

# ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

## PROJECT N°2 REPORT

RANDOMIZED NYSTRÖM

---

---

*Authors:*

Tara FJELLMAN

Amal SEDDAS

*Professor:*

Laura GRIGORI

The EPFL logo is rendered in a bold, red, sans-serif typeface. The letters are thick and blocky, with the 'E' and 'F' featuring a distinctive stepped or 'Z' shape on their right-hand sides.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Randomized Nyström low rank approximation</b>	<b>1</b>
<b>3</b>	<b>Sketching and Sketching Matrices</b>	<b>2</b>
3.1	Gaussian sketching . . . . .	3
<b>4</b>	<b>Block Subsampled Randomized Hadamard Transform (BSRHT)</b>	<b>3</b>
4.1	Subsampled Randomized Hadamard Transform (SRHT) . . . . .	3
<b>5</b>	<b>Stability analysis</b>	<b>4</b>
<b>6</b>	<b>Parallelisation</b>	<b>4</b>
<b>7</b>	<b>Experimental procedure</b>	<b>6</b>
<b>8</b>	<b>Algorithm Performance</b>	<b>6</b>
8.1	Sequential Performance . . . . .	6
8.2	Parallel Performance . . . . .	7
<b>9</b>	<b>Conclusion</b>	<b>8</b>

# 1 Introduction

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

## 2 Randomized Nyström low rank approximation

The goal of the Randomized Nyström Algorithm is to represent a given matrix  $A \in \mathbb{R}^{n \times n}$  by some approximated matrix with a lower rank  $k < n$ . The starting point of the algorithm is a rank- $\ell$  approximation of  $A$ , in the form

$$A_{Nyst} := (A\Omega)(\Omega^T A\Omega)^\dagger(\Omega^T A) \in \mathbb{R}^{n \times n},$$

where  $(\Omega^T A\Omega)^\dagger$  is the pseudoinverse of  $\Omega^T A\Omega$  and  $\Omega \in \mathbb{R}^{n \times \ell}$  is a sketching matrix. We assume  $k < \ell < n$ . To further reduce the rank of  $A_{Nyst}$  to  $k$ , we have two choices: approximate only the core matrix  $\Omega^T A\Omega$  or the whole  $A_{Nyst}$ . In this project, we focus on the second approach. Algorithm 1 shows how this can be achieved.

---

**Algorithm 1** Randomized Nyström approximation using the Cholesky decomposition

---

- 1: Let  $C = A\Omega$  and  $B = \Omega^T C$ .
- 2: Compute the Cholesky decomposition:  $B = LL^T$ .
- 3: Solve the linear system using back substitution:  $Z = C(L^T)^{-1}$ .
- 4: Perform QR decomposition:  $Z = QR$ .
- 5: Compute the singular value decomposition (SVD) of  $R$  and truncate:

$$R = U\Sigma V^T \approx \tilde{U}_k \tilde{\Sigma}_k \tilde{V}_k^T.$$

- 6: Set  $\tilde{U}_k = Q\tilde{U}_k$  (or equivalently  $\tilde{U}_k = Z\tilde{V}_k\tilde{\Sigma}_k^{-1}$ ) and return  $\tilde{U}_k \tilde{\Sigma}_k^2 \tilde{U}_k^T \approx A_{Nyst}$ .
- 

To show why the result of Algorithm 1 is in fact a rank- $k$  approximation of

$A_{Nyst}$ , we write

$$\begin{aligned}\tilde{U}_k \tilde{\Sigma}_k^2 \tilde{U}_k^T &= Q \tilde{U}_k \tilde{\Sigma}_k \tilde{\Sigma}_k^T \tilde{U}_k^T Q^T \\ &\approx Q R R^T Q^T = Z Z^T = A \Omega (L^T)^{-1} (L^T)^{-1} \Omega^T A^T \\ &= A \Omega (L L^T)^{-1} \Omega^T A^T = A (\Omega^T A \Omega)^\dagger \Omega^T A^T.\end{aligned}$$

However, here we have the inverse of  $\Omega^T A \Omega$  instead of its pseudoinverse. Unfortunately, Algorithm 1 fails in Step 2 if  $\Omega^T A \Omega$  is numerically singular. In this case, as in [?], we replace  $L$  by a square root of  $B$  in SVD form. Specifically, we write  $B = U_B \Sigma_B U_B^T$  (since  $B$  is SPSP), and set  $L = U_B \sqrt{\Sigma_B} U_B^T$ . By construction, we have

$$L L^T = U_B \sqrt{\Sigma_B} U_B^T (U_B \sqrt{\Sigma_B} U_B^T)^T = U_B \Sigma_B U_B^T = B.$$

We then replace  $(L^T)^{-1}$  with  $(L^T)^\dagger$ . This can be simply calculated as

$$(L^T)^\dagger = (U_B \sqrt{\Sigma_B} U_B^T)^\dagger,$$

where  $\sqrt{\Sigma_B}$  is a diagonal matrix whose entries are given by

$$(\sqrt{\Sigma_B})_{i,i} = \begin{cases} \frac{1}{(\sqrt{\Sigma_B})_{i,i}} & \text{if } (\sqrt{\Sigma_B})_{i,i} \neq 0, \\ 0 & \text{otherwise.} \end{cases}$$

### 3 Sketching and Sketching Matrices

In this section, we provide a detailed overview of sketching and the sketching matrices utilized in this project. The concept of sketching involves transforming a high-dimensional matrix  $A \in \mathbb{R}^{n \times n}$  into a lower-dimensional representation  $A \Omega$ , where  $\Omega \in \mathbb{R}^{n \times \ell}$  is a tall and skinny sketching matrix. The goal of sketching is to embed the high-dimensional data into a reduced-dimensional space while preserving the essential geometric properties of the data.

More formally, given any two vectors  $x$  and  $y$  in the high-dimensional space, their sketched counterparts  $\hat{x}$  and  $\hat{y}$  should approximately preserve their inner product:

$$|\langle \hat{x}, \hat{y} \rangle - \langle x, y \rangle| \leq \varepsilon \|x\|_2 \|y\|_2 \quad (2)$$

where  $\varepsilon > 0$  is a small approximation error. However, achieving this exact preservation for all  $x$  and  $y$  is generally infeasible due to the reduced dimensionality  $\ell$ . Instead,  $\Omega$  is typically chosen randomly, ensuring that eq. (2) holds with high probability  $1 - \delta$ , where  $\delta < 1$ .

In this project, we employ two specific types of sketching matrices:

### 3.1 Gaussian sketching

The Gaussian sketching matrix is a widely used random projection method in which the entries of the sketching matrix  $\Omega$  are drawn independently from a standard normal distribution. Mathematically, it is defined as:

$$\Omega_{ij} \sim \mathcal{N}(0, 1),$$

where  $\Omega \in \mathbb{R}^{n \times \ell}$ , and  $\ell$  represents the reduced dimensionality of the sketch.

The sketched matrix  $A\Omega$ , for an input matrix  $A \in \mathbb{R}^{n \times d}$ , satisfies approximate preservation of the geometry of the original matrix. Specifically, the sketch guarantees that for any vector  $x \in \mathbb{R}^d$ :

$$(1 - \varepsilon)\|Ax\|_2^2 \leq \|A\Omega x\|_2^2 \leq (1 + \varepsilon)\|Ax\|_2^2,$$

with high probability, where  $\varepsilon > 0$  is a small approximation error. This property ensures that pairwise distances between points in the projected space are approximately preserved.

The Gaussian sketching matrix can also be viewed as a random transformation that approximately satisfies the following subspace embedding property for any subspace  $T \subseteq \mathbb{R}^d$ :

$$(1 - \varepsilon)\|v\|_2^2 \leq \|\Omega v\|_2^2 \leq (1 + \varepsilon)\|v\|_2^2, \quad \forall v \in T.$$

## 4 Block Subsampled Randomized Hadamard Transform (BSRHT)

### 4.1 Subsampled Randomized Hadamard Transform (SRHT)

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

$$\Omega^T = \sqrt{\frac{n}{\ell}} R H D,$$

where:

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

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

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

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

$$\Omega^T = \begin{pmatrix} \Omega_1^T \\ \Omega_2^T \\ \vdots \\ \Omega_P^T \end{pmatrix} = \sqrt{\frac{n}{P\ell}} (D_{L1} \quad \cdots \quad D_{LP}) \begin{pmatrix} RH & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & RH \end{pmatrix} \begin{pmatrix} D_{R1} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & D_{RP} \end{pmatrix}, \quad (2)$$

with:

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

This structure is particularly suited for distributed computations, as it allows for parallelism while maintaining the theoretical properties of the SRHT.

## 5 Stability analysis

- bullet points about comments to include

## 6 Parallelisation

We parallelise the algorithm by distributing among the processors the computation of  $A\Omega$ ,  $\Omega^T A\Omega$ , and all other matrix operations with complexities at least proportional to  $n$ .

The parallelisation of QR factorisation was the topic of the first project. We therefore included the relevant functions for it in the code folder.

The parallelisation of matrix products was however implemented from scratch. It was done as described in algorithm 2, by distributing matrices smartly among the processors.

---

**Algorithm 2** Computes the matrix product of two matrices  $D$  and  $E$  in parallel.

---

**Require:**  $q$  is an integer and the square root of the number of processors,  $D$  is an  $m \times m$  matrix distributed such that processor  $i$  has  $D_{i//q, i \bmod q}$  and  $E$  is a  $m \times n$  matrix distributed such that processor  $i$  has  $E_{i//q}$ . FullProd is true if we also want to compute  $E^T D E$ .

**Ensure:**  $F = DE$  (and  $G = E^T F$  too if FullProd is true).

$F_{ij} \leftarrow P_{ij} E_j$

**if** FullProd **then**

$G_{ij} \leftarrow E_i^T P_{ij}$

**end if**

Row-wise Sum-reduce :  $F_i \leftarrow \sum_j^q F_{ij}$

Column-wise Gather on rank 0 :  $F \leftarrow [F_1^T, \dots, F_q^T]^T$

**if** FullProd **then**

Sum-reduce :  $G \leftarrow \sum_{i,j}^{q,q} G_{ij}$

**end if**

---

This version of the matrix product allows for a fluid computation of both the  $A\Omega, \Omega^T A\Omega$  products involved in the sketching and general matrix products between two matrices  $D$  and  $E$  which appear in the computation of the rank- $k$  approximation.

The pseudo-code for the parallelisation of the randomized Nyström algorithm is then described in algorithm 3.

---

**Algorithm 3** Randomized Nyström algorithm. The syntax was adapted from this Overleaf example.

---

**Require:**  $A$  is an  $n \times n$  symmetric positive semidefinite matrix,  $\Omega$  is a sketching matrix of size  $n \times l$ , and  $k$  is the rank of the approximation

**Ensure:**  $[A_{Nyst}]_k$ , the rank- $k$  randomized Nyström approximation of  $A$ .

$C \leftarrow A\Omega$

$B \leftarrow \Omega^T C$

$L, \text{Failed} \leftarrow \text{Cholesky}(B)$

▷ Locally on rank 0

**if** Failed **then**

$U, \Lambda \leftarrow \text{EigDecomp}(B)$

▷ Locally on rank 0

$B_k^+ \leftarrow U(:, 1:k) \Lambda(1:k, 1:k)^+ U(:, 1:k)^T$

$Q, R \leftarrow \text{QR}(C)$

▷ Using TSQR

$\hat{U}_k \leftarrow QU(:, 1:k)$

▷ In parallel

$[A_{Nyst}]_k \leftarrow \hat{U}_k \Lambda(1:k, 1:k)^+ \hat{U}_k^T$

▷ In parallel

**else**

$Z \leftarrow CL^{-T}$

▷ Computed by substitution :  $(LZ^T)^T = C^T$

$Q, R \leftarrow \text{QR}(Z)$

▷ Using TSQR

$U_k, \Sigma_k, V_k \leftarrow \text{TruncSVD}(R)$

$\hat{U}_k \leftarrow QU(:, 1:k)$

▷ In parallel

$[A_{Nyst}]_k \leftarrow \hat{U}_k \Sigma_k^2(1:k, 1:k) \hat{U}_k^T$

▷ In parallel

**end if**

---

## 7 Experimental procedure

- averaging to increase precision of results - when running in parallel, was limited by number of cores  $p = 2^{2s}$  with  $s \in \mathbb{N}$  (due to perfect square constraint of the parallelised matrix multiplication and the SHRT algorithm using the Hadamard transform), so we ran results for 1,4,16,64 - ran results on the Helvetios cluster -  $k$  was taken often around  $n/20$  as we deemed this value relevant for real applications -  $n$  was made vary as powers of 2 due to the Hadamard transformation while  $l$  was taken logarithmically spaced just to make the visuals more easy to interpret

## 8 Algorithm Performance

### 8.1 Sequential Performance

- in general (both plots) observe that  $k$  rank approximation represents minority of runtime (as expected as of order  $k^3 + nk^2$ ). Indeed for shown examples represents at most 10%. This would be different if one were to take big values of  $k$ . Also does not really seem to depend on sketching method which makes sense given



the computations are the same regardless of which sketching method was used to obtain the B and C matrices

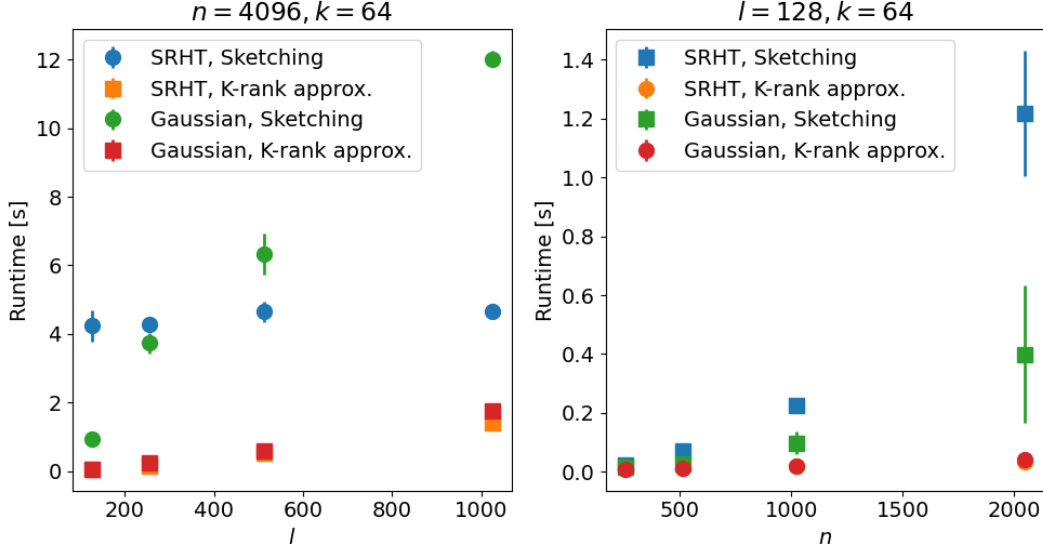


Figure 1: Runtimes associated to the tested sketching methods : as a function of  $l$  (left) and as a function of  $n$  (right). Runtimes are broken down in that associated to the computation of  $A\Omega, \Omega^T A\Omega$  and that associated to the computation of the  $k$  rank approximation.

- different behaviours as a function of  $l$  and  $n$  : as a function of  $l$  SRHT looks almost flat, having really slow increase, while gaussian increases steadily given its complexity is of order  $nl^2 + ln^2$ . This gives advantage for SRHT if we are embedding "dense spaces" as it scales much better than gaussian as function of  $l$ . For  $n$  : both algos show significant increase, but SRHT shows faster runtime growth. This makes sense due to extra  $\log_2(n)$  term in its complexity. This would give advantage to gaussian sketching if we are dealing with sparse information within really high dimensional spaces. -

## 8.2 Parallel Performance

- selected  $l = n/64$  given we are using TSQR for the QR factorisation and we need the matrix to be tall-skinny to actually observe a speed-up. To have a good speed-up for a bigger  $l$ , while keeping numerical stability for ill-conditioned matrices another algo should be used.

- for small example : some speed-up is observed as a function of number of cores but communication costs are non neglectable and even dominate the total runtime for  $p=64$ . This is expected as the computation is rather small to be run

in parallel. The runtimes for the k-rank are the most affected as these represented the smallest part of the computation to start with. for the specific values selected the gaussian sketching is faster (indeed  $l$  is taken only as a small fraction of  $n$ ).

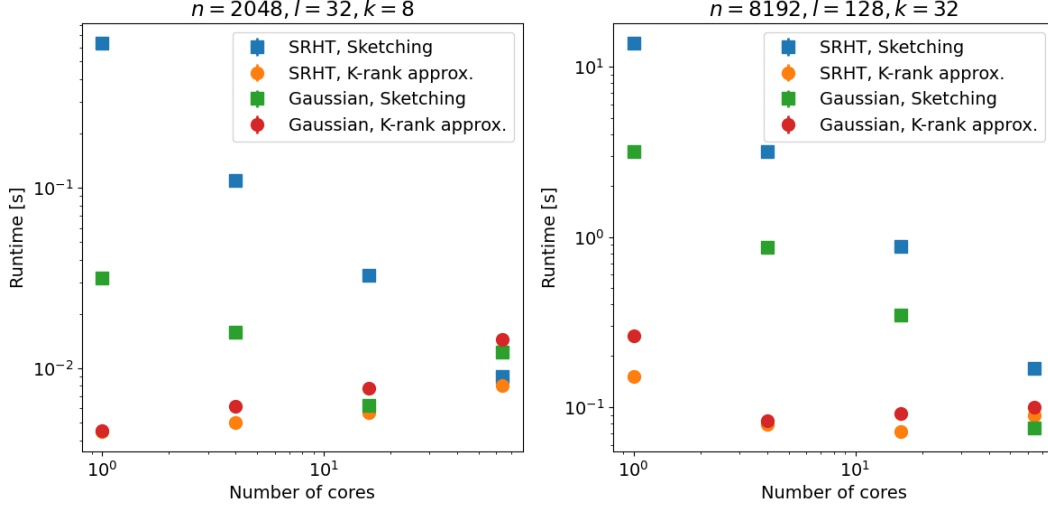


Figure 2: Runtimes associated to the tested sketching methods as a function of the number of cores : for a “small” example (left) and a “large” one (right). Runtimes are broken down in that associated to the computation of  $A\Omega, \Omega^T A\Omega$  and that associated to the computation of the  $k$  rank approximation.

- for the large : communication is not much of an issue, and big speed ups (quasi linear) are observed for the sketching computation. This means the parallelization is successful and scales well for big computations. We can see the speedup is not as good for k-rank partly because the computation was smaller to start with. The fact that the matrix is not that tall-skinny is also expected to have contributed to the low speed-up of that part. -

## 9 Conclusion

## Aknowledegments

## References