $A$ is sparse. We neglect that during the course of the computation $\gamma$ tends to increase, because concomitantly $A_\ell$ shrinks, and computation time is usually dominated by the first few stages. We also assume that entries of sparse matrices can be accessed in constant time. In pMMF, $P$ is the number of stages, $m$ is the number of clusters in each stage, and $c$ is the typical cluster size (thus, $c=\theta(n/m)$). The "pMMF time" columns give the time complexity of the algorithm assuming an architecture that affords $N_{\text{proc}}$–fold parallelism. It is assumed that $k \leq P \leq c \leq n$, but $n = o(c^2)$. Note that in the simplest case of Givens rotations, $k=2$.

tribution to the final error. For simplicity, in pMMF we use

$$\phi(i_2,\ldots,i_k) = \sum_{r=2}^{k} \frac{\langle [A_{\ell-1}]_{:,i_1}, [A_{\ell-1}]_{:,i_r}\rangle}{\| [A_{\ell-1}]_{:,i_r} \|},$$

i.e., $[A_{\ell-1}]_{:,i_1}$ is rotated with the $k-1$ other columns that it has the highest normalized inner product with in absolute value. Similarly to (Kondor et al., 2014), the actual rotation angle (or, in the case of $k$'th order rotations, the $k\times k$ rotation captured by $Q_\ell$) is determined by diagonalizing $[G_\ell]_{(i_1\ldots i_k),(i_1\ldots i_k)}$ at a cost of $O(k^3)$.

This aggressive randomized-greedy strategy reduces the complexity of finding each rotation to $O(c)$, and in our experience does almost as well as exhaustive search. The criterion for elimination is minimal off-diagonal norm, $\|A_{:,i}\|_{\text{off-diag}} = (\|A_{:,i}\|^2 - A_{i,i}^2)^{1/2}$, because $2\,\|A_{:,i}\|^2_{\text{off-diag}}$ is the contribution of eliminating row/column $i$ to the residual. As a by-product, randomization also eliminates the cascade problem of the GREEDYJACOBI algorithm, mentioned in Section 2.2.

### 3.3 Sparsity and Matrix Free MMF Arithmetic

pMMF can compress dense or sparse matrices. Naturally, the $Q_\ell$ elementary rotations are always stored in sparse form, in fact, for maximal efficiency, in the pMMF library they are implemented as separate, specialized objects.

However, when $n$ exceeds several thousand, even just storing the original matrix in memory becomes impossible unless it is sparse. Therefore, maintaining sparsity during the factorization process is critical. As the factorization progresses, due to the rotations, the fill-in (fraction of nonzeros) in $A_p$ will increase. Fortunately, at the same time, the active part of $A_\ell$ shrinks, and assuming multiresolution structure, these two factors balance each other out. In practice, we observe that for most large sparse datasets, the overall complexity of pMMF scales close to linearly with the number of non-zeros in $A$, both in space and in time.

Let us denote the complete factorization appearing on the r.h.s. of (1) by $\tilde{A}$. Even if the original matrix $A$ was sparse, in general, $\tilde{A}$ will not be, therefore computing it explicitly is unfeasible. However, downstream applications, e.g., iterative methods, almost never need $\tilde{A}$ itself, but only need the result of applying $\tilde{A}$ (or e.g., $\tilde{A}^{-1}$) as an operator to vectors. Therefore, we use a so-called matrix free approach: $v$ is stored in the same blocked form as $A$, the rotations are applied individually, and as the different stages are applied to $v$, the vector goes through an analogous reblocking process to that described for $A$. The complexity of matrix-free MMF/vector multiplication is $O(kPn)$.

For certain downstream computations, compressing matrices with pMMF can yield huge savings. For example,

$$\tilde{A}^{-1} \approx Q_1^\top \ldots Q_{L-1}^\top Q_L^\top H^{-1} Q_L Q_{L-1} \ldots Q_1,$$

but since the core of $H$ is only $\delta_L \ll n$ dimensional, $H^{-1}$ can be computed in $O(n+\delta_L^3)$ time, which is negligible compared to the $O(n^3)$ cost of inverting $A$ directly. To compute the matrix exponential, pMMF is used the same way.

The theoretical complexity of the main components of pMMF are summarized in Table 1. Of course, requiring $m^2$–fold parallelism as $m\to\infty$ is an abstraction. Note, however, that for sparse matrices, even the total operation count scales with $\gamma n^2$, which is just the number of nonzeros in $A$. The plots in Figure 2 confirm that on many real world datasets, particularly, matrices coming from sparse network graphs, the wall clock time of pMMF scales close to linearly with the dimension. Also note that while in these experiments $n$ is on the order of $10^5$, the factorization time

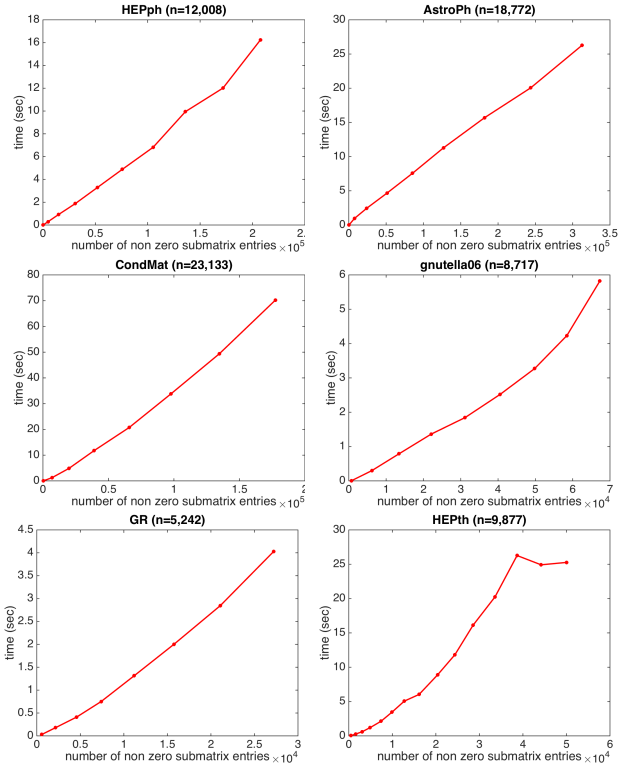


Figure 2: Execution time of pMMF as a function of the number of nonzero entries in the input matrix, $A$. For each of the graph Laplacians of size $n$, listed in Table 2, we take submatrices of varying sizes and compress each of them with pMMF to a $S_L$–core-diagonal matrix with core size of around 100. Each datapoint is averaged over five runs.

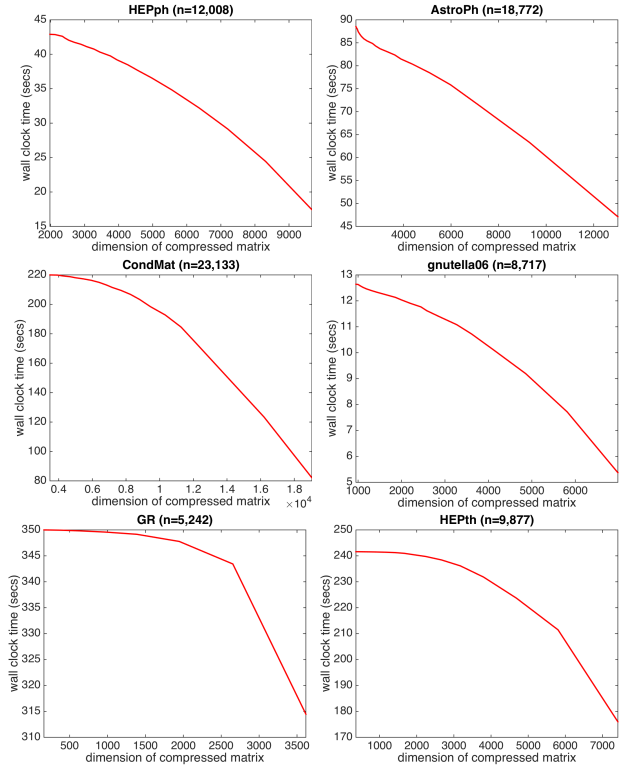


Figure 3: Execution time of pMMF as a function of the size of the compressed submatrix $[H]_{S_L,S_L}$ on the datasets listed in Table 2.

(on a 16-core cluster) is on the order of seconds.

## 4 IMPLEMENTATION

We implemented pMMF in C++11 with the goal of building a general purpose library that scales to large matrices. Critical to the implementation are the data structures used to store blocked vectors and matrices, which must support (a) Sparsity, (b) Block level parallelism, (c) Fast multiplication by Givens rotations and $k$–point rotations from both the left and the right, (d) Fast computation of inner products between columns (and of Gram matrices), (e) Fast reblocking, (f) Fast matrix/vector multiplication. Since we could not find any existing matrix library fulfilling all these requirements, we implemented the blocked matrix data structure from scratch directly on top of the `stl::vector`, `stl::list` and `stl::hash_map` containers.

The main parameters of pMMF are the order of the rotations, $k$, the number of stages, $P$, the target number of clusters per stage, $m$, and the compression ratio, $\eta$, which is the number of rotations to perform in a given cluster, as a function of the cluster size. In most of our experiments, a single row/column is removed from the active set after each rotation. For computational efficiency, we use a rough randomized clustering method, which selects $m$ "anchors columns" uniformly at random from the active set, and clusters the remaining columns to the anchor column with which is has highest normalized inner product. The clustering method has additional parameters intended to ensure that the clustering is approximately even. If the size of any of the clusters falls below $c_{\text{min}}$, then its columns are redistributed amongst the remaining clusters, whereas if the size of any of the clusters exceeds $c_{\text{max}}$, then its columns are recursively reclustered using the same algorithm. The maximum recursion depth is $D_{\text{max}}$. There is also a bypass flag, which, when set, signifies that rows/columns which could not be successfully clustered using the above method at a given stage will simply bypass the stage, with no rotations applied to them. Setting these parameters on a given system requires some experience, but overall our results appear stable w.r.t. the parameter values (including $P$, $m$ and $\eta$), as long as they are in a reasonable range.

## 5 NUMERICAL RESULTS

Figures 4 and 5 show the results of experiments comparing the performance of Multiresolution Matrix Compression to some of the most common matrix sketching algorithms: Nyström with uniform sampling (Williams and Seeger, 2001; Fowlkes et al., 2004), leverage score sampling (Ma-

| Dataset | Dimension ($n$) | Non-zero entries |
|---|---|---|
| **web-NotreDame**: web graph of the University of Notre Dame | $325,729$ | $1,497,134$ |
| **soc-Epinions1**: who-trusts-whom network of Epinions.com | $131,828$ | $841,372$ |
| **Gnutella31**: peer-to-peer network from August 31, 2002 | $62,586$ | $147,892$ |
| **Enron**: Enron email graph | $36,692$ | $367,662$ |
| **as-caida**: CAIDA AS Relationships Datasets | $31,379$ | $106,762$ |
| **CondMat**: ArXiv condensed matter collaboration graph | $23,133$ | $186,936$ |
| **AstroPh**: ArXiv astrophysics collaboration graph | $18,772$ | $396,160$ |
| **HEPph**: ArXiv high energy physics collaboration graph | $12,008$ | $237,010$ |
| **HEPth**: ArXiv high energy physics theory collaboration graph | $9,877$ | $51,971$ |
| **Gnutella06**: peer-to-peer network from August 6, 2002 | $8,717$ | $31,525$ |
| **GR**: ArXiv general relativity & quantum cosmology collaboration graph | $5,242$ | $28,980$ |
| **dexter**: bag of words dataset | $2,000$ | $4,00,000$ |
| **Abalone**: physical measurements of abalones, $\sigma = 0.15$ | $4,177$ | $17,447,329$ |

Table 2: Summary of the datasets used in the pMMF compression experiments (Gittens and Mahoney, 2013; Leskovec and Krevl, 2014; Davis and Hu, 2011).

Table 3: pMMF compression on large datasets. The Frobenius norm error $\mathcal{E}_{\text{Frob}} = \|A - \tilde{A}\|_{\text{Frob}}/\|A\|_{\text{Frob}}$, spectral norm error $\mathcal{E}_{sp} = \|A-\tilde{A}\|_2/\|A\|_2$, time (in secs), and the dimension of the core, $[H]_{S_L,S_L}$, that $A$ is compressed to.

| Dataset | Core size | Time(s) | $\mathcal{E}_{\text{Frob}}$ | $\mathcal{E}_{sp}$ |
|---|---|---|---|---|
| **web-NotreDame** | 1731 | 726.5 | 0.6 | 0.7 |
| **Enron** | 4431 | 530.2 | 0.6 | 0.3 |
| **Gnutella31** | 4207 | 112.2 | 0.8 | 0.6 |
| **as-caida** | 3404 | 91.9 | 0.7 | 0.6 |
| **soc-Epinions1** | 2089 | 1304.4 | 0.5 | 0.5 |

honey and Drineas, 2009; Mahoney, 2011; Gittens and Mahoney, 2013), dense Gaussian projections (Halko et al., 2011), and structured randomness with Fourier transforms (Tropp, 2011). In Figures 4 and 5 these methods are denoted respectively `unif`, `leverage`, `gaussian` and `srft`. These sketches approximate $A\in\mathbb{R}^{n\times n}$ in the form $\tilde{A}=CW^{\dagger}C^{\top}$, where $C$ is a judiciously chosen $\mathbb{R}^{n\times m}$ matrix with $m \ll n$, and $W^{\dagger}$ is the pseudo-inverse of a certain matrix that is of size only $m \times m$. This approximation is effectively a compression of $A$ down to $m$ rows/columns incurring error $\| A - \tilde{A} \|$.

The datasets we used are summarized in Table 2 and are some of the most commonly used in the matrix sketching literature. On most datasets that we tried, pMMF significantly outperforms the other sketching methods in both Frobenius norm error (Figure 4) and spectral norm error (Figure 5). The advantage of pMMF seems to be particularly great on network graphs, perhaps not surprisingly, since it has long been conjectured that networks have multiresolution structure (Ravasz and Barabási, 2003; Coifman and Maggioni, 2006; Savas et al., 2011). However, we find that pMMF often outperforms other methods on kernel matrices in general. On the other hand, on a small fraction of datasets, typically those which explicitly have low rank or are very close to being low rank (e.g., the fourth pane of Figure 4), pMMF performs much worse than expected. In such cases, a combination of the low rank and multiresolution approaches might be most advantageous, which is the subject of ongoing work.

pMMF is also dramatically faster than the other methods. This is only partially explained by the fact that several of the competing algorithms were implemented in MATLAB. For example, leverage score methods require estimating the singular vectors of $A$, which, unless $A$ is very low rank, can be a computational bottleneck. Several of the Nyström experiments took 30 minutes or more to run on 8 cores, whereas our custom C++ pMMF implementation compressed the matrix in at most one or two minutes (Figure 2). The results in Table 3 we couldn't even compare to other methods, because it would have taken too long for them to run.

## 6 CONCLUSIONS

The most common structural assumption about large matrices arising in learning problems is that they are low rank. This paper explores the alternative approach of assuming that they have multiresolution structure. Our results suggest that not only is the multiresolution model often more faithful to the actual structure of data (e.g., as evidenced by much lower approximation error in compression experiments), but it also lends itself to devising efficient parallel algorithms, which is critical to dealing with large scale problems. Our approach bears some similarities to multigrid methods (Brandt, 1973) and structured matrix decompositions (Hackbusch, 1999; Börm, 2009; Chandrasekaran et al., 2005), which are extremely popular in applied mathematics, primarily in the context of solving systems of partial differential equations. A crucial difference, however, is that whereas in these algorithms the multiresolution structure is suggested by the geometry of the domain, in learning problems the structure itself has to be learnt "on the fly".

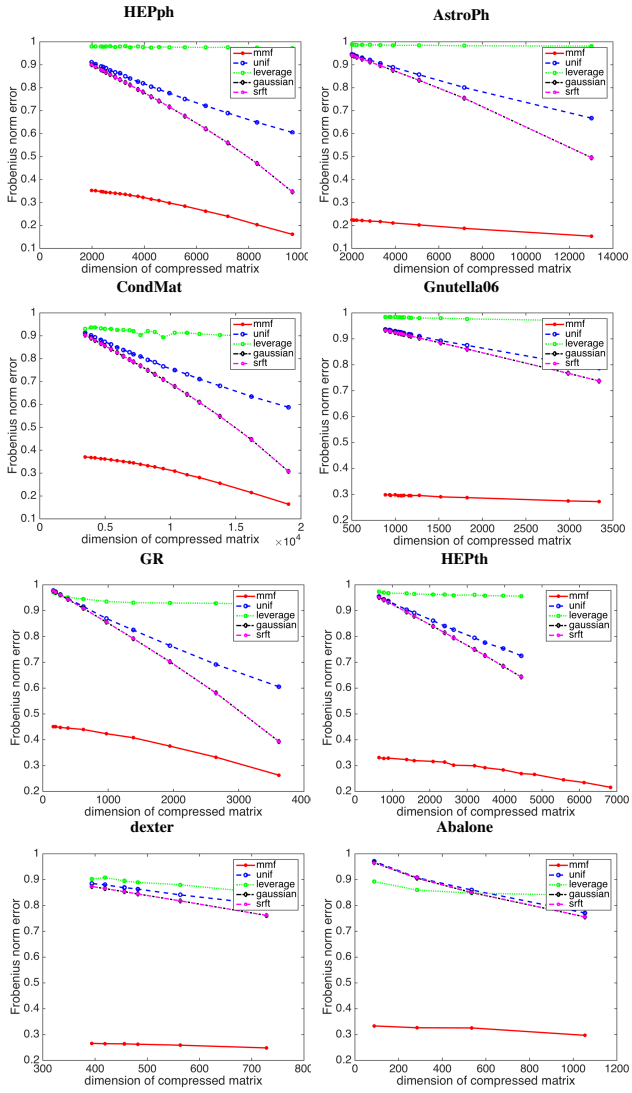


Figure 4: The Frobenius norm error $\|A - \tilde{A}\|_{\text{Frob}}$ of compressing matrices with pMMF vs. other sketching methods, as a function of the dimension of the compressed core. In each figure, the error is normalized by $\|A - A_k\|_{\text{Frob}}$, where $A_k$ is the best rank $k$ approximation to $A$. For the Nyström compression we followed the experiments in (Gittens and Mahoney, 2013) and used $k = 8$ for "dexter" , $k = 20$ for "Gnutella06" and "Abalone", and $k = 100$ for the rest of the datasets.

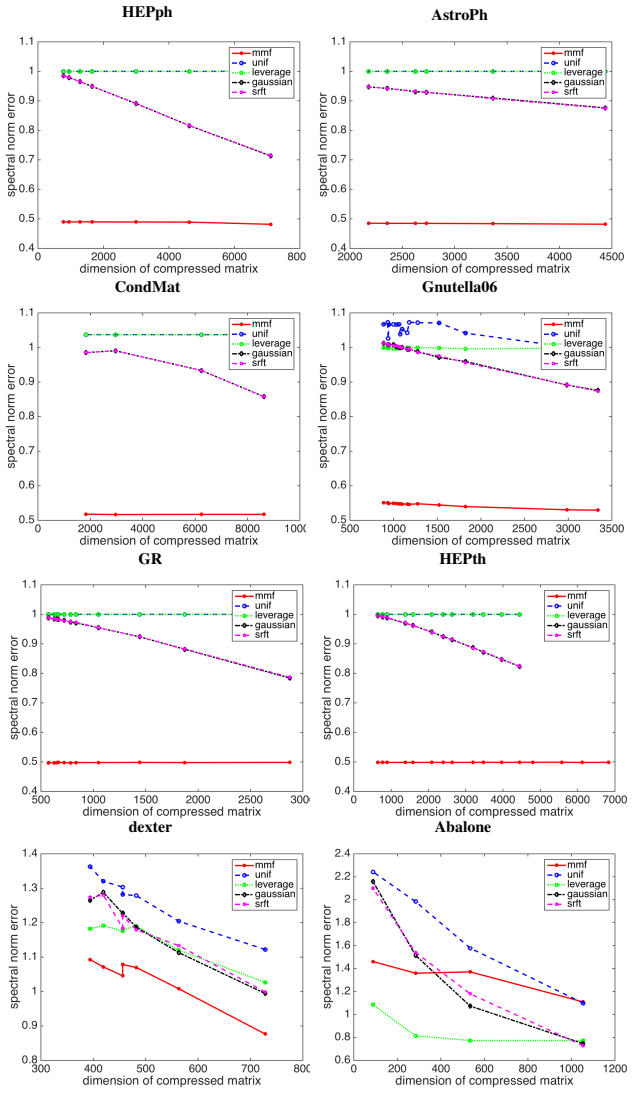


Figure 5: The spectral norm error $\|A - \tilde{A}\|_2$ of compressing matrices with pMMF vs. other sketching methods, as a function of the dimension of the compressed core. In each figure, the error is normalized by $\|A - A_k\|_2$, where $A_k$ is the best rank $k$ approximation to $A$. For the Nyström compression we followed the experiments in (Gittens and Mahoney, 2013) and used $k = 8$ for "dexter" , $k = 20$ for "Gnutella06" and "Abalone", and $k = 100$ for the rest of the datasets.

Empirically, the running time of Multiresolution Matrix Compression often scales linearly in the size of the data. The absolute running time is often orders of magnitude faster than that of other methods. Further work will explore folding entire learning and optimization algorithms into the multiresolution framework, while retaining the same scaling behavior.

Hence, pMMF addresses a different regime than many other Nyström papers: whereas the latter often focus on compressing $\sim 10^3$ dimensional matrices to just 10–100 dimensions, we are more interested in compressing $\sim 10^5$–$10^6$ dimensional matrices to $\sim 10^3$ dimensions. which is more relevant for practical problems, where e.g., kernel matrices need to be compressed to make learning algorithms feasible.

## Acknowledgements

This work was completed in part with resources provided by the University of Chicago Research Computing Center, and supported in part by NSF-1320344.

## References


Ailon, N. and Chazelle, B. (2010). Faster dimension reduction. *Communications of the ACM*, 53(2):97–104.

Börm, S. (2009). Construction of data-sparse Hˆ2-matrices by hierarchical compression. *SIAM Journal on Scientific Computing*, 31(3):1820–1839.

Brandt, A. (1973). Multi-level adaptive technique (MLAT) for fast numerical solution to boundary value problems. In *Proceedings of the Third International Conference on Numerical Methods in Fluid Mechanics*, volume 18 of *Lecture Notes in Physics*, pages 82–89. Springer Berlin Heidelberg.

Chandrasekaran, S., Gu, M., and Lyons, W. (2005). A fast adaptive solver for hierarchically semiseparable representations. *Calcolo*, 42(3-4):171–185.

Coifman, R. R. and Maggioni, M. (2004). Multiresolution analysis associated to diffusion semigroups: Construction and fast algorithms. Technical report, Technical report YALE/DCS/TR-1289, Yale University.

Coifman, R. R. and Maggioni, M. (2006). Diffusion wavelets. *Applied and Computational Harmonic Analysis*, 21(1):53–94.

Davis, T. A. and Hu, Y. (2011). The University of Florida Sparse Matrix Collection. *ACM Trans. Math. Softw.*, 38(1):1:1–1:25.

Deshpande, A., Rademacher, L., Vempala, S., and Wang, G. (2006). Matrix approximation and projective clustering via volume sampling. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 1117–1126. ACM.

Drineas, P. and Mahoney, M. W. (2005). On the Nyström method for approximating a Gram matrix for improved kernel-based learning. *Journal of Machine Learning Research*, 6:2153–2175.

Eckart, C. and Young, G. (1936). The approximation of one matrix by another of low rank. *Psychometrika*, I(3):211–218.

Edmonds, J. (1965). Paths, trees, and flowers. *Canadian Journal of mathematics*, 17(3):449–467.

Fowlkes, C., Belongie, S., Chung, F., and Malik, J. (2004). Spectral grouping using the Nyström method. *IEEE transactions on pattern analysis and machine intelligence*, 26(2):214–25.

Gittens, A. and Mahoney, M. (2013). Revisiting the Nyström method for improved large-scale machine learning. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, volume 28, pages 567–575. JMLR Workshop and Conference Proceedings.

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

Halko, N., Martinsson, P.-G., and Tropp, J. A. (2011). Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review*, 53(2):217–288.

Kondor, R., Teneva, N., and Garg, V. (2014). Multiresolution Matrix Factorization. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*.

Kondor, R., Teneva, N., and Mudrakarta, P. K. (2015). `pMMF`: a high performance parallel MMF library. Hosted at `https://github.com/risi-kondor/pMMF`.

Kumar, S., Mohri, M., and Talwalkar, A. (2009). Ensemble Nyström method. In *Advances in Neural Information Processing Systems*, pages 1060–1068.

Lee, A. B., Nadler, B., and Wasserman, L. (2008). Treelets — An adaptive multi-scale basis for sparse unordered data. *Annals of Applied Statistics*, 2(2):435–471.

Leskovec, J. and Krevl, A. (2014). SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`.

Mahoney, M. W. (2011). Randomized algorithms for matrices and data. *Foundations and Trends in Machine Learning*, 3.

Mahoney, M. W. and Drineas, P. (2009). CUR matrix decompositions for improved data analysis. *Proceedings of the National Academy of Sciences*, 106(3):697–702.

Martinsson, P.-G. (2008). Rapid factorization of structured matrices via randomized sampling. *arXiv preprint arXiv:0806.2339*.

Ravasz, E. and Barabási, A.-L. (2003). Hierarchical organization in complex networks. *Physical Review E*, 67(2):026112.

Savas, B., Dhillon, I., et al. (2011). Clustered low rank approximation of graphs in information science applications. In *Proceedings of the SIAM International Conference on Data Mining (SDM)*.

Shuman, D. I., Narang, S. K., Frossard, P., Ortega, A., and Vandergheynst, P. (2013). The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *Signal Processing Magazine, IEEE*, 30(3):83–98.

Smola, A. J. and Schökopf, B. (2000). Sparse greedy matrix approximation for machine learning. In *Proceedings of the Seventeenth International Conference on Machine Learning*, ICML '00, pages 911–918, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Tropp, J. A. (2011). Improved analysis of the subsamples randomized Hadamard transform. *Advances in Adaptive Data Analysis*, 3(1-2):115–126.

Wang, R., Li, Y., Mahoney, M. W., and Darve, E. (2015). Structured block basis factorization for scalable kernel matrix evaluation. *arXiv preprint arXiv:1505.00398*.

Wang, S. and Zhang, Z. (2013). Improving CUR matrix decomposition and the Nyström approximation via adaptive sampling. *The Journal of Machine Learning Research*, 14(1):2729–2769.

Williams, C. and Seeger, M. (2001). Using the Nyström method to speed up kernel machines. In *Advances in Neural Information Processing Systems 13*.

Woodruff, D. P. (2014). Sketching as a tool for numerical linear algebra. *Foundations and Trends in Theoretical Computer Science*, 10(1–2):1–157.

Zhang, K. and Kwok, J. T. (2010). Clustered Nyström method for large scale manifold learning and dimension reduction. *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, 21(10):1576–87.