Project 2: Randomized Nyström Algorithm

HPC for numerical methods and data analysis

January 6th, 2025

Mathilde Simoni
SCIPER: 371423

Julie Charlet
SCIPER: 314454

1. Introduction

This project aims at implementing and investigating the numerical stability and parallel performance of the randomized Nyström algorithm. This algorithm is used for computing a rank-k approximation of a positive square semi-definite matrix A as a faster alternative to the best rank-k approximation $[[A]]_k = U_k \Sigma_k V_k^T$ where Σ_k is the diagonal matrix containing the k dominant singular values of A and U_k and V_k the associated left and right singular vectors [1]. While other methods, such as the randomized SVD exist [1], the Nyström approach is more efficient as it only requires 1 pass over A and "does not involve any high dimensional operations on A except the computation of the sketch"[1].

The problem of computing a rank-k approximation has many applications in scientific computing and data analysis, such has the implementations of fast solver for integral resolution, data compression or linear system resolution.

1.1. Randomized Nyström low rank approximation

Given a positive semi-definite matrix $A \in \mathbb{R}^{n\times n}$, a sketching matrix $\Omega \in \mathbb{R}^{n\times l}$, where l is the sketch dimension, and $l > k$, the randomized Nyström approximation is

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

Where $(\Omega^T A\Omega)^+$ denotes the pseudo-inverse of $\Omega^T A\Omega$. While several methods exists, in this project the rank-k approximation is computed by a rank-k truncation of the Nyström approximation A_{Nyst}. We denote it $\left[\left[A_{\text{Nyst}}\right]\right]_k$. The sequential algorithm is presented in Figure 1.

Sequential Nyström algorithm

1 Compute $C = A\Omega \in \mathbb{R}^{n\times l}$
2 Compute $B = \Omega^T C \in \mathbb{R}^{l\times l}$ and its Cholesky factorization $B = LL^T$
3 Solve $Z = CL^{-T}$ with substitution
4 Compute the QR factorization $Z = QR$
5 Compute the truncated rank-k SVD $R = U_k \Sigma_k V_k^T$
6 Compute $\hat{U}_k = QU_k$
7 Output the factorization $\left[\left[A_{\text{Nyst}}\right]\right]_k = \hat{U}_k \Sigma_k^2 \hat{U}_k^T$

Figure 1: Pseudocode for the sequential Nyström algorithm

The algorithm output is derived from the following:

$$\begin{aligned} A_{\text{Nyst}} = (A\Omega)(\Omega^T A\Omega)^+(\Omega^T A) &= CB^+C^T \\ &= C(LL^T)^+C^T \\ &= C(L^{-T}L^{-1})C^T \\ &= (CL^{-T})(L^{-1}C^T) \\ &= ZZ^T \\ &= QRR^TQ^T, \end{aligned}$$

where the 3^{rd} line was obtained from knowing that the pseudo-inverse reduces to the standard inverse for the invertible matrix LL^T. Then, taking the rank-k approximation of $R = U_k \Sigma_k V_k^T$:

$$\left[\left[A_{\text{Nyst}}\right]\right]_k = Q(U_k \Sigma_k V_k^T)(V_k \Sigma_k U_k^T)Q^T = QU_k \Sigma_k^2 U_k^T Q^T = \hat{U}_k \Sigma_k^2 \hat{U}_k^T,$$

where we used that V_k is an orthogonal matrix so $V_k^T V_k = I$.

Alternative to Cholesky decomposition

Since A is symmetric semi-positive definite, B is symmetric: $B^T = \left(\Omega^T A \Omega\right)^T = \Omega^T A^T \Omega = \Omega^T A \Omega = B$. However, B is only semi-positive definite, hence the Cholesky method can fail. The first option is to perform the eigenvalue decomposition of the square matrix $B = Q\Sigma Q^T$ where Q is an orthogonal matrix whose columns are the eigenvectors of A and the diagonal entries of Σ are the eigenvalues of A. In addition, since B is semi-positive definite, its eigenvalues are positive or null so we can take their square root: $B = Q\Sigma Q^T = Q\sqrt{\Sigma}\sqrt{\Sigma}Q^T = \left(Q\sqrt{\Sigma}\right)\left(Q\sqrt{\Sigma}\right)^T$ where $\left(Q\sqrt{\Sigma}\right)$ is the L factor used in the algorithm.

However, when implemented, this method does not work well for ill-conditioned matrices: when B is close to singular, small numerical errors are amplified. Hence, we instead use an LDL^T factorization, a variant of the Cholesky factorization for positive semi-definite matrices. In numpy, it is implemented with pivoting options, which allows to reorder rows and columns to ensure numerical stability. Then, since B is semi-positive definite, the elements on the diagonal of D are non-negative. We can therefore write:

$$B = L'DL'^T = L'\sqrt{D}\sqrt{D}L'^T = \left(L'\sqrt{D}\right)\left(L'\sqrt{D}\right)^T \Rightarrow L = L'\sqrt{D}.$$

1.2. Sketching matrices

The sketching matrices $\Omega \in \mathbb{R}^{l\times n}$ are randomized dimension reduction maps. They are defined as oblivious l_2-subspace embeddings OSE(n, ε, δ), if with probability $1-\delta$ for any n-dimensional subspace $V \subset \mathbb{R}^n$:

$$\forall x_i, x_j \in V \quad |\langle \Omega x_i, \Omega x_j\rangle - \langle x_i, x_j\rangle \,| \leq \varepsilon \, \|x_i\|_2 \, \|x_j\|_2 \, .$$

In this project, experiments are conducted on two different sketching matrices are conducted: the Gaussian matrix, and the Subsampled Random Hadamard Transform (SRHT) matrix.

1.2.1. Gaussian sketching Matrix

The gaussian sketching matrix consists of entries drawn as independent standard normal random variables, scaled by a factor of $\frac{1}{\sqrt{l}}$:

$$\Omega \in \mathbb{R}^{l\times n}, \quad \Omega_{ij} \sim \frac{1}{\sqrt{l}}\mathcal{N}(0,1).$$

The matrix Ω satisfies the properties of an OSE(n, ε, δ) if $l = \mathrm{O}\big(e^{-2}\big(n + \log\frac{1}{\delta}\big)\big)$.

Parallelization

Because the entries of Ω are independent, parallel computation is straightforward. Let Ω_i be the portion of Ω assigned to processor i. This local matrix is generated in the same manner, using a distinct seed for the random number generator (for example the rank of the processor).

1.2.2. SRHT sketching Matrix

The Subsampled Random Hadamard Transform is defined as an embedding which brings a vector $w \in \mathbb{R}^n$ to $\mathbb{R}^l$ by multiplication with $\Omega \in \mathbb{R}^{l\times n}$ defined as:

$$\Omega = \sqrt{\frac{n}{l}} \cdot P \cdot H \cdot D,$$

where:

- $D \in \mathbb{R}^{n\times n}$ is a diagonal matrix of uniform random variables $\in \{-1, +1\}$.
- $H \in \mathbb{R}^{n\times n}$ is the normalized Walsh-Hadamard matrix which has the effect of spreading the norm
- $P \in \mathbb{R}^{l\times n}$ is a subset of l rows of the identity matrix chosen uniformly at random.

Then, the matrix Ω satisfies the properties of an OSE(n, ε, δ) if $l = \mathrm{O}\big(e^{-2}\big(n + \ln\frac{n}{\delta}\big)\ln\frac{n}{\delta}\big)$. As explained in [1], the advantage of this method is that it can be be "efficiently applied to a vector in a sequential environment" [1] such as streaming data. In our case, it is applied to each column of A.

Parallelization

This sketching matrix cannot be simply parallelized like the gaussian sketching matrix. However, there exist a method described in [1] to parallelize the computation of Ω: it is called "Block SRHT". As explained in [1], this method combines "the benefits of structured oblivious embeddings, such as the SRHT, with the benefits of unstructured ones, such as Gaussian, from the complexity and performance standpoint."

Suppose Ω is distributed on P processors: $\Omega = [\Omega_1 \ \Omega_2 \ ... \ \Omega_P]$. Then, we can write Ω as follows:

$$\Omega = [\Omega_1 \ \Omega_2 \ ... \ \Omega_P] = \sqrt{\frac{n}{l}}[D_{L1} \ ... \ D_{LP}] \begin{bmatrix} RH & & \\ & ... & \\ & & RH \end{bmatrix} \begin{bmatrix} D_{R1} & & \\ & ... & \\ & & D_{RP} \end{bmatrix},$$

where:

- $D_{Li} \in \mathbb{R}^{l \times l}$ is diagonal with independent random signs $\in \{-1, 1\}$
- $D_{Ri} \in \mathbb{R}^{\frac{n}{P} \times \frac{n}{P}}$ is diagonal with independent random signs $\in \{-1, 1\}$
- $H \in \mathbb{R}^{\frac{n}{P} \times \frac{n}{P}}$ is the normalized Walsh-Hadamard matrix
- $R \in \mathbb{R}^{l \times \frac{n}{P}}$ is the uniform sampling matrix

This way, the multiplication ΩA can easily be parallelized.

1.3. Datasets

We use three different datasets. The two first datasets are diagonal synthetic matrices, specifically the *polynomial decay matrix* and the *exponential decay matrix*. The third dataset is generated from the MNIST dataset using the radial basis function (RBF), as done in [1].

The polynomial decay matrix is generated as:

$$A_1 = \text{diag}(1, ..., 1, 2^{-p}, 3^{-p}, ..., (n - R + 1)^{-p}),$$

where we choose the number of initial ones as R=10 (R is referred as the "effective rank" [2]), and we test values of $p \in \{0.5, 1, 2\}$ where p acts as the rate of polynomial decay.

The exponential decay matrix is:

$$A_2 = \text{diag}\big(1, ..., 1, 10^{-q}, 10^{-2q}, ..., 10^{-(n-R)q}\big),$$

where the initial number of ones is again chosen as R=10 and we let the rate of exponential decay be $q \in \{0.1, 0.25, 1\}$. Regarding the size of the matrices A_1 and A_2, we choose $n = 2048$ and $n = 4096$ since these matrices are used to test stability, hence do not need to be big. A visualization of the diagonal values of A_1 and A_2 for the different parameters is displayed in Figure 2.

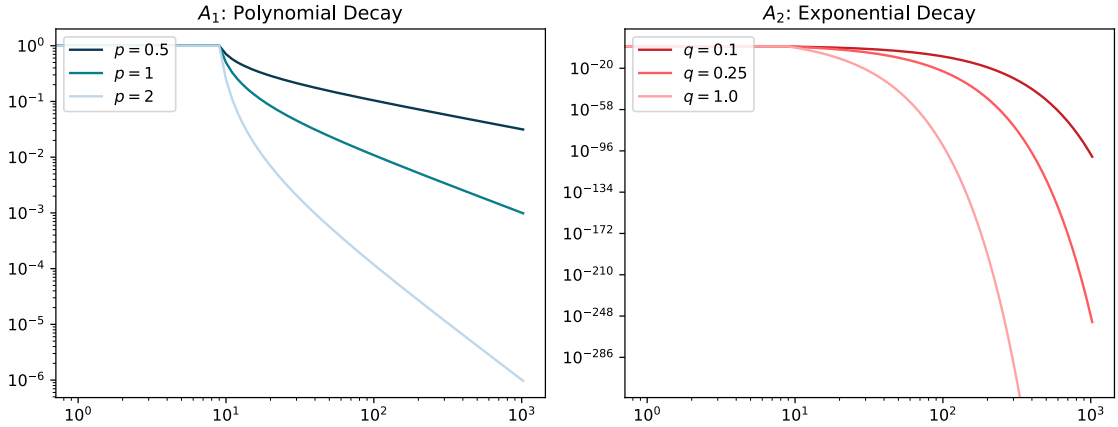

Figure 2: Diagonal entries of A_1 and A_2 for varying rates p and q

The third matrix is generated from the MNIST dataset using the radial basis function to generate a (n, n) dense matrix from n rows of input data, as done in [1]. Then A_3 is generated as:

$$A_3^{i,j} = \text{rbf}(x_i, x_j) = e^{\frac{-\|x_i - x_j\|^2}{c^2}},$$

where (x_i, x_j) are rows of the input MNIST data. Regarding the size of the matrix A_3, we choose powers of 2 for n, from $n = 2^7 = 128$ to $n = 2^{13} = 8192$ as this matrix will be used for the runtime analysis. A visualization of the matrices with $n = 512$ and $n = 8192$ is provided in the Appendix (Figure 14).

2. Parallelization of the randomized Nyström algorithm

A is uniformly distributed over a 2D grid of processors in the following way:

$$A = \begin{bmatrix} A_1 & ... & A_{\sqrt{P}} \\ & ... & \\ & ... & A_P \end{bmatrix},$$

where $A_i \in \mathbb{R}^{r \times r}$ for $i = 1, 2, ...P$ with $r = \frac{n}{P}$.

2.1. Application of the sketching matrix

The application of the sketching matrix is implemented as to take advantage of fast left-multiplication in `python`. Hence, we compute

$$C = (\Omega A)^T = A\Omega^T$$
$$B = \Omega C = \Omega(\Omega A)^T = \Omega A \Omega^T,$$

where we used the symmetry of A ($A = A^T$). In the gaussian case, Ω is distributed according to a normal distribution, and therefore Ω^T follows the same distribution. In the SRHT case, H is a symmetric matrix, D_R and D_L are diagonal hence symmetric, and R follows a uniform distribution. By the same argument as previously regarding R^T, Ω and Ω^T are equivalent. Consequently, this implementation is equivalent to the application of the sketching matrix as $C = A\Omega$ and $B = \Omega^T A \Omega$.

Gaussian sketching matrix:

In the gaussian case, Ω is computed explicitly and simply multiplied to A_{local} to obtain C in parallel.

SRHT sketching matrix:

In the SRHT case, the matrix Ω is not explicitly computed but the transformation is directly applied to A_{local}. Particular attention is required for the random seed used at each step. Indeed, Ω is split along rows when it is applied to C and along columns when it is applied to B. To avoid unnecessary communications, row and column rank are used as *local seed* to generate matrices D_L and D_R. The pseudocode in Figure 15 in the Appendix describes each operation, where $r = \frac{n}{P}$.

PARALLEL NYSTRÖM ALGORITHM

Distribute matrix A across all processors in a 2D grid topology
All processors:
  Compute $C^T = \Omega A$ where $C \in \mathbb{R}^{r \times l}$
  Perform *Sum-Reduce* operation on C^T across processors within the same column
Processors in first column:
  Compute $B = \Omega C$ where $B \in \mathbb{R}^{l \times l}$
  Perform *Sum-Reduce* operation on B across processors within the same row
Root processor:
  Compute the Cholesky factorization $B = LL^T$
  Broadcast L to processors in the first row
Processors in the first row:
  Compute $Z = CL^{-T}$ with substitution
  Perform the QR factorization $Z = QR$ using TSQR
Root processor:
  Compute the singular value decomposition of $R = \tilde{U}_k S_k \tilde{V}_k^T$
  Broadcast $\tilde{U}_k$ to all processors in the first row
Processors in the first row:
  Compute $\hat{U}_k = Q\tilde{U}_k$
end

Figure 3: Pseudocode for the parallelized Nyström algorithm

2.2. Parallel algorithm

Figure 3 presents the parallel implemention of the randomized Nyström algorithm. Based on the distribution of the matrix A in parallel, the number of processors P should be a perfect square $(1, 4, 9, 16, 25, ...)$. In addition, because of the use of TSQR, it should be a power of 2. Therefore, we experiment with $P = 1, 4, 16, 64$ in order to satisfy both conditions.

3. Investigation of the numerical stability

The numerical stability of the randomized Nyström approximation will be studied in terms of the relative error computed in the nuclear norm:

$$\mathrm{E} = \frac{\|A - \left[\left[A_{\mathrm{Nyst}}\right]\right]_k\|_*}{\|A\|_*}, \tag{2}$$

where $\|.\|_*$ denotes the nuclear norm, computed as the sum of the singular values of the matrix. Generally, the best rank-k approximation of A is given by the truncated SVD $[[A]]_k = U_k \Sigma_k V_k^T$ where Σ_k is the diagonal matrix containing the k dominant singular values of A and U_k and V_k the associated left and right singular vectors [1]. In that case, it holds that:

$$\min_{\mathrm{rank}(A_k) \leq k} \|A - A_k\|_* = \|A - [[A]]_k\|_* = \sum_{i=k+1}^{n} \sigma_i, \tag{3}$$

where $\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_n$ are the singular values of A [1]. We will investigate how close the Randomized Nyström approximation is from this optimal error for both sketching matrices and for the data matrices A_1 and A_2 as a function of the truncation rank k and for different sketching dimensions l. We use a 2×2 grid of processors (in [1], they use a grid of 8×8 processors but our matrix is smaller so we reduced this number).

3.1. Gaussian sketching Matrix

The results of the stability analysis are presented in Figure 4. The error E from Equation 2 is displayed for $k = \{5, 10, 25, 50, 100, 150, 200, 300\}$ and $l = \{50, 150, 250, 500, 700\}$. Evidently, sketching dimension $l \geq k$. We also display the relative optimal error using Equation 3 and dividing by the nuclear norm of A:

$$\frac{\|A - [[A]]_k\|_*}{\|A\|_*} = \frac{\sum_{i=k+1}^{n} \sigma_i}{\|A\|_*}$$

3.2. SRHT sketching Matrix

We conduct the exact same analysis with the SRHT sketching matrix, which results are displayed in Figure 16 in the Appendix.

3.3. Discussion of Results

Analysis depending on k and l

For all 6 matrices, we observe that the relative error E decreases as both k and l increase, which aligns with theoretical expectations. Specifically, as k grows, the truncation retains more eigenvalues, leading to improved approximations. In addition, the error reducing with an increasing l is consistent with the theory of oblivious subspace embedding (OSE). A Gaussian matrix Ω is OSE(n, ε, δ) if:

$$\begin{aligned} l &= \mathrm{O}\left(\varepsilon^{-2}\left(n + \log\frac{1}{\delta}\right)\right) && \text{for the Gaussian matrix,} \\ l &= \mathrm{O}\left(\varepsilon^{-2}\left(n + \ln\frac{n}{\delta}\right)\ln\frac{n}{\delta}\right) && \text{for the SRHT matrix.} \end{aligned} \tag{4}$$

In both cases, smaller values of δ and ε correspond to a higher-quality subspace embedding (better geometric preservation). According and Equation 4, a larger l achieves these improvements.

In addition, we observe that for matrices with polynomial decay, the error is optimal for small k. For larger k, it approaches the optimal error if l is large, a behavior more pronounced when the polynomial decay rate is higher. This result aligns with the theory. Indeed the smaller σ_k is compared to $\sum_{i=k+1}^{n} \sigma_i$, the more accurate the Nyström approximation $\left[\left[A_{\mathrm{nyst}}\right]\right]_k$ is with respect to the truncated SVD $[[A]]_k$. A higher polynomial decay rate increases this difference, leading to better approximations. A similar trend is observed for matrices with exponential decay. However, in this case, we also notice a plateau around 10^{-14}, below which the error does not decrease. This behavior is likely caused by numerical limitations, as 10^{-14} is close to machine precision.

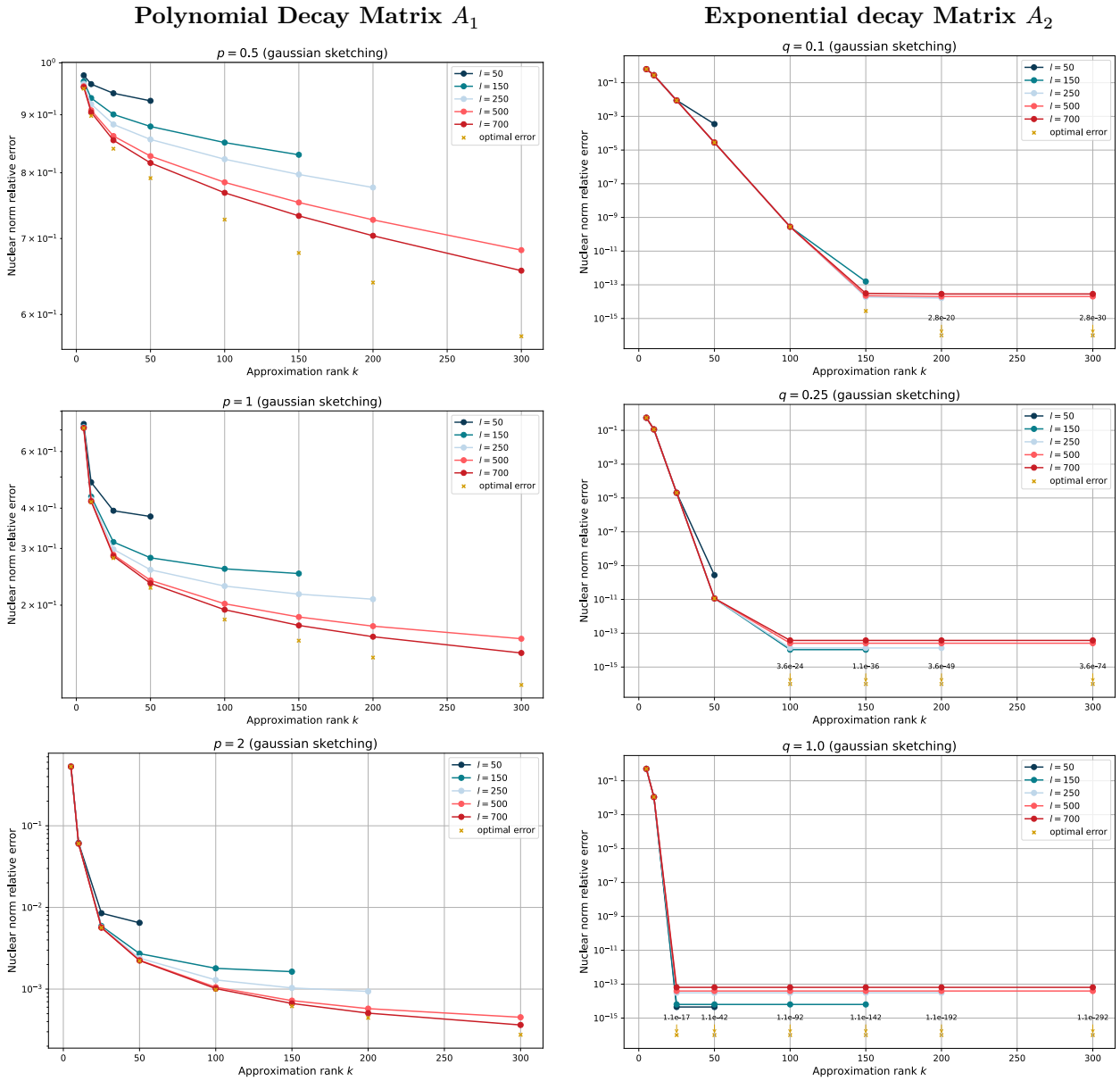

Figure 4: Stability analysis for polynomial (left) and exponential (right) matrices with **Gaussian** sketching

Another observation is that for exponential decay matrices with the highest rate $q = 1$, the error for large l is greater than for smaller l. This behavior is likely due to numerical limitations: when l is large, the algorithm attempts to approximate eigenvalues that are very small, leading to numerical instability.

Finally, we note that for matrices with fast eigenvalue decay, the value of l does not need to be excessively large for a given k. For example, in the case of exponential decay matrices with rate $q = 0.1$ and $q = 0.25$, the errors for $k = 50$ are equal for $l = \{150, 250, 500, 700\}$. This supports the discussion from class, where it was suggested that an optimal ratio between these values is typically $l = 4k$.

Comparison of Gaussian and SRHT sketching matrices

We observe minimal differences in the error between Gaussian and SRHT sketching matrices. Although the Gaussian sketching method shows a slightly lower error in some cases, this difference is negligible. The authors of [1] also obtained equivalent results for both sketching matrices, (although they matrices A were different). The motivation for using SRHT sketching despite the more complex structure is its computational efficiency in specific scenarios, such as streaming data, without impacting stability.

Stability analysis as P grows

An additional analysis was conducted to check whether the number of processors influences the stability of the results (the previous tests were performed using a grid of 2×2 processors). The results in Figure 9 illustrate the behavior for matrix A_1 (polynomial decay) with size $n = 4096$, decay rate $p = 2$, and a fixed

$l = 64$, the largest value which allows running the script with up to 64 processors. We observe minimal to no impact on stability as P increases, even for the Block SRHT Sketching matrix.

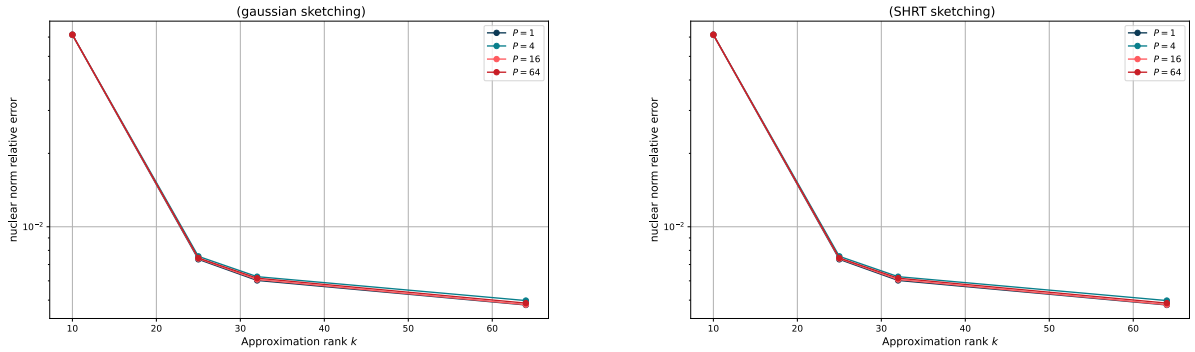

Figure 9: Error for the matrix A_1 with $n = 4096$ depending on P

4. Presentation of the sequential runtimes

This section compares the runtimes of the sequential algorithm using both sketching matrices. The runtime of the sequential algorithm depends on the size of matrix A (n), on the dimension of the sketching matrix Ω (l), and on the choice of the sketching method (SRHT or gaussian). The MNIST dataset is utilized for this comparison.

The runtime of the sequential algorithm will be evaluated for both sketching matrices as a function of n and l. For the matrix size experiments, the size of the matrix is progressively increased ($n = \{1024, 2048, 4096, 8192\}$) while maintaining a constant sketching dimension ($l = 128$). For the sketching dimension experiments, the dimension of matrix A is fixed ($n = 1024$), and the sketching dimension is varied ($l = \{128, 256, 512, 1024\}$). It is noteworthy that the sketching dimension is not required to be a power of two.

To enhance the robustness of the runtime estimations, each reported runtime represents the average of five runs. The total runtime is divided into five components to assess the performance of each part of the algorithm individually. Those components are:

1. ***Sketching***: Computation of $C = A\Omega$ and $B = \Omega C^T$
2. ***Cholesky***: Cholesky decomposition of $B = LL^T$
3. ***Z with substitution***: Computation of $Z = CL^{-T}$ by solving the linear system, with substitution
4. ***QR***: QR decomposition $Z = QR$
5. ***Rank-k truncation***: SVD decomposition of $R = U_k \Sigma_k V_k^T$ and computation of $\hat{U}_k = QU_k$

The runtimes presented exclude the time required to generate the test matrix A. All experiments are run on EPFL's *Scitas* cluster using `python` version 3.9.1, `numpy` version 2.0.2 and `scipy` version 1.13.1.

4.1. Gaussian sketching matrix

Figure 10 presents the runtime results using the Gaussian sketching matrix. In both plots, the dashed line represents the theoretical runtime for comparison.

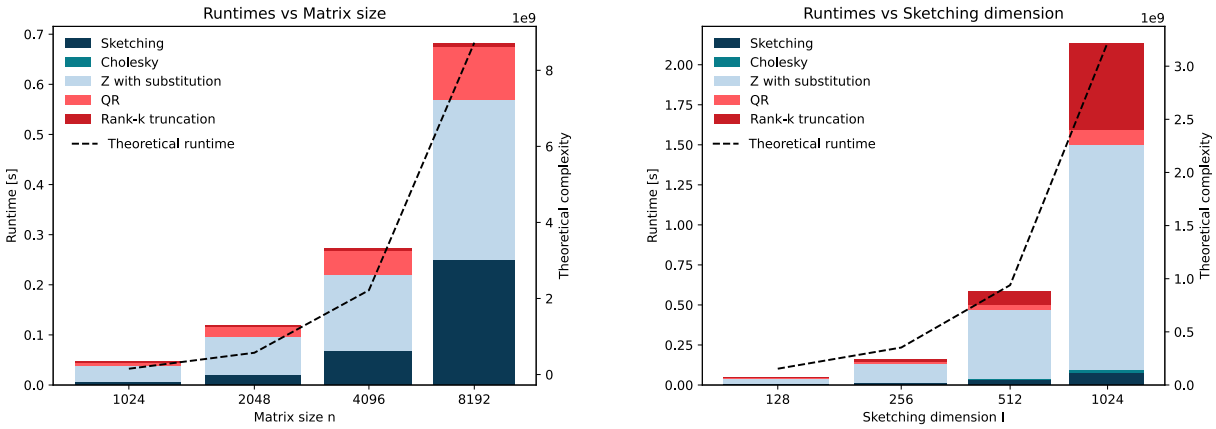

Figure 10: Sequential algorithm runtimes depending on matrix size (left) and sketching dimension (right) with the gaussian sketch matrix

4.2. SRHT sketching matrix

Figure 11 presents the sequential runtimes when using the SRHT sketching matrix. Again, the theoretical complexity is represented as a dashed line.

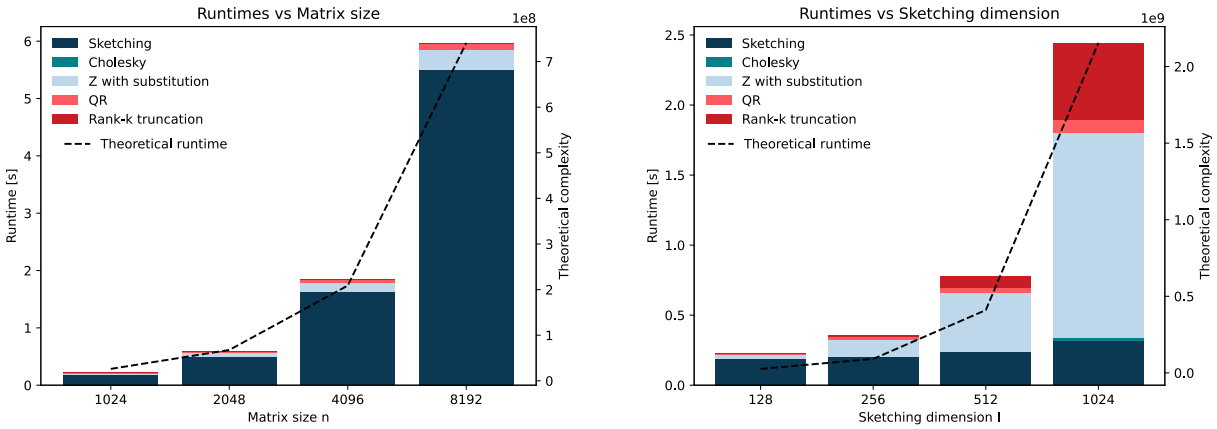

Figure 11: Sequential algorithm runtimes depending on matrix size (left) and sketching dimension (right) with the SRHT sketch matrix

4.3. Discussion of Results

Comparison with theoretical complexity

When using a sequential algorithm, the theoretical cost for applying Ω in the gaussian case is $O(n^2 l)$. When using SRHT, the theoretical complexity is $O(n^2 \log n)$. The Cholesky decomposition and the SVD decomposition of R have a time complexity of $O(l^3)$. Computing Z by solving a linear system and the QR decomposition of Z has a cost of $O(nl^2)$ and computing $\hat{U}_k$ is $O(nlk)$ complexity. Therefore, the overall theoretical runtime of the sequential randomized Nyström algorithm is:

$$\begin{aligned} O(n^2 l + nl^2 + l^3 + nlk) \quad & \text{for the Gaussian matrix,} \\ O(n^2 \log n + nl^2 + l^3 + nlk) \quad & \text{for the SRHT matrix.} \end{aligned}$$

In Figure 10 and Figure 11, the observed runtime evolution corresponds well with the theoretical complexity (dashed line). For the SRHT method, this confirms that, although the current implementation of the Hadamard transform may be slow, it scales correctly with the size of A. However, this similarity holds only in terms of the order of magnitude, as emphasized by the axis graduations.

Matrix size n

For both sketching methods, as the matrix size n increases, the runtime components that grow the most are applying the sketching matrix, performing the QR decomposition, and computing Z. These steps, involving matrix operations of sizes (n, n) and (n, l), have theoretical complexities of $O(n^2 \log n)$ or $O(n^2 l)$ (depending on the sketching method), $O(nl^2)$, and $O(nl^2)$, respectively.

For $n = 8192$, the SRHT method exhibits a total runtime nearly 10 times greater by order od magnitude than the Gaussian method (5.576 s vs. 0.252 s for the sketching part). The substantial runtime of the SRHT sketching matrix is attributed to the Hadamard transform applied to each column of A and C. While more efficient Hadamard transform algorithms could enhance performance, this step inherently depends on n due to the sizes of $A \in \mathbb{R}^{n \times n}$ and $C \in \mathbb{R}^{n \times l}$.

The rank-k truncation is only slightly influenced by n, as its primary operations—computing the SVD of R and matrix multiplication to obtain $\hat{U}_k$ involve matrices of sizes (l, l) and (n, l), respectively.

Overall, the application of the sketching matrix Ω is the most affected by increasing n, particularly in the SRHT case, where it incurs significant performance costs for large n.

Sketching dimension l

As the sketching dimension l increases, Figure 10 and Figure 11 show that the Cholesky decomposition, SVD of R (including rank-k truncation), and solving the linear system to compute Z experience the most runtime growth, consistent with their $O(l^3)$ and $O(nl^2)$ complexities. The rank-k truncation is particularly sensitive to changes in R's size $(l \times l)$.

For the Gaussian sketch matrix (Figure 10), the runtime for sketching increases slightly with l, while it remains nearly constant with the SRHT method (Figure 11). This independence of the SRHT method from l is a key advantage, enabling representation of A in larger OSEs without added performance costs.

5. Discussion of parallel performance

This section will discuss the parallel performance of the Gaussian and SRHT methods with an increasing number of processors $P = \{4, 16, 64\}$. The sequential runtime ($P = 1$) is added as a baseline for comparison.

Speed-up and efficiency are displayed for both sketching methods. The speed-up is defined as the scaling of performance when increasing the number of processors, while the efficiency measures the utilization of processors, indicating how well each processor contributes to speeding up the computation. There are computed as:

$$\text{Speed-up } (P) = \frac{T_1}{T_P}, \qquad \text{Efficiency } (P) = \frac{T_1}{P \cdot T_P},$$

where T_1 is the runtime for 1 processors and T_P for P processors. This given experiment was performed on the MNIST dataset of size $n = 8192$, sketching dimension $l = 128$ and truncation rank $k = 100$. As for the sequential performance, the runtimes are split as before into 5 components, and each value is an average of 5 runs for robustness.

5.1. Gaussian sketching matrix

On the left, Figure 12 displays the runtimes of sequential algorithm and of the parallel algorithm with a gaussian sketching matrix for 1, 4, 16 and 64 processors. On the right, the speed-up and the efficiency are displayed.

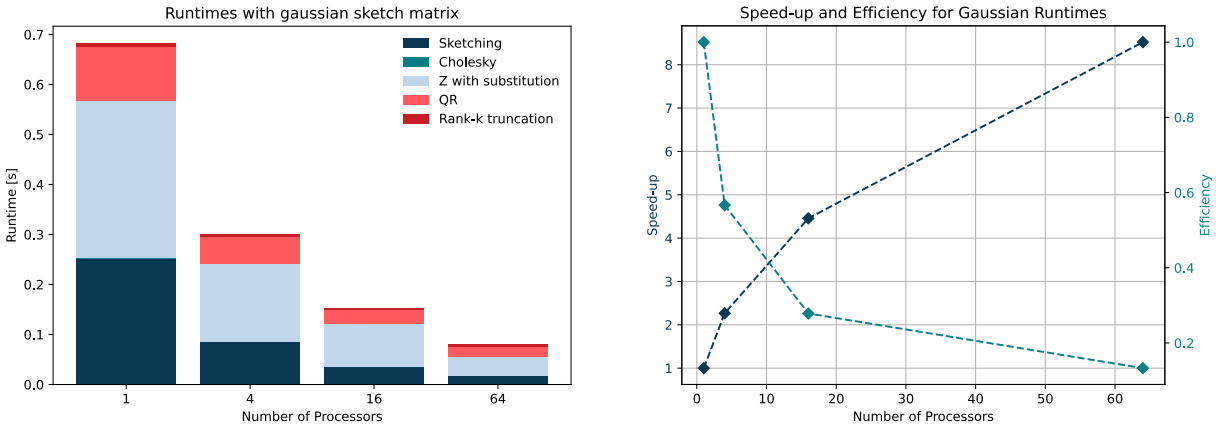

Figure 12: Parallel algorithm runtimes (left), efficiency and speed-up (right) depending on number of processors with gaussian sketching matrix

5.2. SRHT sketching matrix

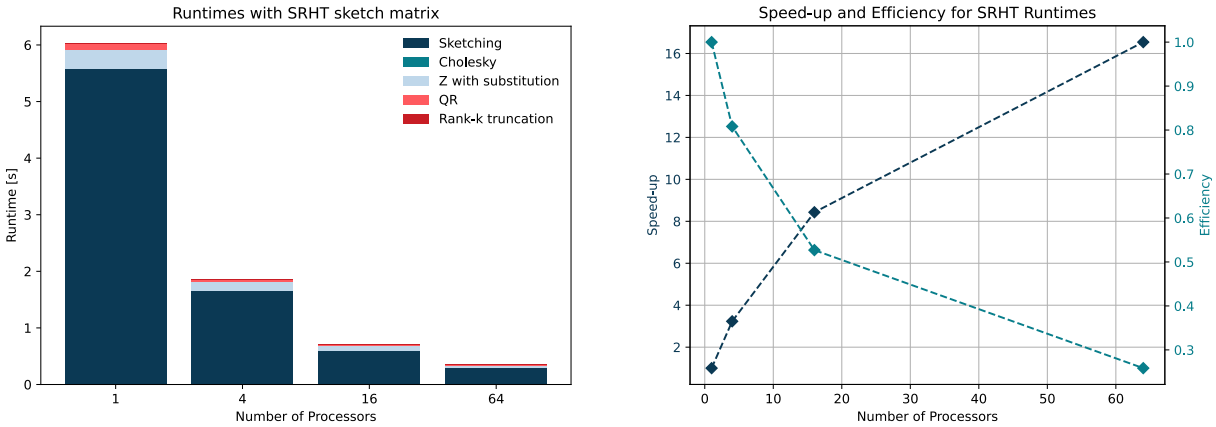

Figure 13: Parallel algorithm runtimes (left), efficiency and speed-up (right) depending on number of processors with SRHT sketching matrix

5.3. Discussion of Results

Overall Runtimes

The primary motivation for parallelizing the randomized Nyström algorithm is to enable computations on a partition of $A \in \mathbb{R}^{r \times r}$, where $r = \frac{n}{p}$ represents the local size of matrix A. Consequently, all operations whose complexity previously depended on n are reduced to $\frac{n}{p}$ in complexity. If p is sufficiently large, this can significantly decrease computation time.

However, communication costs must also be considered. Specifically, the cost of distributing an $m \times n$ matrix to p processors is $O(mnp)$. Nevertheless, due to the 2D grid partitioning of processors, the column/row reductions frequently employed in the parallel algorithm incur a cost of $O(mn \log p))$. Therefore, achieving optimal performance requires a balance between minimizing computation and communication costs.

Figure 12 and Figure 13 demonstrate the decreased overall runtime with an increasing number of processors. With 64 processors, the algorithm reaches a speed-up of 8 with gaussian sketching matrix and of 16 with the SRHT method. As expected, the efficiency decreases with the increasing number of processors, highlighting the cost of communication of the parallel algorithm.

Application of Sketching Matrix

In the Gaussian case, the runtime for applying the sketching matrix decreases from 0.252 s in the sequential algorithm to 0.017 s using 64 processors, achieving a speed-up of 14.8. The SRHT method demonstrates a speed-up of 18.5, with the runtime decreasing from 5.576 s in the sequential algorithm to 0.301 s with 64 processors.

The other components of the algorithm achieve a similar speed-up for both sketching methods, as they are identical in both cases. Thus, the higher overall speed-up of the SRHT method can be attributed to its superior parallel scalability compared to the Gaussian method.

6. Conclusion

This work examines the numerical stability, sequential performance, and parallel scalability of the Randomized Nyström Algorithm using two sketching methods. Both the Gaussian and Subsampled Randomized Hadamard Transform (SRHT) achieve comparable errors in the low-rank SVD decomposition of matrix A, which decrease when k and l grow, approaching the optimal error (given by the truncated SVD) when the decaying rate of the matrices is high. However, the block SRHT demonstrates superior parallel scaling and is independent of the sketching dimension l, making it an efficient OSE method for parallel architectures.

Generative AI

Generative AI (https://chatgpt.com/) has been used to improve the syntax and the formulation of this report.

Bibliography

[1] O. Balabanov, M. Beaupère, L. Grigori, and V. Lederer, "Block subsampled randomized Hadamard transform for low-rank approximation on distributed architectures," *arXiv preprint arXiv:2210.11295*, 2022.

[2] J. A. Tropp, A. Yurtsever, M. Udell, and V. Cevher, "Fixed-rank approximation of a positive-semidefinite matrix from streaming data," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

7. Appendix

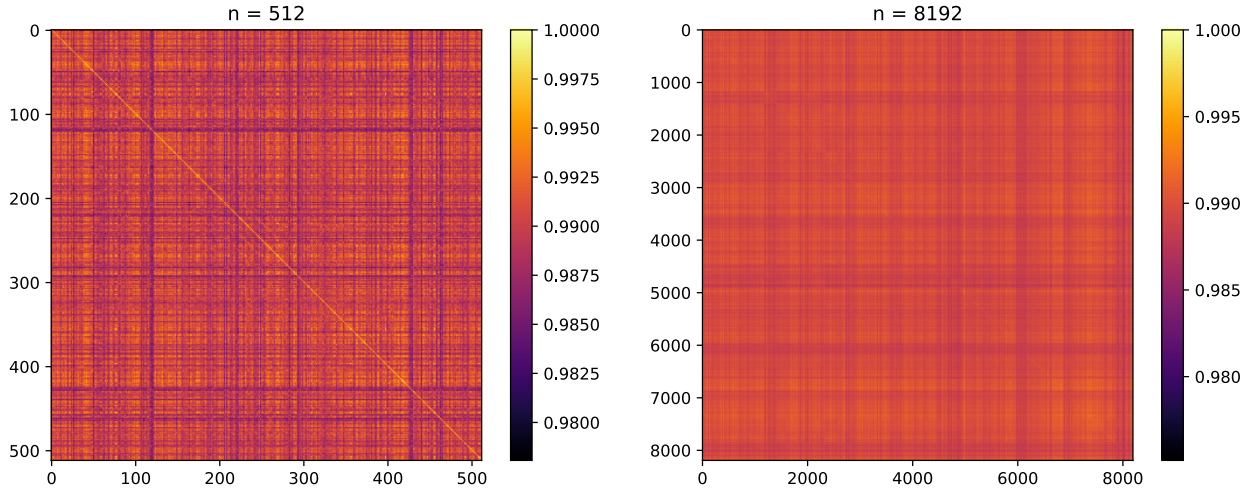

Figure 14: Visualizations of A_3

Parallel application of the SRHT matrix to comoute $C = \Omega A$ and $B = \Omega A \Omega^T$

Distribute matrix A across all processors in a 2D grid topology
All processors:
  Set the *local seed* as the row rank
  Generate $D_r \sim U((0,1)) \in \mathbb{R}^r$ and $D_l \sim U((0,1)) \in \mathbb{R}^l$
  Compute $C^T = \sqrt{\frac{r}{l}} D_r A$
  Compute the Walsh-Hadamard transform of each column of C^T
  Randomly sample l columns of $C^T = C^{T[R,:]}$, with R identical on all processors (*global seed*)
  Compute $C^T = D_L C^T$
  Sum-Reduce C^T across columns
Processors on first row
  Set the *local seed* as the column rank
  Generate $D_r \sim U((0,1)) \in \mathbb{R}^r$ and $D_l \sim U((0,1)) \in \mathbb{R}^l$
  Compute $B = \sqrt{\frac{r}{l}} D_r C$
  Compute the Walsh-Hadamard transform of each column of B
  Randomly sample l columns of $B = B[R,:]$, with R identical on all processors (*global seed*)
  Compute $B = D_L B$
  Sum-Reduce B across rows

Figure 15: Pseudocode for the parallel application of the SRHT sketching matrix

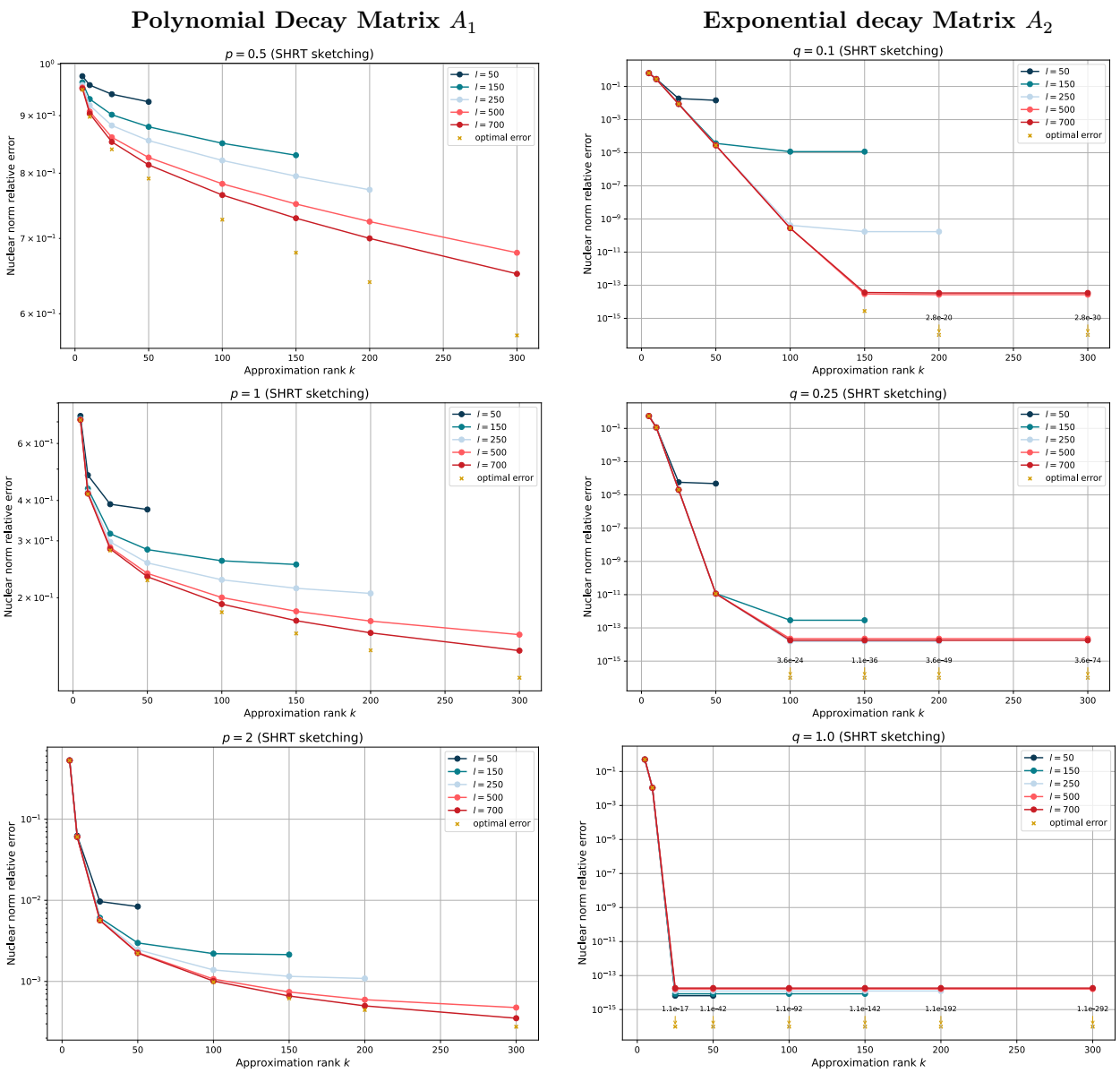

Figure 16: Stability analysis for polynomial (left) and exponential (right) matrices with **SRHT** sketching