École polytechnique fédérale de Lausanne

HPC for Numerical Methods and Data Analysis

Course Project Fall 2023

Randomized Nyström Low-Rank Approximation

Authors:
Christian Mikkelstrup
Julian Schmitt

Supervisor:
Prof. Laura Grigori

Contents

1 Introduction

In this project, we study the randomized Nyström approximation algorithm for the low-rank approximation of a positive-semidefinite matrix $A \in \mathbb{R}^{n \times n}$. There are several important applications for this theory, such as image processing, PCA, or solving integral equations. However, the most common practical setting are kernel methods for large-scale machine-learning problems. Since these algorithms scale at least quadratic in the number of data points, low-rank approximations are essential to obtain reasonable storage usage and computational costs.

The randomized Nyström approximation is based on a random sketching matrix $\Omega \in \mathbb{R}^{n \times l}$ and the formula:

$$A_{Nyst} = (A\Omega)(\Omega^T A\Omega)^{\dagger}(\Omega^T A) \tag{1}$$

where $l \ll n$ is the sketching dimension and $(\Omega^T A\Omega)^{\dagger}$ denotes the pseudo-inverse. In particular, the algorithm we used computes a fixed-rank $k \leq l$ approximation by truncating the sketched matrix A_{Nyst}. Our goal is to efficiently parallelize this alogrithm. Key aspects of our investigations are numerical stability, scalability, performance (in terms of runtime), and the approximation of the leading k singular values of A.

2 Randomized Nyström Algorithm

We start by describing the efficient computation of a rank-k approximation from the sketching formula (1). We obtain such an approximation by simply truncating the *singular value decomposition* (SVD) of A_{Nyst}, as this leads to the best rank-k approximation of (1) [Tro+17]. In the following, we denote this truncation by $[\![A_{Nyst}]\!]_k$ and note that this method is also called *modified fixed-rank Nyström via QR* in literature.

The "naive" approach of computing A_{Nyst} and afterwards the corresponding truncated SVD would be computationally heavy because calculating the SVD of large matrices is very expensive. Algorithm 1 illustrates an alternative which directly computes $[\![A_{Nyst}]\!]_k$ by decomposing A_{Nyst}. First, we calculate the two sketched matrices $C = A\Omega \in \mathbb{R}^{n \times l}$ as well as $B = \Omega^T A\Omega \in \mathbb{R}^{l \times l}$. This is the part of the algorithm with the highest computational cost since it consists of two high-dimensional matrix-matrix products. Nevertheless, in the next section we will present some sketching techniques which lower these costs. Next, we factorize $B = LL^T$, e.g. using the Cholesky decomposition, and $Z = CL^{-T} = QR \in \mathbb{R}^{n \times l}$ via the QR decomposition. This leads to

$$\begin{aligned}
[\![A_{Nyst}]\!]_k &= [\![(A\Omega)(\Omega^T A\Omega)^{\dagger}(\Omega^T A)]\!]_k \\
&= [\![CL^{-T}L^{-1}C^T]\!]_k = [\![ZZ^T]\!]_k \\
&= [\![QRR^TQ^T]\!]_k = QU_k\Sigma_k V_k^T V_k \Sigma_k^T U_k^T Q^T \\
&= QU_k\Sigma_k\Sigma_k U_k^T Q^T = QU_k\Sigma_k^2 U_k^T Q^T
\end{aligned}$$

where $U_k\Sigma_k V_k^T$ is the truncated SVD of $R \in \mathbb{R}^{l \times l}$. Recall $l \ll n$ and note that the SVD is only calculated on an $l \times l$ matrix providing significantly lower runtimes. Moreover, in our Python implementation (for details see Section 5) we do not calculate the entire SVD and throw away the rows and columns we do not need, but instead compute the partial SVD, extracting only the k singular values of the largest magnitude [Sci23]. This approach provides further speedup for low approximation ranks.

Note that in step 7 of Algorithm 1, instead of computing $\hat{U}_k$ as QU_k we could use $\hat{U}_k = ZV_k\Sigma_k^{-1}$, which would provide less numerical stability but entail a smaller computational cost. This smaller computational cost comes because one could omit the computation of the Q-factor of the QR decomposition. However, in practice, we observed that computing $\hat{U}_k$ as $ZV_k\Sigma_k^{-1}$ reduces the runtime of our algorithm by not more than 10%. One reason for this might be that we worked with $n = 2^{13}$ and accordingly $Z \in \mathbb{R}^{8192 \times l}$ is still relatively small. For larger dimensions, it could be worth it to consider $\hat{U}_k = ZV_k\Sigma_k^{-1}$, but we decided to focus on $\hat{U}_k = QU_k$ in this report.

Moreover, it is important to mention that the matrix $B = \Omega^T A\Omega$ can be rank-deficient, e.g. if the rank of A or Ω is smaller than l. In this case, it is not possible to obtain the Cholesky factorization in step 3. Instead, one can use the square root of $\Omega^T A\Omega$ computed via the SVD or eigendecomposition: Let $B = U\Sigma V^T$ be the SVD of B. Since B is symmetric and non-negative definite, it holds $V = U$. We simply set $L = U\sqrt{\Sigma}V^T$ and obtain $LL^T = U\sqrt{\Sigma}V^T V\sqrt{\Sigma}U^T = U\Sigma U^T = U\Sigma V^T = B$. Note that the SVD based approach is accompanied

Algorithm 1 Randomized Nyström

```
Input: $A \in \mathbb{R}^{n\times n}, \Omega \in \mathbb{R}^{n\times l}$
1: $C = A\Omega \in \mathbb{R}^{n\times l}$
2: $B = \Omega^T C \in \mathbb{R}^{l\times l}$
3: $B = LL^T$ (via Cholesky)
4: $Z = CL^{-T}$ (via substitution on triangular system)
5: $Z = QR$ ($QR$ decomposition)
6: Compute rank-k SVD of $R$: $U_k \Sigma_k V_k^T$
7: $\hat{U}_k = QU_k$
Output: $[\![A_{Nyst}]\!]_k = \hat{U}_k \Sigma_k^2 \hat{U}_k^T$
```

by considerably higher computational costs compared to the Cholesky decomposition. Not only is it more expensive to calculate the SVD, but additionally the matrix product $L = U\sqrt{\Sigma}V^T$ must be determined. Further, when using the Cholesky decomposition, one can exploit the fact that L is triangular and calculate $Z = CL^{-T}$ via backward substitution which implies[1] # flops $\doteq l^2$. In contrast, when using the SVD one has to solve a general linear system which does not perform better than $O(l^3)$. Although there are alternatives, e.g. making A full-rank by shifting [Tro+17], we will stick to the SVD approach.

Finally, we remark that Algorithm 1 is a is a *streaming* algorithm since it requires only one pass over the data or A, respectively. This is a major advantage compared to e.g. the randomized singular value decomposition.

2.1 Sketching Techniques

In this section, we describe the construction of the sketching matrix $\Omega \in \mathbb{R}^{n\times l}$. In particular, we are concerned with randomized methods for dimension reduction and it is therefore useful to define the so-called *oblivious l_2-subspace embedding* (OSE):

Definition 2.1 (OSE)**.** Let $0 < \epsilon < 1$ and $0 < \delta < 1$. A random matrix $\Omega \in \mathbb{R}^{n\times l}$ is a (ϵ, δ, d) OSE, if for any fixed d-dimensional subspace $V \subseteq \mathbb{R}^n$,

$$\left| \|x\|_2^2 - \|\Omega^T x\|_2^2 \right| \leq \epsilon \cdot \|x\|_2^2 \quad \forall x \in V$$

holds with probability $1 - \delta$.

For the randomized Nyström approximation, we considered two different sketching techniques: the *block Subsampled Randomized Hadamard Transform* (BSRHT) [Bal+22] and a *Short-Axis-Sparse Sketching Operator* (SASO) [Mur+23]. Both are structured random matrices which aim to reduce the computational costs of the matrix-matrix product $A\Omega$.

The BSRHT is a version of the Subsampled Randomized Hadamard Transform (SRHT) specifically designed for distributed architectures. For n being a power of two, the SRHT can be defined as

$$\Omega^T = \sqrt{\frac{n}{l}} RHD,$$

where

- $H \in \mathbb{R}^{n\times n}$ is the normalized Walsh–Hadamard matrix which is defined recursively as:

$$H_2 = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \qquad H_n = \begin{pmatrix} H_{n/2} & H_{n/2} \\ H_{n/2} & -H_{n/2} \end{pmatrix} \qquad \text{and} \qquad H = n^{-1/2} H_n$$

- $D \in \mathbb{R}^{n\times n}$ is a diagonal matrix with i.i.d. random variables $\sim$ Uniform($\{\pm 1\}$).
- $R \in \mathbb{R}^{n\times l}$ is a subset of l randomly sampled rows from the $n \times n$ identity matrix.

[1] '$\doteq$' indicates equality when omitting lower-order terms

Note that we can always zero-pad the data to ensure that n is a power of 2. Now, for P different processors, the BSRHT can be constructed block-wise from the SRHT as:

$$\Omega^T = \begin{pmatrix}\Omega_1^T & \dots & \Omega_P^T\end{pmatrix} = \sqrt{\frac{n}{Pl}}\begin{pmatrix}D_{L1} & \dots & D_{LP}\end{pmatrix}\begin{pmatrix}RH & & \\ & \ddots & \\ & & RH\end{pmatrix}\begin{pmatrix}D_{R1} & & \\ & \ddots & \\ & & D_{RP}\end{pmatrix} \tag{2}$$

with

- $H \in \mathbb{R}^{n/P \times n/P}$ being the normalized Walsh–Hadamard matrix.
- $D_{Li} \in \mathbb{R}^{l \times l}, D_{Ri} \in \mathbb{R}^{n/P \times n/P}$ being diagonal matrices with i.i.d. Rademacher entries ± 1.
- $R \in \mathbb{R}^{l \times n/P}$ being a uniform sampling matrix, sampling along the rows.

The main advantage of the BRSHT compared to the SRHT is that the matrix-matrix product distributed across P processors with row-wise partitioning can be computed as $A\Omega = \sum_{i=1}^{P} A_i\Omega_i$. In particular, one can compute the local contributions $A_i\Omega_i$ independently on each processor and finally sum-reduce them. Since Ω is distributed across processors, we never have to produce the large concatenated matrices in (2), as we can calculate $\Omega_i = \sqrt{\frac{n}{Pl}} D_{Li} RHD_{Ri}$. Furthermore, we can utilize that D_{Li} and D_{Ri} are diagonal matrices, and that R simply samples rows, such that only H has to be explicitly created and modified to get Ω_i. Also note that because the same R has to be generated on each processor, the root processor has to broadcast a seed for generating this sampling matrix. We chose to sample rows without replacement, but the other case is discussed in [Bal+22]. Moreover, we remark that one can exploit the Fast-Johnson-Lindenstrauß transform to reduce the flops required for applying BRSHT [Bal+22]. However, this requires a special implementation that we do not consider here. Balabanov et al. [Bal+22] showed that the construction of Ω as in (2) yields an (ϵ, δ, d) *oblivious subspace embedding* (OSE) if

$$n \geq l \geq 3.7\epsilon^{-2}\left(\sqrt{d} + 4\sqrt{\log\frac{n}{\delta} + 6.3}\right)^2 \log\frac{5d}{\delta} \tag{3}$$

with $0 < \epsilon < 1$ and $0 < \delta < 1$. This implies compatability with all randomized methods that rely on OSEs and in particular with the Nyström approximation. Note that condition (3) can be used to choose a proper sketching dimension l.

For the construction of the SASO, we call the rows of Ω *short-axis vectors* since $l \ll n$. These short-axis vectors should be independent of one another and have a small, fixed number of non-zero elements. In general, there are different ways of selecting the locations of the non-zero elements. We decided to sample k' indices uniformly from 1 to l without replacement, once for each row. Furthermore, the values of the non-zeros are drawn independently and uniformly from $[-2, -1] \cup [1, 2]$. Compared to i.i.d. Rademachers this can protect against the possibility of a given column of Ω being orthogonal to a row of A [Mur+23]. Finally, we set $k' = \min\{8, l\}$ following experimental results mentioned in [Mur+23]. In theory, the SASO is a faster sketching operator than BRSHT because of its high sparsity. However, to exploit this one would have to implement the corresponding algorithm using sparse vectors (implementation notes are available in [Mur+23]). We did not do so and therefore we expect that that SASO and BRSHT will perform similar w.r.t. the runtime. Last, there are theoretical sketching guaranties for SASO, but these are not as strong as (3) for BRSHT and one might expect that SRHT (and BRSHT) provide better results in terms of numerical stability. Nevertheless, we did not observe this effect in our experiments (see Section 5) which might be due to the fact that we did not use simple Rademachers as non-zero elements for SASO.

3 Parallelization of the Nyström Algorithm

Having seen the (sequential) randomized Nyström approximation in Algorithm 1, we focus now on the corresponding parallelization for a distributed system. For $A \in \mathbb{R}^{n \times n}$ and $\Omega \in \mathbb{R}^{n \times l}$, the Nyström approximation can be divided into 4 steps:

1. Sketching: $C = A\Omega \in \mathbb{R}^{n \times l}, B = \Omega^T C \in \mathbb{R}^{l \times l}$
2. Decomposition: $B = LL^T$ (e.g. via Cholesky) with $L \in \mathbb{R}^{l \times l}$ and $Z = CL^{-T} \in \mathbb{R}^{n \times l}$
3. QR factorization: $Z = QR$ with $Q \in \mathbb{R}^{n \times l}$ and $R \in \mathbb{R}^{l \times l}$

Algorithm 2 Parallel Sketching

```
Input: $A \in \mathbb{R}^{n\times n}, \Omega \in \mathbb{R}^{n\times l}$
1: Root processor: broadcast information for generating blocks of $\Omega$
2: for all processors $P_{ij}$ with $i, j = 1 : \sqrt{P}$ in parallel do
3:     Generate block $\Omega_j$
4:     Compute $C_{ij} = A_{ij}\Omega_j$
5:     Sum-Reduce among rows $C_i = \sum_{j=1}^{\sqrt{P}} C_{ij}$ with root being $P_{ii}$
6:     Scatter $C_i$ into $C_{ki}$ among processors in same column
7:     Compute $B_{ki} = \Omega_j[(k-1)n/P : kn/P, :]^T \; C_{ki}$
8: end for
9: Sum-Reduce $B = \sum_{k,i=1}^{\sqrt{P},\sqrt{P}} B_{ki}$
Output: $B, C$
```

4. Rank-k SVD of R: $U_k\Sigma_k V_k^T$ and $\hat{U}_k = QU_k$

Considering $l \ll n$, we aim to parallelize the sketching of A and use the TSQR algorithm for the QR factorization of the tall-skinny matrix Z. The decomposition of B and the SVD of R can be computed sequentially since both matrices are small (dimensions only dependent on l).

For the parallel sketching, we work with a 2D block distribution of A and a row-block distribution of Ω. From now on, we assume that the number of processors P is a power of 4 because we need $\sqrt{P}$ to be an integer for the 2D distribution and later we will need a power of 2 for TSQR. For $P = 4$, the matrix partitioning is the following:

$$C = A\Omega = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} \Omega_1 \\ \Omega_2 \end{pmatrix} = \begin{pmatrix} A_{11}\Omega_1 + A_{12}\Omega_2 \\ A_{21}\Omega_1 + A_{22}\Omega_2 \end{pmatrix}$$
$$B = \Omega^T C = \begin{pmatrix} \Omega_1^T & \Omega_2^T \end{pmatrix} \begin{pmatrix} C_1 \\ C_2 \end{pmatrix} = \begin{pmatrix} \Omega_1^T & \Omega_2^T \end{pmatrix} \begin{pmatrix} A_{11}\Omega_1 + A_{12}\Omega_2 \\ A_{21}\Omega_1 + A_{22}\Omega_2 \end{pmatrix}$$

The sketching algorithm works by first computing C and then using C to compute B. Algorithm 2 illustrates the procedure. For each processor owning a block $A_{ij} \in \mathbb{R}^{n/\sqrt{P}\times n/\sqrt{P}}$, we start by generating the corresponding block $\Omega_j \in \mathbb{R}^{n/\sqrt{P}\times l}$ of the sketching matrix. Then, we compute $C_{ij} = A_{ij}\Omega_j$ in parallel and sum-reduce among processors in the same row $C_i = \sum_{j=1}^{\sqrt{P}} C_{ij}$. For the computation of B, we need to sketch each C_i from the left by multiplying Ω_i. Note that for processors owning a diagonal block A_{ii}, it holds $\Omega_i = \Omega_j$ and we do not have to generate Ω_i. Therefore, we choose the processors owning a diagonal block A_{ii} of A as roots for the reduce operation. Now, we could simply compute $\Omega_i^T C_i$ on the $\sqrt{P}$ processors owning a C_i block and then sum-reduce among these to obtain B. However, later we want to execute the TSQR algorithm on all P processors using a row-block distribution of C. Therefore, we chose to scatter C already during sketching and exploit the additional processing power for the computation of B. We scatter each C_i among processors in the same column because these processors do all own the same sketching matrix Ω_i. Since we partitioned $C_i \in \mathbb{R}^{n/\sqrt{P}\times l}$ into $C_{ki} \in \mathbb{R}^{n/P\times l}$, each processor uses only a row-block of Ω_i for sketching and has to compute $B_{ki} = \Omega_{ki}^T C_{ki}$. Finally, we sum-reduce $B = \sum_{k,i=1}^{\sqrt{P},\sqrt{P}} B_{ki}$ among all processors. Note that the matrix C is row-block distributed and B is owned exclusively by the root processor after sketching.

Next, the decomposition of $B = LL^T$ is computed sequentially only on the root processor. The matrix L is then broadcasted to compute $Z = CL^{-T}$ in parallel. The algebra for 4 processors is:

$$Z = CL^{-T} = \begin{pmatrix} C_1 \\ C_2 \\ C_3 \\ C_4 \end{pmatrix} L^{-T} = \begin{pmatrix} C_1L^{-T} \\ C_2L^{-T} \\ C_3L^{-T} \\ C_4L^{-T} \end{pmatrix}$$

Here we utilize that C_iL^{-T} can be found by solving a linear system, thus never having to find L^{-1}. The advantage of the parallelized approach is that we do not need to gather C together and can compute Z directly distributed row-block wise which we need anyway for the TSQR algorithm in the next step.

In the third step, we compute the QR factorization of the tall and skinny matrix Z. We use the (parallel) TSQR algorithm which uses a reduction-like operation among a binary tree of processors to compute Q and

R in parallel (the binary tree is the reason why we need P being a power of 2). It is communication optimal and as stable as Householder QR. For details, we refer to [Dem+08]. At the end, the Q factor is returned distributed row-block wise while R is owned by the root processor.

In the final step, we copute the rank-k truncated SVD $U_k \Sigma_k V_k^T$ of R and Σ_k^2 sequentially on the root processor. Then, we broadcast U_k to compute $\hat{U}_k = QU_k$ in parallel (Q is still row-block distributed) and gather $\hat{U}_k$ together. We return Σ_k^2 as well as $\hat{U}_k$ on the root processor.

4 Datasets

In the following sections, we will investigate the performance of the Nyström algorithm on 5 different datasets. The positive semi-definite matrices, A, will be made from both real and synthetic data, as described below. For all datasets we chose to subsample the datapoints for the real data, or simply choose the dimension for the synthetic data such that we had $n = 8.192$, leading to a positive semi-definite matrix $A \in \mathbb{R}^{n \times n}$. Note that the side lengths of A, i.e. (n), should be divisible by P, i.e. a power of 4 (see Section 3). Thus setting $n = 2^{13}$ satisfies both this requirement for $P = 4$ and the fact that the SRHT matrices require n as a power of 2. You can always satisfy the requirements for n by zero-padding the input data.

In all experiments, all datasets were run with the different sketching matrices (Ω), sketching dimensions (l), and approximation ranks (k). We had a total of 19 different (l, k) pairs with $l \in \{400, 600, 1000, 2000\}$ and $k \in \{100, 200, 350, 500, 700, 900\}$. Note that this choice of parameters is specific to us using $P = 4$ processors with TSQR. This enables us to use the *tall and skinny* QR algorithm computing the thin QR factorization of $Z \in \mathbb{R}^{n \times l}$ for all our parameters since $n \geq 4 \cdot l$.

For the matrices created from real data, we used the MNIST (normalized to have values in the range $[0, 1]$) and YearPredictionMSD datasets [Ber+11; Lec+98]. We then used a subset of n datapoints and used the radial basis function (RBF), $A_{ij} = e^{-\|x_i - x_j\|_2^2}/\sigma^2$ to create a symmetric and positive semi-definite matrix. For MNIST, we chose a value of $\sigma = 100$, and for YearPredictionMSD we chose $\sigma = 10^4$ and $\sigma = 10^5$ for a total of 3 datasets, matching the approach in [Bal+22]. These datasets will in the following be referred to as `MNIST`, `YearMSD-1e4`, and `YearMSD-1e5`, respectively.
For the synthetic datasets, we choose diagonal matrices containing positive elements with different rates of decay for the singular values. Since A is a diagonal matrix, we can easily select the singular values, and thus also choose an effective rank, R, of the matrices. We used both a *Polynomial decay* and an *Exponential decay* of the singular values. The matrix with *Polynomial decay* has the form

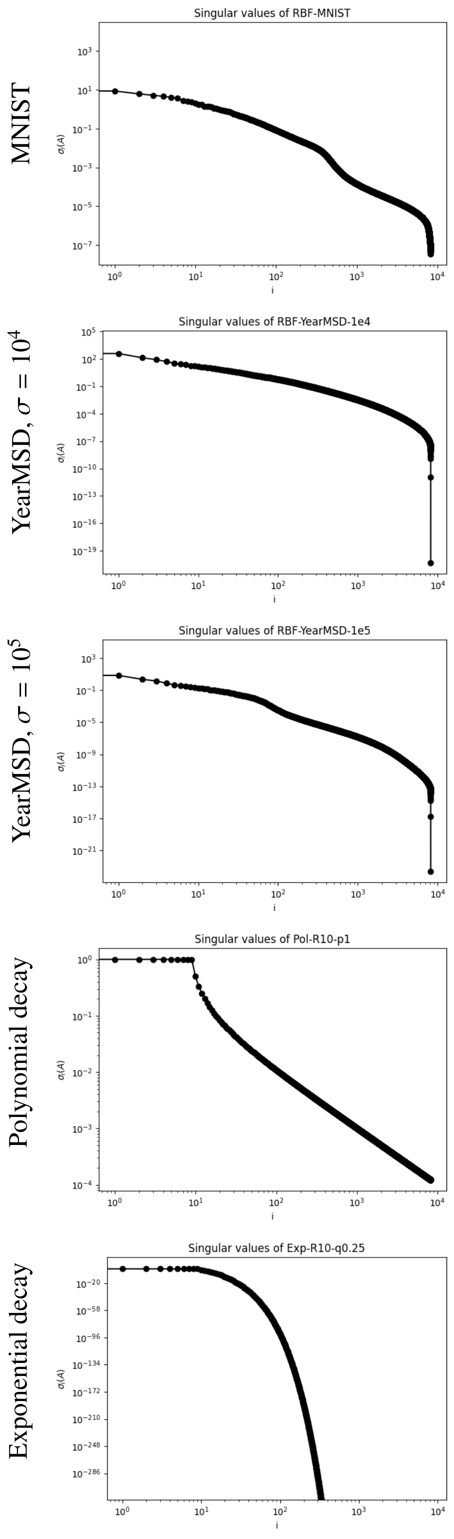


Figure 1: Singular values of the datasets.

$$A = \mathrm{diag}(\underbrace{1, ..., 1}_{R}, 2^{-p}, 3^{-p}, ..., (n - R + 1)^{-p}) \in \mathbb{R}^{n \times n}.$$

The parameter $p > 0$ is a hyperparameter used to control the decay of the polynomially decaying singular values. The matrix with *Exponential decay* has the form

$$A = \mathrm{diag}(\underbrace{1, ..., 1}_{R}, 10^{-q}, 10^{-2q}, ..., 10^{-(n-R)q}) \in \mathbb{R}^{n \times n}$$

The parameter $q > 0$ again controls the rate of decay, now of the exponentially decaying singular values. To match the datasets in [Tro+17], we set $R = 10$, $p = 1$, and $q = 0.25$ for all experiments, also matching the real datasets in size by setting $n = 8.192$, leading to 2 synthetic datasets, referred to as `Polynomial` and `Exponential`.

To investigate the datasets, Figure 1 displays the decay of the singular values for the A matrices. We computed the singular values via the SVD for the real datasets, and via the diagonal elements for the diagonal matrices. Looking firstly at the synthetic datasets, we see that a polynomial decay in singular values leads to a straight line, maybe even a bit of curvature upwards, in a log-log plot with the smallest singular values at 10^{-4}. On the other hand, the exponential dataset has a downward curvature on the singular values in the log-log plot with the smallest singular values being less than machine precision. With these two (synthetic) decays in mind, we can take a look at the datasets created by the RBF kernel. For all of them, we see a curvature downward on the singular values indicating a decay faster than a polynomial decay. We see very similar decay and magnitude between the `MNIST` and `YearMSD-1e4` (ignoring the very last singular value), but find that for `YearMSD-1e5` (simply adjusting σ), we have significantly smaller singular values and even have similar behavior to `Exponential` for the singular values above $i = 1000$. If we were to find the best rank k approximation to these datasets and get a 2-norm error below 10^{-5}, we would expect to use an approximation rank of a few thousand for `YearMSD-1e4` and `MNIST`, a rank of a few hundred for `YearMSD-1e5`, a rank of less than 100 for `Exponential`, whereas no approximation could be found for `Polynomial`.

5 Numerical Stability

For the numerical analysis and the performance evaluation, we executed all experiments on a single machine with an Intel Core i5 6500 (4 × 3.6 GHz, 6MB L3 cache) processor and 16 GB of DDR4-RAM powered by Fedora 38 (64-Bit, Linux Kernel 6.5.6). Furthermore, the algorithms have been implemented in Python 3.11.5 using NumPy 1.26.0. The parallelization has been done via the Python implementation *mpi4py* 3.1.4 of the well-known Message Passing Interface (MPI) (OpenMPI 4.1.4). The machine precision is $2.220446049250313 \cdot 10^{-16}$ for the type *float* (or *single precision*) on our system. Moreover, we limit the threads available to the BLAS (Basic Linear Algebra Subprograms) library used by NumPy to 1.

The numerical performance is tested on the datasets described in Section 4. The performance is measured using the relative error with the nuclear norm,

$$\textit{Relative error:}\ \frac{\|A - [\![A_{nyst}]\!]_k\|_*}{\|A\|_*}. \tag{4}$$

Since the nuclear norm is the sum of the singular values, $\|A\|_* = \sigma_1(A) + ... + \sigma_n(A)$, this value describes the ratio of the sum of the singular values from the approximation error of $[\![A_{nyst}]\!]_k$, compared to the original matrix A. If this value is low, we have a good low-rank approximation of A.

We used the parallel randomized Nyström with Cholesky for all datasets except for `Exponential`. For this dataset, Cholesky failed since B was classified as a singular matrix. It also makes sense that `Exponential` would run into problems with singular matrices when looking at the decay of the singular values in Figure 1, as its singular values are significantly lower than for the other datasets. For this dataset, we instead used the SVD approach. Because of the runtime differences, as seen in Section 6, we chose to use the SVD approach only for this dataset.

The relative error for all datasets and both sketching matrices can be seen in Figure 2. For all the tests except for `Exponential`, we could see a monotonic reduction in relative error using a larger size of l or k, if the other value was kept fixed. However, when keeping l fixed, even though the relative error was reduced when increasing k, we found a reduction in the improvement for each step. That is, we get diminishing returns on the improvement in the relative error when increasing the approximation rank, k. The exception to this was `Exponential`, where the effects of changing the approximation rank or the sketching dimension were inconclusive. However, for all tests run with `Exponential`, the relative error was close to or below 10^{-14}, which is nearing machine precision, and significantly lower than all other considered datasets.

Comparing across the different datasets, we see a big difference in the magnitude of the relative errors. `Polynomial` has a much larger relative error, compared to all other datasets, and `YearMSD-1e5` is significantly better than all other datasets (except `Exponential` mentioned before), across all combinations of different hyperparameters. This also shows that for the YearPredictionMSD dataset, the value for σ has a much larger impact on the relative error than the values for the sketching dimension or the approximation rank. It seems that datasets created with a smaller standard deviation in the radial basis function, are easier to approximate using the randomized Nyström approximation. However, in general, for the correct value of σ, the randomized Nyström algorithm works well in combination with kernel methods using the RBF kernel.

Comparing with the decay of the singular values in Figure 1, we find a close relationship between the decay of the singular values and the numerical relative error. The datasets with the fastest decay of singular values have the lowest relative error. Since we are working with a fixed-rank approximation, this is to be expected, as a slow decay of singular values makes the matrix harder to approximate with a low-rank approximation. Assuming best-rank approximation, $[\![A]\!]_k$, we have that for a dataset like `Polynomial`, with a large tail of singular values, it should have a worse performance than for `Exponential`, where a low-rank approximation results in a small error on the singular values.

Another interesting finding across datasets is that when fixing a low approximation error, the difference in relative error when varying the sketching dimension, depended heavily on the initial performance. That is, if we found a low relative error for a low approximation rank, increasing the sketch dimension made little to no difference, but when the initial performance was poor, as with `Polynomial`, we did see some difference. So if one were to select between increasing the sketch dimension or the approximation rank to get the most improvement in the relative error, the results were not conclusive. However, as a general trend, for a low approximation rank, we found the most improvement by increasing the approximation rank, and for medium to high approximation ranks we found more significant improvements by increasing the sketch dimension. This also explains why `YearMSD-1e5` has a better performance than the other two RBF datasets, as it has a faster decay on the singular values with the higher value of σ.

Comparing across different sketching matrices, we find next to identical results. That is, in our testing, we did not find any significant numerical differences between the two different sketching matrices. The only visible difference across sketching matrices was for the exponential decay dataset, but the numerical difference is small, and the error for both is close to machine precision.

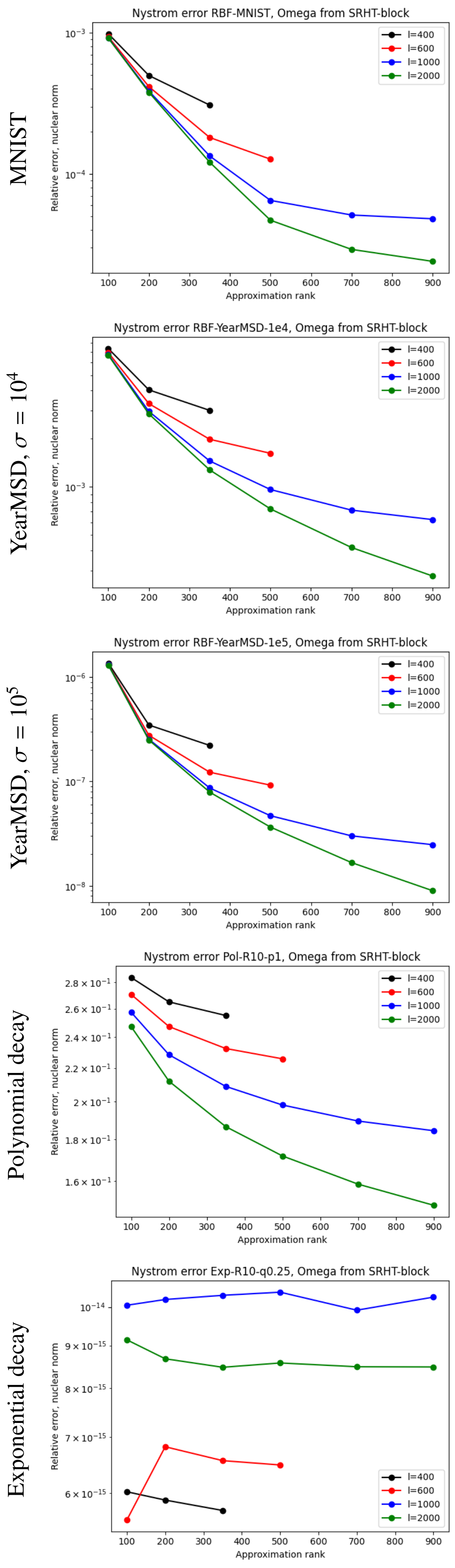
6 Runtime

We evaluated the runtime of the randomized Nyström algorithm on the same parameters as in Section 5. The runtime was measured as the time to do the steps described in Sections 2 and 3, and thus does not include the time to generate the positive semi-definite matrix A from the data, as is required when using kernel methods. To quantify the variation in runtimes, all tests were run 5 times to show the mean and the standard deviation of the runtimes at the different parameters.

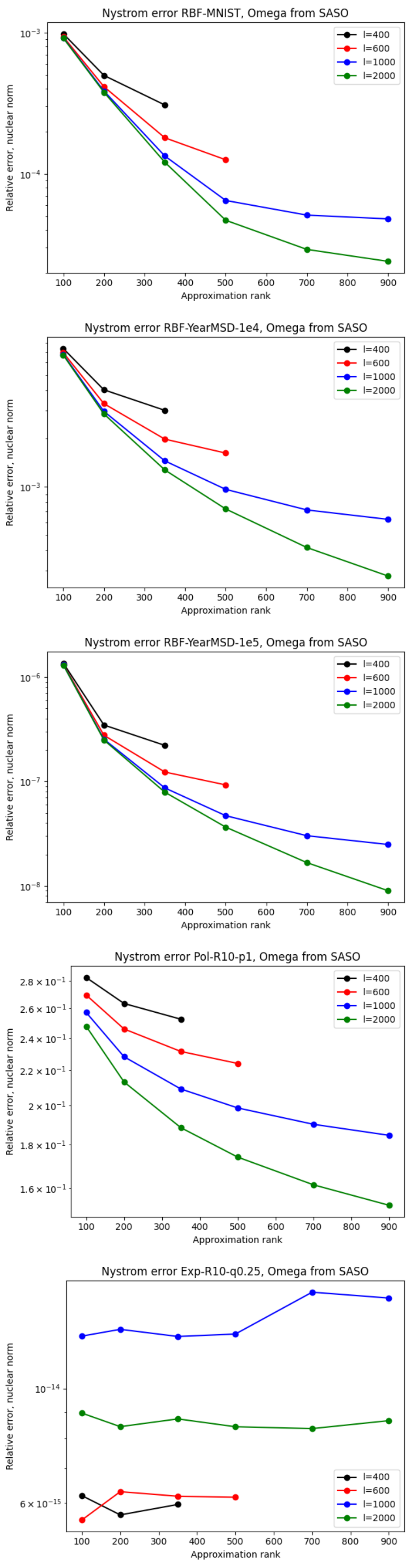
6.1 Parallel runtime

The parallel runtime, with the sketching matrices created using BSRHT and SASO, are shown in Figure 3. We find that `Exponential` is significantly slower to run than the other datasets. This difference comes down to the difference between the algorithms used, as the SVD approach has a higher computational cost, as described in Section 2. Besides the difference found between the two algorithms, we found little variation in the runtime across datasets.

When selecting between the sketching dimension and the approximation rank, we see a large difference in runtimes. When increasing the sketching dimension, and thus working with larger sketched matrices B and C, it takes significantly longer to run the approximation. We consistently saw an increase of runtime of more



(a) Ω created using BSRHT



(b) Ω created using SASO

Figure 2: Relative error of the Nyström algorithm.

than a factor 3 when doubling the sketching dimension from 1000 to 2000. We also found that the increase in runtime when increasing the sketch dimension was larger than any of our increases in the approximation rank. This makes sense when looking back at Section 3, where an increase in sketching dimension is directly related to the size of the matrices, $C_i \in \mathbb{R}^{n/\sqrt{P}\times l}$ and $B_{ij} \in \mathbb{R}^{l\times l}$, and thus the size of matrices that we find the Cholesky/SVD decomposition of in the parallel randomized Nyström algorithm. On the other hand, the difference in runtime across the approximation rank only affects the final step described in Section 3, where the rank-k truncated SVD is found, and the $[\![A_{Nyst}]\!]_k$ is calculated. Therefore, we conclude that the sketching dimension has the largest effect on the runtime of the randomized Nyström algorithm.

6.2 Sequential runtime

The sequential runtime, with the sketching matrices created using SRHT and SASO, are shown in Figure 4. We find very similar internal comparisons between the different datasets. Since `Exponential` is run using the SVD approach, this has a much higher runtime than the other datasets. As with the parallel runtimes, we also see that increasing the approximation rank or the sketching dimension leads to a higher runtime, with the sketching dimension having the largest influence.

Comparing against the parallel runtime, we find that going from the parallel algorithm (with 4 processors) to the sequential algorithm, the runtime increases by 30-50%. Thus switching to the parallel algorithm gives a significant improvement, especially when the runtime is upwards of 60+ seconds for some parameter settings with the sequential algorithm. However, as we have increased the number of processors by a factor of 4, our parallelization has some overhead. Note that we measured the runtime of the parallel algorithm including scattering the matrix A and gathering $[\![A]\!]_k$ back together across processors. Ignoring this data distribution would further reduce the runtime.

Comparing the two different sketching matrices, we find little to no difference in runtime between the SRHT and SASO sketching operators. That is, getting a low-rank approximation, $[\![A_{nyst}]\!]_k$, is not significantly affected by the choice between SRHT and SASO. However, we did see a slightly shorter runtime for SASO than for SRHT for some of the larger runtimes, for instance for the `YearMSD-1e5` with the highest approximation rank and sketching dimension. Since we have seen in the previous section that both sketching operators produce similar results w.r.t. accuracy, the SASO is the preferable choice for our test matrices when looking only at the running time.

7 Approximation of leading Singular Values

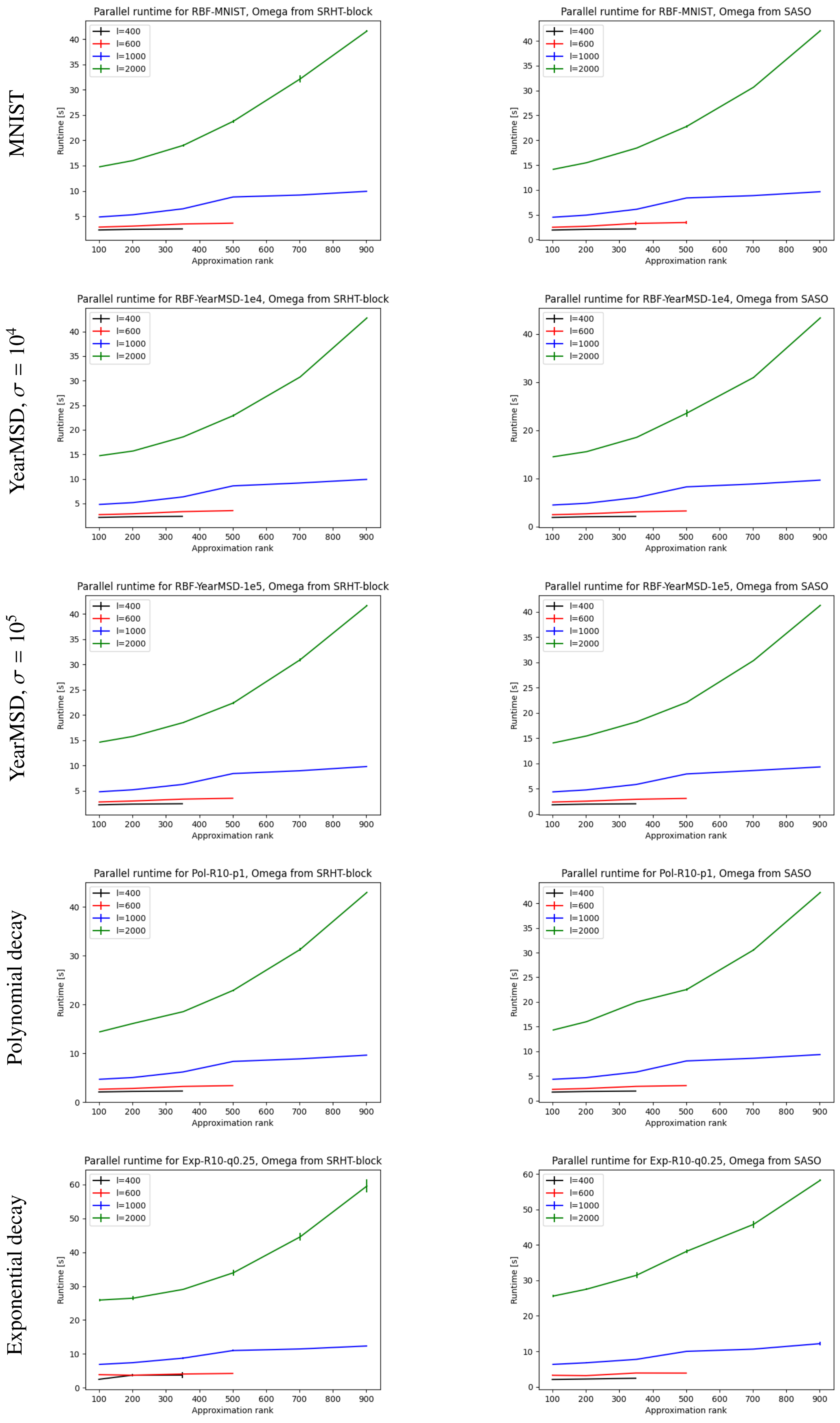
In this section, we address the question of whether the leading k singular values of A can be approximated using $[\![A_{Nystr}]\!]_k$. In particular, we compare the largest k singular values of A and $[\![A_{Nystr}]\!]_k$ (ordered non-increasingly) and measure their ratio

$$\text{ratio}_{\sigma_i} := \frac{\sigma_i([\![A_{Nystr}]\!]_k)}{\sigma_i(A)} \quad \forall i \in \{1, ..., k\}. \tag{5}$$

A ratio close to 1 would lead to an error of the singular values close to 0. Figure 5 shows the ratio of singular values ratio_{σ_i} from equation (5) for both sketching matrices and all datasets. We fixed the approximation rank to $k = 400$, once again computing the singular values via the SVD. Furthermore, we restrict our investigations to the parallelized Nyström algorithm from Section 3 since we have seen in Section 5 that the sequential version (Algorithm 1) produces similar results.

For the 3 datasets using the RBF kernel, we see very similar results. Here we find that (except the last few singular values for `YearMSD-1e5`) for $l = 400$ the ratio drops below 0.3, whereas for $l = 2000$ the first 400 singular values are approximated with almost no loss. For $l = 600$ and $l = 1000$, we find that the ratio stays above 0.7 and 0.9 respectively. That is, we find a large difference between the sketching dimensions, with a worse ratio (larger error) for the smaller singular values. For `Polynomial`, we see similar behavior when altering the sketching dimension, but we are no longer able to recreate the leading singular values for $l = 2000$, so we do not expect to be able to recreate the singular values for this dataset. For `Exponential`, we see a good approximation for the first ~ 70 singular values, whereafter we see a steep decrease followed by a steep increase. When comparing with the singular values of this dataset in Figure 1, its singular values above $i = 70$ are also significantly below machine precision, so the error on the singular values is expected to be low.

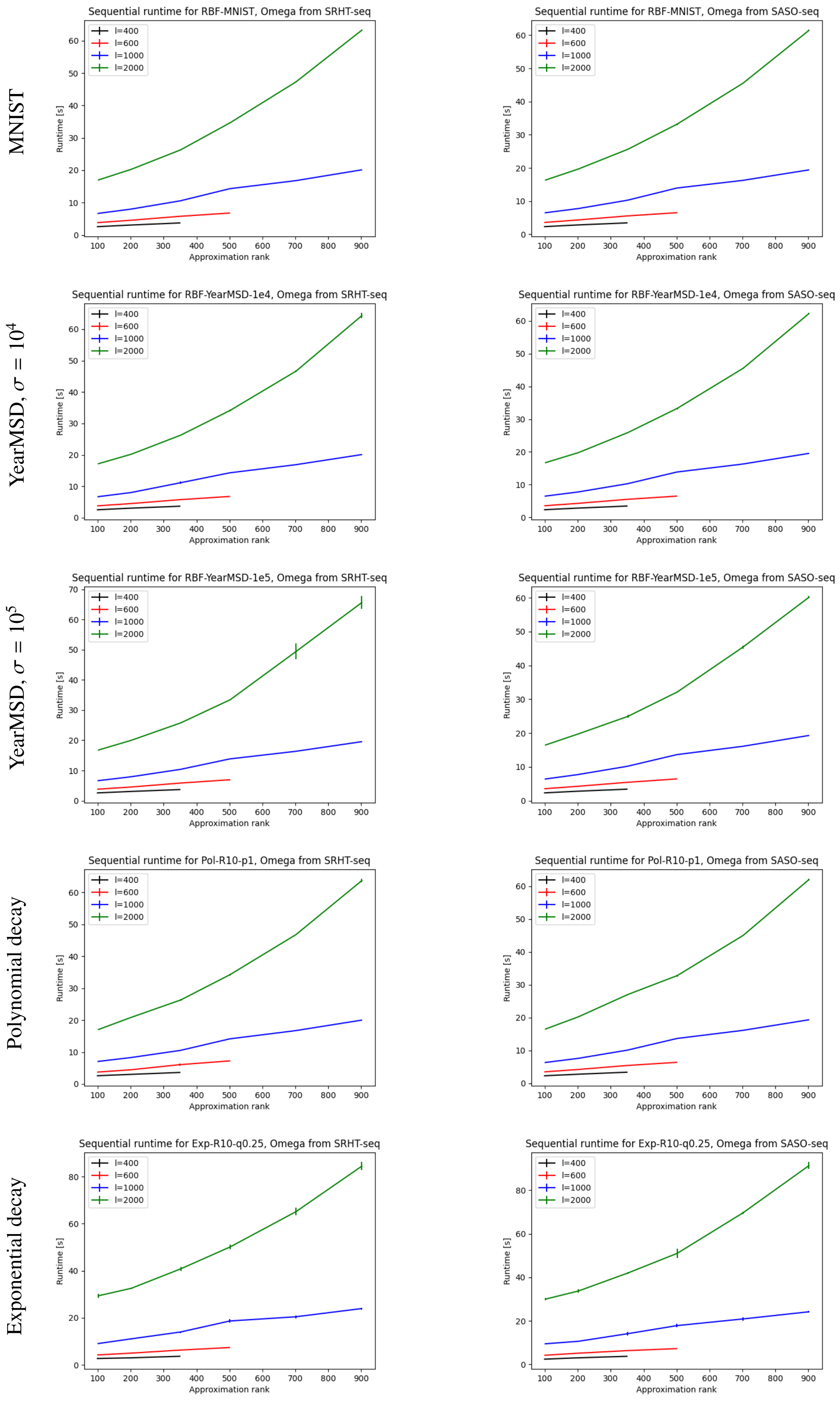
Both sketching operators perform very similarly, with the only visible difference for `Polynomial` for low sketching dimension, so this test does not reveal which sketching matrix to choose. However, we do find that choosing the sketching dimension l as $5 \cdot k$ looks to be a good and safe rule of thumb. However, if you are



(a) Ω created using BSRHT

(b) Ω created using SASO

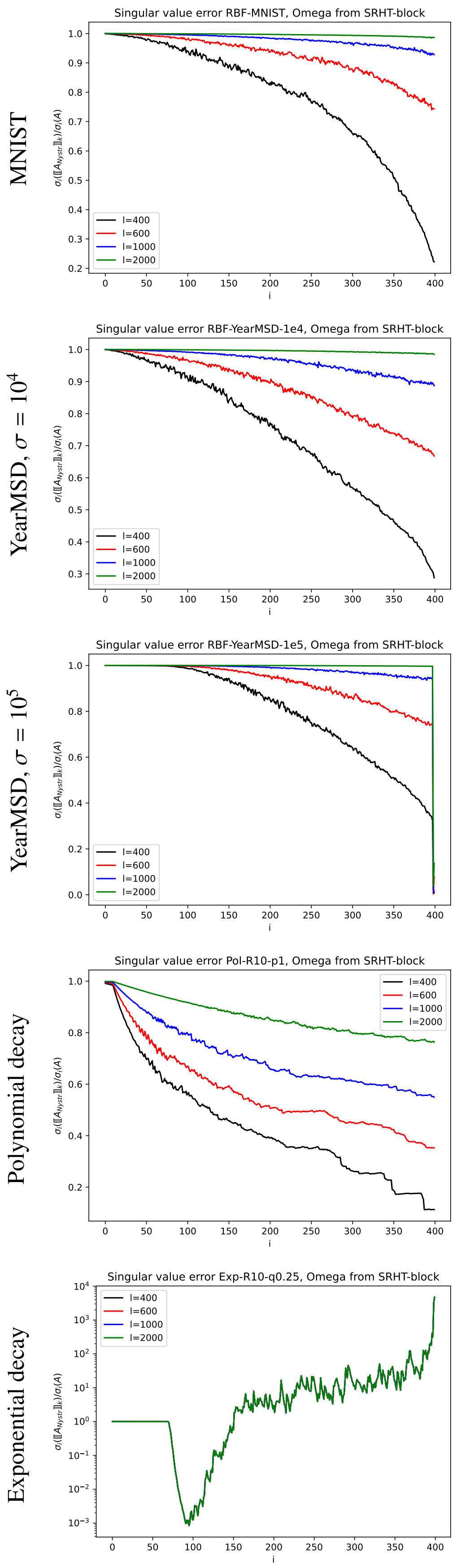
Figure 3: Runtime of the parallel Nyström algorithm.



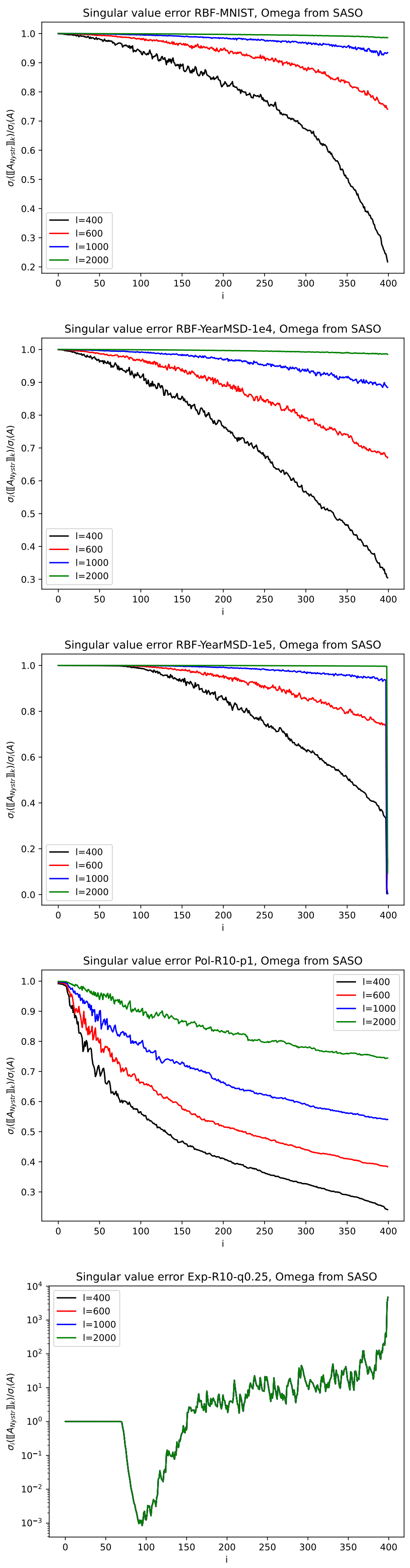
(a) Ω created using SRHT

(b) Ω created using SASO

Figure 4: Runtime of the sequential version of the randomized Nyström algorithm.



(a) Ω created using BSRHT



(b) Ω created using SASO

Figure 5: Ratio of singular values for $k = 400$ and different sketching dimensions between Nyström approximation and original matrix.

working with these datasets, depending on the usage, one would consider using $l = 1000$ for the RBF datasets (l being $2 \cdot k$ to $3 \cdot k$ since $k = 400$). Here $l = 1000$ is chosen as the ratio of approximated singular values is above 0.9 for all test cases and the computational effort is significantly reduced when the sketching dimension is halved compared to $l = 2000$.

8 Discussion and Conclusion

To summarize, we found an efficient parallelization of the randomized Nyström approximation which is stable and scales well w.r.t. the sketching dimension l as well as the approximation rank k. We tested our approach on 5 different datasets, real and synthetic to find and compare the numerical accuracy and runtime performance of the algorithm. In particular, our parallel algorithm produces similar numerical results as the sequential version but achieves a speedup of up to 30% on 4 processors. Furthermore, we compared the output processed using the two sketching matrices SASO and SRHT (or BRSHT respectively). Although one could expect SASO to provide lower overall runtimes compared to SRHT, we observed this effect only partially and very limited in practice. One might expect to see this effect more clearly with an implementation exploiting sparsity. Regarding numerical stability, both sketching operators produced similar results. Since there are stronger theoretical approximation guarantees for BRSHT such as (3), BRSHT should be preferred when using our implementation.

For the numerical stability of the algorithm, we found a low relative error when the decay of singular values was big enough, and found the Nyström approximation to be good for datasets created using the RBF kernel with a properly chosen σ value. However, when the decay of singular values was big enough to lead to a singular matrix, a change of algorithm was required for the approximation, leading to a higher runtime, but still achieving a good approximation.

Approximating the leading k singular values of A using the Nyström approximation is possible for fast decay of singular values, noting that the ratio might be big if the singular values get very small. Judging from our experiments, one should set l between $2.5 \cdot k$ and $5 \cdot k$ to obtain a decent approximation of the leading singular values, with the last few singular values having a magnitude of ~ 0.9 of the original singular values.

Finally, we want to point out some possible improvements for future work. First, we considered only the randomized Nyström approximation which incorporates the Q-factor of the QR decomposition because it is sufficient for our relatively small data dimension of $n = 2^{13}$ (see Section 2). However, an investigation of the Q vs. no-Q version on a large-scale distributed system with high-dimensional data, e.g. $n \geq 2^{16}$, would be interesting. Second, all of our experiments were done on 4 processors on a single machine. Therefore, we were unable to fully test the scaling capabilities of our parallelization. Lastly, one could theoretically analyze the computational cost of our parallel Nyström approximation in terms of floating-point operations and communication cost and compare it with the found numerical experiments.

References

[Bal+22] O. Balabanov et al. *Block subsampled randomized Hadamard transform for low-rank approximation on distributed architectures*. 2022. arXiv: 2210.11295 [math.NA].

[Ber+11] T. Bertin-Mahieux et al. “The Million Song Dataset”. In: *Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR 2011)*. 2011.

[Dem+08] J. Demmel et al. *Communication-optimal parallel and sequential QR and LU factorizations: theory and practice*. 2008. arXiv: 0806.2159 [cs.NA].

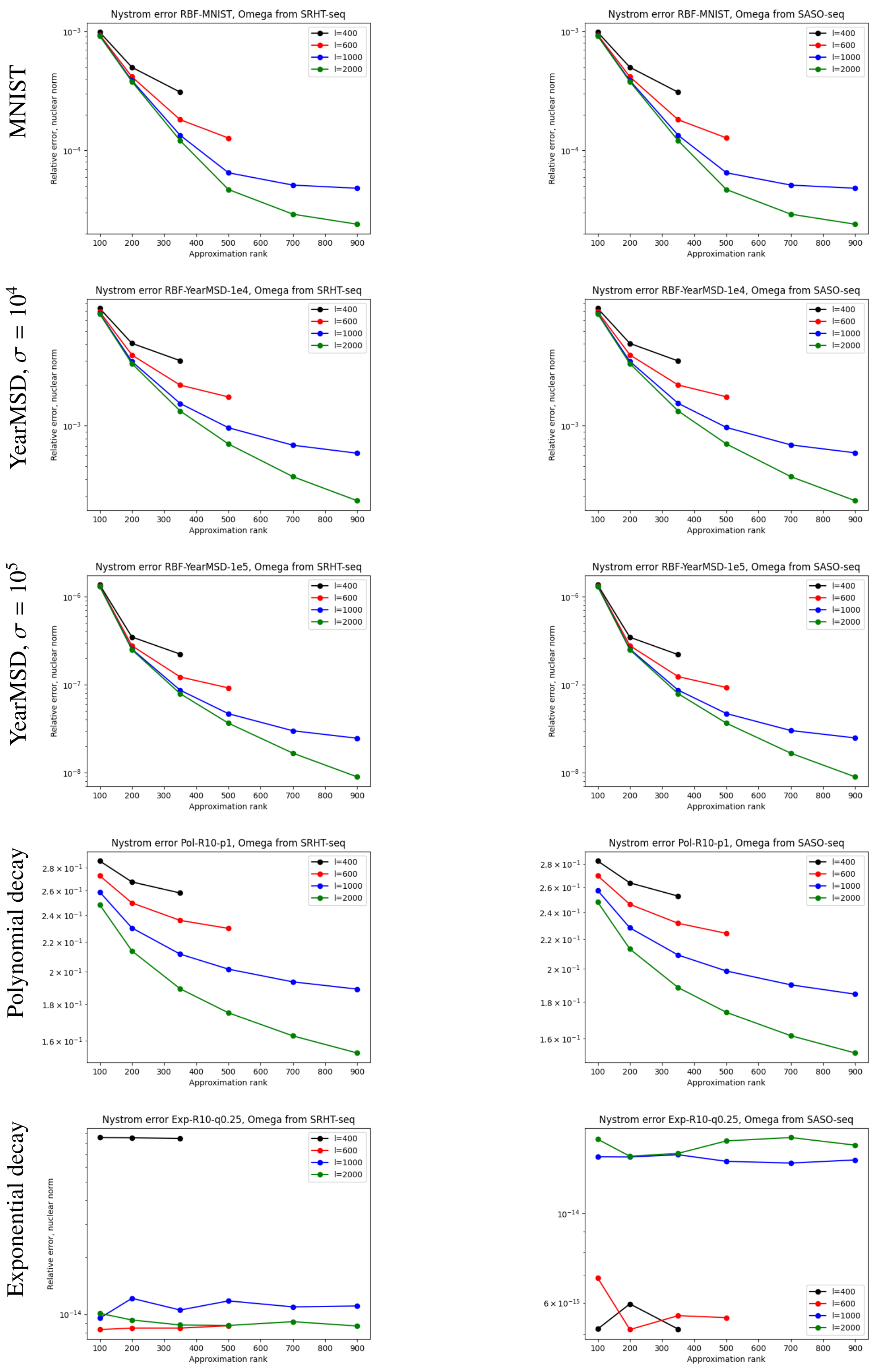
[Lec+98] Y. Lecun et al. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[Mur+23] R. Murray et al. *Randomized Numerical Linear Algebra : A Perspective on the Field With an Eye to Software*. 2023. arXiv: 2302.11474 [math.NA].

[Sci23] SciPy. *scipy.sparse.linalg.svds - SciPy v1.11.4 Manual*. Nov. 2023. URL: https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.linalg.svds.html.

[Tro+17] J. A. Tropp et al. *Fixed-Rank Approximation of a Positive-Semidefinite Matrix from Streaming Data*. 2017. arXiv: 1706.05736 [cs.NA].

Appendix A Numerical Stability of Sequential Nyström



(a) Ω created using BSRHT

(b) Ω created using SASO

Figure 6: Relative error of the sequential version of the randomized Nyström algorithm with different sketching matrices. Created for comparison with Figure 2, and is found to have very similar (almost identical) performance.