On the Accuracy and Robustness of Randomised Methods for Matrix Computations

Taejun Park

Worcester College

University of Oxford

A thesis submitted for

Doctor of Philosophy

Trinity 2025

Acknowledgements

First and foremost, I would like to express my deepest gratitude to my supervisor, Yuji Nakatsukasa, for his unwavering support and guidance. His passion for research, boundless enthusiasm, and never-ending ideas have been a constant source of inspiration and motivation throughout my DPhil. I truly owe the researcher I am today to his mentorship.

I am grateful to my examiners, Hussam Al Daas and Stefano Massei, for kindly agreeing to examine my thesis.

I would also like to express my sincere thanks to the professors who have inspired and supported me along the way: Nick Trefethen, Per-Gunnar Martinsson, Coralia Cartis, Jared Tanner, Andy Wathen, Patrick Farrell, Kathryn Gillow, Endre Süli, and Laura Grigori. Their insights and encouragement have played an essential role in shaping my academic journey.

I am thankful to the Heilbronn Institute for Mathematical Research for sponsoring my DPhil and supporting my research.

To my friends in Oxford and those I have met at conferences–India, Aaron, Umberto, Pablo, Kars, Mingdong, Irina, Ryan, Alan, Malena, Nicolas, Alice, Astrid, Ethan, Alberto, and Diana–thank you for your companionship and the many memorable moments we shared. Each of you has contributed to making this journey truly unforgettable. I am especially grateful to Maike, Charlie, Boris, Karl, Lorenzo, Nathaniel, and Jung Eun–your friendship and support have been invaluable. I know that my journey would not have been the same without each and every one of you.

Finally, and most importantly, I want to thank my family–my mum Kyung Mi, my dad Han Young, my brother Yongjun, and my grandparents–for their love, support, and encouragement over the years. Their boundless belief in me has been a constant source of strength, and I am truly grateful for everything they have done. I would not be who I am today without them.

Abstract

Randomised numerical linear algebra provides scalable and robust tools for tackling large-scale problems in the computational sciences. This thesis investigates the accuracy, stability, and robustness of various randomised algorithms, with a particular focus on low-rank matrix approximations.

We begin with a sketch-and-solve approach for approximating trailing right singular vectors of a tall matrix. Using multiplicative perturbation theory, we show that accuracy depends on the relative spectral gap. To improve efficiency, we introduce a sketch update technique that handles matrix modifications without recomputing the full sketch.

Next, we explore the Nyström method for symmetric indefinite matrices. We identify a key source of instability in the standard approach, caused by inaccurate singular value estimation in the core matrix. To address this, we propose a robust variant that truncates the core matrix based on the target rank–representing, to our knowledge, the first stable Nyström method designed for symmetric indefinite matrices.

The rest of the thesis focuses on the CUR decomposition, where a matrix is approximated using a subset of its rows and columns. We analyse both its accuracy and numerical stability, emphasising the role of dependent index selection and the benefits of oversampling. Our theoretical results lead to a numerically stable CUR implementation, highlighting how oversampling mitigates the effects of poorly chosen indices.

Finally, we introduce two efficient, rank-adaptive algorithms–AdaCUR and FastAdaCUR–for low-rank approximation of parameter-dependent matrices via CUR decomposition. These methods aim to reuse previously selected indices across parameter values, thereby greatly reducing computational cost. While AdaCUR offers adaptive rank selection and error control, FastAdaCUR achieves faster runtimes at the expense of robustness.

Notations

We use MATLAB style notation for matrices and vectors. For example, for the ith to jth columns of a matrix $\boldsymbol{A}$, we write $\boldsymbol{A}(:, i:j)$. All matrices and vectors in this thesis are boldfaced. Below is a list of commonly used notations.

Notation	**Description**
k	target rank or number of singular vectors approximated
$[\![\boldsymbol{A}]\!]_k$	best rank-k approximation to $\boldsymbol{A}$
$\boldsymbol{U}$	left singular vectors
$\boldsymbol{\Sigma}$	diagonal matrix of singular values
$\boldsymbol{V}$	right singular vectors
$\sigma_i(\boldsymbol{A})$	ith largest singular value of $\boldsymbol{A}$
$\lambda_i(\boldsymbol{A})$	ith largest eigenvalue of $\boldsymbol{A}$ in magnitude
$\boldsymbol{A}^\dagger$	Moore-Penrose pseudoinverse of matrix $\boldsymbol{A}$
$\boldsymbol{I}_n$	$n \times n$ identity matrix
$\boldsymbol{Q}, \boldsymbol{W}$	orthonormal matrix
$\mathrm{nnz}(\boldsymbol{A})$	number of nonzero elements in matrix $\boldsymbol{A}$
u	unit roundoff, $u \approx 10^{-16}$ in double-precision
$\|\cdot\|_2$	spectral norm
$\|\cdot\|_{\mathrm{F}}$	Frobenius norm
$\|\cdot\|_*$	nuclear/trace norm
$\|\cdot\|$	unitarily invariant norm

Notation	Description
s	sketch size
$\boldsymbol{S}$	subspace embedding
$\boldsymbol{S}_G$	Gaussian embedding
$\boldsymbol{S}_T$	subsampled randomized trigonometric transform (SRTT)
$\boldsymbol{S}_S$	sparse map
$\boldsymbol{X}$	row sketch of $\boldsymbol{A}$, $\boldsymbol{X} = \boldsymbol{S}^\top \boldsymbol{A}$
$\boldsymbol{Y}$	column sketch of $\boldsymbol{A}$, $\boldsymbol{Y} = \boldsymbol{A}\boldsymbol{S}$
$\boldsymbol{\mathcal{P}}$	projection matrix, $\boldsymbol{\mathcal{P}}^2 = \boldsymbol{\mathcal{P}}$
$\boldsymbol{\mathcal{P}}_{\boldsymbol{Y}}$	orthogonal projector, $\boldsymbol{\mathcal{P}}_{\boldsymbol{Y}} = \boldsymbol{Y}\boldsymbol{Y}^\dagger$
$\boldsymbol{\mathcal{P}}_{\boldsymbol{Y},\boldsymbol{G}}$	oblique projector, $\boldsymbol{\mathcal{P}}_{\boldsymbol{Y},\boldsymbol{G}} = \boldsymbol{Y}(\boldsymbol{G}^\top \boldsymbol{Y})^\dagger \boldsymbol{G}^\top$
$\boldsymbol{C}$	column subset of a matrix
$\boldsymbol{R}$	row subset of a matrix,
	or upper triangular factor in the QR decomposition
$\boldsymbol{M}$	middle matrix, intersection of the columns $\boldsymbol{C}$ and rows $\boldsymbol{R}$
I	row index set
J	column index set
$\lvert I \rvert$	cardinality of the index set I
$\boldsymbol{\Pi}_J$	$\boldsymbol{\Pi}_J = \boldsymbol{I}_n(:, J)$ for index set J
$\boldsymbol{A}_{IJ}$	$\boldsymbol{A}_{IJ} = \boldsymbol{A}(:, J)\boldsymbol{A}(I, J)^\dagger \boldsymbol{A}(I, :) = \boldsymbol{A}\boldsymbol{\Pi}_J(\boldsymbol{\Pi}_I^\top \boldsymbol{A}\boldsymbol{\Pi}_J)^\dagger \boldsymbol{\Pi}_I^\top \boldsymbol{A}$
$[n]$	$[n] = \{1, 2, ..., n\}$
$T_{\boldsymbol{A}}$	cost of matrix-vector product with $\boldsymbol{A}$ or $\boldsymbol{A}^\top$
$T_{sketch}^{(s)}$	cost of sketching with sketch size s

Abbreviations

Abbreviation	**Expansion**
NLA	Numerical Linear Algebra
RandNLA	Randomised Numerical Linear Algebra
SRTT	Subsampled Randomised Trigonometric Transforms
SVD	Singular Value Decomposition, $\boldsymbol{U\Sigma V}^\top$
TSVD	Truncated Singular Value Decomposition, $\boldsymbol{U}_k\boldsymbol{\Sigma}_k\boldsymbol{V}_k^\top$
SPSD	Symmetric Positive Semi-Definite
CSSP	Column Subset Selection Problem
CPQR	Column-Pivoted QR
LUPP	LU with Partial Pivoting
CUR-BA	CUR with Best Approximation, $\boldsymbol{CC}^\dagger\boldsymbol{AR}^\dagger\boldsymbol{R}$
CUR-CA	CUR with Cross Approximation, $\boldsymbol{CM}^\dagger\boldsymbol{R}$
SCUR-CA	Stabilised CUR with Cross Approximation, $\boldsymbol{CM}_\epsilon^\dagger\boldsymbol{R}$

Contents

1

Introduction

Numerical linear algebra (NLA) is at the core of computational sciences. In many computing platforms, we can now routinely solve small- to medium-scale linear algebra problems (typically involving up to 10^6 unknowns) reliably and robustly [Dem97, ABB+99, GVL13].

In modern computational sciences, the volume of available data is growing at a remarkable rate. Despite advancements in computational resources, many classical NLA algorithms struggle to keep up with the sheer volume of data, rendering them increasingly prohibitive. Given these computational challenges, there is an increasing demand for algorithms capable of addressing large-scale problems. One promising framework is to introduce *randomisation* to algorithms, giving rise to the field of randomised numerical linear algebra [WLRT08, HMT11, Woo14, MT20].

Randomised numerical linear algebra (RandNLA) is a rapidly emerging field that uses

probabilistic techniques to reduce the computational complexity of many NLA algorithms, making them more efficient than classical NLA algorithms. The central idea in RandNLA is to *embed* vectors or matrices onto a *lower-dimensional space*, where the problem can be tackled more efficiently. For instance, if a matrix exhibits underlying low-dimensional properties such as sparsity or low-rank, we can compress it while retaining its essential information. This technique of embedding a high-dimensional vector onto a lower dimension is called *sketching*. The aim of sketching is to retain as much information as possible from the original matrix, thereby creating a smaller representation that serves as a good surrogate for a given problem. Sketching has proven to be a powerful tool in various fundamental linear algebraic tasks such as low-rank approximation [WS00, HMT11], least-squares problems [Sar06, RT08], linear systems [ARS21, BG22, FTU23, NT24], and eigenvalue problems [BG22, NT24].

RandNLA algorithms offer various advantages that allow us to effectively leverage computational resources [MDM+23]. For example, randomised algorithms scale well in parallel and distributed environments [YMM16], exploit cloud-computing architectures [MSM14], and account for computational constraints, such as when data are streamed [CW09, TYUC19]. However, such efficiency gains come with trade-offs, often at the cost of some loss of accuracy. Nevertheless, for very large matrices, sacrificing accuracy for speed is often acceptable. Therefore, it is crucial to balance the two when tackling problems using RandNLA algorithms.

1.1 Outline

This thesis aims to analyse and design RandNLA algorithms, focusing on their accuracy, robustness, and stability. We concern ourselves with two topics in numerical linear algebra: singular value and singular vector estimation, and low-rank approximation, with an emphasis on low-rank approximation. All experiments were performed in MATLAB version 2021a using double precision arithmetic on a workstation with Intel® Xeon® Silver 4314 CPU @ 2.40GHz (16

cores) and 512GB memory. Below we briefly outline each chapter and its main contributions.

In Chapter 2, we introduce the foundational concepts in classical NLA and RandNLA that are integral to the thesis. This chapter provides essential background material to aid in understanding the key ideas explored later. Specifically, we discuss subspace embeddings, low-rank approximation, and column subset selection problem. These topics form the theoretical framework necessary for the subsequent developments in the thesis.

In Chapter 3, we examine a fast algorithm for computing an approximate null space of a tall-skinny matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ ($m \gg n$), specifically aiming to approximate the trailing right singular subspace of $\boldsymbol{A}$. Using the sketch-and-solve paradigm, we obtain an approximate null space, and its accuracy is analysed using multiplicative perturbation theory, extending the work in [GPW12] to subspaces. The effectiveness and accuracy of this algorithm are demonstrated through numerical experiments using the total least squares problem, and the AAA algorithm for rational approximation. In addition, we propose a method for efficiently updating the sketch when a column or row update/downdate is made to the original matrix, which is particularly useful for problems like the AAA algorithm where the matrix undergoes such updates or downdates. This chapter is based on the paper [PN23]:

- **A fast randomized algorithm for computing an approximate null space**, Taejun Park and Yuji Nakatsukasa, *BIT Numerical Mathematics*, 2023

Chapter 4 examines the Nyström method, a widely used algorithm for computing a low-rank approximation of a symmetric positive semidefinite matrix, and explores how it can be adapted for symmetric indefinite matrices. The Nyström method can become unstable and may fail when applied to symmetric indefinite matrices, potentially leading to unbounded errors. We begin by identifying the key challenges in finding a stable Nyström approximation for symmetric indefinite matrices, and prove the existence of a variant that overcomes the instability. In particular, we establish a relative-error nuclear norm bound

that holds when the singular values decay rapidly. This analysis naturally leads to the development of a practical algorithm, whose robustness is demonstrated through experiments. This chapter is based on the paper [NP23]:

- **Randomized Low-Rank Approximation for Symmetric Indefinite Matrices**, Taejun Park and Yuji Nakatsukasa, *SIAM Journal on Matrix Analysis and Applications*, 2023

In Chapters 5 and 6, we investigate the accuracy and numerical stability of CUR decompositions with oversampling. The CUR decomposition approximates a matrix by selecting a subset of its columns and rows. However, because the low-rank representation involves a matrix inverse, careful implementation is necessary to preserve accuracy, and prevent potential numerical instability. In Chapter 5, we show that the CUR decomposition can be implemented in a numerically stable manner by proving a numerical stability result. Furthermore, in Chapter 6, we show that oversampling, which increases the number of either columns or rows in the CUR decomposition can be employed to improve both the accuracy and stability of the decomposition. This analysis leads to an oversampling algorithm, whose competitiveness is illustrated through experiments. Chapters 5 and 6 are based on the paper [PN25a]:

- **Accuracy and Stability of CUR decompositions with Oversampling**, Taejun Park and Yuji Nakatsukasa, *SIAM Journal on Matrix Analysis and Applications*, 2025

In Chapter 7, we present efficient randomised algorithms for low-rank approximations of parameter-dependent matrices $\boldsymbol{A}(t)$ using the CUR decomposition. The key idea is that for nearby parameter values, the same row and column indices can often be reused. Our algorithms retain or minimally update these indices as the parameter changes, recomputing them only when the approximation error exceeds a predefined error tolerance. The resulting algorithm is rank-adaptive and incorporates error control, allowing it to meet the specified error tolerance.

Its complexity is competitive with existing methods. We also introduce a faster alternative that forgoes error control in favour of speed–still rank-adaptive, but with a complexity at most linear in the number of rows and columns. This chapter is based on the paper [PN25b]:

- **Low-rank approximation of parameter-dependent matrices via CUR decomposition**, Taejun Park and Yuji Nakatsukasa, *SIAM Journal on Scientific Computing*, 2025

Finally, we conclude the thesis in Chapter 8 with a summary and a discussion.

2

Preliminaries

In this section, we review key ideas and results in numerical linear algebra (NLA) and randomised numerical linear algebra (RandNLA) that complement this thesis. We focus on four main concepts: (i) the singular value decomposition (SVD), a fundamental matrix decomposition in NLA; (ii) *subspace embeddings*, which are the building blocks of many RandNLA algorithms that allow us to preserve geometric structure when embedding data into lower dimensions; (iii) *low-rank approximations*, which aim to efficiently represent a matrix using matrices of lower rank; and (iv) the *column subset selection problem*, which identifies representative columns of a matrix to form a near-optimal low-rank approximation.

2.1 Singular value decomposition

The singular value decomposition (SVD) of a matrix $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ with $m \geq n$ is a matrix factorisation of the form

$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top = \sum_{i=1}^{n} \sigma_i(\boldsymbol{A})\, \boldsymbol{u}_i \boldsymbol{v}_i^\top,$$

where $\boldsymbol{U} \in \mathbb{R}^{m\times n}$ is an orthonormal matrix of left singular vectors, $\boldsymbol{\Sigma} \in \mathbb{R}^{n\times n}$ is a diagonal matrix containing the singular values $\sigma_1(\boldsymbol{A}) \geq \cdots \geq \sigma_n(\boldsymbol{A}) \geq 0$, and $\boldsymbol{V} \in \mathbb{R}^{n\times n}$ is an orthogonal matrix of right singular vectors. Here, $\boldsymbol{u}_i$'s and $\boldsymbol{v}_i$'s are the ith columns of left and right singular vectors, respectively. This form is often called the thin SVD.

Alternatively, by completing the orthonormal basis of $\boldsymbol{U}$, the SVD can be expressed as

$$\boldsymbol{A} = [\boldsymbol{U}, \boldsymbol{U}_\perp] \begin{bmatrix} \boldsymbol{\Sigma} \\ \boldsymbol{0} \end{bmatrix} \boldsymbol{V}^\top,$$

where $[\boldsymbol{U}, \boldsymbol{U}_\perp] \in \mathbb{R}^{m\times m}$ is an orthogonal matrix. This version is commonly referred to as the full SVD of $\boldsymbol{A}$. In this thesis, we focus on the thin SVD.

When $\boldsymbol{A}$ has rank-r where $r < n$, the singular values satisfy $\sigma_1(\boldsymbol{A}) \geq \cdots \geq \sigma_r(\boldsymbol{A}) > 0 = \sigma_{r+1}(\boldsymbol{A}) = \cdots = \sigma_n(\boldsymbol{A})$ and the thin SVD can be written compactly as,

$$\boldsymbol{A} = \boldsymbol{U}_r\boldsymbol{\Sigma}_r\boldsymbol{V}_r^\top = \sum_{i=1}^{r} \sigma_i(\boldsymbol{A})\, \boldsymbol{u}_i \boldsymbol{v}_i^\top,$$

where $\boldsymbol{U}_r = [\boldsymbol{u}_1, ..., \boldsymbol{u}_r] \in \mathbb{R}^{m\times r}$, $\boldsymbol{\Sigma}_r = \mathrm{diag}(\sigma_1(\boldsymbol{A}), ..., \sigma_r(\boldsymbol{A}))$, and $\boldsymbol{V}_r = [\boldsymbol{v}_1, ..., \boldsymbol{v}_r] \in \mathbb{R}^{n\times r}$. This compact form is particularly useful for data compression [EY36, GVL13] and will be revisited in the discussion of low-rank approximation in Section 2.3.

The Moore-Penrose pseudoinverse (or simply the pseudoinverse) of a rank-r matrix

$\boldsymbol{A} \in \mathbb{R}^{m\times n}$, denoted by $\boldsymbol{A}^\dagger \in \mathbb{R}^{n\times m}$, can be formulated in terms of the thin SVD as

$$\boldsymbol{A}^\dagger = \boldsymbol{V}_r \boldsymbol{\Sigma}_r^{-1} \boldsymbol{U}_r^\top.$$

Moreover, the pseudoinverse satisfies $\boldsymbol{A}^\dagger = (\boldsymbol{A}^\top \boldsymbol{A})^\dagger \boldsymbol{A}^\top$, which can be verified using the SVD:

$$(\boldsymbol{A}^\top \boldsymbol{A})^\dagger \boldsymbol{A}^\top = (\boldsymbol{V}_r \boldsymbol{\Sigma}_r^2 \boldsymbol{V}_r^\top)^\dagger \boldsymbol{V}_r \boldsymbol{\Sigma}_r \boldsymbol{U}_r^\top = \boldsymbol{V}_r \boldsymbol{\Sigma}_r^{-2} \boldsymbol{V}_r^\top \boldsymbol{V}_r \boldsymbol{\Sigma}_r \boldsymbol{U}_r^\top = \boldsymbol{V}_r \boldsymbol{\Sigma}_r^{-1} \boldsymbol{U}_r^\top = \boldsymbol{A}^\dagger.$$

2.2 Subspace embeddings

A central idea in RandNLA is to embed a high-dimensional problem into a space of lower dimension, while approximately preserving the geometry of the original space. Our hope is that by projecting onto a lower dimensional space, we are able to well-approximate the solution to the original problem using fewer computational resources. A subspace embedding is a linear map that achieves this by preserving the 2-norm of every vector in a given subspace. The definition below follows [Woo14, MT20], and originates from the work of Sarlós [Sar06], which uses the Johnson–Lindenstrauss transform.

Definition 2.2.1 (Subspace embedding)**.** *Let $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ be a matrix of rank $r \leq \min\{m, n\}$. Then $\boldsymbol{S} \in \mathbb{R}^{m\times s}$ $(r \leq s)$ is a* subspace embedding *for $\boldsymbol{A}$ with distortion $0 < \eta < 1$ if*

$$(1-\eta)\left\|\boldsymbol{A}\boldsymbol{x}\right\|_2 \leq \left\|\boldsymbol{S}^\top \boldsymbol{A}\boldsymbol{x}\right\|_2 \leq (1+\eta)\left\|\boldsymbol{A}\boldsymbol{x}\right\|_2 \tag{2.1}$$

for all $\boldsymbol{x} \in \mathbb{R}^n$ where $\|\cdot\|_2$ is the 2-norm.

Subspace embeddings are the building blocks of many RandNLA algorithms and we typically have $r \leq s \ll \min\{m, n\}$. *Sketching* is a technique where we multiply $\boldsymbol{A}$ by its subspace embedding $\boldsymbol{S}$, i.e. $\boldsymbol{S}^\top \boldsymbol{A}$ or $\boldsymbol{A}\boldsymbol{S}$, to preserve certain quantities of $\boldsymbol{A}$ in a lower

dimensional subspace, for example, the column space of $\boldsymbol{A}$ or the norm of $\boldsymbol{A}$. In Chapter 3, we analyse how the row sketch $\boldsymbol{S}^\top \boldsymbol{A}$ approximately preserves the singular values and the right singular subspace of a tall matrix $\boldsymbol{A}$.

Many subspace embeddings are constructed using randomness. We refer to these as *random embeddings*, which are embeddings drawn at random that, with high probability, satisfy Equation (2.1) for any matrix $\boldsymbol{A}$. Random embeddings are oblivious (data-independent) as they are designed to work for an arbitrary matrix $\boldsymbol{A}$ with high probability without seeing $\boldsymbol{A}$. There are also subspace embeddings that are data-dependent such as leverage score sampling [MD09, DMIMW12]. In many settings, we do not know the column/row space of $\boldsymbol{A}$ in advance, making it difficult to predict how $\boldsymbol{S}$ would interact with the column/row space of $\boldsymbol{A}$. Therefore it is much easier to use randomness to construct an embedding that satisfies (2.1) with high probability for any subspace. Random embeddings are closely related to various concepts like the restricted isometry property [CT06] and the Johnson-Lindenstrauss lemma [JL84]. Below, we discuss a few important examples of random embeddings.

2.2.1 Gaussian embeddings

A Gaussian embedding is a random matrix $\boldsymbol{S}_G \in \mathbb{R}^{m\times s}$ with independent and identically distributed (i.i.d.) standard normal entries $[\boldsymbol{S}_G]_{ij} \sim \mathcal{N}(0, 1/s)$. Note that the variance is equal to $1/s$, where s is the number of columns. This scaling ensures that $\mathbb{E}\left\|\boldsymbol{S}_G^\top \boldsymbol{x}\right\|_2^2 = \|\boldsymbol{x}\|_2^2$ for each fixed $\boldsymbol{x} \in \mathbb{R}^m$. A key property of Gaussian embeddings is their rotational invariance [Mec19]. Specifically, if $\boldsymbol{U} \in \mathbb{R}^{m\times n}$ is a matrix with orthonormal columns, i.e. $\boldsymbol{U}^\top \boldsymbol{U} = \boldsymbol{I}_n$ with $m \geq s \geq n$, then $\boldsymbol{S}_G^\top \boldsymbol{U} \in \mathbb{R}^{s\times n}$ is also a Gaussian embedding. The following theoretical result demonstrates that Gaussian embeddings form subspace embeddings with high probability.

Theorem 2.2.1 (Marčenko & Pastur 1967; [MP67], Davidson & Szarek 2001; [DS01])**.** *Consider an $m \times s$ Gaussian embedding $\boldsymbol{S}_G$, and an $m \times n$ orthonormal matrix $\boldsymbol{U}$ with $s \geq n$.*

Then

$$1 - \sqrt{\frac{n}{s}} \leq \mathbb{E}\left[\sigma_n(\boldsymbol{S}_G^\top \boldsymbol{U})\right] \leq \mathbb{E}\left[\sigma_1(\boldsymbol{S}_G^\top \boldsymbol{U})\right] \leq 1 + \sqrt{\frac{n}{s}}.$$

Furthermore, for any $t > 0$,

$$\mathbb{P}\left(1 - \sqrt{\frac{n}{s}} - t \leq \sigma_n(\boldsymbol{S}_G^\top \boldsymbol{U}) \leq \sigma_1(\boldsymbol{S}_G^\top \boldsymbol{U}) \leq 1 + \sqrt{\frac{n}{s}} + t\right) \geq 1 - e^{-st^2/2}.$$

Theorem 2.2.1 implies that rectangular Gaussian embeddings with aspect ratio $s/n > 1$ are well-conditioned with singular values that lie in the interval $\left[1 - \sqrt{\frac{n}{s}} - t, 1 + \sqrt{\frac{n}{s}} + t\right]$ with failure probability that decreases exponentially with t^2. Notice that to obtain a distortion factor of η, we require $s \approx \frac{1}{\eta^2} n$ as setting $s = \frac{1}{\eta^2} n$ in Theorem 2.2.1 gives

$$1 - \eta \leq \mathbb{E}\left[\sigma_n(\boldsymbol{S}_G^\top \boldsymbol{U})\right] \leq \mathbb{E}\left[\sigma_1(\boldsymbol{S}_G^\top \boldsymbol{U})\right] \leq 1 + \eta.$$

Therefore to achieve a moderate distortion, say $\eta = \frac{1}{\sqrt{2}}$, the cost is not too high while if we require a small distortion, say $\eta = 0.01$, then s needs to be substantially larger than n. In summary, random embeddings should only be used in scenarios where a large distortion factor is acceptable.

Theorem 2.2.1 can be used to prove that the singular values of $\boldsymbol{A}$ and the sketch $\boldsymbol{S}_G^\top \boldsymbol{A}$ is similar in magnitude. Before proving this statement, we first prove a lemma that extends [EI95, Thm. 3.1] for rectangular matrices, which is also a generalisation of [Ost59].

Lemma 2.2.1. *Let* $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ $(m \geq n)$ *be a rank-r matrix with factorisation,* $\boldsymbol{A} = \boldsymbol{A}_1 \boldsymbol{A}_2 \boldsymbol{A}_3^\top$ *where* $\boldsymbol{A}_1 \in \mathbb{R}^{m \times k}$, $\boldsymbol{A}_2 \in \mathbb{R}^{k \times k}$, *and* $\boldsymbol{A}_3 \in \mathbb{R}^{n \times k}$ *with* $k \geq r$. *Suppose* $\boldsymbol{D}_1 \in \mathbb{R}^{m \times s_1}$ *and* $\boldsymbol{D}_2 \in \mathbb{R}^{n \times s_2}$ *are matrices with* $s_1, s_2 \geq k$ *such that* $\boldsymbol{D}_1^\top \boldsymbol{A}_1 \in \mathbb{R}^{s_1 \times k}$, *and* $\boldsymbol{D}_2^\top \boldsymbol{A}_3 \in \mathbb{R}^{s_2 \times k}$ *have full column rank. Then*

$$\sigma_k(\boldsymbol{D}_1^\top \boldsymbol{A}_1)\sigma_k(\boldsymbol{D}_2^\top \boldsymbol{A}_3)\sigma_i(\boldsymbol{A}_2) \leq \sigma_i(\boldsymbol{D}_1^\top \boldsymbol{A} \boldsymbol{D}_2) \leq \sigma_1(\boldsymbol{D}_1^\top \boldsymbol{A}_1)\sigma_1(\boldsymbol{D}_2^\top \boldsymbol{A}_3)\sigma_i(\boldsymbol{A}_2)$$

for all $i \in [k]$.

Proof. Let $\boldsymbol{D}_1^\top \boldsymbol{A}_1 = \boldsymbol{Q}_1 \boldsymbol{R}_1$, and $\boldsymbol{D}_2^\top \boldsymbol{A}_3 = \boldsymbol{Q}_2 \boldsymbol{R}_2$ be the thin QR factorisation of $\boldsymbol{D}_1^\top \boldsymbol{A}_1$, and $\boldsymbol{D}_2^\top \boldsymbol{A}_3$, respectively. By unitary invariance of the spectral norm, we have

$$\sigma_i(\boldsymbol{D}_1^\top \boldsymbol{A} \boldsymbol{D}_2) = \sigma_i(\boldsymbol{Q}_1 \boldsymbol{R}_1 \boldsymbol{A}_2 \boldsymbol{R}_2^\top \boldsymbol{Q}_2^\top) = \sigma_i(\boldsymbol{R}_1 \boldsymbol{A}_2 \boldsymbol{R}_2^\top).$$

Now by assumption, $\boldsymbol{R}_1, \boldsymbol{R}_2 \in \mathbb{R}^{k \times k}$ are both nonsingular matrices. Therefore by [EI95, Thm. 3.1],

$$\sigma_k(\boldsymbol{R}_1)\sigma_k(\boldsymbol{R}_2)\sigma_i(\boldsymbol{A}_2) \leq \sigma_i(\boldsymbol{R}_1 \boldsymbol{A}_2 \boldsymbol{R}_2^\top) \leq \sigma_1(\boldsymbol{R}_1)\sigma_1(\boldsymbol{R}_2)\sigma_i(\boldsymbol{A}_2),$$

for all $i \in [k]$. The result follows by noting $\sigma_j(\boldsymbol{D}_1^\top \boldsymbol{A}_1) = \sigma_j(\boldsymbol{R}_1)$, and $\sigma_j(\boldsymbol{D}_2^\top \boldsymbol{A}_3) = \sigma_j(\boldsymbol{R}_2)$ for all $j \in [k]$. □

We now prove a proposition, which shows that the singular values of $\boldsymbol{A}$ and the sketch $\boldsymbol{S}_G^\top \boldsymbol{A}$ is similar for a moderately large s.

Proposition 2.2.1. *Let* $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ *be a matrix and* $\boldsymbol{S}_G \in \mathbb{R}^{m \times 4n}$ *be a Gaussian embedding with* $m \geq 4n$ *then*

$$0.4\sigma_i(\boldsymbol{A}) \leq \sigma_i\left(\boldsymbol{S}_G^\top \boldsymbol{A}\right) \leq 1.6\sigma_i(\boldsymbol{A})$$

for all $i \in [n]$ *with probability at least* $1 - e^{-n/50}$.

Proof. Let $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top$ be a thin SVD of $\boldsymbol{A}$ where $\boldsymbol{U} \in \mathbb{R}^{m \times n}$ and $\boldsymbol{\Sigma}, \boldsymbol{V} \in \mathbb{R}^{n \times n}$. Then by Lemma 2.2.1, we have

$$\sigma_n\left(\boldsymbol{S}_G^\top \boldsymbol{U}\right)\sigma_i\left(\boldsymbol{\Sigma}\boldsymbol{V}^\top\right) \leq \sigma_i\left(\boldsymbol{S}_G^\top \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top\right) \leq \sigma_1\left(\boldsymbol{S}_G^\top \boldsymbol{U}\right)\sigma_i\left(\boldsymbol{\Sigma}\boldsymbol{V}^\top\right)$$

for all i since $\boldsymbol{S}_G^\top \boldsymbol{U}$ has full column rank with probability 1. Therefore by Theorem 2.2.1,

$$0.5 - t \leq \sigma_n\left(\boldsymbol{S}_G^\top \boldsymbol{U}\right) \leq \sigma_1\left(\boldsymbol{S}_G^\top \boldsymbol{U}\right) \leq 1.5 + t$$

with probability at least $1 - e^{-2nt^2}$. Now setting $t = 0.1$, we get the desired result. $\square$

Proposition 2.2.1 shows us that sketching with a Gaussian embedding approximately preserves the singular values of the original matrix with high probability. Notice that for $n = 200$, the singular values of $\boldsymbol{A}$ are approximately preserved with probability at least 98%.

Gaussian embeddings are the most widely used random embedding for theoretical analysis as they are known to have elegant results and often has optimal guarantees [HMT11]. Other random embeddings such as those discussed later in this section often lack strong theoretical guarantees, however they behave similarly to a Gaussian embedding in practice [MT20]. As a result, the theory for Gaussian embeddings are often used to provide a rule of thumb for the general behaviour of other random embeddings.

For a Gaussian embedding, the sketch size needs to be $s = \mathcal{O}(n)$ for theoretical guarantees, but in practice $s = n + \mathcal{O}(1)$ often suffices [HMT11, MT20]. Despite its attractive sketch size and elegant analysis, Gaussian embeddings are often impractical as we spend $\mathcal{O}(mns)$ operations to apply it on a dense $m \times n$ matrix, which is prohibitively expensive for large m and n. In addition, the cost of generating and storing ms Gaussian entries is also unattractive. As a result more structured random embeddings are often used in practice.

2.2.2 Randomised trigonometric transforms

A subsampled randomised trigonometric transform (SRTT) [AC09] is an $m \times s$ matrix with $m \geq s$ that takes the form

$$\boldsymbol{S}_T = \sqrt{\frac{m}{s}} \boldsymbol{\Pi} \boldsymbol{D} \boldsymbol{F}^* \boldsymbol{R},$$

where $\boldsymbol{\Pi} \in \mathbb{R}^{m \times m}$ is a random permutation, $\boldsymbol{D} \in \mathbb{R}^{m \times m}$ is a random diagonal matrix whose entries are i.i.d. Rademacher random variables, i.e. $\boldsymbol{D}_{ii} = \pm 1$ with equal probability, $\boldsymbol{F} \in \mathbb{C}^{m \times m}$ is an unitary trigonometric transform, and $\boldsymbol{R} \in \mathbb{R}^{m \times s}$ is a random restriction

that selects s out of m columns uniformly at random. In the complex case, $\boldsymbol{F}$ is the unitary discrete Fourier transform (DFT) and in the real case, $\boldsymbol{F}$ is commonly the discrete cosine transform (DCT) or the Hadamard transform. For SRTT matrices, we have the following theoretical result.

Theorem 2.2.2 (Tropp 2011; [Tro11])**.** *Let $\boldsymbol{U}$ be an $m \times n$ orthonormal matrix with $m \geq n$ and $\boldsymbol{S}_T$ be an $m \times s$ SRTT where the sketch size s satisfies*

$$s \geq 4\left[\sqrt{n} + \sqrt{8\log(mn)}\right]^2 \log n.$$

Then

$$0.4 \leq \sigma_n\left(\boldsymbol{S}_T^\top \boldsymbol{U}\right) \leq \sigma_1\left(\boldsymbol{S}_T^\top \boldsymbol{U}\right) \leq 1.48$$

with probability at least $1 - \frac{3}{n}$.

Theorem 2.2.2 implies that SRTTs are subspace embeddings with distortion factor $\eta = 0.6$ and failure probability $3n^{-1}$. The theorem also implies that the sketch size of SRTTs needs to be $s = \mathcal{O}(n \log n)$ for theoretical guarantees which is larger than Gaussian embeddings by a factor of $\log n$. Unfortunately, the $\log n$ factor cannot be removed in general [Tro11] as there exists counterexample matrices with high coherence where the $\log n$ factor is necessary. The coherence of a rank-r matrix is the largest row norm squared of the dominant r left singular vectors of the matrix [DMIMW12], i.e., $\mu(\boldsymbol{A}) := \max_{j\in[m]} \|\boldsymbol{U}(j,:)\|_2^2$ where $\boldsymbol{U} \in \mathbb{R}^{m\times r}$ are the dominant r left singular vectors. Note that $\frac{r}{n} \leq \mu \leq 1$ [DMIMW12] and the matrix is considered to have high coherence if $\mu(\boldsymbol{A}) \approx 1$ and low coherence if $\mu(\boldsymbol{A}) \approx \frac{r}{n}$. Nonetheless, the sketch size $s = \mathcal{O}(n)$ works well in practice [HMT11, NT24].

The cost of applying an SRTT to an $m \times n$ matrix is $\mathcal{O}(mn \log s)$ operations [BG13] using a fast subsampled trigonometric transform algorithm [WLRT08]. However, it is easier to get $\mathcal{O}(mn \log m)$ operations in practical implementations. This is still cheaper than Gaussian

embeddings as long as m is only polynomially larger than n. Therefore SRTTs are preferred for practical reasons. SRTTs also have an advantage that only $s + 2m$ parameters are required to represent it. The random restriction $\boldsymbol{R}$ requires s parameters while m parameters each are required for the random diagonal $\boldsymbol{D}$ and the random permutation matrix $\boldsymbol{\Pi}$.

Other variants of randomised trigonometric transforms.

There are other variants of SRTTs. One can replace the random restriction $\boldsymbol{R}$ by a random 1-hashing matrix to obtain Hashed Randomised Trigonometric Transform (HRTT) [CFS21]. A random 1-hashing matrix has exactly one nonzero entry equal to ± 1 with equal probability placed in uniformly random coordinates in each row, whereas a random restriction has this property in each column. As long as the coherence of an orthonormal matrix $\boldsymbol{U} \in \mathbb{R}^{m \times n}$ is $\mathcal{O}(n^{-1})$, this modification enables us to eliminate the $\log n$ factor in the sketch size, achieving the optimal sketch size of $\mathcal{O}(n)$. Another variant called the scrambled SRTT (SSRTT) applies the trigonometric transform, a random diagonal matrix, and a random permutation matrix (independent from the first) once more to obtain $\sqrt{\frac{m}{s}}\boldsymbol{\Pi}_1\boldsymbol{D}_1\boldsymbol{F}^*\boldsymbol{\Pi}_2\boldsymbol{D}_2\boldsymbol{F}^*\boldsymbol{R}$ [TYUC17a, MT20].

2.2.3 Sparse sign embeddings

A sparse sign embedding is a sparse matrix with nonzero entries that are random signs [NN13, KN14, Woo14]. They are particularly useful for sparse data and take the form:

$$\boldsymbol{S}_S = \frac{1}{\sqrt{\xi}} \begin{bmatrix} \boldsymbol{s}_1 \\ \vdots \\ \boldsymbol{s}_m \end{bmatrix} \in \mathbb{R}^{m \times s}$$

where the rows of $\boldsymbol{S}_S$, $\boldsymbol{s}_i$'s, are i.i.d. sparse rows with ξ independent Rademacher random variables placed in uniformly random coordinates.[1] Here, ξ is the sparsity parameter. The special case when $\xi = 1$ is also referred to as the CountSketch matrix [NN13, CW17].

For sparse sign embeddings to be a subspace embedding for an $m \times n$ matrix $\boldsymbol{A}$, we require the sketch size to be $s = \mathcal{O}(n \log n)$ and the sparsity parameter to be $\xi = \mathcal{O}(\log n)$ for theoretical guarantees; see Theorem 4.2 in [Coh16]. Here, the failure probability is a constant. In practice, however, $s = \mathcal{O}(n)$ and $\xi = \min\{s, 8\}$ is recommended [TYUC19].

The cost of applying a sparse sign embedding to a matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ is $\mathcal{O}(\xi \cdot \mathrm{nnz}(\boldsymbol{A}))$ if sparse data structures and arithmetic are available where $\mathrm{nnz}(\boldsymbol{A})$ denotes the number of nonzero entries of $\boldsymbol{A}$. We obtain the cost of $\mathcal{O}(\xi \cdot \mathrm{nnz}(\boldsymbol{A}))$ by noting that $\boldsymbol{S}_S^\top \boldsymbol{A} = \sum_{i=1}^{m} \boldsymbol{s}_i^\top \boldsymbol{a}_i$ where $\boldsymbol{s}_i$'s and $\boldsymbol{a}_i$'s are the rows of $\boldsymbol{S}_S$ and $\boldsymbol{A}$, respectively, and computing $\boldsymbol{s}_i^\top \boldsymbol{a}_i$ costs $\mathcal{O}(\xi \cdot \mathrm{nnz}(\boldsymbol{a}_i))$. With a good sparse matrix library, sparse sign embeddings are often the fastest random embedding to apply especially when $\xi \ll s$. An empirical study has been conducted in [DM23] where the authors show that sparse sign embeddings outperform Gaussian embeddings and SRTTs in terms of speed, while performing similarly in the quality of the output.

2.3 Low-rank Approximation

Low-rank approximation serves as a powerful tool for reducing the complexity of large matrices while preserving their most important features. Let $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ be a matrix with $m \geq n$ and $k \leq n$ be the target rank. Then we seek a rank-k matrix that best approximates $\boldsymbol{A}$ under a chosen norm, such as the spectral or Frobenius norm. The advantages of low-rank approximation include data compression [CGMR05, HMT11, GVL13], and noise reduction [Han87, CLMW11].

The best rank-k approximation to $\boldsymbol{A}$ in the Frobenius norm is a rank-k matrix $[\![\boldsymbol{A}]\!]_k \in \mathbb{R}^{m \times n}$

[1] The sparse sign embedding $\boldsymbol{S}_S$ is also referred to as ξ-hashing matrices.

that solves the following minimization problem:

$$\min_{\text{rank}(\boldsymbol{A}_k)\leq k} \|\boldsymbol{A}-\boldsymbol{A}_k\|_{\text{F}}\,,$$

where $\|\cdot\|_{\text{F}}$ is the Frobenius norm. The truncated SVD (TSVD) [EY36] achieves this optimal approximation, as stated in the following theorem.

Theorem 2.3.1 (Eckart & Young 1936; [EY36])**.** *Let $\boldsymbol{A}\in\mathbb{R}^{m\times n}$ be a matrix with the SVD, $\boldsymbol{A}=\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top$, and denote the best rank-k approximation to $\boldsymbol{A}$ in Frobenius norm by*

$$[\![\boldsymbol{A}]\!]_k = \operatorname*{arg\,min}_{\text{rank}(\boldsymbol{A}_k)\leq k} \|\boldsymbol{A}-\boldsymbol{A}_k\|_{\text{F}}\,.$$

Then $[\![\boldsymbol{A}]\!]_k$ is given by the truncated SVD namely,

$$[\![\boldsymbol{A}]\!]_k = \boldsymbol{U}_k\boldsymbol{\Sigma}_k\boldsymbol{V}_k^\top,$$

where $\boldsymbol{U}_k=\boldsymbol{U}(:,1{:}\,k)$ contains the first k left singular vectors, $\boldsymbol{\Sigma}_k=\boldsymbol{\Sigma}(1{:}\,k,1{:}\,k)$ contains the largest k singular values on its diagonal, and $\boldsymbol{V}_k=\boldsymbol{V}(:,1{:}\,k)$ contains the first k right singular vectors, and satisfies,

$$\left\|\boldsymbol{A}-\boldsymbol{U}_k\boldsymbol{\Sigma}_k\boldsymbol{V}_k^\top\right\|_{\text{F}} = \sqrt{\sum_{j=k+1}^{n}\sigma_j(\boldsymbol{A})^2}.$$

This theorem generalises to any unitarily invariant norm [Mir60], where

$$[\![\boldsymbol{A}]\!]_k = \boldsymbol{U}_k\boldsymbol{\Sigma}_k\boldsymbol{V}_k^\top = \operatorname*{arg\,min}_{\text{rank}(\boldsymbol{A}_k)\leq k} \|\boldsymbol{A}-\boldsymbol{A}_k\|$$

for any unitarily invariant norm $\|\cdot\|$. For the rest of the thesis, the best rank-k approximation of a matrix $\boldsymbol{A}$ is denoted by $[\![\boldsymbol{A}]\!]_k$ and is given by the truncated SVD, i.e., $[\![\boldsymbol{A}]\!]_k=\boldsymbol{U}_k\boldsymbol{\Sigma}_k\boldsymbol{V}_k^\top$.

Note that the best low-rank approximation $[\![\boldsymbol{A}]\!]_k$ may not be unique when $\sigma_k(\boldsymbol{A}) = \sigma_{k+1}(\boldsymbol{A})$, since in this case, the kth singular value has multiplicity greater than one, and any orthonormal basis of the corresponding singular subspace can be used to construct the approximation.

2.3.1 Projections

To form a good low-rank approximation of a matrix $\boldsymbol{A} \in \mathbb{R}^{m\times n}$, we require a subspace that well-approximates the column space of $\boldsymbol{A}$. Once this subspace has been found, we can project $\boldsymbol{A}$ onto this space using a projection matrix $\boldsymbol{\mathcal{P}} \in \mathbb{R}^{m\times m}$. A projection matrix is defined as a matrix $\boldsymbol{\mathcal{P}}$ that satisfies $\boldsymbol{\mathcal{P}}^2 = \boldsymbol{\mathcal{P}}$. Projection matrices have the property that $\boldsymbol{I}_m - \boldsymbol{\mathcal{P}}$ is also a projection matrix, and moreover, if $\boldsymbol{\mathcal{P}} \neq \boldsymbol{0}, \boldsymbol{I}_m$, then $\|\boldsymbol{\mathcal{P}}\|_2 = \|\boldsymbol{I}_m - \boldsymbol{\mathcal{P}}\|_2$ [Szy06].

Let $\boldsymbol{Y} \in \mathbb{R}^{m\times k}$ be a rank-k matrix (full column rank) that approximates the dominant column space of $\boldsymbol{A}$. Then we can project $\boldsymbol{A}$ *orthogonally* onto the column space of $\boldsymbol{Y}$ by solving the following optimisation problem:

$$\underset{\boldsymbol{F}\in\mathbb{R}^{k\times n}}{\arg\min} \|\boldsymbol{A} - \boldsymbol{Y}\boldsymbol{F}\|_{\mathrm{F}}\,. \tag{2.2}$$

The solution to this optimisation problem is given by $\boldsymbol{F} = \boldsymbol{Y}^\dagger\boldsymbol{A}$, which approximates $\boldsymbol{A}$ by a matrix $\hat{\boldsymbol{A}} = \boldsymbol{Y}\boldsymbol{Y}^\dagger\boldsymbol{A}$ of rank at most k. We denote the orthogonal projector onto the column space of $\boldsymbol{Y}$ by $\boldsymbol{\mathcal{P}}_{\boldsymbol{Y}} = \boldsymbol{Y}\boldsymbol{Y}^\dagger$; indeed $\boldsymbol{\mathcal{P}}_{\boldsymbol{Y}}$ is a projector since $\boldsymbol{\mathcal{P}}_{\boldsymbol{Y}}^2 = \boldsymbol{Y}\boldsymbol{Y}^\dagger\boldsymbol{Y}\boldsymbol{Y}^\dagger = \boldsymbol{Y}\boldsymbol{Y}^\dagger = \boldsymbol{\mathcal{P}}_{\boldsymbol{Y}}$. Note that by taking the thin QR decomposition of $\boldsymbol{Y} = \boldsymbol{Q}_{\boldsymbol{Y}}\boldsymbol{R}_{\boldsymbol{Y}}$ where $\boldsymbol{Q}_{\boldsymbol{Y}} \in \mathbb{R}^{m\times k}$ and $\boldsymbol{R}_{\boldsymbol{Y}} \in \mathbb{R}^{k\times k}$, we have $\boldsymbol{Y}\boldsymbol{Y}^\dagger = \boldsymbol{Q}_{\boldsymbol{Y}}\boldsymbol{Q}_{\boldsymbol{Y}}^\top$. Therefore, any orthogonal projector is *symmetric* and satisfy $\|\boldsymbol{\mathcal{P}}_{\boldsymbol{Y}}\|_2 = 1$.

In many settings, applying orthogonal projectors to a matrix can become prohibitive for large m as we require the thin QR decomposition of $\boldsymbol{Y}$. An alternative efficient approach is to use *oblique* projectors. Oblique projectors approximate orthogonal projectors by approximating the pseudoinverse $\boldsymbol{Y}^\dagger$ and we do this through modifying the optimisation problem given in

Equation (2.2) by projecting $\boldsymbol{A}$ and $\boldsymbol{Y}$ onto a lower dimension:

$$\underset{\boldsymbol{F}\in\mathbb{R}^{k\times n}}{\arg\min} \left\|\boldsymbol{G}^\top(\boldsymbol{A}-\boldsymbol{Y}\boldsymbol{F})\right\|_{\mathrm{F}}$$

where $\boldsymbol{G}\in\mathbb{R}^{m\times s}$ with $s\geq k$. Here, s is usually a modest constant multiple of k and $\boldsymbol{G}$ is chosen to capture most of the information contained in $\boldsymbol{A}$ and $\boldsymbol{Y}$; see, for example, generalised Nyström method in Section 2.3.2 where we take $\boldsymbol{G}$ to be a subspace embedding. Note that when $\boldsymbol{G}=\boldsymbol{I}_n$, we obtain Equation (2.2). The solution to this problem is given by $\boldsymbol{F}=(\boldsymbol{G}^\top\boldsymbol{Y})^\dagger\boldsymbol{G}^\top\boldsymbol{A}$, which approximates $\boldsymbol{A}$ by a matrix $\tilde{\boldsymbol{A}}=\boldsymbol{Y}(\boldsymbol{G}^\top\boldsymbol{Y})^\dagger\boldsymbol{G}^\top\boldsymbol{A}$ of rank at most k. By taking the pseudoinverse of $\boldsymbol{G}^\top\boldsymbol{Y}\in\mathbb{R}^{s\times k}$ instead of the larger matrix $\boldsymbol{Y}$, $\tilde{\boldsymbol{A}}$ is cheaper to compute than $\hat{\boldsymbol{A}}$. We denote the oblique projector (using the matrix $\boldsymbol{G}$) onto the column space of $\boldsymbol{Y}$ by $\boldsymbol{\mathcal{P}}_{\boldsymbol{Y},\boldsymbol{G}}:=\boldsymbol{Y}(\boldsymbol{G}^\top\boldsymbol{Y})^\dagger\boldsymbol{G}^\top$. Here, $\boldsymbol{\mathcal{P}}_{\boldsymbol{Y},\boldsymbol{G}}$ is a projector since $\boldsymbol{\mathcal{P}}^2_{\boldsymbol{Y},\boldsymbol{G}}=\boldsymbol{Y}(\boldsymbol{G}^\top\boldsymbol{Y})^\dagger\boldsymbol{G}^\top\boldsymbol{Y}(\boldsymbol{G}^\top\boldsymbol{Y})^\dagger\boldsymbol{G}^\top=\boldsymbol{Y}(\boldsymbol{G}^\top\boldsymbol{Y})^\dagger\boldsymbol{G}^\top=\boldsymbol{\mathcal{P}}_{\boldsymbol{Y},\boldsymbol{G}}$. Note that $\boldsymbol{\mathcal{P}}_{\boldsymbol{Y},\boldsymbol{Y}}=\boldsymbol{Y}\left(\boldsymbol{Y}^\top\boldsymbol{Y}\right)^\dagger\boldsymbol{Y}^\top=\boldsymbol{Y}\boldsymbol{Y}^\dagger=\boldsymbol{\mathcal{P}}_{\boldsymbol{Y}}$. Unlike orthogonal projectors, oblique projectors are not symmetric and in general their spectral norm is larger than 1, i.e. $\|\boldsymbol{\mathcal{P}}_{\boldsymbol{Y},\boldsymbol{G}}\|_2\geq 1$, [Szy06].

2.3.2 Randomised low-rank approximations

The computation of a low-rank approximation to a matrix is omnipresent in the computational sciences [UT19]. However, in large-scale problems, computing the best rank-k matrix using the truncated SVD becomes prohibitive. We therefore seek methods that are accurate, robust and scale well. Over the past two decades, a popular and provably efficient approach has been randomised methods. They are simple and fast, yet powerful methods that come with elegant theoretical guarantees.

One of the most popular randomised algorithm for low-rank approximation is the randomised SVD [HMT11]. The idea is simple. We take a random embedding $\boldsymbol{S}$, e.g. a Gaussian embedding, form a sketch $\boldsymbol{A}\boldsymbol{S}$, and then project $\boldsymbol{A}$ orthogonally onto $\boldsymbol{A}\boldsymbol{S}$ to get

$\boldsymbol{\mathcal{P}}_{\boldsymbol{AS}}\boldsymbol{A} = \boldsymbol{AS}(\boldsymbol{AS})^{\dagger}\boldsymbol{A}$. While simple[2], this is a powerful idea that gives strong guarantees on the quality of low-rank approximation. Due to the nature of the approximation, the accuracy bounds are probabilistic. The theoretical results that are shown below are from Theorem 10.5 and Theorem 10.7 in [HMT11].

Theorem 2.3.2 (Halko, Martinsson & Tropp 2011; [HMT11])**.** *Let $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ be a matrix, k be a target rank and s be a sketch size with $k+2 \leq s \leq \min\{m,n\}$. Draw a Gaussian embedding $\boldsymbol{S}_G \in \mathbb{R}^{n\times s}$ and construct a sketch $\boldsymbol{Y} = \boldsymbol{AS}_G \in \mathbb{R}^{m\times s}$. Then the expected approximation error is bounded by*

$$\mathbb{E}\,\|\boldsymbol{A} - \boldsymbol{\mathcal{P}}_{\boldsymbol{Y}}\boldsymbol{A}\|_{\mathrm{F}}^2 \leq \left(1 + \frac{k}{s-k-1}\right)\|\boldsymbol{A} - [\![\boldsymbol{A}]\!]_k\|_{\mathrm{F}}^2\,.$$

Assume further that $s \geq k+4$. Then for all $u, t \geq 1$,

$$\|\boldsymbol{A} - \boldsymbol{\mathcal{P}}_{\boldsymbol{Y}}\boldsymbol{A}\|_{\mathrm{F}} \leq \left(1 + t\sqrt{\frac{12k}{s-k}}\right)\|\boldsymbol{A} - [\![\boldsymbol{A}]\!]_k\|_{\mathrm{F}} + ut\frac{e\sqrt{s}}{s-k+1}\sigma_{k+1}(\boldsymbol{A})$$

with failure probability at most $5t^{-(s-k)} + 2e^{-u^2/2}$.

Theorem 2.3.2 gives an expectation bound as well as a tail bound in the Frobenius norm for the accuracy of randomised SVD. The spectral norm bounds are also available in [HMT11]. When s is a constant multiple of the target rank, e.g. $s = \lfloor 1.2k \rfloor$, the randomised SVD, $\boldsymbol{\mathcal{P}}_{\boldsymbol{Y}}\boldsymbol{A}$ is, in expectation, only a modest constant worse than the best rank-k approximation in the Frobenius norm.

Despite its simplicity and elegant analysis, a downside of randomised SVD is that it is *adaptive*, which means that we need to access the matrix at at least two separate occasions with one dependent on the other. Typically, in the randomised SVD, we take the QR factorisation of the sketch $\boldsymbol{AS} = \boldsymbol{Q}_{\boldsymbol{Y}}\boldsymbol{R}_{\boldsymbol{Y}}$ and then project $\boldsymbol{A}$ onto $\boldsymbol{Q}_{\boldsymbol{Y}}$ by $\boldsymbol{Q}_{\boldsymbol{Y}}(\boldsymbol{Q}_{\boldsymbol{Y}}^{\top}\boldsymbol{A})$, making the randomised SVD an adaptive algorithm, which requires two passes. In addition,

[2]The randomised low-rank approximations discussed in this section can be improved using power iteration [HMT11], or by constructing a Krylov subspace using the power iterates [TW23].

forming $\boldsymbol{Q}_{\boldsymbol{Y}}^{\top}\boldsymbol{A}$ involves an expensive cost of $\mathcal{O}(mns)$ for a dense matrix $\boldsymbol{A}$. Note $\boldsymbol{Q}_{\boldsymbol{Y}}$ is unstructured. To overcome this downside, we project obliquely onto the sketch to obtain the generalised Nyström method [Nak20, TYUC17b]:

$$\boldsymbol{\mathcal{P}}_{\boldsymbol{AS}_1,\boldsymbol{S}_2}\boldsymbol{A} = \boldsymbol{AS}_1\left(\boldsymbol{S}_2^{\top}\boldsymbol{AS}_1\right)^{\dagger}\boldsymbol{S}_2^{\top}\boldsymbol{A}.$$

Instead of a single sketch, the generalised Nyström method constructs two independent sketches, $\boldsymbol{AS}_1$ and $\boldsymbol{A}^{\top}\boldsymbol{S}_2$, to approximate the column space and the row space, respectively. Unlike randomised SVD, generalised Nyström is non-adaptive (or one-pass), since the necessary matrix-vector products can be computed all at once at the start, rather than separately over multiple occasions. This, however, comes at a slight loss in accuracy, as we see below. In general, opting for an oblique projection–as done in generalised Nyström–trades some accuracy for improved efficiency compared to orthogonal projection. Theoretical guarantees for this method are provided in [Nak20, TYUC17b].

Theorem 2.3.3 (Tropp, Yurtsever, Udell & Cevher 2017; [TYUC17b])**.** *Let $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ be a matrix, k be a target rank, and s_1, s_2 be sketch sizes satisfying $k+2 \leq s_1$ and $s_1 + 2 \leq s_2 \leq \min\{m,n\}$. Draw two independent Gaussian embeddings $\boldsymbol{S}_1 \in \mathbb{R}^{n\times s_1}$ and $\boldsymbol{S}_2 \in \mathbb{R}^{m\times s_2}$ and construct sketches $\boldsymbol{Y}_1 = \boldsymbol{AS}_1 \in \mathbb{R}^{m\times s_1}$ and $\boldsymbol{Y}_2 = \boldsymbol{A}^{\top}\boldsymbol{S}_2 \in \mathbb{R}^{n\times s_2}$. Then the expected approximation error is bounded by*

$$\mathbb{E}\left\|\boldsymbol{A} - \boldsymbol{Y}_1\left(\boldsymbol{S}_2^{\top}\boldsymbol{Y}_1\right)^{\dagger}\boldsymbol{Y}_2^{\top}\right\|_{\mathrm{F}}^2 \leq \left(1+\frac{s_1}{s_2-s_1-1}\right)\left(1+\frac{k}{s_1-k-1}\right)\left\|\boldsymbol{A}-[\![\boldsymbol{A}]\!]_k\right\|_{\mathrm{F}}^2.$$

Theorem 2.3.3 demonstrates that the expected error bound for the generalised Nyström method is worse than that of randomised SVD by a factor of $\left(1+\frac{s_1}{s_2-s_1-1}\right)$. This difference essentially stems from the distinction between orthogonal and oblique projectors. Specifically, randomised SVD uses an orthogonal projector that satisfies $\|\boldsymbol{\mathcal{P}}_{\boldsymbol{Y}}\|_2 = 1$, whereas the

generalised Nyström method uses an oblique projector, which generally satisfies $\|\boldsymbol{\mathcal{P}}_{\boldsymbol{AS}_1,\boldsymbol{S}_2}\| > 1$, contributing to the increased error. For accuracy, we require s_2 to be proportionally larger than s_1. The difference $(s_2 - s_1)$ is often referred to as oversampling, which we discuss in the context of column sampling matrices in Chapter 6.

In the special case when $\boldsymbol{A}$ is a symmetric positive semi-definite matrix (SPSD) we can symmetrise the generalised Nyström approximation to:

$$\boldsymbol{\mathcal{P}}_{\boldsymbol{AS},\boldsymbol{S}}\boldsymbol{A} = \boldsymbol{A}\boldsymbol{\mathcal{P}}_{\boldsymbol{S},\boldsymbol{AS}} = \boldsymbol{AS}\left(\boldsymbol{S}^\top\boldsymbol{AS}\right)^\dagger\boldsymbol{S}^\top\boldsymbol{A}.$$

This is the Nyström method [Nys30, WS00, GM16]. It takes advantage of the symmetry and uses only a single sketch $\boldsymbol{AS}$ to form a low-rank approximation. The expected error result is given by the following theorem from [GM16].

Theorem 2.3.4 (Gittens & Mahoney 2016; [GM16])**.** *Let $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ be a SPSD matrix, k be a target rank and s be a sketch size with $k+2 \leq s \leq n$. Draw a Gaussian embeddings $\boldsymbol{S}_G$ and construct a sketch $\boldsymbol{Y} = \boldsymbol{AS}_G \in \mathbb{R}^{n\times s}$. Then the expected approximation error of the Nyström method is bounded by*

$$\mathbb{E}\left\|\boldsymbol{A} - \boldsymbol{Y}\left(\boldsymbol{S}_G^\top\boldsymbol{Y}\right)^\dagger\boldsymbol{Y}^\top\right\|_* \leq \left(1 + \frac{k}{s-k-1}\right)\|\boldsymbol{A} - [\![\boldsymbol{A}]\!]_k\|_*.$$

The proof of Theorem 2.3.4 uses the Gram correspondence [Epp, Git11, GM16] and Theorem 2.3.2. The Gram correspondence links the nuclear norm of the error matrix with the Frobenius norm squared of the square root of the error matrix as shown below:

$$\begin{aligned}
\left\|\boldsymbol{A} - \boldsymbol{AS}\left(\boldsymbol{S}^\top\boldsymbol{AS}\right)^\dagger\boldsymbol{S}^\top\boldsymbol{A}\right\|_* &= \left\|\boldsymbol{A}^{1/2}\left(\boldsymbol{I}_n - \boldsymbol{A}^{1/2}\boldsymbol{S}\left(\boldsymbol{S}^\top\boldsymbol{AS}\right)^\dagger\boldsymbol{S}^\top\boldsymbol{A}^{1/2}\right)\boldsymbol{A}^{1/2}\right\|_* \\
&= \left\|\boldsymbol{A}^{1/2}\left(\boldsymbol{I}_n - \boldsymbol{\mathcal{P}}_{\boldsymbol{A}^{1/2}\boldsymbol{S},\boldsymbol{A}^{1/2}\boldsymbol{S}}\right)\boldsymbol{A}^{1/2}\right\|_* \\
&= \left\|\left(\boldsymbol{I}_n - \boldsymbol{\mathcal{P}}_{\boldsymbol{A}^{1/2}\boldsymbol{S}}\right)\boldsymbol{A}^{1/2}\right\|_{\mathrm{F}}^2
\end{aligned}$$

Theorem 2.3.2 is used on the last expression to obtain Theorem 2.3.4.

The expression for the Nyström method, $\boldsymbol{A}\boldsymbol{S}\left(\boldsymbol{S}^\top\boldsymbol{A}\boldsymbol{S}\right)^\dagger\boldsymbol{S}^\top\boldsymbol{A}$, can be applied to any symmetric matrix. However, the Nyström method may fail and become unstable when applied to symmetric indefinite matrices, that is, matrices with both positive and negative eigenvalues. In such cases, the standard theoretical guarantees no longer hold, as the Gram correspondence can not be used; indefinite matrices do not have a (real) matrix square root. In Chapter 4, we examine the application of Nyström methods to symmetric indefinite matrices by identifying potential issues and exploring strategies to overcome these challenges.

2.4 Column Subset Selection Problem and CUR

In applications such as recommendation systems [DKR02, WS17] and imaging [CHHN21], it is often preferable to approximate a matrix using a subset of its columns or rows, rather than a sketch obtained via random embeddings. Selecting actual columns or rows has the advantage of preserving structural properties of the original matrix $\boldsymbol{A}$, such as sparsity and non-negativity [DMM08, MD09]. Moreover, this approach aids interpretability by highlighting important rows and columns.

The Column Subset Selection Problem (CSSP) addresses this by identifying the optimal set of columns that best approximate the matrix. We formally define the problem below.

Problem 2.4.1 (Column Subset Selection Problem (CSSP))**.** *Given a matrix $\boldsymbol{A} \in \mathbb{R}^{m\times n}$, find a set $J \subseteq [n]$ of k column indices that minimises*

$$\left\|\boldsymbol{A} - \boldsymbol{A}(:,J)\boldsymbol{A}(:,J)^\dagger\boldsymbol{A}\right\|_{\mathrm{F}} = \left\|\boldsymbol{A} - \boldsymbol{\mathcal{P}}_{\boldsymbol{A}(:,J)}\boldsymbol{A}\right\|_{\mathrm{F}}.$$

The resulting approximation, $\boldsymbol{\mathcal{P}}_{\boldsymbol{A}(:,J)}\boldsymbol{A}$, serves as a low-rank approximation of $\boldsymbol{A}$. A theoretical bound for this approximation is provided in several works, such as Lemma 4.1

of [SE16], and Theorem 1 of [DM23], which is given below.

Theorem 2.4.1 (Sorensen & Embree 2016; [SE16], Dong & Martinsson 2023; [DM23])**.** *Let* $\boldsymbol{X} \in \mathbb{R}^{k\times n}$ *be a full-rank row space approximator of* $\boldsymbol{A} \in \mathbb{R}^{m\times n}$*, and* $J \subseteq [n]$ *be a set of column indices such that* $\boldsymbol{X}(:, J)$ *admits full row rank. Then,*

$$\left\|\boldsymbol{A} - \boldsymbol{\mathcal{P}}_{\boldsymbol{A}(:,J)}\boldsymbol{A}\right\| \leq \left\|\boldsymbol{Q}_{\boldsymbol{X}}(J,:)^{\dagger}\right\|_2 \left\|\boldsymbol{A} - \boldsymbol{A}\boldsymbol{X}^{\dagger}\boldsymbol{X}\right\|$$

for any unitarily invariant norm $\|\cdot\|$ *where* $\boldsymbol{X}^{\top} = \boldsymbol{Q}_{\boldsymbol{X}}\boldsymbol{R}_{\boldsymbol{X}}$ *is the thin QR factorisation of* $\boldsymbol{X}^{\top}$*.*

The prefactor in the upper bound of Theorem 2.4.1 is analogous to $\sqrt{1 + \frac{k}{s-k-1}}$ in Theorem 2.3.2 for the randomised SVD. Furthermore, if the subspace embedding is chosen to be the column sampling matrix corresponding to the indices J, then the randomised SVD reduces to the approximation $\boldsymbol{\mathcal{P}}_{\boldsymbol{A}(:,J)}\boldsymbol{A}$.

The CSSP is widely studied [GE03, BMD09, BRN10, DKM20], and is known to be computationally challenging. In fact, it is NP-hard [Shi21]. Additionally, randomised low-rank approximations based on column selection often suffer from high variance [FTU23, MT20], which can make them less reliable. Nonetheless, several practical strategies have been developed that yield high-quality column subsets.

Since the prefactor, $\left\|\boldsymbol{Q}_{\boldsymbol{X}}(J,:)^{\dagger}\right\|_2$, typically governs the accuracy of the low-rank approximation $\boldsymbol{\mathcal{P}}_{\boldsymbol{A}(:,J)}\boldsymbol{A}$, most methods aim to select the index set J so as to minimise its influence. Broadly, these methods can be split: (i) *pivoting-based* methods, which apply pivoting schemes to use the pivots as row or column indices, or (ii) *sampling-based* methods, which sample row or column indices from a probability distribution informed by properties of the matrix.

In pivoting-based methods, we apply pivoting schemes to $\boldsymbol{A}$ or its proxy such as a row sketch $\boldsymbol{S}^{\top}\boldsymbol{A}$ or the dominant singular vectors of $\boldsymbol{A}$ to obtain the pivots, which then get used as row or column indices. Examples of such schemes include column-pivoted QR

(CPQR) [GVL13], LU with partial or complete pivoting [TB97], and strong rank-revealing factorisations [HP92, GE96, Pan00]. The Discrete Empirical Interpolation Method (DEIM) [CS10, DG16, SE16] is a notable example that uses LU with partial pivoting (LUPP) on the dominant left singular vectors of $\boldsymbol{A}$ to select column indices. Directly applying these schemes on $\boldsymbol{A}$ can be computationally expensive for large matrices. To address this, randomised algorithms that apply pivoting to a sketch such as $\boldsymbol{S}^\top \boldsymbol{A}$ have been proposed [VM17, DG20, DM23]. A comparative study of such methods appears in [DM23].

In sampling-based methods, column or row indices are drawn from a distribution based on certain properties of the matrix $\boldsymbol{A}$. One widely used technique is leverage score sampling, which samples indices in proportional to the row norms of the dominant singular vectors of $\boldsymbol{A}$ [DMM08, MD09]. Other strategies include uniform sampling [CD13], volume sampling [DRVW06, DR10, CK20, Mas22], determinantal point process (DPP) sampling [DM21], BSS sampling [BSS12, BDMI14], and k-means++ sampling [OG17]. Since sampling can also be expensive for large matrices, proxy-based methods using a sketch have been proposed [DMIMW12]. There are also a deterministic variant for leverage scores [PKB14] and hybrid methods such as L-DEIM [GH22].

Notably, there exists a set of column indices for which the CSSP satisfies the following tight near-optimal guarantee [ZO18]:

$$\left\| \boldsymbol{A} - \boldsymbol{\mathcal{P}}_{\boldsymbol{A}(:,J)} \boldsymbol{A} \right\|_{\mathrm{F}} \leq \sqrt{1+k}\, \left\| \boldsymbol{A} - [\![\boldsymbol{A}]\!]_k \right\|_{\mathrm{F}} .$$

This bound shows that a low-rank approximation based on actual columns of $\boldsymbol{A}$ can be nearly as good as the best possible rank-k approximation, provided the indices J are chosen appropriately. Polynomial-time algorithms for computing such set of indices are given in [CK20, Osi23], and a randomised version that achieve this bound in expectation appears in [CK24].

The CSSP can also be formulated for rows. A particularly important case involves

selecting both rows and columns. Specifically, we seek row and column indices, denoted by I and J, respectively that minimise

$$\left\|\boldsymbol{A} - \boldsymbol{C}\boldsymbol{C}^{\dagger}\boldsymbol{A}\boldsymbol{R}^{\dagger}\boldsymbol{R}\right\|_{\mathrm{F}} = \|\boldsymbol{A} - \boldsymbol{\mathcal{P}}_{\boldsymbol{C}}\boldsymbol{A}\boldsymbol{\mathcal{P}}_{\boldsymbol{R}^{\top}}\|_{\mathrm{F}}\,,$$

where $\boldsymbol{C} = \boldsymbol{A}(:, J)$ and $\boldsymbol{R} = \boldsymbol{A}(I, :)$. Notice that $\boldsymbol{C}^{\dagger}\boldsymbol{A}\boldsymbol{R}^{\dagger}$ corresponds to the solution of the optimisation problem $\min_{\boldsymbol{Z}} \|\boldsymbol{A} - \boldsymbol{C}\boldsymbol{Z}\boldsymbol{R}\|_{\mathrm{F}}$, which gives a low-rank approximation $\boldsymbol{A} \approx \boldsymbol{C}\left(\boldsymbol{C}^{\dagger}\boldsymbol{A}\boldsymbol{R}^{\dagger}\right)\boldsymbol{R}$. We refer to the choice $\boldsymbol{Z} = \boldsymbol{C}^{\dagger}\boldsymbol{A}\boldsymbol{R}^{\dagger}$ as CUR-BA (CUR with Best Approximation). A theoretical bound is given in Lemma 4.2 of [SE16], which we discuss further in Section 6.4. While robust, this choice requires access to the full matrix $\boldsymbol{A}$, which can be computationally expensive even if $\boldsymbol{C}$ and $\boldsymbol{R}$ are known in advance.

To address this, a more efficient variant uses $\boldsymbol{Z} = \boldsymbol{A}(I, J)^{\dagger}$, yielding the approximation $\boldsymbol{A} \approx \boldsymbol{C}\boldsymbol{M}^{\dagger}\boldsymbol{R}$ where $\boldsymbol{M} = \boldsymbol{A}(I, J)$.[3] This choice is often referred to as the cross approximation [Beb00, GOS+10, HPS12, CK20], and we denote it as CUR-CA (CUR with Cross Approximation) in this thesis. CUR-CA is more efficient than CUR-BA, as it requires only the submatrix $\boldsymbol{M} = \boldsymbol{A}(I, J)$. Moreover, computing the pseudoinverse of the smaller matrix $\boldsymbol{M}$ is generally much less expensive than computing the pseudoinverses of $\boldsymbol{C}$ and $\boldsymbol{R}$, or computing $\boldsymbol{C}^{\dagger}\boldsymbol{A}\boldsymbol{R}^{\dagger}$. However, a known drawback of CUR-CA is that $\boldsymbol{A}(I, J)$ can be (nearly) singular, which may lead to poor approximation quality. This concern has been raised in several works [SE16, MT20, DM23], and we examine it further in Chapters 5 and 6. In the context of CUR-BA, there exist a numerically stable implementation called StableCUR introduced in [ADM+15]. While stable, StableCUR has the drawback that the resulting approximation is no longer expressed explicitly as a product of $\boldsymbol{C}$ and $\boldsymbol{R}$, which can be undesirable in settings where interpretability or explicit factorisation is important.

[3]The notation $\boldsymbol{M} = \boldsymbol{A}(I, J)$ is a non-standard choice. In this thesis, this notation is adopted to avoid conflict with the conventional use of $\boldsymbol{U}$ to denote the left singular vectors of a matrix. The symbol $\boldsymbol{M}$ is chosen to represent the *middle* matrix in the CUR decomposition (and the Nyström method) and will often be referred to as the core matrix throughout this thesis.

Analogous to the relationship between $\boldsymbol{C}\boldsymbol{C}^{\dagger}\boldsymbol{A}$ and randomised SVD, CUR-CA is related to the generalised Nyström method. When the subspace embedding is chosen as a column sampling matrix, the generalised Nyström method coincides with CUR-CA. Both methods rely on oblique projections and tend to prioritise computational efficiency over accuracy. Note that CUR-CA involves oblique projection, similarly to the generalised Nyström method as

$$\boldsymbol{C}\boldsymbol{M}^{\dagger}\boldsymbol{R} = \boldsymbol{C}\left(\boldsymbol{\Pi}_I^{\top}\boldsymbol{C}\right)^{\dagger}\boldsymbol{\Pi}_I^{\top}\boldsymbol{A} = \boldsymbol{\mathcal{P}}_{\boldsymbol{C},\boldsymbol{\Pi}_I}\boldsymbol{A},$$

where $\boldsymbol{\Pi}_I = \boldsymbol{I}_m(:, I)$ is a column sampling matrix that selects indices corresponding to the index set I, i.e., $\boldsymbol{\Pi}_I^{\top}\boldsymbol{A} = \boldsymbol{A}(I, :)$. Nonetheless, with carefully chosen indices I and J, we empirically observe that CUR-CA achieves accuracy comparable to that of CUR-BA. This is further discussed in Chapter 5. The theoretical guarantees for CUR-CA are analysed in Chapter 5. Similar to the generalised Nyström method—which involves two prefactors, $\left(1 + \frac{s_1}{s_2 - s_1 - 1}\right)$ and $\left(1 + \frac{k}{s_1 - k - 1}\right)$, in the bound of Theorem 2.3.3—the error bound for CUR-CA also contains two prefactors, as will be shown in Corollary 5.1.1. Like the generalised Nyström method, these prefactors can be improved by oversampling, which increases the set of either row or column indices, but not both. We discuss this further in Chapter 6. The theoretical work for CUR-CA presented in Chapters 5 and 6 motivates the use of similar (or the same) set of indices for matrices that are closely related, which we apply in the context of parameter-dependent matrices in Chapter 7.

3

Fast approximate null space computation

Many algorithms in RandNLA such as the randomised SVD and the vast literature on randomised low-rank approximations (see Section 2.3.2) focus on approximating the top few singular values and their corresponding singular vectors. On the other hand, relatively little attention has been paid to the *smallest* singular values and their corresponding singular vectors. These are often required in problems such as total least squares [GVL80, VHV91], and rational approximation via the AAA algorithm [NST18], among others. In particular, the right singular vector(s) corresponding to the zero singular value span the null space. In this chapter, we focus on approximating the singular vector(s) corresponding to the smallest singular value(s) of a tall matrix $\boldsymbol{A}$.

Let us begin by formally defining the problem.

Problem 3.0.1 (Approximate Null Space Problem)**.** *Let* $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ *where* $m \geq n$ *and*

$k \in \mathbb{Z}^+$. *Find* $\boldsymbol{W} \in \mathbb{R}^{n \times k}$ *that solves the following optimisation problem*

$$\min_{\boldsymbol{V}^\top \boldsymbol{V} = \boldsymbol{I}_k} \|\boldsymbol{A}\boldsymbol{V}\|_{\mathrm{F}}.$$

This problem statement does not find the null space of $\boldsymbol{A}$ exactly; indeed the null space of $\boldsymbol{A}$ is usually the trivial 0, but to consider a broader class of problems we will call this problem the approximate null space problem. The solution, $\boldsymbol{W}$, in the problem statement, to the null space problem is the trailing k right singular vectors, that is, the k right singular vectors corresponding to the smallest k singular values. When the k smallest singular values are distinct from the other $(n-k)$ singular values, the orthonormal matrix $\boldsymbol{W}$ is unique up to right multiplication by an orthogonal matrix, that is, $\boldsymbol{W}\boldsymbol{Q}$ is also a solution to Problem 3.0.1 for any orthogonal matrix $\boldsymbol{Q} \in \mathbb{R}^{k \times k}$.

The standard method for computing the null space or the trailing right singular vectors of a matrix is to use the SVD or rank-revealing factorisations [Cha87, HP92, GE96], which costs $\mathcal{O}(mn^2)$ flops for an $m \times n$ matrix with $m \geq n$. Other methods such as the TSQR [DGHL12] and Cholesky QR [Ste98, Ch. 4] have the same complexity $\mathcal{O}(mn^2)$, but can be improved with parallelization. Unfortunately, Cholesky QR suffers from squaring of the condition number as it uses the Gram matrix [SW02].

For $m \gg n$, the standard $\mathcal{O}(mn^2)$ methods can become prohibitive, so we use *sketching* to get a near-optimal solution at a lower complexity. In this chapter, we study the sketch-and-solve method from [GPW12] by analysing its accuracy using multiplicative perturbation theory developed by Li [Li98b]. The sketch-and-solve method gives us the complexity of $\mathcal{O}(mn \log n + n^3)$, which scales better than the standard $\mathcal{O}(mn^2)$ methods.

3.1 Sketch-and-solve method

Now we approach Problem 3.0.1 using the sketch-and-solve method. The sketch-and-solve method [Sar06, DMMS11] transforms a highly overdetermined least squares problem into a problem of lower dimension using sketching, i.e.,

$$\min_{\boldsymbol{V}^\top \boldsymbol{V} = \boldsymbol{I}_k} \|\boldsymbol{A}\boldsymbol{V}\|_{\mathrm{F}} \quad \longmapsto \quad \min_{\boldsymbol{V}^\top \boldsymbol{V} = \boldsymbol{I}_k} \left\|\boldsymbol{S}^\top \boldsymbol{A}\boldsymbol{V}\right\|_{\mathrm{F}}$$

where $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ and $\boldsymbol{S} \in \mathbb{R}^{m\times s}$ is a subspace embedding with $m \geq s \geq n$. Typically, we have $m \gg s = cn$ where $c > 1$ is a modest constant, say $c = 2$ or $c = 4$. To solve Problem 3.0.1 approximately, we work with the sketch $\boldsymbol{S}^\top \boldsymbol{A} \in \mathbb{R}^{s\times n}$ by taking its SVD, and finding its trailing right singular vectors. By working with a smaller-sized sketch $\boldsymbol{S}^\top \boldsymbol{A}$, we can overcome certain constraints, such as when $\boldsymbol{A}$ is too large to fit in memory or when $\boldsymbol{A}$ is streamed. In Section 3.4, we demonstrate how the sketch can be efficiently updated or downdated in a streaming setting, and this idea is leveraged in the AAA algorithm described in Section 3.5. These techniques improve computational efficiency in terms of both speed and storage. The overall algorithm is given in Algorithm 1.

Algorithm 1 Solving the approximate null space problem using sketch-and-solve

Input: $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ with $m \gg n$, s ($> n$) the sketch size (recommended $s = 2n$), k the number of right singular vectors desired

Output: $\boldsymbol{W} \in \mathbb{R}^{n\times k}$ the trailing k right singular vectors

`function` $\boldsymbol{W} =$ `Rand_Nullspace`$(\boldsymbol{A}, s, k)$

1: Draw a random embedding $\boldsymbol{S} \in \mathbb{R}^{m\times s}$ with sketch size s,
2: Sketch $\boldsymbol{X} = \boldsymbol{S}^\top \boldsymbol{A} \in \mathbb{R}^{s\times n}$,
3: $[\boldsymbol{U}, \boldsymbol{\Sigma}, \boldsymbol{V}] =$ `svd`$(\boldsymbol{X})$,
4: $\boldsymbol{W} = \boldsymbol{V}(:, n-k+1:n)$, the trailing k right singular vectors

There is also a version of Algorithm 1 with ϵ tolerance rather than taking k as input. This version finds all the singular values that are less than ϵ and set $\boldsymbol{W}$ to be their corresponding right singular vectors. If we set $\epsilon = \mathcal{O}(u)$ where u is the unit roundoff,

then we get the numerical null space.

For the sketch size $s = cn$ for a constant $c > 1$, the complexity of Algorithm 1 is $\mathcal{O}(mn \log n)$ for performing the sketch (line 2) using an SRTT and $\mathcal{O}(n^3)$ for calculating the trailing right singular vectors of $\boldsymbol{S}^\top \boldsymbol{A}$ using the SVD (line 3). This gives us an overall complexity of $\mathcal{O}(mn \log n + n^3)$, which is faster than the traditional methods, $\mathcal{O}(mn^2)$. In line 2, sparse sign embeddings can be used to give an overall complexity of $\mathcal{O}(\xi \cdot \text{nnz}(\boldsymbol{A}) + n^3)$ for Algorithm 1 where ξ is the sparsity parameter; see Section 2.2.3.

For Algorithm 1, the quality of the solution needs to be assessed. Since subspace embeddings preserve norms of the original space (Equation (2.1)), it is straightforward to see that the residual norm of the output of Algorithm 1 will be on a similar order as the residual norm of the true solution of Problem 3.0.1. This is made precise for SRTTs in the following theorem.

Theorem 3.1.1. *Let $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ be a matrix and $\boldsymbol{S}_T \in \mathbb{R}^{m \times s}$ be an SRTT with the sketch size s satisfying*

$$4\left[\sqrt{n} + \sqrt{8 \log(mn)}\right]^2 \log n \leq s \leq m.$$

Then

$$\left\|\boldsymbol{A}\tilde{\boldsymbol{W}}\right\|_{\text{F}} < 4 \left\|\boldsymbol{A}\boldsymbol{W}\right\|_{\text{F}}$$

with failure probability at most $O(n^{-1})$ where $\tilde{\boldsymbol{W}}$ is the output from Algorithm 1 and $\boldsymbol{W}$ is the exact solution to the approximate null space problem (Problem 3.0.1).

Proof. Let $\boldsymbol{A} = \boldsymbol{U}_{\boldsymbol{A}} \boldsymbol{\Sigma}_{\boldsymbol{A}} \boldsymbol{V}_{\boldsymbol{A}}^\top$ be a thin SVD of $\boldsymbol{A}$ such that $\boldsymbol{U}_{\boldsymbol{A}} \in \mathbb{R}^{m \times n}$ and $\boldsymbol{\Sigma}_{\boldsymbol{A}}, \boldsymbol{V}_{\boldsymbol{A}} \in \mathbb{R}^{n \times n}$. Then by Lemma 2.2.1, we have

$$\sigma_n(\boldsymbol{S}_T^\top \boldsymbol{U}_{\boldsymbol{A}}) \sigma_i(\boldsymbol{\Sigma}_{\boldsymbol{A}}) \leq \sigma_i\left(\boldsymbol{S}_T^\top \boldsymbol{A}\right) \leq \sigma_1(\boldsymbol{S}_T^\top \boldsymbol{U}_{\boldsymbol{A}}) \sigma_i(\boldsymbol{\Sigma}_{\boldsymbol{A}})$$

for $i = 1, 2, ..., n$, and

$$\sigma_n(\boldsymbol{S}_T^\top \boldsymbol{U}_{\boldsymbol{A}}) \left\| \boldsymbol{A}\tilde{\boldsymbol{W}} \right\|_{\mathrm{F}} \leq \left\| \boldsymbol{S}_T^\top \boldsymbol{A}\tilde{\boldsymbol{W}} \right\|_{\mathrm{F}}.$$

Therefore, by Theorem 2.2.2, with failure probability at most $O(n^{-1})$ we have

$$\begin{aligned}
\left\| \boldsymbol{A}\tilde{\boldsymbol{W}} \right\|_{\mathrm{F}} &\leq \frac{1}{\sigma_n(\boldsymbol{S}_T^\top \boldsymbol{U}_{\boldsymbol{A}})} \left\| \boldsymbol{S}_T^\top \boldsymbol{A}\tilde{\boldsymbol{W}} \right\|_{\mathrm{F}} \\
&= \frac{1}{\sigma_{\min}(\boldsymbol{S}_T^\top \boldsymbol{U}_{\boldsymbol{A}})} \sqrt{\sum_{j=1}^{k} \sigma_{n-j+1} \left(\boldsymbol{S}_T^\top \boldsymbol{A} \right)^2} \\
&\leq \frac{\sigma_{\max}(\boldsymbol{S}_T^\top \boldsymbol{U}_{\boldsymbol{A}})}{\sigma_{\min}(\boldsymbol{S}_T^\top \boldsymbol{U}_{\boldsymbol{A}})} \sqrt{\sum_{j=1}^{k} \sigma_{n-j+1}(\boldsymbol{A})^2} \\
&\leq \frac{1.48}{0.4} \left\| \boldsymbol{A}\boldsymbol{W} \right\|_{\mathrm{F}} < 4 \left\| \boldsymbol{A}\boldsymbol{W} \right\|_{\mathrm{F}}
\end{aligned}$$

□

Remark 3.1.1.

1. *Theorem 3.1.1 suggests that the sketch size needs to be $\mathcal{O}(n \log n)$, but a constant multiple of n, say $s = 2n$, works well in practice; see discussion in Section 2.2.2.*

2. *A similar result can be obtained for other random embeddings. In general, if the random embedding $\boldsymbol{S}$ satisfies*

$$(1 - \eta)\sigma_i(\boldsymbol{A}) \leq \sigma_i(\boldsymbol{S}^\top \boldsymbol{A}) \leq (1 + \eta)\sigma_i(\boldsymbol{A})$$

for some $0 < \eta < 1$ with high probability then

$$\left\| \boldsymbol{A}\tilde{\boldsymbol{W}} \right\|_{\mathrm{F}} \leq \frac{1 + \eta}{1 - \eta} \left\| \boldsymbol{A}\boldsymbol{W} \right\|_{\mathrm{F}}$$

with high probability where $\tilde{\boldsymbol{W}}$ is the output of Algorithm 1 when using $\boldsymbol{S}$ as the random

embedding.

Theorem 3.1.1 tells us that the residual norm of Algorithm 1 will be a modest constant larger than the residual norm of the exact solution of Problem 3.0.1. However, this does not immediately imply that the the two solution vectors are close to each other. We will therefore assess the quality of the sketch-and solve solution by deriving bounds for the sine of the angle between the sketched solution and the original solution; see Section 3.2. Specifically, we will quantify this bound using multiplicative perturbation theory from [Li98b]. The perturbation is multiplicative because the original matrix gets multiplied by a subspace embedding rather than undergoing an additive perturbation. This is different from the classical perturbation theory in [DK70, Sec. 2] and [Wed72, Sec. 3], which is additive. The classical result scales poorly for small singular values when the perturbation is close to a unitary matrix, whereas the multiplicative perturbation theory can overcome this issue.

Algorithm 1 is not new and to our knowledge first appeared in [GPW12]. In [GPW12], Gilbert, Park[1] and Wakin analyse the accuracy of the singular values and the right singular vectors obtained using Algorithm 1. They use the spectral norm error to quantify the accuracy of the right singular vectors obtained this way, whereas we use the canonical angles which is arguably more natural in this setting. Furthermore, we extend the analysis to comparing subspaces of either the same or different dimensions (Section 3.2). This is particularly useful if we are extracting a subspace from the sketch $\boldsymbol{S}^\top \boldsymbol{A}$ rather than a single vector, for example, when we are solving total least squares problems with multiple right-hand sides (Section 3.5). It is also important to note that the condition number of computing a singular vector is inversely proportional to the gap [DK70, Wed72], so if we want to extract the right singular subspace of dimension larger than one corresponding to a singular value (for example, singular values corresponding to zero in the case of computing the null space of a matrix) then the bound used to quantify the accuracy of a single right singular vector is useless as the

[1]This is not me, unfortunately.

gap is zero. The gap refers to the distance between the approximated singular values and the singular values of the original matrix that is not being approximated. In our context, $\text{gap} = \min_{\substack{i=1,2,\ldots,n-k \\ j=n-k+1,\ldots,n}} |\sigma_i(\boldsymbol{A}) - \sigma_j(\boldsymbol{S}^\top \boldsymbol{A})|$. However, when we compare the subspace as a whole, we can get meaningful bound as we shall see in Section 3.2. Moreover, when the gap between the target singular value and the rest is small, the corresponding target singular vector is ill-conditioned. In such cases it is often difficult to obtain nontrivial bounds for the accuracy; however, we show that useful bounds can be obtained by examining the angle between the target vector(s) and a computed (approximate) subspace of larger dimension.

Now we look at the quality of the solution. We will examine how much the sketched solution (output of Algorithm 1) deviates from the original solution (SVD of $\boldsymbol{A}$).

3.2 Accuracy of the right singular subspace

The standard way of quantifying the distance between two subspaces is to use the *canonical angles* between subspaces.

Definition 3.2.1 (Canonical angles [SS90, Ch. I.5])**.** *Let $\boldsymbol{U}, \boldsymbol{V} \in \mathbb{R}^{n\times k}$ with $n \geq k$ be two matrices with orthonormal columns. Then the canonical angles between the subspaces spanned by $\boldsymbol{U}$ and $\boldsymbol{V}$ are $\{\theta_i\}_{i=1}^k$ where $\theta_i = \arccos\left(\sigma_i(\boldsymbol{U}^\top \boldsymbol{V})\right)$, that is, the arccosine of the singular values of $\boldsymbol{U}^\top \boldsymbol{V}$. We let $\boldsymbol{\Theta}(\boldsymbol{U}, \boldsymbol{V}) = \operatorname{diag}(\theta_1, ..., \theta_k)$ and $\sin\boldsymbol{\Theta}(\boldsymbol{U}, \boldsymbol{V})$ and $\cos\boldsymbol{\Theta}(\boldsymbol{U}, \boldsymbol{V})$ be defined element-wise for the diagonal entries only.*

Remark 3.2.1.

1. *When $k = 1$, we get the standard definition where $\cos\theta(\boldsymbol{x}, \boldsymbol{y}) = \frac{|\boldsymbol{x}^\top \boldsymbol{y}|}{\|\boldsymbol{x}\|_2 \|\boldsymbol{y}\|_2}$.*
2. *$\boldsymbol{\Theta}(\boldsymbol{U}, \boldsymbol{V}) = \boldsymbol{\Theta}(\boldsymbol{V}, \boldsymbol{U})$ since $\boldsymbol{U}^\top \boldsymbol{V}$ and $\boldsymbol{V}^\top \boldsymbol{U}$ have the same singular values.*
3. *By the cosine-sine decomposition (Ch. I, [SS90]), the elements on the diagonal of $\sin\boldsymbol{\Theta}(\boldsymbol{U}, \boldsymbol{V})$ are the singular values of $\boldsymbol{U}_\perp^\top \boldsymbol{V}$ or $\boldsymbol{U}^\top \boldsymbol{V}_\perp$ where $\boldsymbol{U}_\perp \in \mathbb{R}^{n\times(n-k)}$ and*

$\boldsymbol{V}_\perp \in \mathbb{R}^{n\times(n-k)}$ have columns that give orthonormal bases to the orthogonal complements of range($\boldsymbol{U}$) and range($\boldsymbol{V}$), respectively. This implies that $\sin\boldsymbol{\Theta}(\boldsymbol{U},\boldsymbol{V})$ *and* $\sin\boldsymbol{\Theta}(\boldsymbol{U}_\perp,\boldsymbol{V}_\perp)$ *have the same number of nonzero entries on their diagonal.*

Now we look at the perturbation of right singular vectors using canonical angles. There are two different types of perturbations, namely additive and multiplicative perturbations. The results for additive perturbation was first proved in 1970 by Davis and Kahan [DK70] for eigenvectors of symmetric matrices, and two years later Wedin [Wed72] derived analogous results for singular vectors. We focus on multiplicative perturbation bounds [Li98b], as they arise naturally in our context, and give superior bounds in our setting. This type of bound for Algorithm 1 has been studied in [GPW12] for vectors, however there is no known study for subspaces of dimension larger than 1.

3.2.1 Computing subspaces of the same dimension

Let $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ and $\tilde{\boldsymbol{A}} \in \mathbb{R}^{s\times n}$ be matrices with $m \geq s \geq n$ and suppose that they have SVDs of the form

$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top = [\boldsymbol{U}_1,\boldsymbol{U}_2]\begin{bmatrix}\boldsymbol{\Sigma}_1 & \\ & \boldsymbol{\Sigma}_2\end{bmatrix}[\boldsymbol{V}_1,\boldsymbol{V}_2]^\top, \tag{3.1}$$

$$\tilde{\boldsymbol{A}} = \tilde{\boldsymbol{U}}\tilde{\boldsymbol{\Sigma}}\tilde{\boldsymbol{V}}^\top = [\tilde{\boldsymbol{U}}_1,\tilde{\boldsymbol{U}}_2]\begin{bmatrix}\tilde{\boldsymbol{\Sigma}}_1 & \\ & \tilde{\boldsymbol{\Sigma}}_2\end{bmatrix}[\tilde{\boldsymbol{V}}_1,\tilde{\boldsymbol{V}}_2]^\top, \tag{3.2}$$

where $\boldsymbol{U} \in \mathbb{R}^{m\times n}$, $\tilde{\boldsymbol{U}} \in \mathbb{R}^{s\times n}$, $\boldsymbol{V},\tilde{\boldsymbol{V}} \in \mathbb{R}^{n\times n}$ and the matrices with subscript 1 have $(n-k)$ columns and subscript 2 have k columns with $k < n$ and

$$\boldsymbol{\Sigma}_1 = \mathrm{diag}(\sigma_1,...,\sigma_{n-k}), \quad \boldsymbol{\Sigma}_2 = \mathrm{diag}(\sigma_{n-k+1},...,\sigma_n), \tag{3.3}$$

$$\tilde{\boldsymbol{\Sigma}}_1 = \operatorname{diag}(\tilde{\sigma}_1, ..., \tilde{\sigma}_{n-k}), \quad \tilde{\boldsymbol{\Sigma}}_2 = \operatorname{diag}(\tilde{\sigma}_{n-k+1}, ..., \tilde{\sigma}_n), \tag{3.4}$$

where the singular values are ordered in non-increasing order.

Next we define the relative distance χ. Let $a, b \in \mathbb{R}$ and define

$$\chi(a, b) = \frac{|a - b|}{\sqrt{|ab|}}$$

with the convention $0/0 = 0$ and $1/0 = \infty$. Note that χ is not a metric on $\mathbb{R}$ [Li98a] as it violates the triangle inequality. We also recall a useful matrix norm inequality from [Mir60], which states that for any matrices $\boldsymbol{A}, \boldsymbol{B}$ and $\boldsymbol{C}$ such that the product $\boldsymbol{ABC}$ is defined, we have

$$\|\boldsymbol{ABC}\| \leq \|\boldsymbol{A}\|_2 \|\boldsymbol{B}\| \|\boldsymbol{C}\|_2$$

for any unitarily invariant norm $\|\cdot\|$. Lastly, we review a key lemma from Li [Li98b].

Lemma 3.2.1 (Lemma 2.5; [Li98b])**.** *Let $\boldsymbol{A} \in \mathbb{R}^{s\times s}$ and $\boldsymbol{B} \in \mathbb{R}^{t\times t}$ be two positive semi-definite Hermitian matrices and let $\boldsymbol{E} \in \mathbb{R}^{s\times t}$. Suppose that there exists $\alpha > 0$ and $\delta > 0$ such that*

$$\|\boldsymbol{A}\|_2 \leq \alpha \text{ and } \left\|\boldsymbol{B}^{-1}\right\|_2^{-1} \geq \alpha + \delta$$

or

$$\|\boldsymbol{B}\|_2 \leq \alpha \text{ and } \left\|\boldsymbol{A}^{-1}\right\|_2^{-1} \geq \alpha + \delta.$$

Then the Sylvester equation $\boldsymbol{AX} - \boldsymbol{XB} = \boldsymbol{A}^{1/2}\boldsymbol{E}\boldsymbol{B}^{1/2}$ has a unique solution $\boldsymbol{X} \in \mathbb{R}^{s\times t}$, and moreover $\|\boldsymbol{X}\| \leq \|\boldsymbol{E}\| / \chi(\alpha, \alpha + \delta)$ for any unitarily invariant norm.

Now we prove a theorem that assesses the quality of the right singular subspace obtained using the sketch-and-solve method. This is a modification of Theorem 4.2 in [Li98b]. The modification allows multiplicative perturbation of an $m \times n$ matrix $\boldsymbol{A}$ by a $s \times m$ rectangular matrix $\boldsymbol{S}^\top$ with $m \geq s \geq n$ such that $\boldsymbol{S}^\top \boldsymbol{U}$ has full column rank, whereas Li only considers non-

singular square matrices. Our bound also tightens Li's bound by removing extra simplifications made in the proof. In our context, $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ corresponds to the original matrix and $\boldsymbol{S}^\top \boldsymbol{A} \in \mathbb{R}^{s\times n}$ corresponds to the sketch where $\boldsymbol{S}$ is a subspace embedding.

Theorem 3.2.1. *Let $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ and $\tilde{\boldsymbol{A}} = \boldsymbol{S}^\top \boldsymbol{A} \in \mathbb{R}^{s\times n}$ $(m \geq s \geq n)$ with SVDs as in Equations* (3.1) *to* (3.4) *where $\boldsymbol{S} \in \mathbb{R}^{m\times s}$ is a matrix such that $\boldsymbol{S}^\top \boldsymbol{U}$ has full column rank. Let the thin QR decomposition of $\boldsymbol{S}^\top \boldsymbol{U}$ be $\boldsymbol{QR}$. Suppose that there exists $\alpha > 0$ and $\delta > 0$ such that*

$$\sigma_{n-k} \geq \alpha + \delta \qquad \text{and} \qquad \tilde{\sigma}_{n-k+1} \leq \alpha,$$

or

$$\tilde{\sigma}_{n-k} \geq \alpha + \delta \qquad \text{and} \qquad \sigma_{n-k+1} \leq \alpha.$$

Then for any unitarily invariant norm we have

$$\left\|\sin \boldsymbol{\Theta}(\boldsymbol{V}_2, \tilde{\boldsymbol{V}}_2)\right\| \leq \frac{\left\|\boldsymbol{R} - \boldsymbol{R}^{-\top}\right\|}{\chi\left(\alpha^2, (\alpha+\delta)^2\right)} \tag{3.5}$$

where $\boldsymbol{V}_2$ and $\tilde{\boldsymbol{V}}_2$ are as in Equations (3.1) *and* (3.2)*.*

Proof. We define two matrices $\boldsymbol{B} := \boldsymbol{\Sigma}\boldsymbol{V}^\top$ and $\tilde{\boldsymbol{B}} := \boldsymbol{R}\boldsymbol{\Sigma}\boldsymbol{V}^\top$. Notice that $\boldsymbol{A}$ and $\boldsymbol{B}$ have the same singular values and the right singular vectors, and since $\tilde{\boldsymbol{A}} = \boldsymbol{S}^\top \boldsymbol{A} = \boldsymbol{QR}\boldsymbol{\Sigma}\boldsymbol{V}^\top$, $\tilde{\boldsymbol{A}}$ and $\tilde{\boldsymbol{B}}$ also have the same singular values and the right singular vectors. Therefore to prove Equation (3.5) it suffices to work with $\boldsymbol{B}$ and $\tilde{\boldsymbol{B}}$. We have

$$\tilde{\boldsymbol{B}}^\top \tilde{\boldsymbol{B}} - \boldsymbol{B}^\top \boldsymbol{B} = \tilde{\boldsymbol{B}}^\top \boldsymbol{R}\boldsymbol{B} - \tilde{\boldsymbol{B}}^\top \boldsymbol{R}^{-\top}\boldsymbol{B} = \tilde{\boldsymbol{B}}^\top (\boldsymbol{R} - \boldsymbol{R}^{-\top})\boldsymbol{B},$$

since $\boldsymbol{R}$ is invertible as $\boldsymbol{S}^\top \boldsymbol{U}$ has full column rank. This is equivalent to

$$\tilde{\boldsymbol{V}}\tilde{\boldsymbol{\Sigma}}^2\tilde{\boldsymbol{V}}^\top - \boldsymbol{V}\boldsymbol{\Sigma}^2\boldsymbol{V}^\top = \tilde{\boldsymbol{V}}\tilde{\boldsymbol{\Sigma}}\left(\boldsymbol{Q}^\top \tilde{\boldsymbol{U}}\right)^\top (\boldsymbol{R} - \boldsymbol{R}^{-\top})\boldsymbol{\Sigma}\boldsymbol{V}^\top,$$

since $\tilde{\boldsymbol{B}} = \boldsymbol{Q}^\top \tilde{\boldsymbol{A}} = \boldsymbol{Q}^\top \tilde{\boldsymbol{U}} \tilde{\boldsymbol{\Sigma}} \tilde{\boldsymbol{V}}^\top$. Left-multiplying $\tilde{\boldsymbol{V}}^\top$ and right-multiplying $\boldsymbol{V}$ gives

$$\tilde{\boldsymbol{\Sigma}}^2 \tilde{\boldsymbol{V}}^\top \boldsymbol{V} - \tilde{\boldsymbol{V}}^\top \boldsymbol{V} \boldsymbol{\Sigma}^2 = \tilde{\boldsymbol{\Sigma}} \left(\boldsymbol{Q}^\top \tilde{\boldsymbol{U}} \right)^\top (\boldsymbol{R} - \boldsymbol{R}^{-\top}) \boldsymbol{\Sigma}.$$

Now define $\boldsymbol{K} := \boldsymbol{R} - \boldsymbol{R}^{-\top}$ for shorthand then the above matrix equation can be represented as a 2×2 block matrix as

$$\begin{bmatrix} \tilde{\boldsymbol{\Sigma}}_1^2 \tilde{\boldsymbol{V}}_1^\top \boldsymbol{V}_1 - \tilde{\boldsymbol{V}}_1^\top \boldsymbol{V}_1 \boldsymbol{\Sigma}_1^2 & \tilde{\boldsymbol{\Sigma}}_1^2 \tilde{\boldsymbol{V}}_1^\top \boldsymbol{V}_2 - \tilde{\boldsymbol{V}}_1^\top \boldsymbol{V}_2 \boldsymbol{\Sigma}_2^2 \\ \tilde{\boldsymbol{\Sigma}}_2^2 \tilde{\boldsymbol{V}}_2^\top \boldsymbol{V}_1 - \tilde{\boldsymbol{V}}_2^\top \boldsymbol{V}_1 \boldsymbol{\Sigma}_1^2 & \tilde{\boldsymbol{\Sigma}}_2^2 \tilde{\boldsymbol{V}}_2^\top \boldsymbol{V}_2 - \tilde{\boldsymbol{V}}_2^\top \boldsymbol{V}_2 \boldsymbol{\Sigma}_2^2 \end{bmatrix}$$

$$= \begin{bmatrix} \tilde{\boldsymbol{\Sigma}}_1 \tilde{\boldsymbol{U}}_1^\top \boldsymbol{Q} \boldsymbol{K} \begin{bmatrix} \boldsymbol{\Sigma}_1 \\ \boldsymbol{0}_{k \times (n-k)} \end{bmatrix} & \tilde{\boldsymbol{\Sigma}}_1 \tilde{\boldsymbol{U}}_1^\top \boldsymbol{Q} \boldsymbol{K} \begin{bmatrix} \boldsymbol{0}_{(n-k) \times k} \\ \boldsymbol{\Sigma}_2 \end{bmatrix} \\ \tilde{\boldsymbol{\Sigma}}_2 \tilde{\boldsymbol{U}}_2^\top \boldsymbol{Q} \boldsymbol{K} \begin{bmatrix} \boldsymbol{\Sigma}_1 \\ \boldsymbol{0}_{k \times (n-k)} \end{bmatrix} & \tilde{\boldsymbol{\Sigma}}_2 \tilde{\boldsymbol{U}}_2^\top \boldsymbol{Q} \boldsymbol{K} \begin{bmatrix} \boldsymbol{0}_{(n-k) \times k} \\ \boldsymbol{\Sigma}_2 \end{bmatrix} \end{bmatrix}.$$

Taking the $(2,1)$-entry of the block matrix we get

$$\tilde{\boldsymbol{\Sigma}}_2^2 \tilde{\boldsymbol{V}}_2^\top \boldsymbol{V}_1 - \tilde{\boldsymbol{V}}_2^\top \boldsymbol{V}_1 \boldsymbol{\Sigma}_1^2 = \tilde{\boldsymbol{\Sigma}}_2 \tilde{\boldsymbol{U}}_2^\top \boldsymbol{Q} \boldsymbol{K} \begin{bmatrix} \boldsymbol{I}_{n-k} \\ \boldsymbol{0}_{k \times (n-k)} \end{bmatrix} \boldsymbol{\Sigma}_1.$$

This is in the form of the Sylvester equation in Lemma 3.2.1. Thus, using Lemma 3.2.1 we get for any unitarily invariant norm

$$\left\| \tilde{\boldsymbol{V}}_2^\top \boldsymbol{V}_1 \right\| \leq \frac{\left\| \tilde{\boldsymbol{U}}_2^\top \boldsymbol{Q} \boldsymbol{K} \begin{bmatrix} \boldsymbol{I}_{n-k} \\ \boldsymbol{0}_{k \times (n-k)} \end{bmatrix} \right\|}{\chi\left(\alpha^2, (\alpha + \delta)^2\right)}.$$

Therefore,

$$\begin{aligned}\left\|\sin\boldsymbol{\Theta}(\boldsymbol{V}_2,\tilde{\boldsymbol{V}}_2)\right\| = \left\|\tilde{\boldsymbol{V}}_2^\top \boldsymbol{V}_1\right\| &\leq \frac{\left\|\tilde{\boldsymbol{U}}_2^\top \boldsymbol{Q}\boldsymbol{K}\begin{bmatrix}\boldsymbol{I}_{n-k}\\ \boldsymbol{0}_{k\times(n-k)}\end{bmatrix}\right\|}{\chi\big(\alpha^2,(\alpha+\delta)^2\big)}\\ &\leq \frac{\left\|\tilde{\boldsymbol{U}}_2^\top \boldsymbol{Q}\right\|_2 \|\boldsymbol{K}\| \left\|\begin{bmatrix}\boldsymbol{I}_{n-k}\\ \boldsymbol{0}_{k\times(n-k)}\end{bmatrix}\right\|_2}{\chi\big(\alpha^2,(\alpha+\delta)^2\big)}\\ &\leq \frac{\left\|\boldsymbol{R}-\boldsymbol{R}^{-\top}\right\|}{\chi\big(\alpha^2,(\alpha+\delta)^2\big)}\end{aligned}$$

for any unitarily invariant norm. □

Corollary 3.2.1. *In the setting of Theorem 3.2.1, we have*

$$\left\|\sin\boldsymbol{\Theta}(\boldsymbol{V}_2,\tilde{\boldsymbol{V}}_2)\right\|_2 \leq \frac{2.1}{\chi(\alpha^2,(\alpha+\delta)^2)} \tag{3.6}$$

with failure probability at most $O(n^{-1})$ for an SRTT matrix with sketch size s satisfying $4[\sqrt{n}+\sqrt{8\log(mn)}]^2\log n \leq s \leq m$. For a Gaussian embedding, Equation (3.6) *is satisfied with failure probability at most* $\exp(-n/50)$ *with sketch size $s = 4n$.*

Proof. The proof follows by noting that if the SVD of $\boldsymbol{R} = \boldsymbol{U}_{\boldsymbol{R}}\boldsymbol{\Sigma}_{\boldsymbol{R}}\boldsymbol{V}_{\boldsymbol{R}}^\top$ where $\boldsymbol{R}$ is defined as in Theorem 3.2.1, then

$$\left\|\boldsymbol{R}-\boldsymbol{R}^{-\top}\right\|_2 = \left\|\boldsymbol{U}_{\boldsymbol{R}}\boldsymbol{\Sigma}_{\boldsymbol{R}}\boldsymbol{V}_{\boldsymbol{R}}^\top - \boldsymbol{U}_{\boldsymbol{R}}\boldsymbol{\Sigma}_{\boldsymbol{R}}^{-1}\boldsymbol{V}_{\boldsymbol{R}}^\top\right\|_2 \leq \max_{\sigma\in[0.4,1.6]}|\sigma-\sigma^{-1}| = 2.1$$

where $\sigma_i(\boldsymbol{R}) \in [0.4, 1.6]$ follows from the discussion on random embeddings in Section 2.2. □

A priori bound

In certain cases, we might be interested in a priori bounds. We can obtain them by substituting the singular values in place of α and δ in Corollary 3.2.1. There are two a priori bounds, which are given below. If $\sigma_{n-k} > 1.6\ \sigma_{n-k+1}$ then

$$\left\|\sin\boldsymbol{\Theta}(\boldsymbol{V}_2, \tilde{\boldsymbol{V}}_2)\right\|_2 \leq \frac{2.1 \cdot \sigma_{n-k}\ \tilde{\sigma}_{n-k+1}}{\sigma_{n-k}^2 - \tilde{\sigma}_{n-k+1}^2} \leq \frac{3.36 \cdot \sigma_{n-k}\ \sigma_{n-k+1}}{\sigma_{n-k}^2 - 2.56 \cdot \sigma_{n-k+1}^2}, \tag{3.7}$$

and if $0.4\ \sigma_{n-k} > \sigma_{n-k+1}$ then

$$\left\|\sin\boldsymbol{\Theta}(\boldsymbol{V}_2, \tilde{\boldsymbol{V}}_2)\right\|_2 \leq \frac{2.1 \cdot \tilde{\sigma}_{n-k}\ \sigma_{n-k+1}}{\tilde{\sigma}_{n-k}^2 - \sigma_{n-k+1}^2} \leq \frac{3.36 \cdot \sigma_{n-k}\ \sigma_{n-k+1}}{0.16 \cdot \sigma_{n-k}^2 - \sigma_{n-k+1}^2}, \tag{3.8}$$

since in the setting of Corollary 3.2.1, we have

$$0.4\ \sigma_i \leq \tilde{\sigma}_i \leq 1.6\ \sigma_i$$

for all i.

The upper bounds, Equations (3.7) and (3.8) are informative ($\ll 1$) if $\sigma_{n-k} \gg \sigma_{n-k+1}$. In this case, the upper bounds are $\approx \frac{\sigma_{n-k+1}}{\sigma_{n-k}}$, which make them both much less than 1. In particular, if $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_{n-k} > \sigma_{n-k+1} = \cdots = \sigma_n = 0$ then the multiplicative perturbation by $\boldsymbol{S}^\top$ preserves the null space exactly as the upper bound becomes zero. In the presence of rounding errors, a backward stable solution would correspond to $\sigma_{n-k+1} = \mathcal{O}(u)$ where u is the unit roundoff. Assuming σ_{n-k} is sufficiently larger than u, the last k right singular vectors of the sketch give an excellent approximation for the null space of the original matrix.

To illustrate the theoretical result, Figure 3.1 shows the accuracy of the bound in Corollary 3.2.1 for the case when $k = 1$. We generated a random 1000×100 matrix with Haar-distributed right singular vectors. Haar-distributed vectors are vectors sampled

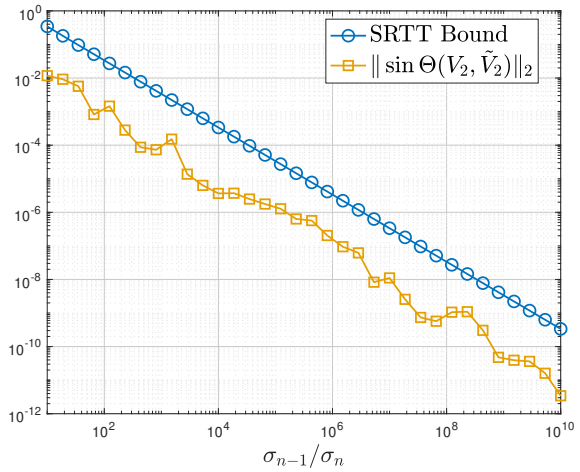
uniformed from n-sphere [Mec19, Chp. 1.2]. They can be generated in MATLAB using $[\mathtt{Q},\mathtt{R}] = \mathtt{qr}(\mathtt{randn}(\mathtt{m},\mathtt{n}),0);\ \ \mathtt{U} = \mathtt{Q} * \mathtt{sign}(\mathtt{diag}(\mathtt{R}))$, where $m \geq n$ and $\mathtt{U} \in \mathbb{R}^{m\times n}$ is a Haar-distributed subspace of dimension n. The left singular vectors in the top left plot are also Haar-distributed while the other 3 plots were generated with the left singular vectors equal to $[\boldsymbol{I}_{100}, \boldsymbol{0}]^\top \in \mathbb{R}^{1000\times 100}$, which is a difficult (coherent) example for SRTTs [Tro11, BG13]. We set the singular values as $(\sigma_1, \sigma_2, ..., \sigma_{n-1}, \sigma_n) = (1, 1, ..., 1, 10^{-1}, \sigma_n)$ where $\sigma_{n-1}/\sigma_n \in [10, 10^{10}]$. We then sketch the matrix with a Gaussian matrix and an SRTT matrix. In Figure 3.1, we observe that except in the bottom left plot, the bounds in Corollary 3.2.1 give us the correct decay rate with a factor that is not too large. As seen in the bottom left plot, the SRTT sketch can *fail*, that is, the SRTT sketch fails to be a subspace embedding if we set the sketch size equal to cn for a small constant c, say $c = 1.5$, for a difficult example (coherent) [HMT11, Tro11]. However, we can overcome this issue if we make the SRTT sketch size $\mathcal{O}(n \log n)$ as shown in the bottom right plot of Figure 3.1. This shows us that when $\sigma_{n-1} \gg \sigma_n$ we expect the sketched solution to give a very good approximation to the exact solution.

Up until now, we have examined the canonical angles between two subspaces with the same dimension. We next extend Theorem 3.2.1 to subspaces of different dimensions. The extension will be useful when the bound in Theorem 3.2.1 is not informative, $\mathcal{O}(1)$, and we want to search for a bigger subspace that can be shown to contain the exact subspace that we are looking for.

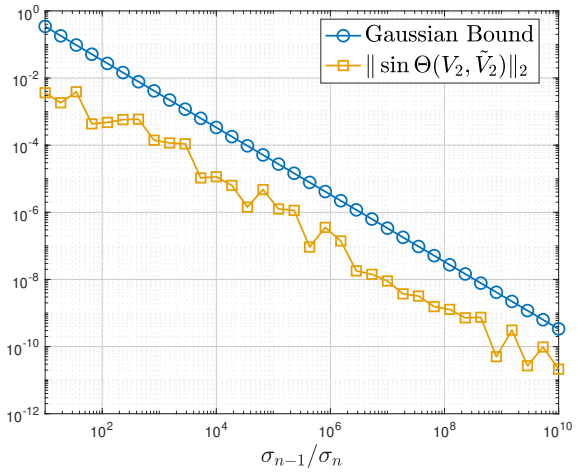
3.2.2 Computing subspaces of different dimensions

Let $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ and $\tilde{\boldsymbol{A}} \in \mathbb{R}^{s\times n}$ be matrices with $m \geq s \geq n$ and suppose that they have SVDs of the form

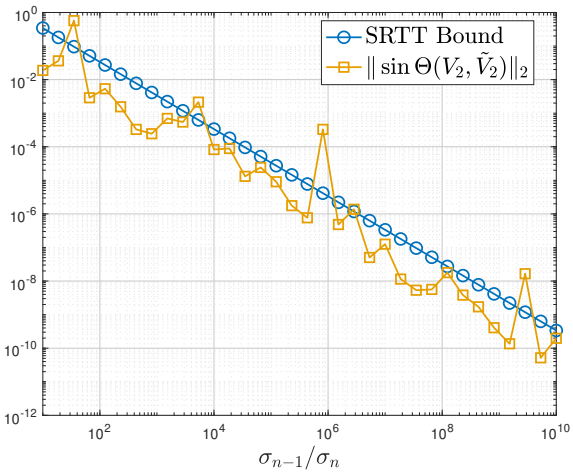
$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top = [\boldsymbol{U}_1, \boldsymbol{U}_2, \boldsymbol{U}_3] \begin{bmatrix} \boldsymbol{\Sigma}_1 & & \\ & \boldsymbol{\Sigma}_2 & \\ & & \boldsymbol{\Sigma}_3 \end{bmatrix} [\boldsymbol{V}_1, \boldsymbol{V}_2, \boldsymbol{V}_3]^\top \tag{3.9}$$



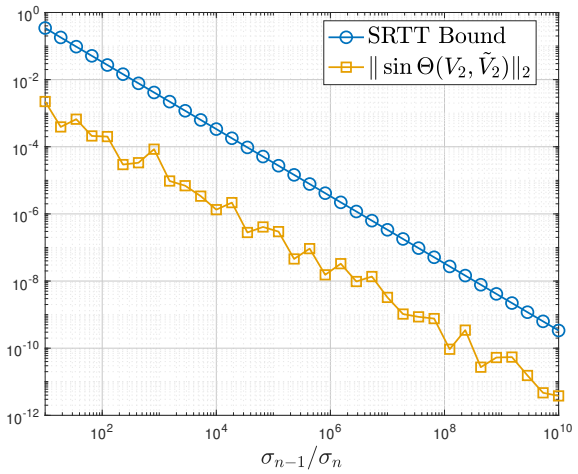
(a) SRTT ($s = \lfloor 1.5n \rfloor$): Incoherent



(b) Gaussian ($s = \lfloor 1.5n \rfloor$): Coherent



(c) SRTT ($s = \lfloor 1.5n \rfloor$): Coherent



(d) SRTT ($s = \lfloor 1.5n \log n \rfloor$): Coherent

Figure 3.1: Loglog plot of the bound Corollary 3.2.1 against the actual sine values for $m = 1000$, $n = 100$ and $k = 1$. The matrix was generated with singular values $\text{diag}(1, 1, ..., 1, 10^{-1}, \sigma_n)$ where $\sigma_{n-1}/\sigma_n \in [10, 10^{10}]$. The bottom left plot shows a difficult example (coherent) for the SRTT sketch which can fail if the sketch size is not large enough. However, as seen on the bottom right plot it can be fixed by enlarging the sketch size to $\mathcal{O}(n \log n)$.

$$\tilde{\boldsymbol{A}} = \tilde{\boldsymbol{U}}\tilde{\boldsymbol{\Sigma}}\tilde{\boldsymbol{V}}^\top = \left[\tilde{\boldsymbol{U}}_1, \tilde{\boldsymbol{U}}_2, \tilde{\boldsymbol{U}}_3\right] \begin{bmatrix} \tilde{\boldsymbol{\Sigma}}_1 & & \\ & \tilde{\boldsymbol{\Sigma}}_2 & \\ & & \tilde{\boldsymbol{\Sigma}}_3 \end{bmatrix} \left[\tilde{\boldsymbol{V}}_1, \tilde{\boldsymbol{V}}_2, \tilde{\boldsymbol{V}}_3\right]^\top \tag{3.10}$$

where $\boldsymbol{U} \in \mathbb{R}^{m\times n}$, $\tilde{\boldsymbol{U}} \in \mathbb{R}^{s\times n}$, $\boldsymbol{V}, \tilde{\boldsymbol{V}} \in \mathbb{R}^{n\times n}$ and the matrices with subscript 1 have $(n-k)$ columns, those with subscript 2 have $(k-\ell)$ columns and subscript 3 have ℓ columns with $1 \leq \ell < k < n$, and

$$\boldsymbol{\Sigma}_1 = \mathrm{diag}(\sigma_1, ..., \sigma_{n-k}), \boldsymbol{\Sigma}_2 = \mathrm{diag}(\sigma_{n-k+1}, ..., \sigma_{n-\ell}), \boldsymbol{\Sigma}_3 = \mathrm{diag}(\sigma_{n-\ell+1}, ..., \sigma_n), \tag{3.11}$$

$$\tilde{\boldsymbol{\Sigma}}_1 = \mathrm{diag}(\tilde{\sigma}_1, ..., \tilde{\sigma}_{n-k}), \tilde{\boldsymbol{\Sigma}}_2 = \mathrm{diag}(\tilde{\sigma}_{n-k+1}, ..., \tilde{\sigma}_{n-\ell}), \tilde{\boldsymbol{\Sigma}}_3 = \mathrm{diag}(\tilde{\sigma}_{n-\ell+1}, ..., \tilde{\sigma}_n), \tag{3.12}$$

where the singular values are arranged in non-increasing order.

Theorem 3.2.2. *Let $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ and $\tilde{\boldsymbol{A}} = \boldsymbol{S}^\top\boldsymbol{A} \in \mathbb{R}^{s\times n}$ $(m \geq s \geq n)$ with SVDs as in Equations* (3.9) *to* (3.12) *where $\boldsymbol{S} \in \mathbb{R}^{m\times s}$ is a matrix such that $\boldsymbol{S}^\top\boldsymbol{U}$ has full column rank. Let the thin QR decomposition of $\boldsymbol{S}^\top\boldsymbol{U}$ be $\boldsymbol{QR}$. Suppose that there exist $\alpha > 0$ and $\delta > 0$ such that*

$$\sigma_{n-k} \geq \alpha + \delta \qquad \textit{and} \qquad \tilde{\sigma}_{n-\ell+1} \leq \alpha.$$

Then for any unitarily invariant norm we have

$$\left\|\sin\boldsymbol{\Theta}\left(\left[\boldsymbol{V}_2, \boldsymbol{V}_3\right], \tilde{\boldsymbol{V}}_3\right)\right\| \leq \frac{\left\|\boldsymbol{R} - \boldsymbol{R}^{-\top}\right\|}{\chi\left(\alpha^2, (\alpha+\delta)^2\right)}.$$

Alternatively, if there exist $\alpha > 0$ and $\delta > 0$ such that

$$\tilde{\sigma}_{n-k} \geq \alpha + \delta \qquad \textit{and} \qquad \sigma_{n-\ell+1} \leq \alpha,$$

then for any unitarily invariant norm we have

$$\left\|\sin\boldsymbol{\Theta}\left(\boldsymbol{V}_3, \left[\tilde{\boldsymbol{V}}_2, \tilde{\boldsymbol{V}}_3\right]\right)\right\| \leq \frac{\left\|\boldsymbol{R} - \boldsymbol{R}^{-\top}\right\|}{\chi\left(\alpha^2, (\alpha+\delta)^2\right)},$$

where $\boldsymbol{V}_2, \boldsymbol{V}_3, \tilde{\boldsymbol{V}}_2$ *and* $\tilde{\boldsymbol{V}}_3$ *are as in Equations* (3.9) *and* (3.10)*.*

Proof. We follow the proof of Theorem 3.2.1. We consider the case

$$\sigma_{n-k} \geq \alpha + \delta \qquad \text{and} \qquad \tilde{\sigma}_{n-\ell+1} \leq \alpha.$$

The other case follows similarly. As in Theorem 3.2.1, we define two matrices $\boldsymbol{B} := \boldsymbol{\Sigma}\boldsymbol{V}^\top$ and $\tilde{\boldsymbol{B}} := \boldsymbol{R}\boldsymbol{\Sigma}\boldsymbol{V}^\top$ that have the same singular values and the right singular vectors as $\boldsymbol{A}$ and $\tilde{\boldsymbol{A}}$ respectively. Therefore, it suffices to work with $\boldsymbol{B}$ and $\tilde{\boldsymbol{B}}$. We first note that

$$\tilde{\boldsymbol{B}}^\top\tilde{\boldsymbol{B}} - \boldsymbol{B}^\top\boldsymbol{B} = \tilde{\boldsymbol{B}}^\top\boldsymbol{R}\boldsymbol{B} - \tilde{\boldsymbol{B}}^\top\boldsymbol{R}^{-\top}\boldsymbol{B} = \tilde{\boldsymbol{B}}^\top(\boldsymbol{R} - \boldsymbol{R}^{-\top})\boldsymbol{B}.$$

This is equivalent to

$$\tilde{\boldsymbol{V}}\tilde{\boldsymbol{\Sigma}}^2\tilde{\boldsymbol{V}}^\top - \boldsymbol{V}\boldsymbol{\Sigma}^2\boldsymbol{V}^\top = \tilde{\boldsymbol{V}}\tilde{\boldsymbol{\Sigma}}\left(\boldsymbol{Q}^\top\tilde{\boldsymbol{U}}\right)^\top(\boldsymbol{R} - \boldsymbol{R}^{-\top})\boldsymbol{\Sigma}\boldsymbol{V}^\top,$$

since $\tilde{\boldsymbol{B}} = \boldsymbol{Q}^\top\tilde{\boldsymbol{A}} = \boldsymbol{Q}^\top\tilde{\boldsymbol{U}}\tilde{\boldsymbol{\Sigma}}\tilde{\boldsymbol{V}}^\top$. Left-multiplying $\tilde{\boldsymbol{V}}^\top$ and right-multiplying $\boldsymbol{V}$ gives

$$\tilde{\boldsymbol{\Sigma}}^2\tilde{\boldsymbol{V}}^\top\boldsymbol{V} - \tilde{\boldsymbol{V}}^\top\boldsymbol{V}\boldsymbol{\Sigma}^2 = \tilde{\boldsymbol{\Sigma}}\left(\boldsymbol{Q}^\top\tilde{\boldsymbol{U}}\right)^\top(\boldsymbol{R} - \boldsymbol{R}^{-\top})\boldsymbol{\Sigma}.$$

Now the above matrix equation can be represented as a 3×3 block matrix using Equations (3.9) and (3.10) in a similar manner to the proof of Theorem 3.2.1. Taking the $(3,1)$-entry of the

block matrix we get

$$\tilde{\boldsymbol{\Sigma}}_3^2\tilde{\boldsymbol{V}}_3^\top\boldsymbol{V}_1 - \tilde{\boldsymbol{V}}_3^\top\boldsymbol{V}_1\boldsymbol{\Sigma}_1^2 = \tilde{\boldsymbol{\Sigma}}_3\tilde{\boldsymbol{U}}_3^\top\boldsymbol{Q}(\boldsymbol{R}-\boldsymbol{R}^{-\top})\begin{bmatrix}\boldsymbol{I}_{n-k}\\ \boldsymbol{0}_{k\times(n-k)}\end{bmatrix}\boldsymbol{\Sigma}_1.$$

This is again in the form of the Sylvester equation in Lemma 3.2.1. Thus, using Lemma 3.2.1, we get that for any unitarily invariant norm

$$\left\|\tilde{\boldsymbol{V}}_3^\top\boldsymbol{V}_1\right\| \leq \frac{\left\|\tilde{\boldsymbol{U}}_3^\top\boldsymbol{Q}(\boldsymbol{R}-\boldsymbol{R}^{-\top})\begin{bmatrix}\boldsymbol{I}_{n-k}\\ \boldsymbol{0}_{k\times(n-k)}\end{bmatrix}\right\|}{\chi\big(\alpha^2,(\alpha+\delta)^2\big)}.$$

Finally we get

$$\begin{aligned}\left\|\sin\boldsymbol{\Theta}\left(\left[\boldsymbol{V}_2,\boldsymbol{V}_3\right],\tilde{\boldsymbol{V}}_3\right)\right\| = \left\|\tilde{\boldsymbol{V}}_3^\top\boldsymbol{V}_1\right\| &\leq \frac{\left\|\tilde{\boldsymbol{U}}_3^\top\boldsymbol{Q}(\boldsymbol{R}-\boldsymbol{R}^{-\top})\begin{bmatrix}\boldsymbol{I}_{n-k}\\ \boldsymbol{0}_{k\times(n-k)}\end{bmatrix}\right\|}{\chi\big(\alpha^2,(\alpha+\delta)^2\big)}\\ &\leq \frac{\left\|\tilde{\boldsymbol{U}}_3^\top\boldsymbol{Q}\right\|_2\left\|\boldsymbol{R}-\boldsymbol{R}^{-\top}\right\|\left\|\begin{bmatrix}\boldsymbol{I}_{n-k}\\ \boldsymbol{0}_{k\times(n-k)}\end{bmatrix}\right\|_2}{\chi\big(\alpha^2,(\alpha+\delta)^2\big)}\\ &\leq \frac{\left\|\boldsymbol{R}-\boldsymbol{R}^{-\top}\right\|}{\chi\big(\alpha^2,(\alpha+\delta)^2\big)}\end{aligned}$$

for any unitarily invariant norm. □

Corollary 3.2.2. *In the setting of Theorem 3.2.2, we have*

$$\left\|\sin\boldsymbol{\Theta}\left(\boldsymbol{V}_3,\left[\tilde{\boldsymbol{V}}_2,\tilde{\boldsymbol{V}}_3\right]\right)\right\|_2 \leq \frac{2.1}{\chi(\alpha^2,(\alpha+\delta)^2)} \tag{3.13}$$

with failure probability at most $O(n^{-1})$ for an SRTT matrix with sketch size s satisfying

$4[\sqrt{n}+\sqrt{8\log(mn)}]^2\log n \le s \le m$. *For a Gaussian embedding, Equation* (3.13) *is satisfied with failure probability at most* $\exp(-n/50)$ *with sketch size* $s = 4n$.

Proof. The proof is identical to the proof of Corollary 3.2.1. □

A priori bound

Similarly as before, we may be interested in a priori bounds when the subspaces have different sizes. We can obtain them by substituting the singular values in place of α and δ in Corollary 3.2.2. There are two a priori bounds, which are given below. If $0.4\ \sigma_{n-k} > \sigma_{n-\ell+1}$ then

$$\left\|\sin\boldsymbol{\Theta}\left(\boldsymbol{V}_3, \left[\tilde{\boldsymbol{V}}_2, \tilde{\boldsymbol{V}}_3\right]\right)\right\|_2 \le \frac{2.1\cdot\tilde{\sigma}_{n-k}\ \sigma_{n-\ell+1}}{\tilde{\sigma}_{n-k}^2-\sigma_{n-\ell+1}^2} \le \frac{3.36\cdot\sigma_{n-k}\ \sigma_{n-\ell+1}}{0.16\cdot\sigma_{n-k}^2-\sigma_{n-\ell+1}^2}, \tag{3.14}$$

and if $\sigma_{n-k} > 1.6\ \sigma_{n-\ell+1}$ then

$$\left\|\sin\boldsymbol{\Theta}\left(\left[\boldsymbol{V}_2, \boldsymbol{V}_3\right], \tilde{\boldsymbol{V}}_3\right)\right\|_2 \le \frac{2.1\cdot\sigma_{n-k}\ \tilde{\sigma}_{n-\ell+1}}{\sigma_{n-k}^2-\tilde{\sigma}_{n-\ell+1}^2} \le \frac{3.36\cdot\sigma_{n-k}\ \sigma_{n-\ell+1}}{\sigma_{n-k}^2-2.56\cdot\sigma_{n-\ell+1}^2}, \tag{3.15}$$

since in the setting of Corollary 3.2.2, we have

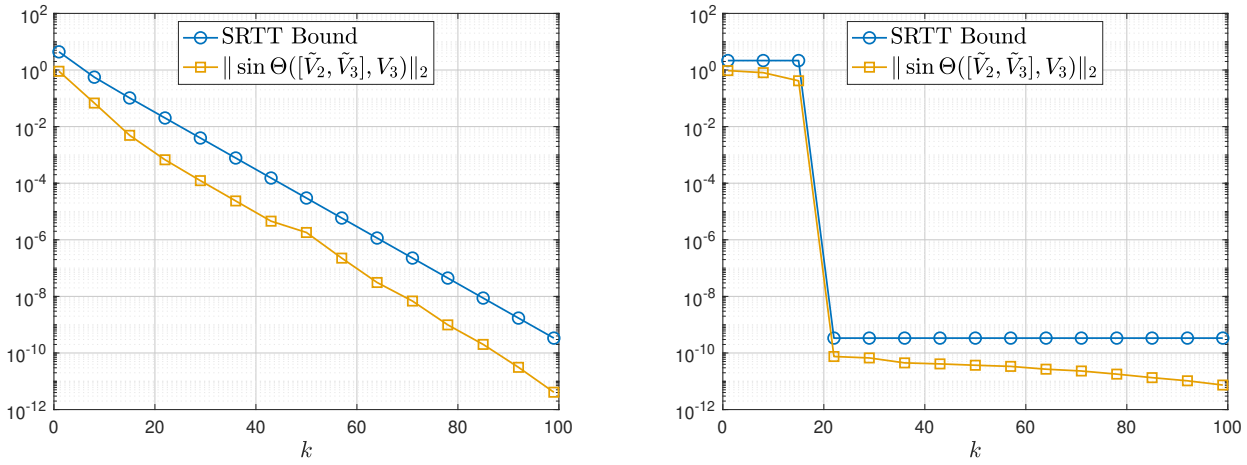
$$0.4\ \sigma_i \le \tilde{\sigma}_i \le 1.6\ \sigma_i$$

for all i.

The upper bounds Equations (3.14) and (3.15) are informative ($\ll 1$) if $\sigma_{n-k} \gg \sigma_{n-\ell+1}$. In this case, the upper bounds are $\approx \frac{\sigma_{n-\ell+1}}{\sigma_{n-k}}$, which are both much less than 1. In contrast to Corollary 3.2.1, Corollary 3.2.2 may still give meaningful bounds even when $\sigma_{n-k} \approx \sigma_{n-k+1}$.

Now we illustrate Corollary 3.2.2 through numerical experiments. Figure 3.2 shows the accuracy of the bound in Corollary 3.2.2 using an SRTT in the case when $\ell = 1$ and $k = 2, ..., n-1$. We generated a random 1000×100 matrix by sampling the left and the right singular vectors as before for the coherent example in Figure 3.1, but the singular values now

decay geometrically from 1 to 10^{-10} for the left plot and the right plot has singular values equal to 1 for the first 80 singular values and 10^{-10} for the last 20 singular values. In both of these cases, we have $\sigma_n/\sigma_{n-1} \approx 1$, so the previous bound in Corollary 3.2.1 is useless. However, the bound in Corollary 3.2.2 becomes meaningful as we increase the subspace dimension k. In Figure 3.2, we see that the bounds in Corollary 3.2.2 give us the correct decay rate with a modest factor for the left plot. For the right plot, we see that after the subspace becomes large enough (> 20) to contain the right singular subspace corresponding to the singular value 10^{-10}, we get a good subspace which approximately contains the right singular vector corresponding to the smallest singular value of the original matrix. This illustrates us that even when $\sigma_n/\sigma_{n-1} \approx 1$, as long as $\sigma_{n-k} \gg \sigma_n$, we can find a larger subspace of dimension k that is expected to contain the right singular vector corresponding to the smallest singular value.



(a) SRTT ($s = 2n$): Coherent

(b) SRTT ($s = 2n$): Coherent

Figure 3.2: Semilogy plot of the bound in Corollary 3.2.2 against the actual sine values for $m = 1000, n = 100, \ell = 1$ and subspace size $k = 2, ..., n-1$. The matrix was generated with singular values that decay geometrically from 1 to 10^{-10} for the left plot and the right plot has singular values equal to 1 for the first 80 singular values and the last 20 singular values are equal to 10^{-10}. The bound in Corollary 3.2.2 gives us a good indication as to how much the computed subspace contains the trailing right singular vector.

3.3 Application: Total Least Squares

Total least squares (TLS) [GVL80] is a type of least squares problem where errors in both independent and dependent variables are allowed. TLS is also known as errors-in-variables models in statistics. This is different from the standard least-squares problem where errors only occur in dependent variables. In a standard least-squares problem, we are interested in solving the following optimization problem

$$\min_{\boldsymbol{x}\in\mathbb{R}^n} \|\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b}\|_2$$

where $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ and $\boldsymbol{b} \in \mathbb{R}^m$ with $m \geq n$. The errors only occur in the dependent variables $\boldsymbol{b}$, so the problem can be restated as

$$\min_{\boldsymbol{b}+\boldsymbol{r}\in\mathrm{range}(\boldsymbol{A})} \|\boldsymbol{r}\|_2 .$$

Now if we also allow errors in the independent variables, represented by $\boldsymbol{A}$, we get the optimization problem formulation for the TLS problem, which is

$$\min_{\boldsymbol{b}+\boldsymbol{r}\in\mathrm{range}(\boldsymbol{A}+\boldsymbol{E})} \|[\boldsymbol{E}|\boldsymbol{r}]\|_{\mathrm{F}}$$

where $[\cdot|\cdot]$ represents the augmented matrix of size $m \times (n+1)$.

Now, if we allow multiple right-hand sides, say k, we get

$$\min_{\boldsymbol{B}+\boldsymbol{R}\in\mathrm{range}(\boldsymbol{A}+\boldsymbol{E})} \|[\boldsymbol{E}|\boldsymbol{R}]\|_{\mathrm{F}} \tag{3.16}$$

where $\boldsymbol{A}, \boldsymbol{E} \in \mathbb{R}^{m\times n}$ and $\boldsymbol{B}, \boldsymbol{R} \in \mathbb{R}^{m\times k}$. We will also impose $m \geq n+k$ and $n \geq k$. If $[\boldsymbol{E}_0|\boldsymbol{R}_0]$ solves the optimization problem, Equation (3.16), then any $\boldsymbol{X} \in \mathbb{R}^{n\times k}$ satisfying $(\boldsymbol{A}+\boldsymbol{E}_0)\boldsymbol{X} = \boldsymbol{B}_0 + \boldsymbol{R}_0$ is called the TLS solution.

In 1980, Golub and Van Loan [GVL80] gave a solution to the TLS problem. The algorithm solves the approximate null space problem,

$$\boldsymbol{V}_k = \underset{\boldsymbol{V}^\top \boldsymbol{V} = \boldsymbol{I}_k}{\arg\min} \left\| [\boldsymbol{A}|\boldsymbol{B}]\, \boldsymbol{V} \right\|_{\mathrm{F}}$$

with $\boldsymbol{V}_k = \begin{bmatrix} \boldsymbol{V}_{k,1} \\ \boldsymbol{V}_{k,2} \end{bmatrix} \in \mathbb{R}^{(n+k)\times k}$ where $\boldsymbol{V}_{k,1} \in \mathbb{R}^{n\times k}$ and $\boldsymbol{V}_{k,2} \in \mathbb{R}^{k\times k}$ and sets the solution to $\boldsymbol{X} = -\boldsymbol{V}_{k,1}\boldsymbol{V}_{k,2}^{-1} \in \mathbb{R}^{n\times k}$. Note that since we are inverting $\boldsymbol{V}_{k,2}$, the solution obtained with this approach may not exist. There are known conditions for existence and uniqueness, for example when $\sigma_n(\boldsymbol{A}) > \sigma_{n+1}([\boldsymbol{A}|\boldsymbol{B}])$. For more information see [GVL80, VHV91]. Throughout this section, we assume $\sigma_n(\boldsymbol{A}) > \sigma_{n+1}([\boldsymbol{A}|\boldsymbol{B}])$ so that the solution for the TLS problem exists. The cost of computing the TLS solution is $\mathcal{O}(m(n+k)^2)$ flops for computing the SVD of $[\boldsymbol{A}|\boldsymbol{B}] \in \mathbb{R}^{m\times(n+k)}$.

For the sketched version, we sketch the augmented matrix $[\boldsymbol{A}|\boldsymbol{B}]$ using an SRTT sketch in $\mathcal{O}(m(n+k)\log(n+k))$ flops and solve the TLS problem for a smaller-sized matrix using $\mathcal{O}((n+k)^3)$ flops. Overall, the sketched version costs $\mathcal{O}(\ m(n+k)\log(n+k) + (n+k)^3)$, which becomes effective when $m \gg (n+k)$.

To assess the accuracy of the sketched TLS solution, we consider the condition number of the TLS problem, which depends on the gap between $\sigma_n(\boldsymbol{A})$ and $\sigma_{n+1}([\boldsymbol{A}|\boldsymbol{B}])$ [GVL80]. Specifically, we expect both the relative error $\frac{\left\|\boldsymbol{X}-\tilde{\boldsymbol{X}}\right\|_2}{\|\boldsymbol{X}\|_2}$, and $\left\|\sin\boldsymbol{\Theta}(\boldsymbol{X},\tilde{\boldsymbol{X}})\right\|_2$ to be proportional to the relative gap $\frac{\sigma_{n+1}([\boldsymbol{A}|\boldsymbol{B}])}{\sigma_n(\boldsymbol{A})}$. Here, $\boldsymbol{X} \in \mathbb{R}^{n\times k}$ is the TLS solution and $\tilde{\boldsymbol{X}} \in \mathbb{R}^{n\times k}$ is the sketched TLS solution. In addition, we expect $\left\|\sin\boldsymbol{\Theta}(\boldsymbol{V}_k,\tilde{\boldsymbol{V}}_k)\right\|_2$ to be proportional to the relative gap $\frac{\sigma_{n+1}([\boldsymbol{A}|\boldsymbol{B}])}{\sigma_n([\boldsymbol{A}|\boldsymbol{B}])}$ by our analysis in Section 3.2, where $\boldsymbol{V}_k, \tilde{\boldsymbol{V}}_k \in \mathbb{R}^{(n+k)\times k}$ are the trailing right singular vectors of the original and sketched TLS problems, respectively. Now, by Cauchy interlace theorem, we have $\sigma_n([\boldsymbol{A}|\boldsymbol{B}]) \geq \sigma_n(\boldsymbol{A})$, and under the assumption

$\sigma_n(\boldsymbol{A}) > \sigma_{n+1}([\boldsymbol{A}|\boldsymbol{B}])$, it follows that

$$\frac{\sigma_{n+1}([\boldsymbol{A}|\boldsymbol{B}])}{\sigma_n([\boldsymbol{A}|\boldsymbol{B}])} \leq \frac{\sigma_{n+1}([\boldsymbol{A}|\boldsymbol{B}])}{\sigma_n(\boldsymbol{A})} < 1.$$

Therefore, when $\sigma_n([\boldsymbol{A}|\boldsymbol{B}]) \approx \sigma_n(\boldsymbol{A})$, the error quantities $\frac{\|\boldsymbol{X}-\tilde{\boldsymbol{X}}\|_2}{\|\boldsymbol{X}\|_2}$, $\left\|\sin\boldsymbol{\Theta}(\boldsymbol{X},\tilde{\boldsymbol{X}})\right\|_2$, and $\left\|\sin\boldsymbol{\Theta}(\boldsymbol{V}_k,\tilde{\boldsymbol{V}}_k)\right\|_2$ are expected to behave similarly. We empirically demonstrate the relationship between the error quantities by varying the relative gap $\frac{\sigma_{n+1}([\boldsymbol{A}|\boldsymbol{B}])}{\sigma_n(\boldsymbol{A})}$.

Figure 3.3 was generated using $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ and $\boldsymbol{B} \in \mathbb{R}^{m\times k}$ with $m = 10^5$, $n = 10^3$ and $k = 10$ where $\boldsymbol{A}$ is as in the previous experiments with the singular values that decay geometrically from 1 to 10^{-3}, and $\boldsymbol{B}$ is a sum of a matrix in the span of $\boldsymbol{A}$ with the same norm as $\boldsymbol{A}$ and a Gaussian noise matrix of varying noise level between 10^{-3} and 10^{-12}. In this experiment, the sketch size was $s = 2020$, which is 2 times the sum of the number of columns of $\boldsymbol{A}$ and $\boldsymbol{B}$.

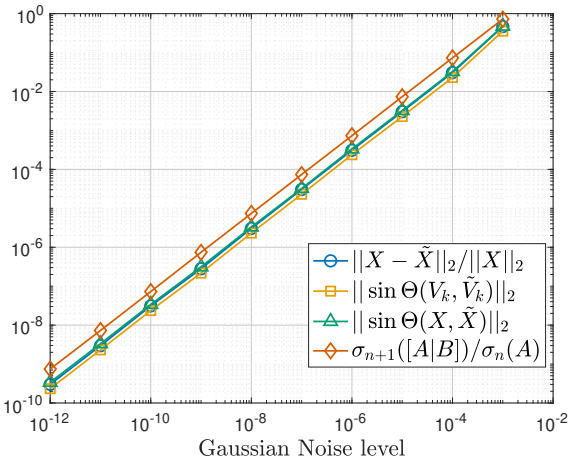


Figure 3.3: A plot demonstrating the similarity between three different error metrics: relative error $\frac{\|\boldsymbol{X}-\tilde{\boldsymbol{X}}\|_2}{\|\boldsymbol{X}\|_2}$ and the angles $\left\|\sin\boldsymbol{\Theta}(\boldsymbol{X},\tilde{\boldsymbol{X}})\right\|_2$ and $\left\|\sin\Theta(\boldsymbol{V}_k,\tilde{\boldsymbol{V}}_k)\right\|_2$. As we increase the relative gap $\frac{\sigma_{n+1}([\boldsymbol{A}|\boldsymbol{B}])}{\sigma_n(\boldsymbol{A})}$, we observe that all three error metrics behave similarly and have similar magnitudes.

We observe in Figure 3.3 that as we vary the noise level and consequently the relative gap $\frac{\sigma_{n+1}([\boldsymbol{A}|\boldsymbol{B}])}{\sigma_n([\boldsymbol{A}|\boldsymbol{B}])}$, the error metrics $\frac{\|\boldsymbol{X}-\tilde{\boldsymbol{X}}\|_2}{\|\boldsymbol{X}\|_2}$, $\left\|\sin\boldsymbol{\Theta}(\boldsymbol{X},\tilde{\boldsymbol{X}})\right\|_2$, and $\left\|\sin\boldsymbol{\Theta}(\boldsymbol{V}_k,\tilde{\boldsymbol{V}}_k)\right\|_2$ all behave

similarly. All three error metrics are proportional to the relative gap and have similar magnitudes. In view of the theory developed in Section 3.2, we use $\left\|\sin\Theta(\boldsymbol{V}_k, \tilde{\boldsymbol{V}}_k)\right\|_2$ as an error metric for analysing the accuracy of the sketched TLS solution. From Section 3.2.1, we have an a priori bound

$$\left\|\sin\boldsymbol{\Theta}(\boldsymbol{V}_k, \tilde{\boldsymbol{V}}_k)\right\|_2 \lesssim \frac{\sigma_{n+1}([\boldsymbol{A}|\boldsymbol{B}])}{\sigma_n([\boldsymbol{A}|\boldsymbol{B}])} \leq \frac{\sigma_{n+1}([\boldsymbol{A}|\boldsymbol{B}])}{\sigma_n(\boldsymbol{A})},$$

where $\lesssim$ is used to hide the constants in the bound. Now we demonstrate the speedup obtained using Algorithm 1 with an experiment. Table 3.1 was generated using the same setup as in Figure 3.3 with $\boldsymbol{A} \in \mathbb{R}^{m\times 1000}$ and $\boldsymbol{B} \in \mathbb{R}^{m\times 10}$, but with varying values of $m = 2^{14}, 2^{15}$, $2^{16}, 2^{17}, 2^{18}$ and a fixed Gaussian noise level of 10^{-3} for the matrix $\boldsymbol{B}$. In Table 3.1, as we increase the value of m, we observe up to $\mathbf{16}\times$ speedup, demonstrating the benefit of sketching. We also notice that the relative error, and the sine of the angle between the trailing right singular vectors of the original TLS problem and the sketched TLS problem are small and similar in magnitude, which is consistent with Figure 3.3. The accuracy of the sketched solution can be improved by enlarging the sketch size at the cost of increasing the overall complexity. Lastly, we see that the sketched solution residual error is only a modest constant larger than the original error; this is expected since sketching gives a near-optimal solution (Theorem 3.1.1). Therefore, the sketched solution is a good approximation for the TLS problem.

3.4 Updating and downdating of the sketch

We now look at how row/column updates or downdates of the original matrix influence the sketch. Let $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ with $m \geq n$. We have been sketching from the left to get $\boldsymbol{S}^\top\boldsymbol{A} \in \mathbb{R}^{s\times n}$ where $\boldsymbol{S} \in \mathbb{R}^{m\times s}$ is a subspace embedding with sketch size s. Sometimes, in problems such as the AAA algorithm [NST18] which we discuss in Section 3.5, we want

Table 3.1: Total least squares problem with $\boldsymbol{A} \in \mathbb{R}^{m\times 1000}$ and $\boldsymbol{B} \in \mathbb{R}^{m\times 10}$ $(n = 1000, k = 10)$. $\tilde{\boldsymbol{X}} \in \mathbb{R}^{1000\times 10}$ is the sketched TLS solution with the TLS error $\left\|[\tilde{\boldsymbol{E}}|\tilde{\boldsymbol{R}}]\right\|_{\mathrm{F}}$ and $\boldsymbol{X} \in \mathbb{R}^{1000\times 10}$ is the original TLS solution with the TLS error $\|[\boldsymbol{E}|\boldsymbol{R}]\|_{\mathrm{F}}$. The orthonormal matrices $\boldsymbol{V}_k$ and $\tilde{\boldsymbol{V}}_k$ are the k trailing right singular vectors of the original TLS problem and the sketched TLS problem, respectively.

m	Speedup	$\|[\boldsymbol{E}\|\boldsymbol{R}]\|_{\mathrm{F}}$	$\frac{\left\|[\tilde{\boldsymbol{E}}\|\tilde{\boldsymbol{R}}]\right\|_{\mathrm{F}}}{\|[\boldsymbol{E}\|\boldsymbol{R}]\|_{\mathrm{F}}}$	$\frac{\left\|\boldsymbol{X}-\tilde{\boldsymbol{X}}\right\|_2}{\|\boldsymbol{X}\|_2}$	$\left\|\sin\boldsymbol{\Theta}(\boldsymbol{V}_k, \tilde{\boldsymbol{V}}_k)\right\|_2$
2^{14}	3.71×	$2.16 \cdot 10^{-8}$	1.39	$2.65 \cdot 10^{-6}$	$2.21 \cdot 10^{-6}$
2^{15}	6.07×	$2.21 \cdot 10^{-8}$	1.40	$2.98 \cdot 10^{-6}$	$2.46 \cdot 10^{-6}$
2^{16}	8.55×	$2.22 \cdot 10^{-8}$	1.40	$3.00 \cdot 10^{-6}$	$2.32 \cdot 10^{-6}$
2^{17}	14.12×	$2.22 \cdot 10^{-8}$	1.40	$2.89 \cdot 10^{-6}$	$2.39 \cdot 10^{-6}$
2^{18}	16.10×	$2.22 \cdot 10^{-8}$	1.41	$2.91 \cdot 10^{-6}$	$2.24 \cdot 10^{-6}$

to update $\boldsymbol{A}$, either by adding or removing a column and/or a row. The null space of the updated matrix is then sought.

Here, we devise a strategy in a similar spirit to the ones used for data streaming [CW09, TYUC19] to efficiently update the sketch $\boldsymbol{S}^\top\boldsymbol{A}$ rather than sketching $\boldsymbol{A}$ from scratch, so that the solution for Problem 3.0.1 can be updated efficiently. The strategies which will be discussed below are feasible because, as long as the updates or downdates to the original matrix are independent of the random embedding, the embedding will continue to be a valid subspace embedding for the updated or downdated matrix with high probability. In particular, if Gaussian embeddings are used, the probability of failure remains bounded by a sum of exponentially small terms, by a union bound argument.

First, adding or removing a column is straightforward. If we add or remove a column from the original matrix then we can do the same for the sketch.

By contrast, adding or removing a row is not so simple. For simplicity, we focus on row updates and downdates using Gaussian embeddings $\boldsymbol{S}_G \in \mathbb{R}^{m\times s}$. We first observe that regardless of how many row updates and downdates are being done to the original matrix, the

sketch $\boldsymbol{S}_G^\top \boldsymbol{A}$ will always be an $s \times n$ matrix. One caution is that if too many row downdates are done to $\boldsymbol{A}$ so that $m \approx s$ then sketching becomes pointless.

3.4.1 Row updating

Let us consider a row update first. Without loss of generality, suppose that a row $\boldsymbol{a} \in \mathbb{R}^{1\times n}$ is added to the bottom of $\boldsymbol{A}$ and let $\bar{\boldsymbol{A}} := \begin{bmatrix} \boldsymbol{A} \\ \boldsymbol{a} \end{bmatrix} \in \mathbb{R}^{(m+1)\times n}$ be the updated matrix. If we did not have the sketch of $\boldsymbol{A}$ and if we had to sketch from scratch then we need to draw a new Gaussian embedding $\tilde{\boldsymbol{S}}_G \in \mathbb{R}^{(m+1)\times s}$ and left-multiply it to $\bar{\boldsymbol{A}}$. Now we find an equivalent process by reusing the sketch $\boldsymbol{Y} := \boldsymbol{S}_G^\top \boldsymbol{A} \in \mathbb{R}^{s\times n}$. Since a row is appended to the end of $\boldsymbol{A}$, we add a row to the end of $\boldsymbol{S}_G$ giving us $\bar{\boldsymbol{S}}_G = \begin{bmatrix} \boldsymbol{S}_G \\ \boldsymbol{g} \end{bmatrix} \in \mathbb{R}^{(m+1)\times s}$ where $\boldsymbol{g} \in \mathbb{R}^{1\times s}$ is a Gaussian row vector independent of $\boldsymbol{S}_G$ and $\boldsymbol{A}$ with entries i.i.d. $\mathcal{N}(0, 1/s)$. Notice that $\bar{\boldsymbol{S}}_G$ and $\tilde{\boldsymbol{S}}_G$ are equal in distribution. Therefore, the updated sketch for $\bar{\boldsymbol{A}}$ becomes

$$\bar{\boldsymbol{Y}} = \bar{\boldsymbol{S}}_G^\top \bar{\boldsymbol{A}} = \begin{bmatrix} \boldsymbol{S}_G \\ \boldsymbol{g} \end{bmatrix}^\top \begin{bmatrix} \boldsymbol{A} \\ \boldsymbol{a} \end{bmatrix} = \boldsymbol{S}_G^\top \boldsymbol{A} + \boldsymbol{g}^\top \boldsymbol{a} = \boldsymbol{Y} + \boldsymbol{g}^\top \boldsymbol{a} \in \mathbb{R}^{s\times n}$$

with the updated Gaussian embedding $\bar{\boldsymbol{S}}_G = \begin{bmatrix} \boldsymbol{S}_G \\ \boldsymbol{g} \end{bmatrix}$. If a row is inserted into the middle of the matrix, say at the jth position, then the formula for the sketch is the same as above with the Gaussian row vector inserted at the jth position. Algorithm 2 shows this.

3.4.2 Row downdating

Row downdating is similar to a row update. Let $\boldsymbol{A}_j \in \mathbb{R}^{(m-1)\times n}$ be the matrix with the jth row removed from $\boldsymbol{A}$. Using a similar idea as a row update, we let $\boldsymbol{S}_j \in \mathbb{R}^{(m-1)\times s}$ be the

Algorithm 2 Sketch update: A row update on $\boldsymbol{A}$

Input: $\boldsymbol{S} \in \mathbb{R}^{m\times s}$ subspace embedding with sketch size s, $\boldsymbol{a} \in \mathbb{R}^{1\times n}$ the row being added, j the index where $\boldsymbol{a}$ is being added, $\boldsymbol{Y} \in \mathbb{R}^{s\times n}$ the sketch of $\boldsymbol{A}$

Output: $\bar{\boldsymbol{S}} \in \mathbb{R}^{(m+1)\times s}$ updated subspace embedding, $\bar{\boldsymbol{Y}} \in \mathbb{R}^{s\times n}$ updated sketch

`function` $[\bar{\boldsymbol{S}}, \bar{\boldsymbol{Y}}] =$ `Row_Update`$(\boldsymbol{S}, \boldsymbol{Y}, \boldsymbol{a}, j)$

1: Draw $\boldsymbol{g} \in \mathbb{R}^{1\times s}$ with i.i.d. $\mathcal{N}(0, 1/s)$ independent of $\boldsymbol{S}$,

2: Set $\bar{\boldsymbol{S}} = \begin{bmatrix} \boldsymbol{S}(1:j-1,\ :) \\ \boldsymbol{g} \\ \boldsymbol{S}(j:s,\ :) \end{bmatrix} \in \mathbb{R}^{(m+1)\times s}$, ▷ Updated subspace embedding

3: Set $\bar{\boldsymbol{Y}} = \boldsymbol{Y} + \boldsymbol{g}^\top \boldsymbol{a}$ ▷ Updated sketch

matrix with the jth row removed from $\boldsymbol{S}$. Then the updated sketch for $\boldsymbol{A}_j$ becomes

$$\boldsymbol{Y}_j := \boldsymbol{S}_j \boldsymbol{A}_j = \begin{bmatrix} \boldsymbol{S}_G(1:j-1,\ :) \\ \boldsymbol{S}_G(j+1:m,\ :) \end{bmatrix}^\top \begin{bmatrix} \boldsymbol{A}(1:j-1,\ :) \\ \boldsymbol{A}(j+1:m,\ :) \end{bmatrix} = \boldsymbol{Y} - \boldsymbol{S}_G(j,\ :)^\top \boldsymbol{A}(j,\ :).$$

where $\boldsymbol{Y} = \boldsymbol{S}_G^\top \boldsymbol{A}$ is the sketch for $\boldsymbol{A}$. Thus, the updated sketching matrix becomes $\boldsymbol{S}_j$ and the updated sketch becomes $\boldsymbol{Y}_j = \boldsymbol{Y} - \boldsymbol{S}_G(j,:)^\top \boldsymbol{A}(j,:)$. Algorithm 3 shows this.

Algorithm 3 Sketch update: A row downdate on $\boldsymbol{A}$

Input: $\boldsymbol{S} \in \mathbb{R}^{m\times s}$ subspace embedding with sketch size s, j the index to be removed, $\boldsymbol{Y} \in \mathbb{R}^{s\times n}$ the sketch of $\boldsymbol{A}$

Output: $\boldsymbol{S}_j \in \mathbb{R}^{(m-1)\times s}$ downdated subspace embedding, $\boldsymbol{Y}_j \in \mathbb{R}^{s\times n}$ downdated sketch

`function` $[\boldsymbol{S}_j, \boldsymbol{Y}_j] =$ `Row_Downdate`$(\boldsymbol{S}, \boldsymbol{Y}, j)$

1: Set $\boldsymbol{S}_j = \begin{bmatrix} \boldsymbol{S}(1:j-1,\ :) \\ \boldsymbol{S}(j+1:m,\ :) \end{bmatrix}$, ▷ Downdated subspace embedding

2: Set $\boldsymbol{Y}_j = \boldsymbol{Y} - \boldsymbol{S}(j,:)^\top \boldsymbol{A}(j,:)$ ▷ Downdated sketch

3.4.3 Non-Gaussian sketch

We have considered Gaussian embeddings for row updates and downdates. For other random embeddings such as SRTTs, the analysis is more difficult. However, in practice many random embeddings behave similarly to the Gaussian embedding and often Gaussian analysis reflects

the performance in practice [MT20]. For column updates and downdates, the strategy is the same for all random embeddings, but not necessarily for rows. The strategy for row updates and downdates can be the same for certain classes of random embeddings such as those with i.i.d. rows, with sparse sign embeddings being an example. The strategy we suggest for other sketching matrices for row updates and downdates is the following.

For a row update, we append a Gaussian row vector (or a row of a sparse sign embedding for efficiency) to the subspace embedding at the index where the new row gets appended to $\boldsymbol{A}$, and set the result to be the updated subspace embedding. The updated sketch will then be $\boldsymbol{Y} + \boldsymbol{g}^\top \boldsymbol{a}$ where $\boldsymbol{Y} \in \mathbb{R}^{s \times n}$ is the previous sketch, $\boldsymbol{g} \in \mathbb{R}^{1 \times s}$ is a Gaussian row vector independent of $\boldsymbol{S}$ with entries i.i.d. $\mathcal{N}(0, 1/s)$, and $\boldsymbol{a} \in \mathbb{R}^{1 \times n}$ is the row appended to $\boldsymbol{A}$.

For a row downdate, we would need to remove the row from the subspace embedding corresponding to the row that will be removed from $\boldsymbol{A}$. However, removing this row is essentially the same as zeroing out its corresponding row in $\boldsymbol{A}$ and keeping the original subspace embedding. More specifically, we have the following equivalence for removing the jth row of $\boldsymbol{A}$,

$$\begin{aligned}
\begin{bmatrix} \boldsymbol{S}(1:j-1,\ :) \\ \boldsymbol{S}(j+1:m,\ :) \end{bmatrix}^\top \begin{bmatrix} \boldsymbol{A}(1:j-1,\ :) \\ \boldsymbol{A}(j+1:m,\ :) \end{bmatrix} &= \boldsymbol{S}^\top \begin{bmatrix} \boldsymbol{A}(1:j-1,\ :) \\ \boldsymbol{0} \\ \boldsymbol{A}(j+1:m,\ :) \end{bmatrix} \\
&= \boldsymbol{S}^\top \boldsymbol{A} - \boldsymbol{S}(j,:)^\top \boldsymbol{A}(j,:) \\
&= \boldsymbol{S}^\top \boldsymbol{A} - \boldsymbol{S}^\top \boldsymbol{e}_j \boldsymbol{A}(j,:),
\end{aligned}$$

where $\boldsymbol{e}_j$ is the jth canonical basis vector. Following this observation, we propose the following for row downdates. We first take the canonical basis vector corresponding to the index of the row that will be removed from $\boldsymbol{A}$, say $\boldsymbol{e}_j$, and sketch it using the original subspace embedding $\boldsymbol{S}$ giving $\boldsymbol{S}^\top \boldsymbol{e}_j \in \mathbb{R}^s$. We then subtract $(\boldsymbol{S}^\top \boldsymbol{e}_j)\boldsymbol{A}(j,:) \in \mathbb{R}^{s \times n}$ from the original sketch to obtain the updated sketch, $\boldsymbol{Y}_j = \boldsymbol{Y} - (\boldsymbol{S}^\top \boldsymbol{e}_j)\boldsymbol{A}(j,:)$. This process removes the

contribution from the removed row in the original sketch.

3.4.4 Complexity of updating the sketch

We discuss the complexity of reusing the sketch. We assume that the random embedding is an SRTT with the sketch size $s = 2n$ and we are sketching $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ ($m \gg n$). For a column downdate, it is essentially free because we only need to remove a column from the sketch. For a column update, we need to sketch a column which costs $\mathcal{O}(m \log m)$ flops. For a row update we need to do a column-row multiplication followed by an addition of two $s \times n$ matrices which cost an overall $\mathcal{O}(sn)$ flops. For a row downdate, we need to sketch a canonical basis vector which costs $\mathcal{O}(m \log m)$ flops, and perform a column-row multiplication followed by a subtraction of two $s \times n$ matrices which cost $\mathcal{O}(sn)$ flops. Overall, a row downdate costs $\mathcal{O}(m \log m + sn)$ flops. This is all better than resketching, which costs $\mathcal{O}(mn \log n)$ flops. Therefore, whenever an update or a downdate is made to the original matrix, updating the sketch is about $\mathcal{O}(n)$ times better than resketching the updated/downdated matrix from scratch.

We now use the ideas in this section to speed up the AAA algorithm for rational approximation by reusing the sketch in the sketch-and-solve method.

3.5 Application: AAA algorithm

The goal of rational approximation is: given a (possibly complicated) function $\mathtt{f} : \mathbb{R} \to \mathbb{R}$, find a rational function $\mathtt{r} : \mathbb{R} \to \mathbb{R}$ that approximates $\mathtt{f} \approx \mathtt{r}$, in a domain $\Omega \subseteq \mathbb{R}$. The AAA algorithm [NST18] is a powerful algorithm for this task, requiring only a set of distinct sample points, $z_1, z_2, ..., z_m \in \Omega$, and their function values, $f_1, f_2, ..., f_m$, that is, $f_i = \mathtt{f}(z_i)$. The flexibility and empirical near-optimality of AAA have resulted in its use in a large number of applications, including nonlinear eigenvalue problems [LMPV22], conformal maps [GT19], model order reduction [GG20], and signal processing [DPP23]. Rational approximation can

significantly outperform the more standard polynomial approximation when, for example, Ω is unbounded or when $\mathtt{f}$ has singularities [Tre19].

3.5.1 A brief summary of the AAA algorithm

Let us outline the AAA algorithm, emphasizing how it results in the approximate null space problem (Problem 3.0.1). The AAA algorithm has two key ingredients. The first is to use the barycentric representation of rational functions, instead of the standard quotient of polynomials:

$$\mathtt{r}(z) = \mathtt{n}(z)/\mathtt{d}(z) = \sum_{j=1}^{n} \frac{f_{i_j} w_j}{z - z_{i_j}} \Bigg/ \sum_{j=1}^{n} \frac{w_j}{z - z_{i_j}}$$

where n is the degree/type of rational approximation, $\boldsymbol{w} = [w_1, w_2, ..., w_n]^\top \in \mathbb{R}^n$ are weights, and $i_1, i_2, ..., i_n \in \{1, 2, ..., m\}$ are distinct indices. Here, m is the number of sample points and usually, we have $m \gg n$. The second key ingredient is the greedy selection of the so-called *support points* z_{i_j}, which are chosen as the points where the current error $|\mathtt{f}(z_i) - \mathtt{r}(z_i)|$ is maximised. This greedy selection is done by finding the minimiser of the linearised least-squares error $\|\mathtt{fd} - \mathtt{n}\|_2$, where the norm is the ℓ_2 norm on the sample points. This is equivalent to forming a matrix called the Loewner matrix [NST18], whose (i, j) entry is $\frac{\mathtt{f}(z_i) - \mathtt{f}(z_j)}{z_i - z_j}$, and finding the minimiser of $\|\boldsymbol{A}\boldsymbol{w}\|_2 = \|\mathtt{fd} - \mathtt{n}\|_2$ over unit-norm vectors $\boldsymbol{w}$, i.e., an approximate null space problem with $k = 1$. More specifically, as the algorithm iterates, a column is added (the degree of $\mathtt{r}$ is increased) while a row is removed (a support point is added, hence removed from the least-squares equation) from the Loewner matrix from the previous iteration, making the Loewner matrix $\boldsymbol{A}^{(\ell)}$ an $(m - \ell) \times \ell$ matrix at iteration ℓ. The iteration is terminated once the tolerance is met and the resulting rational function interpolates (apart from removable singularities) $f(x)$ at the support points. The precise details are covered in the original paper [NST18].

To summarize, the main computational task in AAA is that at each iteration, say ℓ, we solve for the weights $\boldsymbol{w} \in \mathbb{R}^\ell$ that solve the following approximate null space problem

$$\min_{\|\boldsymbol{w}\|_2=1} \left\| \boldsymbol{A}^{(\ell)} \boldsymbol{w} \right\|_2 .$$

The original algorithm computes the SVD of an $(m-\ell) \times \ell$ matrix at iteration ℓ, giving us the overall complexity of $O(mn^3)$ for the AAA algorithm where n is the degree/type of the rational approximant `r`.

3.5.2 Speeding up AAA by reusing the sketch

Given that the approximate null space problem is the main computational task in AAA, it is natural to apply the algorithms in this paper to speed it up. The first straightforward idea is as follows: At the ℓth iteration, when we solve for the right singular vector corresponding to the smallest singular value of $\boldsymbol{A}^{(\ell)}$, we can sketch the matrix $\boldsymbol{A}^{(\ell)}$ and then take the SVD of a smaller-sized matrix instead. This will reduce the complexity from $\mathcal{O}(mn^3)$ to $\mathcal{O}(mn^2 \log n + n^4)$ when $m \gg n$. Note that by resketching the matrix at each iteration using an SRTT, we achieve a theoretical flop count of $\mathcal{O}(m\ell \log \ell + \ell^3)$ at iteration ℓ. This limits the speedup we can obtain as n is usually not too large ($\lesssim 200$) in most applications [NST18].

Fortunately, we can improve the speed further by noting that at each iteration of the AAA algorithm a column is added while a row is deleted from the Loewner matrix. Therefore we can update the sketch using the strategies discussed in Section 3.4. For deleting a row, we can use Algorithm 3 and for adding a column, we can sketch the new column and append it to the sketch. The overall sketch then becomes

$$\boldsymbol{Y}^{(\ell)} = \left[\boldsymbol{Y}^{(\ell-1)} - (\boldsymbol{S}^\top \boldsymbol{e}_j) \boldsymbol{A}^{(\ell-1)}(j,:), \;\; \boldsymbol{S}^\top \boldsymbol{A}_\ell \right] \in \mathbb{R}^{s \times \ell}$$

at iteration ℓ, where $\boldsymbol{Y}^{(\ell-1)}$ is the sketch of $\boldsymbol{A}^{(\ell-1)}$, $\boldsymbol{S}$ is the original subspace embedding, j is the index of the row of $\boldsymbol{A}^{(\ell-1)}$ being removed and $\boldsymbol{A}_\ell$ is the added column. After the sketch is made, we take the SVD of the sketch $\boldsymbol{Y}^{(\ell)}$ and extract its trailing singular vector. The overall complexity is $\mathcal{O}(mn \log m + n^4)$ using the SRTT with the sketch size $s = 2n$. Thus, when $m \gg n$, and m is at most exponentially larger than n, we get a lower complexity than both the original AAA algorithm with $O(mn^3)$ flops, and the version where we resketch the entire Loewner matrix $A^{(\ell)}$ at each iteration, requiring $O(mn^2 \log n + n^4)$ flops.

Note that at each iteration, a column update and a row downdate to the Loewner matrix is the result of choosing a support point. Since we choose our support points based on trailing right singular vector of our sketch, there is weak dependency between our sketch and the support point. However, since we begin with a random embedding independent to the initial Loewner matrix, the failure probability for the column update and the row downdate concerning any independently chosen support point is exponentially small using the failure probability for the Gaussian embedding (Section 2.2.1). Therefore, regardless of the dependency between our sketch and the chosen support point, the sketch would succeed, in the worst case with all but a sum of exponentially small failure probabilities by union bound. For n not too large, we expect the original random embedding to be a subspace embedding for future Loewner matrices, which is observed in practice; see Figure 3.4.

Other approaches for speeding up AAA

In [Hoc17], Hochman designs an algorithm to speed up the AAA algorithm based on Cholesky update/downdate of the Gram matrix of $\boldsymbol{A}^{(\ell)}$. In the AAA algorithm, the Loewner matrix $\boldsymbol{A}^{(\ell)}$ can become extremely ill-conditioned so Cholesky update/downdate can be numerically unstable [EP94]. Also, the complexity of Hochman's algorithm is $\mathcal{O}(mn^2)$ and our algorithm, with complexity $\mathcal{O}(mn \log m + n^4)$, is faster as long as m is larger than n^2.

3.5.3 AAA Experiments

In Table 3.2, we conducted experiments with various functions using $m = 10^6$ points sampled uniformly from the domain of each function. These functions were chosen so that the functions give different values of n in the rational approximation. This will demonstrate how the sketched version of the AAA algorithm performs in the high-dimensional case when compared with the original AAA algorithm. By reusing the sketch we already see a great speedup even for small values of n. The function $f(z) = \log(2 + z^4)/(1 - 16z^4)$ is estimated by a rational function with $n = 32$ and we achieved more than $10\times$ speedup. For a larger value of n, in the case $f(z) = \tan(256z)$ with $n = 190$, we get more than $30\times$ speedup. This experiment shows the efficiency of reusing the sketch for the sketch-and-solve method.[2]

Table 3.2: AAA speedup for four functions that give different values of n with $m = 10^6$ points. The points were sampled uniformly at random from the domain of each function.

$f(z)$	n	Speedup	Domain
$\log(2 + z^4)/(1 - 16z^4)$	32	$10.49\times$	$\{z : \lvert z\rvert = 1\}$
$\sqrt{z(1-z)}\sqrt{(z-i)(1+i-z)}$	60	$14.00\times$	$\{z = x + iy : x, y \in [0, 1]\}$
$\tan(128z)$	105	$19.40\times$	$\{z : \lvert z\rvert \leq 1\}$
$\tan(256z)$	190	$32.69\times$	$\{z : \lvert z\rvert \leq 1\}$

Figure 3.4 illustrates the qualitative similarity between the rational approximations of $f(z) = \sqrt{z(1-z)}\sqrt{(z-i)(1+i-z)}$ produced by the original AAA algorithm and the version where we have reused the sketch. Both the standard AAA approximant r_1 and the sketched AAA approximant r_2 provide good approximation to the original function f, with r_2 being computed approximately 14 times faster than r_1.

[2]It is often excessive and unnecessary to take a million sample points in AAA. In many cases, say 10^3–10^4 points would suffice. Nonetheless, when the function has singularities on or near the domain of interest, and the precise location of the singularity is unknown, it is sensible to take as many sample points as possible to ensure sufficiently many sample points are near the singularity.

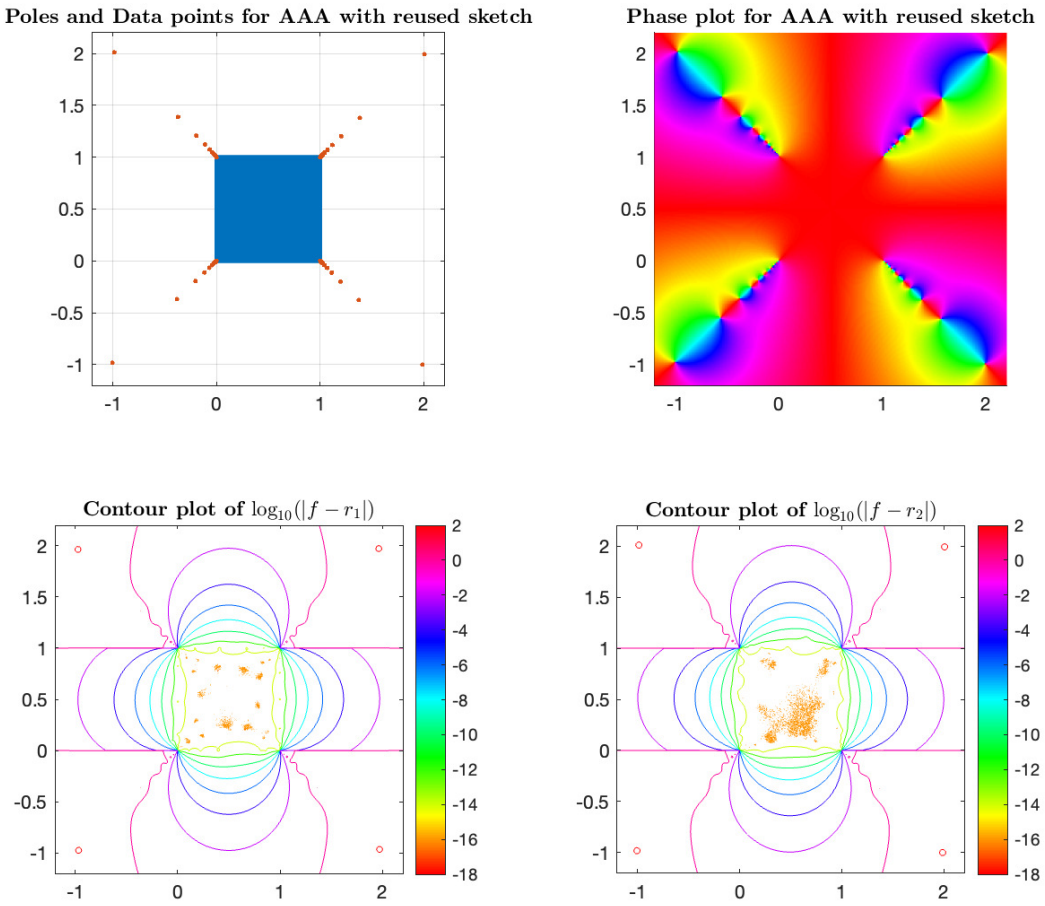


Figure 3.4: $f(z) = \sqrt{z(1-z)}\sqrt{(z-i)(1+i-z)}$ with 10^6 points randomly sampled from $\{z = x + iy : x, y \in [0,1]\}$. r_1 and r_2 are the rational approximation obtained using the original AAA algorithm and the AAA algorithm with reused sketch respectively. The bottom two contour plots show the logarithm (base 10) of the approximation error. We see that the two versions give approximately the same quality of approximation. The top left plot shows the data points (blue) and the poles (red) for r_2 and the top right plot shows the phase plot of r_2.

4

Nyström method for indefinite matrices

The Nyström method [Nys30, WS00, GM16] (Section 2.3.2) has been a popular choice for finding low-rank approximations to symmetric positive semi-definite (SPSD) matrices, especially in the machine learning community for kernel-based methods. However, it can fail when applied to symmetric *indefinite* matrices for which the approximation error can be unboundedly large. Low-rank approximation of symmetric indefinite matrices arises in many applications, such as learning in reproducing kernel Kreĭn spaces [OG19], natural language processing [PR14, DCLT19], and non-metric proximity transformations [GS15], which has applications in bioinformatics and social networks.

Let $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ be a SPSD matrix and k be the target rank. Then the Nyström

approximation is given by

$$\boldsymbol{A}_{nys}^{(s)} = \boldsymbol{C}\boldsymbol{M}^{\dagger}\boldsymbol{C}^{\top},$$

where $\boldsymbol{C} = \boldsymbol{A}\boldsymbol{S} \in \mathbb{R}^{n\times s}$, and $\boldsymbol{M} = \boldsymbol{S}^{\top}\boldsymbol{A}\boldsymbol{S} \in \mathbb{R}^{s\times s}$ with $k \leq s < n$ and $\boldsymbol{S} \in \mathbb{R}^{n\times s}$ is a subspace embedding with sketch size s. Traditionally, $\boldsymbol{S}$ is chosen to be a column sampling matrix, which make $\boldsymbol{C}$ a subset of s columns of $\boldsymbol{A}$ and $\boldsymbol{M}$ an $s \times s$ principal submatrix of $\boldsymbol{A}$; see Section 2.4 and Chapter 5. While generally cheaper to compute, column sampling matrices typically have higher variance result than random embeddings [MT20, FTU23]. As a result, in view of obtaining a stable, robust Nyström method for symmetric indefinite matrices, we focus on random embeddings in this chapter.

The goal of this chapter is to demonstrate that a judiciously constructed rank-restricted variant of the Nyström approximation (Algorithm 4), when used with random embeddings, remains robust even for symmetric indefinite matrices. This is demonstrated empirically through numerical experiments in Section 4.5. Additionally, we show that there exists an orthogonal projection matrix for the core matrix $\boldsymbol{M}$ such that the Nyström approximation gives a good low-rank approximation to *any* symmetric matrix, provided its singular values decay sufficiently rapidly. While this analysis does not directly establish the accuracy of our proposed algorithm, it does support a closely related variant of the Nyström method.

It is important to note that the original matrix $\boldsymbol{A}$ need not be SPSD for one to form the Nyström approximation. However, existing theory does not translate directly to symmetric indefinite matrices, as it relies on the assumption that $\boldsymbol{A}$ is SPSD [Git11, GM16, MW17].

4.1 Rank-restricted variants of the Nyström method

There are several variants of the Nyström method for SPSD matrices. There are two rank-restricted versions that give a rank-k approximation to $\boldsymbol{A}_{nys}^{(s)}$ where $k < s$. The first version, which is more traditional, is defined by $\boldsymbol{A}_{nys}^{(s,k)} = \boldsymbol{C}\,[\![\boldsymbol{M}]\!]_k^{\dagger}\,\boldsymbol{C}^{\top}$ [DM05, LBKL15, GM16] where $[\![\boldsymbol{M}]\!]_k$ denotes the best rank-k approximation to $\boldsymbol{M}$. The second version is given by $\left[\!\left[\boldsymbol{A}_{nys}^{(s)}\right]\!\right]_k = [\![\boldsymbol{C}\boldsymbol{M}^{\dagger}\boldsymbol{C}^{\top}]\!]_k$ [TYUC17a, PABW18, WGM19], which was suggested more recently. The difference between the two methods is that $\boldsymbol{A}_{nys}^{(s,k)}$ performs rank-truncation in the core matrix, $\boldsymbol{M}$, which makes this method cheaper to compute, while $\left[\!\left[\boldsymbol{A}_{nys}^{(s)}\right]\!\right]_k$ performs rank-truncation in the entire Nyström approximation $\boldsymbol{A}_{nys}^{(s)}$. We mostly focus on $\boldsymbol{A}_{nys}^{(s,k)}$ in this chapter.

For an SPSD matrix, $\boldsymbol{A}$, $\boldsymbol{A}_{nys}^{(s)}$ (Theorem 2.3.4) and $\left[\!\left[\boldsymbol{A}_{nys}^{(s)}\right]\!\right]_k$ [WGM19] satisfy relative-error bounds in the nuclear norm. Contrarily, it is not known whether $\boldsymbol{A}_{nys}^{(s,k)}$ satisfies a relative-error norm bound [WGM19]. In [PABW18], an example of a 3×3 SPSD matrix is given, showing the downside of using $\boldsymbol{A}_{nys}^{(s,k)}$ for kernel approximations, which commonly uses a column sampling matrix. The authors propose $\left[\!\left[\boldsymbol{A}_{nys}^{(s)}\right]\!\right]_k$ as an alternative, for which later Wang, Gittens and Mahoney derived a relative-error norm bound [WGM19]. For this 3×3 example, the problem persists even if we use random embeddings. However, this is a small example that can yield high variance results, and random embeddings do give a smaller expected relative-error in the nuclear norm and a smaller variance result than column sampling. These types of phenomena have been discussed before, for example in [MT20], where the authors point out that column sampling is less reliable than random embeddings due to their relatively high variance results.

For symmetric indefinite matrices, not much has been shown. It is however known that the problem is rather difficult. We can easily see that the plain Nyström approximation, $\boldsymbol{A}_{nys}^{(s)}$

can behave poorly for symmetric indefinite matrices. For example, let $0 < \epsilon < 1$ and

$$\boldsymbol{A} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \boldsymbol{S} = \begin{bmatrix} \epsilon \\ \sqrt{1-\epsilon^2} \end{bmatrix}$$

where $\boldsymbol{A}$ has eigenvalues ± 1. Then the plain rank-1 Nyström approximation to $\boldsymbol{A}$ is

$$\boldsymbol{A}_{nys}^{(1)} = \boldsymbol{A}\boldsymbol{S}(\boldsymbol{S}^\top \boldsymbol{A}\boldsymbol{S})^\dagger \boldsymbol{S}^\top \boldsymbol{A} = \frac{1}{2\epsilon\sqrt{1-\epsilon^2}} \begin{bmatrix} 1-\epsilon^2 & \epsilon\sqrt{1-\epsilon^2} \\ \epsilon\sqrt{1-\epsilon^2} & \epsilon^2 \end{bmatrix}$$

and therefore

$$\left\| \boldsymbol{A} - \boldsymbol{A}_{nys}^{(1)} \right\|_* = \frac{1}{2\epsilon\sqrt{1-\epsilon^2}},$$

which can become arbitrarily large as $\epsilon \to 0$, whereas the best rank-1 nuclear norm error of $\boldsymbol{A}$ is 1. This kind of issue does not arise for SPSD matrices. A similar phenomenon has been observed in a different context involving CUR approximations of rectangular matrices [CK20], which we discuss in greater detail in Chapter 5. Essentially, the issue arises from the presence of an eigenvalue of $\boldsymbol{M}$ close to (or even equal to) 0, much smaller than $\sigma_k(\boldsymbol{A})$ or even $\sigma_{\min}(\boldsymbol{A})$—a phenomenon that is absent when $\boldsymbol{A}$ is SPSD. This leads to a blow-up in the norm of the pseudoinverse of the core matrix, $\boldsymbol{M}^\dagger$, causing instability. While this is admittedly a contrived example, the difficulty can also be easily observed in experiments. In Figure 4.1, the two plots were generated using 100×100 symmetric indefinite matrices with Haar distributed eigenvectors. In the left plot, the eigenvalues decay geometrically from 1 to 10^{-8} with random signs, and in the right plot, the first 20 eigenvalues are equal to ± 1 and the other 80 eigenvalues are equal to $\pm 10^{-10}$, where the signs were applied randomly with equal probability. We apply the plain Nyström approximation $\boldsymbol{A}_{nys}^{(k)}$ using the Gaussian embedding to $\boldsymbol{A}$. The peaks of $\boldsymbol{A}_{nys}^{(k)}$ in Figure 4.1 indicate that the plain Nyström approximation can be unstable.

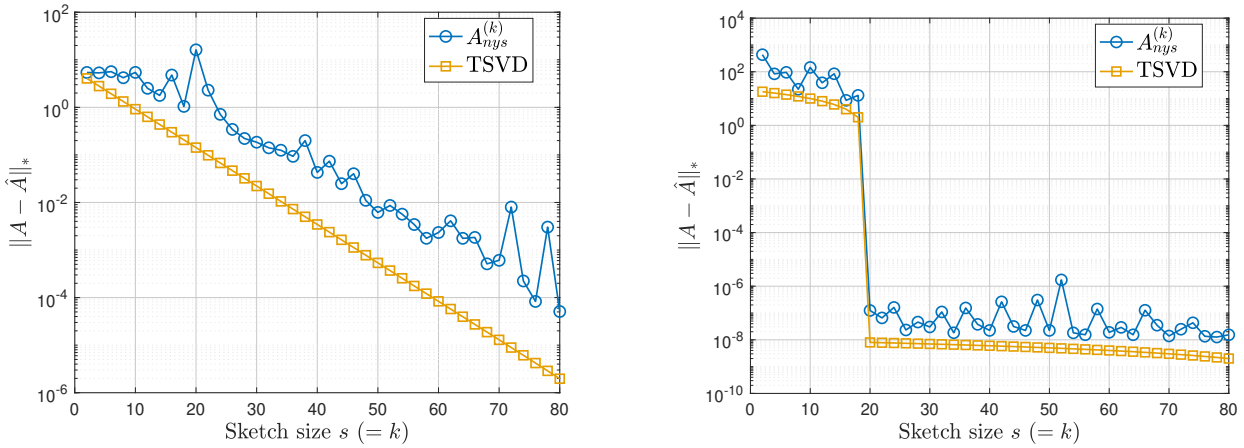


Figure 4.1: Plain Nyström approximation $\boldsymbol{A}_{nys}^{(k)}$ using the Gaussian embedding to 100×100 symmetric indefinite matrices. We can see that $\boldsymbol{A}_{nys}^{(k)}$ can be unstable.

4.2 Existing methods

There are several existing ideas for using the Nyström method for symmetric indefinite matrices. Cai, Nagy and Xi [CNX22] derive an error bound for the Nyström method $\boldsymbol{A}_{nys}^{(s)}$ for symmetric indefinite matrices that arise from a symmetric function. This bound depends on how close the function values of the sampled points are, which is not an attractive dependence and may not be very useful in more general or practical situations. They suggest the plain Nyström method $\boldsymbol{A}_{nys}^{(k)}$, which can be unstable as illustrated in Figure 4.1. They also suggest $\boldsymbol{AS}(\boldsymbol{S}^\top \boldsymbol{AS})_\epsilon^\dagger \boldsymbol{S}^\top \boldsymbol{A}$ for the Nyström approximation motivated by [Nak20] with the aim of improving the stability. This version truncates the core matrix $\boldsymbol{M} = \boldsymbol{S}^\top \boldsymbol{AS}$ so that $\sigma_{\min}\left((\boldsymbol{S}^\top \boldsymbol{AS})_\epsilon\right) > \epsilon$ where ϵ is on the order of the unit roundoff. However, this version can give worse approximations than $\boldsymbol{A}_{nys}^{(s)}$ as illustrated in their paper [CNX22] and does not always improve the stability of the Nyström approximation. In another work, Ray *et al.* [RMMM22] suggest submatrix-shifted (SMS) Nyström to provide an efficient algorithm that deals with symmetric matrices that have only few negative eigenvalues. This method uses an eigenvalue shift based on the minimum eigenvalue of a small principal submatrix before applying the plain Nyström method $\boldsymbol{A}_{nys}^{(k)}$. A major drawback of this approach is that the eigenvalue shift

can adversely affect the quality of the low-rank approximation by potentially destroying the inherent decay structure of the singular values. Lastly, the authors in [GS15, OG19] devise several strategies to form Nyström approximations to symmetric indefinite matrices. However, these methods use eigenvalue information of the original matrix, which is expensive to compute. The three existing methods described above use column sampling matrices for $\boldsymbol{S}$, which is different from our approach which focuses on random embeddings.

Non-Nyström approaches. In [HMT11], a low-rank approximation for symmetric matrices in the form of the randomised SVD is given. This approximation is given by $\boldsymbol{Q}\boldsymbol{Q}^\top\boldsymbol{A}\boldsymbol{Q}\boldsymbol{Q}^\top$ where $\boldsymbol{Q} \in \mathbb{R}^{n\times s}$ is the orthonormal matrix in the thin QR decomposition of $\boldsymbol{A}\boldsymbol{S}$ and is known to satisfy a relative-error norm bound. The dominant cost is $\mathcal{O}(n^2 s)$ flops for forming $\boldsymbol{Q}^\top\boldsymbol{A}$ (assuming $\boldsymbol{A}$ is dense), which becomes prohibitive when n, s are large. Wang, Luo and Zhang derived in [WLZ14] a relative-error norm bound to any symmetric matrices (possibly indefinite) for the prototype model. This model computes the low-rank approximation by first forming the sketch $\boldsymbol{C} = \boldsymbol{A}\boldsymbol{S}$ and then approximating $\boldsymbol{A}$ by $\boldsymbol{C}\boldsymbol{C}^\dagger\boldsymbol{A}(\boldsymbol{C}^\dagger)^\top\boldsymbol{C}^\top$. The dominant costs are $\mathcal{O}(n^2 k \log k)$ for computing $\boldsymbol{C}$ and $\mathcal{O}(n^2 k)$ for computing $\boldsymbol{C}^\dagger\boldsymbol{A}$, which becomes very costly with large n. Both of these methods are two-pass algorithms compared with the Nyström method which is a one-pass algorithm. Two-pass algorithms, generally require more storage and computational complexity than one pass algorithms.

Non-symmetric approaches. Non-symmetric low-rank approximation can be used for symmetric indefinite matrices. Examples include the randomised SVD [HMT11] and the generalised Nyström method [TYUC17b, Nak20]; see Section 2.3. For both methods, since their representation is not symmetric, if we want to force symmetry in their representations (e.g. by taking the symmetric part $(\hat{\boldsymbol{A}}^\top + \hat{\boldsymbol{A}})/2$), we may risk doubling the rank in the approximation and recompression is needed to truncate the symmetric part back to the target

rank. In addition, as before, the randomised SVD has the cost of computing $\boldsymbol{Q}^\top \boldsymbol{A}$, which becomes prohibitive when n, s are large. For generalised Nyström, we approximately double the number of matrix-vector multiplications needed as $\boldsymbol{A}$ needs to be multiplied by two independent random embeddings and this, in turn doubles the storage requirement (in fact, more than double because one of the sketch size is recommended to be proportionally larger than the target rank [Nak20]). Therefore, we focus on symmetric low-rank approximations as they are natural for symmetric matrices.

4.3 Proposed method

The Nyström method may fail on symmetric indefinite matrices. The main concern is in the core matrix $\boldsymbol{M} = \boldsymbol{S}^\top \boldsymbol{A}\boldsymbol{S}$, because the positive and negative eigenvalues of $\boldsymbol{A}$ can 'cancel' each other out when forming $\boldsymbol{M}$, making the eigenvalues of $\boldsymbol{M}$ much smaller than $\sigma_k(\boldsymbol{A})$. This causes inaccuracies and instabilities when computing the pseudoinverse of $\boldsymbol{M}$. For the sake of argument, suppose $\boldsymbol{S}$ is a column sampling matrix then $\boldsymbol{M}$ would be a principal submatrix of $\boldsymbol{A}$. By Cauchy's interlacing theorem, the spectrum of $\boldsymbol{M}$ is contained in the interval $[\lambda_{\min}(\boldsymbol{A}), \lambda_{\max}(\boldsymbol{A})]$, which contains both positive and negative values since $\boldsymbol{A}$ is indefinite. Therefore the magnitude of the eigenvalues of $\boldsymbol{M}$ can be significantly smaller from those of $\boldsymbol{A}$, potentially resulting in the matrix $\boldsymbol{M}^\dagger$ blowing up. In addition, the computation of the pseudoinverse of $\boldsymbol{M}$ can be numerically unstable if $\sigma_{\min}(\boldsymbol{M}) < u$ where u is the unit roundoff. Thus, the main challenge is to ensure that $\boldsymbol{M}^\dagger$ does not ruin the Nyström approximation quality. One approach is to introduce a potentially large shift to make $\boldsymbol{A}$ SPSD, but this can severely affect the approximation quality unless $\boldsymbol{A}$ is nearly definite, that is, the negative eigenvalues of $\boldsymbol{A}$ are very small in magnitude, for example, on the order of the unit roundoff. This idea is used for SPSD matrices where a small shift is introduced to gain numerical stability, however the shift here needs to be small enough

to ensure that accuracy is still high [LLS+17, TYUC17a].

In light of these observations, we propose

$$\boldsymbol{A}_{indef}^{(c,k)} = \boldsymbol{A}\boldsymbol{S}\left[\!\left[\boldsymbol{S}^\top \boldsymbol{A}\boldsymbol{S}\right]\!\right]_k^\dagger \boldsymbol{S}^\top \boldsymbol{A}$$

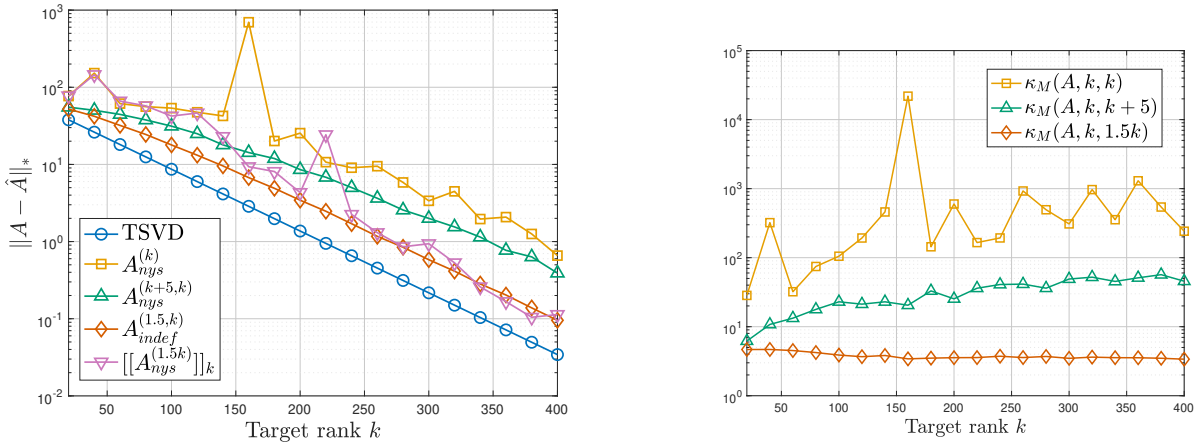
for symmetric *indefinite* matrices $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ where $\boldsymbol{S} \in \mathbb{R}^{n\times ck}$ is a random embedding, $c > 1$ is a modest constant, say $c = 1.5$ or $c = 2$, and k is the target rank. Here, we assume that ck is an integer. When $\boldsymbol{A}$ is SPSD and the sketch size s is proportional to the target rank, $\boldsymbol{A}_{indef}^{(c,k)}$ is equivalent to $\boldsymbol{A}_{nys}^{(ck,k)}$. This rank-restricted version truncates the bottom $(c-1)k$ singular values of $\boldsymbol{M} = \boldsymbol{S}^\top \boldsymbol{A}\boldsymbol{S} \in \mathbb{R}^{ck\times ck}$, which can potentially be harmful even if they are sufficiently larger than the unit roundoff. This is different to the truncation used in [CNX22] as they use truncation based on the magnitudes of the singular values of $\boldsymbol{M}$, whereas for our method, the number of smallest singular values we truncate is proportional to the target rank. This intuition is justified by Andoni and Nguyễn [AN13], who prove that the largest eigenvalues (those proportional to the sketch size) of symmetric matrices with rapidly decaying singular values are approximately preserved under conjugation by a Gaussian embedding, i.e. $\boldsymbol{S}_G^\top \boldsymbol{A}\boldsymbol{S}_G$, with an appropriate normalisation factor.

To assess the preservation of singular values, we define a metric that quantifies how accurately the core matrix $\boldsymbol{M}$ of the Nyström method captures the singular values of $\boldsymbol{A}$. For a symmetric matrix $\boldsymbol{A} \in \mathbb{R}^{n\times n}$, a target rank k and a sketch size $s \geq k$, define

$$\kappa_{\boldsymbol{M}}(\boldsymbol{A}, k, s) := \frac{\max\limits_{1\leq i\leq k} \sigma_i(\boldsymbol{S}_G^\top \boldsymbol{A}\boldsymbol{S}_G)/\sigma_i(\boldsymbol{A})}{\min\limits_{1\leq j\leq k} \sigma_j(\boldsymbol{S}_G^\top \boldsymbol{A}\boldsymbol{S}_G)/\sigma_j(\boldsymbol{A})} = \max_{1\leq i,j\leq k} \frac{\sigma_i(\boldsymbol{S}_G^\top \boldsymbol{A}\boldsymbol{S}_G)}{\sigma_j(\boldsymbol{S}_G^\top \boldsymbol{A}\boldsymbol{S}_G)} \frac{\sigma_j(\boldsymbol{A})}{\sigma_i(\boldsymbol{A})}$$

where $\boldsymbol{S}_G \in \mathbb{R}^{n\times s}$ is a Gaussian embedding. This quantity measures the ratio between the worst over-approximation and the worst under-approximation of the leading singular values of $\boldsymbol{A}$ using the singular values in the core matrix $\boldsymbol{M}$. The quantity $\kappa_{\boldsymbol{M}}(\boldsymbol{A}, k, s)$ will help us

see how much the singular values of $\boldsymbol{M}$ have deviated from the leading singular values of $\boldsymbol{A}$, which directly affects the Nyström approximation quality as we illustrate in Figure 4.2.



(a) **(b)**

Figure 4.2: Accuracy of the Nyström approximations $\boldsymbol{A}_{nys}^{(k)}$, $\boldsymbol{A}_{nys}^{(k+5,k)}$, $\boldsymbol{A}_{indef}^{(1.5,k)}$, and $\left[\!\left[\boldsymbol{A}_{nys}^{(1.5k)}\right]\!\right]_k$ to a symmetric indefinite matrix $\boldsymbol{A} \in \mathbb{R}^{1000\times 1000}$. Figure 4.2a shows the Nyström error in the nuclear norm and Figure 4.2b shows the accuracy of the singular values of $\boldsymbol{M} = \boldsymbol{S}_G^\top \boldsymbol{A}\boldsymbol{S}_G$ when compared with the singular values of $\boldsymbol{A}$. We observe that the truncation in the core matrix $\boldsymbol{M}$ can significantly increase the robustness and the accuracy of the Nyström approximation.

In Figure 4.2, we show how important it is to ensure that the spectrum of $\boldsymbol{M}$ does not ruin the approximation quality. In this experiment, $\boldsymbol{A} \in \mathbb{R}^{1000\times 1000}$ is a symmetric indefinite matrix constructed as in the left plot of Figure 4.1. The smallest singular value in the core matrix was larger than 10^{-7} throughout this experiment. For the truncated cases, $\boldsymbol{A}_{indef}^{(1.5,k)}$ and $\boldsymbol{A}_{nys}^{(k+5,k)}$, the approximation is robust as seen in Figure 4.2a. This robustness we see is illustrated in Figure 4.2b where the singular values of $\boldsymbol{M} = \boldsymbol{S}_G^\top \boldsymbol{A}\boldsymbol{S}_G$ behaves well in the sense that there is no wild fluctuations in $\kappa_{\boldsymbol{M}}(\boldsymbol{A}, k, k+5)$ and $\kappa_{\boldsymbol{M}}(\boldsymbol{A}, k, 1.5k)$. However, when the sketch size is not proportional to the target rank ($s = k+5$), the relative approximation error for $\boldsymbol{A}_{nys}^{(k+5,k)}$ (when compared with the truncated SVD) and $\kappa_{\boldsymbol{M}}(\boldsymbol{A}, k, k+5)$ grow as we increase the target rank. This issue can be further exacerbated when structured random embeddings, such as the SRTT embedding, are used with the sketch size $s = k+5$; see Figure 4.3 and Section 2.2. When the sketch size is proportional to the target rank,

$\kappa_{\boldsymbol{M}}(\boldsymbol{A}, k, 1.5k)$ and the relative approximation error for $\boldsymbol{A}_{indef}^{(1.5,k)}$ are approximately a constant, which motivates us to choose the oversample size to be proportional to the target rank. On the other hand, without the truncation in the core matrix we see that $\kappa_{\boldsymbol{M}}(\boldsymbol{A}, k, k)$ behaves wildly. This indicates that the singular values of $\boldsymbol{M}$ inaccurately approximates the leading singular values of $\boldsymbol{A}$. As a result, the Nyström approximations, $\boldsymbol{A}_{nys}^{(k)}$ and $\left[\!\left[\boldsymbol{A}_{nys}^{(1.5k)}\right]\!\right]_k$ yield unstable results. Empirically, this provides a reason to favour $\boldsymbol{A}_{indef}^{(c,k)}$ over other variants of the Nyström method for symmetric indefinite matrices.

4.3.1 Suggested algorithm

For a general symmetric matrix $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ with the target rank k, we suggest

$$\boldsymbol{A}_{indef}^{(c,k)} = \boldsymbol{A}\boldsymbol{S}\left[\!\left[\boldsymbol{S}^\top\boldsymbol{A}\boldsymbol{S}\right]\!\right]_k^\dagger \boldsymbol{S}^\top\boldsymbol{A} = \boldsymbol{C}\left[\!\left[\boldsymbol{M}\right]\!\right]_k^\dagger \boldsymbol{C}^\top, \tag{4.1}$$

where $\boldsymbol{S} \in \mathbb{R}^{n\times s}$ is a random embedding with the sketch size $s = ck$ where $c > 1$ is a modest constant. The algorithm is given in Algorithm 4.

Algorithm 4 Judiciously truncated Nyström approximation for indefinite matrices

Input: Symmetric matrix $\boldsymbol{A} \in \mathbb{R}^{n\times n}$, target rank k, $c > 1$ (rec. $c = 1.5$)
Output: $\bar{\boldsymbol{C}} \in \mathbb{R}^{n\times k}$ and $\boldsymbol{\Lambda}_k \in \mathbb{R}^{k\times k}$ such that $\boldsymbol{A}_{indef}^{(c,k)} = \bar{\boldsymbol{C}}\boldsymbol{\Lambda}_k^\dagger\bar{\boldsymbol{C}}^\top$

`function` $[\bar{\boldsymbol{C}}, \boldsymbol{\Lambda}_k] =$ `Indef_Nystrom`$(\boldsymbol{A}, k, c)$
1: Draw a random embedding $\boldsymbol{S} \in \mathbb{R}^{n\times ck}$,
2: Sketch $\boldsymbol{C} = \boldsymbol{A}\boldsymbol{S}$,
3: $\boldsymbol{M} = \boldsymbol{S}^\top\boldsymbol{C}$,
4: $[\boldsymbol{V}, \boldsymbol{\Lambda}] =$ `eig`$(\boldsymbol{M})$, ▷ eigendecomposition of $\boldsymbol{M}$
5: $\boldsymbol{\Lambda}_k = \boldsymbol{\Lambda}(1\!:\!k, 1\!:\!k) \in \mathbb{R}^{k\times k}$,
6: $\bar{\boldsymbol{C}} = \boldsymbol{C}\boldsymbol{V}(:, 1\!:\!k) \in \mathbb{R}^{n\times k}$

Algorithm 4 forms Equation (4.1) such that the eigenvectors of the truncated eigendecomposition of $\boldsymbol{M}$ is absorbed into the $\boldsymbol{C}$ factor. This is done to ensure numerical

stability, as the matrix multiplication

$$\boldsymbol{C} \cdot \left(\boldsymbol{V}(:, 1:k) \cdot \boldsymbol{\Lambda}(1:k, 1:k)^{\dagger} \cdot \boldsymbol{V}(:, 1:k)^{\top} \right) \cdot \boldsymbol{C}^{\top}$$

can be numerically unstable. This type of numerical instability will be revisited in Chapter 5, where we examine a more general setting involving rectangular matrices. There, we show that depending on the implementation, forming the pseudoinverse of the core matrix can similarly lead to numerically unstable approximation. The recommended sketch size for Algorithm 4 is $s = 1.5k$ for efficiency, but if one wants a better approximation quality guarantee, then the sketch size can be increased to, for example, $s = 2k$ or $s = 4k$. Note that the truncation is performed irrespectively of the singular values of $\boldsymbol{M}$ (unlike previous studies, e.g. [CNX22]); our analysis in Section 4.4 suggests that it is important that the number of singular values to be truncated $(s - k) = (c - 1)k$ is proportional to k.

Complexity. The cost of Algorithm 4 is $\mathcal{O}\left(T_{sketch}^{(ck)} + k^3\right)$, where $T_{sketch}^{(ck)}$ is the cost of forming a sketch of size ck in line 2. For example, $T_{sketch}^{(ck)} = \mathcal{O}(n^2 \log k)$ for SRTTs and $T_{sketch}^{(ck)} = \mathcal{O}(\xi \cdot \mathrm{nnz}(\boldsymbol{A}))$ for sparse sign embeddings, where ξ is the sparsity parameter. The cost $\mathcal{O}(k^3)$ is for computing the eigendecomposition of the core matrix $\boldsymbol{M}$ in line 4.

Eigendecomposition of $\boldsymbol{A}_{indef}^{(c,k)}$. Algorithm 4 as presented does not output the eigendecomposition of $\boldsymbol{A}_{indef}^{(c,k)}$. To do this, we require an extra $\mathcal{O}(nk^2 + k^3)$ flops. We need $\mathcal{O}(nk^2)$ flops to compute the thin QR decomposition of $\boldsymbol{C} = \boldsymbol{Q}_{\boldsymbol{C}}\boldsymbol{R}_{\boldsymbol{C}}$, $\mathcal{O}(k^3)$ flops to form and compute the eigendecomposition of $\boldsymbol{R}_{\boldsymbol{C}} [\![\boldsymbol{M}]\!]_k^{\dagger} \boldsymbol{R}_{\boldsymbol{C}}^{\top} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{U}^{\top}$ and $\mathcal{O}(nk^2)$ flops to form $\boldsymbol{U}_1 = \boldsymbol{Q}_{\boldsymbol{C}}\boldsymbol{U}$ giving us the eigendecomposition, $\boldsymbol{A}_{indef}^{(c,k)} = \boldsymbol{U}_1\boldsymbol{\Sigma}\boldsymbol{U}_1^{\top}$.

In Figure 4.3, we illustrate Algorithm 4 using an SRTT and a sparse sign embedding. The experiment was conducted with synthetic 2000×2000 symmetric indefinite matrices. The top two plots have eigenvalues that decay geometrically from 1 to 10^{-12} each assigned

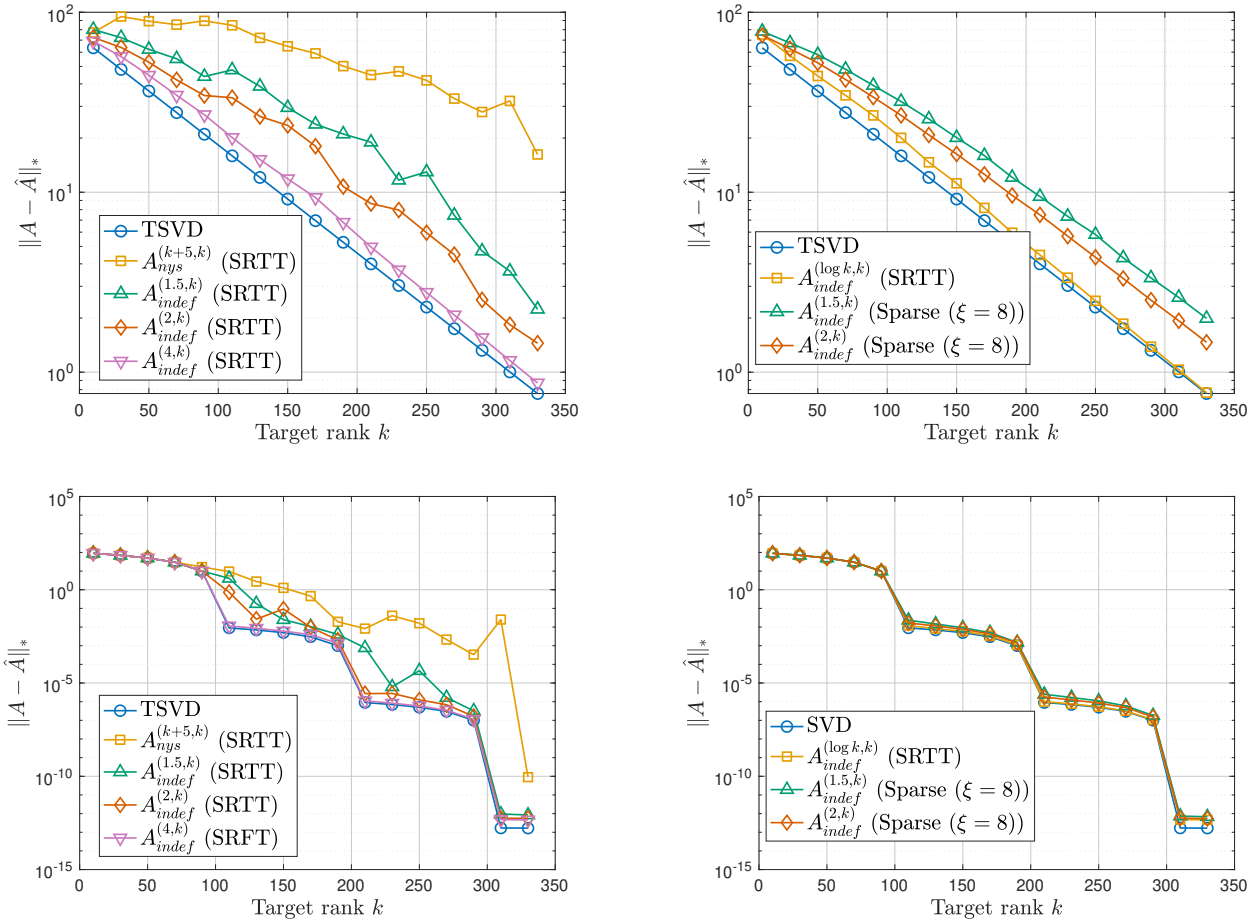


Figure 4.3: Algorithm 4: A difficult (coherent) example for the SRTT sketch. The approximation can be unstable if the sketch size is too small for the SRTT sketch (left plots). This problem can be fixed by enlarging the sketch size. The right plots show that sparse maps have no issue with this example and the approximation is robust.

a random sign with equal probability and the eigenvectors are in a 2×2 block diagonal form, $\text{diag}(\boldsymbol{I}_{200}, \boldsymbol{U})$ where $\boldsymbol{I}_{200}$ is the 200×200 identity matrix and $\boldsymbol{U} \in \mathbb{R}^{1800 \times 1800}$ is a Haar distributed orthogonal matrix. This eigenvector matrix is a more coherent example than our previous examples and is known to be a difficult example for SRTTs [BG13]. The bottom two plots were generated using the same eigenvector matrix, but with eigenvalues equal to ± 1 for the first 100, $\pm 10^{-4}$ for the next 100, $\pm 10^{-8}$ for the 100 eigenvalues after that, and $\pm 10^{-16}$ for the last 1700 eigenvalues each assigned a random sign with equal probability.

In the two left plots, we see that the SRTT sketch can fail if the sketch size is not large enough. This instability in the approximation can be fixed by enlarging the sketch size. We see that $s = k + 5$ does not do well, but when $s = 4k$ the approximation becomes more accurate and robust. In the right plots, we see that the SRTT sketch with the sketch size $s = \lfloor k \log k \rfloor$, which comes with theoretical guarantees, has excellent approximation quality. Finally, we see that the sparse map with sparsity $\xi = 8$ gives a robust approximation throughout, which can be improved by enlarging the sketch size.

4.4 Analysis

For a general symmetric matrix $\boldsymbol{A} \in \mathbb{R}^{n \times n}$, there is no known relative-error norm bounds for the Nyström method. Here, we show that for general symmetric matrices, the Nyström method with a Gaussian embedding achieves, in expectation, a relative-error bound in nuclear norm after an orthogonal projection in the core matrix, provided the singular values decay sufficiently fast. The analysis that follows establishes the accuracy not of Algorithm 4, but of a closely related variant of the Nyström method. The last paragraph of this section discusses this in more detail.

Let $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ be a symmetric matrix and let the eigendecomposition of $\boldsymbol{A}$ be

$$\boldsymbol{A} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^\top = [\boldsymbol{V}_1, \boldsymbol{V}_2, \boldsymbol{V}_3] \begin{bmatrix} \boldsymbol{\Lambda}_1 & & \\ & \boldsymbol{\Lambda}_2 & \\ & & \boldsymbol{\Lambda}_3 \end{bmatrix} [\boldsymbol{V}_1, \boldsymbol{V}_2, \boldsymbol{V}_3]^\top \tag{4.2}$$

where $\boldsymbol{V} \in \mathbb{R}^{n \times n}$ is the orthogonal eigenvector matrix of $\boldsymbol{A}$ and $\boldsymbol{\Lambda} \in \mathbb{R}^{n \times n}$ is a diagonal matrix containing the eigenvalues of $\boldsymbol{A}$. The matrices with subscript 1 have k columns, those with subscript 2 have $(c_1 - 1)k$ columns and subscript 3 have $(n - c_1 k)$ columns where $k < c_1 k < n$ and $c_1 > 1$ is a constant such that $c_1 k$ is a positive integer. The eigenvalues are ordered in

non-increasing order with respect to their magnitude, so we have $\sigma_i(\boldsymbol{A}) = |\lambda_i(\boldsymbol{A})|$ for all i.

Now we state our main theorem, and discuss the three key facts that will accompany our proof before getting to the proof immediately.

Theorem 4.4.1. *Let $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ be a symmetric matrix as in Equation (4.2) and assume that $\lambda_k(\boldsymbol{A}) \neq 0$. Let c_1 and c_2 be constants with $1 < c_1 < c_2 < \frac{n}{k} - 1$ such that $c_1 k$ and $c_2 k$ are positive integers. Define $\boldsymbol{S}_i := \boldsymbol{V}_i^\top \boldsymbol{S}_G$ for $i = 1, 2, 3$ where $\boldsymbol{S}_G \in \mathbb{R}^{n\times c_2 k}$ is a Gaussian matrix, and set $\boldsymbol{B} = \boldsymbol{S}_3\boldsymbol{Q}_\perp(\boldsymbol{S}_1\boldsymbol{Q}_\perp)^\dagger \in \mathbb{R}^{(n-c_1k)\times k}$ where $\boldsymbol{Q}_\perp \in \mathbb{R}^{c_2k\times(c_2-c_1+1)k}$ is an orthogonal complement of $\boldsymbol{S}_2^\top \in \mathbb{R}^{c_2k\times(c_1-1)k}$. Let $(\boldsymbol{S}_1\boldsymbol{Q}_\perp)^\dagger = \hat{\boldsymbol{Q}}\hat{\boldsymbol{R}}$ be the thin QR decomposition of $(\boldsymbol{S}_1\boldsymbol{Q}_\perp)^\dagger$ and set $\boldsymbol{U} := \boldsymbol{Q}_\perp\hat{\boldsymbol{Q}} \in \mathbb{R}^{c_2k\times k}$. Then the orthogonal projector $\boldsymbol{P} = \boldsymbol{U}\boldsymbol{U}^\top \in \mathbb{R}^{c_2k\times c_2k}$ satisfies*

$$\mathbb{E}\left[\|\boldsymbol{E}\|_* \,|\Omega_F\right] \leq (1 + \epsilon_{k,\boldsymbol{A}}) \left\|\boldsymbol{A} - [\![\boldsymbol{A}]\!]_k\right\|_* \tag{4.3}$$

where

$$\boldsymbol{E} := \boldsymbol{A} - \boldsymbol{A}\boldsymbol{S}_G(\boldsymbol{P}\boldsymbol{S}_G^\top\boldsymbol{A}\boldsymbol{S}_G\boldsymbol{P})^\dagger\boldsymbol{S}_G^\top\boldsymbol{A}$$

is the associated Nyström error, Ω_F is an event defined as

$$\Omega_F := \left\{\left\||\boldsymbol{\Lambda}_3|^{1/2}\boldsymbol{B}\right\|_{\mathrm{F}}^2 \leq 0.5|\lambda_k(\boldsymbol{A})|\right\}$$

where $|\boldsymbol{\Lambda_3}|$ is defined element-wise and

$$\epsilon_{k,\boldsymbol{A}} := 2b\sqrt{k}\left(1 + \sqrt{\frac{2}{b}} + \frac{|\lambda_{c_1k+1}(\boldsymbol{A})|}{|\lambda_k(\boldsymbol{A})|}\right)\frac{\|\boldsymbol{\Lambda}_3\|_*}{\|\boldsymbol{\Lambda}_2\|_* + \|\boldsymbol{\Lambda}_3\|_*}$$

where $b = \frac{k}{(c_2-c_1)k-1}$.

In the above theorem, c_1 and c_2 are oversampling factors which are of modest size, say $c_1 = 1.5$ and $c_2 = 2$. We need two factors because we need $\boldsymbol{S}_1\boldsymbol{Q}_\perp \in \mathbb{R}^{k\times(c_2-c_1+1)k}$ to be *rectangular* Gaussian matrices, which makes them well-conditioned with high probability

(Theorem 2.2.1). We can view c_1 as c in Algorithm 4 and c_2 to be the oversampling factor introduced to make the analysis possible. By making c_1, c_2 and $(c_2 - c_1)$ larger, we can improve the bound in the above theorem. The orthogonal projector $\boldsymbol{P} = \boldsymbol{U}\boldsymbol{U}^\top$ truncates the core matrix $\boldsymbol{M} = \boldsymbol{S}_G^\top \boldsymbol{A}\boldsymbol{S}_G$ by removing the largest 'unwanted' eigenvalues of $\boldsymbol{A}$, i.e. the eigenvalues in $\boldsymbol{\Lambda}_2$, using $\boldsymbol{Q}_\perp$ factor in $\boldsymbol{U}$. This helps the core matrix to not be corrupted by the interaction between the target and the large 'unwanted' singular values and singular vectors of $\boldsymbol{A}$, which can happen when forming the core matrix M. Lastly, the term $\epsilon_{k,\boldsymbol{A}}$ plays a similar role to the distortion factor $\frac{k}{s-k-1}$ in Theorem 2.3.4, and Ω_F is roughly the event that the eigenvalues of $\boldsymbol{A}$ decay sufficiently quickly. If we assume that $\boldsymbol{A}$ has a low-rank structure, for example, $|\lambda_k(\boldsymbol{A})| \gg |\lambda_{c_1 k+1}(\boldsymbol{A})|$, then Ω_F would hold with high probability, and $\epsilon_{k,\boldsymbol{A}}$ would be a moderately-sized constant, which tells us that the relative-error nuclear norm bound in Equation (4.3) is good.

We now introduce three key facts that will be useful for our proof. The first fact follows closely the analysis in [Nak20]. Let $\boldsymbol{\mathcal{P}} := \boldsymbol{\Lambda}\boldsymbol{V}^\top \boldsymbol{S}_G(\boldsymbol{P}\boldsymbol{S}_G^\top \boldsymbol{A}\boldsymbol{S}_G\boldsymbol{P})^\dagger \boldsymbol{S}_G^\top \boldsymbol{V}$. Then since $\boldsymbol{U} \in \mathbb{R}^{c_2 k \times k}$ is an orthonormal matrix, we have

$$\begin{aligned}
\boldsymbol{\mathcal{P}} &= \boldsymbol{\Lambda}\boldsymbol{V}^\top \boldsymbol{S}_G(\boldsymbol{P}\boldsymbol{S}_G^\top \boldsymbol{A}\boldsymbol{S}_G\boldsymbol{P})^\dagger \boldsymbol{S}_G^\top \boldsymbol{V} \\
&= \boldsymbol{\Lambda}\boldsymbol{V}^\top \boldsymbol{S}_G(\boldsymbol{U}\boldsymbol{U}^\top \boldsymbol{S}_G^\top \boldsymbol{A}\boldsymbol{S}_G\boldsymbol{U}\boldsymbol{U}^\top)^\dagger \boldsymbol{S}_G^\top \boldsymbol{V} \\
&= \boldsymbol{\Lambda}\boldsymbol{V}^\top \boldsymbol{S}_G\boldsymbol{U}(\boldsymbol{U}^\top \boldsymbol{S}_G^\top \boldsymbol{A}\boldsymbol{S}_G\boldsymbol{U})^\dagger \boldsymbol{U}^\top \boldsymbol{S}_G^\top \boldsymbol{V}
\end{aligned}$$

and $\boldsymbol{\mathcal{P}}$ is an oblique projector, since

$$\begin{aligned}
\boldsymbol{\mathcal{P}}^2 &= \boldsymbol{\Lambda}\boldsymbol{V}^\top \boldsymbol{S}_G\boldsymbol{U}(\boldsymbol{U}^\top \boldsymbol{S}_G^\top \boldsymbol{A}\boldsymbol{S}_G\boldsymbol{U})^\dagger \boldsymbol{U}^\top \boldsymbol{S}_G^\top \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^\top \boldsymbol{S}_G\boldsymbol{U}(\boldsymbol{U}^\top \boldsymbol{S}_G^\top \boldsymbol{A}\boldsymbol{S}_G\boldsymbol{U})^\dagger \boldsymbol{U}^\top \boldsymbol{S}_G^\top \boldsymbol{V} \\
&= \boldsymbol{\Lambda}\boldsymbol{V}^\top \boldsymbol{S}_G\boldsymbol{U}(\boldsymbol{U}^\top \boldsymbol{S}_G^\top \boldsymbol{A}\boldsymbol{S}_G\boldsymbol{U})^\dagger \boldsymbol{U}^\top \boldsymbol{S}_G^\top \boldsymbol{V} = \boldsymbol{\mathcal{P}}.
\end{aligned}$$

As in [Nak20], we can rewrite the associated Nyström error as

$$\boldsymbol{V}^\top \boldsymbol{E}\boldsymbol{V} = (\boldsymbol{I}_n - \boldsymbol{\mathcal{P}})\boldsymbol{\Lambda} = (\boldsymbol{I}_n - \boldsymbol{\mathcal{P}})\boldsymbol{\Lambda}(\boldsymbol{I}_n - \boldsymbol{V}^\top \boldsymbol{S}_G \boldsymbol{U}\boldsymbol{N})$$

for any $\boldsymbol{N} \in \mathbb{R}^{k\times n}$. Let $\boldsymbol{V}_k = [\boldsymbol{I}_k, \boldsymbol{0}]^\top \in \mathbb{R}^{n\times k}$ and set $\boldsymbol{N} = \left(\boldsymbol{V}_k^\top \boldsymbol{V}^\top \boldsymbol{S}_G \boldsymbol{U}\right)^\dagger \boldsymbol{V}_k^\top$; we then find

$$\boldsymbol{V}^\top \boldsymbol{E}\boldsymbol{V} = (\boldsymbol{I}_n - \boldsymbol{\mathcal{P}})\boldsymbol{\Lambda}(\boldsymbol{I}_n - \boldsymbol{V}_k\boldsymbol{V}_k^\top)\left(\boldsymbol{I}_n - \boldsymbol{V}^\top \boldsymbol{S}_G \boldsymbol{U}(\boldsymbol{V}_k^\top \boldsymbol{V}^\top \boldsymbol{S}_G \boldsymbol{U})^\dagger \boldsymbol{V}_k^\top\right). \tag{4.4}$$

This modification of the associated Nyström error will be important for our proof.

The second fact is the following. Let $f(x)$ be convex in the interval $[x_1, x_2]$ with $x_1 < x_2$. Define $g(x)$ on $[x_1, x_2]$ to be the linear function joining the endpoints of f on $[x_1, x_2]$, that is, $g(x) = \frac{f(x_2)-f(x_1)}{x_2-x_1}x + \frac{f(x_1)x_2 - f(x_2)x_1}{x_2-x_1}$. Then $f(x) \leq g(x)$ on $[x_1, x_2]$. Let $\boldsymbol{Y}$ be a random variable with $\boldsymbol{Y} \in [x_1, x_2]$ almost surely. Then $f(\boldsymbol{Y}) \leq g(\boldsymbol{Y})$ almost surely. Furthermore, if $\boldsymbol{Y} \in [x_1, x_2]$ conditional on an event Ω, then conditional on Ω we get

$$f(\boldsymbol{Y}) \leq g(\boldsymbol{Y}). \tag{4.5}$$

The final fact is based on expected norm bounds for Gaussian matrices from [HMT11, App. A]. We can deduce the following lemma.

Lemma 4.4.1. *Let $\boldsymbol{B}$ be the matrix as in Theorem 4.4.1, and let $\boldsymbol{G}$ be a fixed real matrix such that $\boldsymbol{GB}$ is defined. Then*

$$\mathbb{E}\,\|\boldsymbol{GB}\|_{\mathrm{F}}^2 = b\,\|\boldsymbol{G}\|_{\mathrm{F}}^2$$

where $b = \frac{k}{(c_2-c_1)k-1}$ as in Theorem 4.4.1.

Proof. First note that $\boldsymbol{Q}_\perp$ only depends on $\boldsymbol{S}_2$. Now since conditional on $\boldsymbol{S}_2$, $\boldsymbol{S}_3\boldsymbol{Q}_\perp$, and

$\boldsymbol{S}_1\boldsymbol{Q}_\perp$ are two independent Gaussian matrices, we have

$$\begin{aligned}\mathbb{E}_{\boldsymbol{S}_1,\boldsymbol{Q}_\perp,\boldsymbol{S}_3}\|\boldsymbol{GB}\|_{\mathrm{F}}^2 &= \mathbb{E}_{\boldsymbol{S}_1}\left[\mathbb{E}_{\boldsymbol{S}_3}\left[\left\|\boldsymbol{G}\boldsymbol{S}_3\boldsymbol{Q}_\perp(\boldsymbol{S}_1\boldsymbol{Q}_\perp)^\dagger\right\|_{\mathrm{F}}^2\middle|\boldsymbol{S}_1,\boldsymbol{Q}_\perp\right]\right]\\ &= \|\boldsymbol{G}\|_{\mathrm{F}}^2\,\mathbb{E}_{\boldsymbol{S}_1,\boldsymbol{Q}_\perp}\left\|(\boldsymbol{S}_1\boldsymbol{Q}_\perp)^\dagger\right\|_{\mathrm{F}}^2\\ &= \frac{k}{(c_2-c_1)k-1}\|\boldsymbol{G}\|_{\mathrm{F}}^2\end{aligned}$$

using the tower property and the propositions in [HMT11, App. A]. □

Now using these three key facts we are ready to prove Theorem 4.4.1.

Proof of Theorem 4.4.1. We begin by noting that since $\boldsymbol{S}_1\boldsymbol{Q}_\perp \in \mathbb{R}^{k\times(c_2-c_1+1)k}$ is a fat rectangular Gaussian matrix, hence full rank with probability 1, we have $\boldsymbol{S}_1\boldsymbol{Q}_\perp(\boldsymbol{S}_1\boldsymbol{Q}_\perp)^\dagger = \boldsymbol{I}_k$. Therefore

$$\boldsymbol{S}_1\boldsymbol{Q}_\perp\hat{\boldsymbol{Q}} = \hat{\boldsymbol{R}}^{-1}$$

and we get

$$\boldsymbol{V}^\top\boldsymbol{S}_G\boldsymbol{U} = \begin{bmatrix}\boldsymbol{S}_1\\ \boldsymbol{S}_2\\ \boldsymbol{S}_3\end{bmatrix}\boldsymbol{Q}_\perp\hat{\boldsymbol{Q}} = \begin{bmatrix}\boldsymbol{S}_1\boldsymbol{Q}_\perp\hat{\boldsymbol{Q}}\\ \boldsymbol{0}\\ \boldsymbol{S}_3\boldsymbol{Q}_\perp\hat{\boldsymbol{Q}}\end{bmatrix} = \begin{bmatrix}\hat{\boldsymbol{R}}^{-1}\\ \boldsymbol{0}\\ \boldsymbol{B}\hat{\boldsymbol{R}}^{-1}\end{bmatrix},$$

where $\boldsymbol{B} = \boldsymbol{S}_3\boldsymbol{Q}_\perp(\boldsymbol{S}_1\boldsymbol{Q}_\perp)^\dagger \in \mathbb{R}^{(n-c_1k)\times k}$.

Now we use the first key fact (Equation (4.4)) and get

$$\boldsymbol{V}^\top\boldsymbol{E}\boldsymbol{V} = (\boldsymbol{I}_n-\boldsymbol{\mathcal{P}})\boldsymbol{\Lambda}(\boldsymbol{I}_n-\boldsymbol{V}_k\boldsymbol{V}_k^\top)\left(\boldsymbol{I}_n-\boldsymbol{V}^\top\boldsymbol{S}_G\boldsymbol{U}\left(\boldsymbol{V}_k^\top\boldsymbol{V}^\top\boldsymbol{S}_G\boldsymbol{U}\right)^\dagger\boldsymbol{V}_k^\top\right),$$

where $\boldsymbol{\mathcal{P}} = \boldsymbol{\Lambda}\boldsymbol{V}^\top\boldsymbol{S}_G\boldsymbol{U}\left(\boldsymbol{U}^\top\boldsymbol{S}_G^\top\boldsymbol{A}\boldsymbol{S}_G\boldsymbol{U}\right)^\dagger\boldsymbol{U}^\top\boldsymbol{S}_G^\top\boldsymbol{V}$ and set $\boldsymbol{V}_k = [\boldsymbol{I}_k,\boldsymbol{0}]^\top \in \mathbb{R}^{n\times k}$.

Therefore,

$$\boldsymbol{V}^\top \boldsymbol{E}\boldsymbol{V} = (\boldsymbol{I}_n - \boldsymbol{\mathcal{P}})\boldsymbol{\Lambda} \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{I}_{n-k} \end{bmatrix} [\boldsymbol{0}, \boldsymbol{I}_{n-k}] \left(\boldsymbol{I}_n - \begin{bmatrix} \hat{\boldsymbol{R}}^{-1} \\ \boldsymbol{0} \\ \boldsymbol{B}\hat{\boldsymbol{R}}^{-1} \end{bmatrix} \left(\hat{\boldsymbol{R}}^{-1}\right)^\dagger [\boldsymbol{I}_k, \boldsymbol{0}] \right)$$
$$= (\boldsymbol{I}_n - \boldsymbol{\mathcal{P}}) \begin{bmatrix} \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{\Lambda}_2 & \boldsymbol{0} \\ -\boldsymbol{\Lambda}_3\boldsymbol{B} & \boldsymbol{0} & \boldsymbol{\Lambda_3} \end{bmatrix},$$

and by conditioning on Ω_F,

$$\boldsymbol{\mathcal{P}} = \begin{bmatrix} \boldsymbol{\Lambda}_1\hat{\boldsymbol{R}}^{-1} \\ \boldsymbol{0} \\ \boldsymbol{B}\hat{\boldsymbol{R}}^{-1} \end{bmatrix} \left(\begin{bmatrix} \hat{\boldsymbol{R}}^{-1} \\ \boldsymbol{0} \\ \boldsymbol{B}\hat{\boldsymbol{R}}^{-1} \end{bmatrix}^\top \begin{bmatrix} \boldsymbol{\Lambda}_1\hat{\boldsymbol{R}}^{-1} \\ \boldsymbol{0} \\ \boldsymbol{\Lambda}_3\boldsymbol{B}\hat{\boldsymbol{R}}^{-1} \end{bmatrix} \right)^\dagger \begin{bmatrix} \hat{\boldsymbol{R}}^{-1} \\ \boldsymbol{0} \\ \boldsymbol{B}\hat{\boldsymbol{R}}^{-1} \end{bmatrix}^\top$$
$$= \begin{bmatrix} \boldsymbol{\Lambda}_1\hat{\boldsymbol{R}}^{-1} \\ \boldsymbol{0} \\ \boldsymbol{\Lambda}_3\boldsymbol{B}\hat{\boldsymbol{R}}^{-1} \end{bmatrix} \left(\hat{\boldsymbol{R}}^{-\top} \left(\boldsymbol{\Lambda}_1 + \boldsymbol{B}^\top\boldsymbol{\Lambda}_3\boldsymbol{B} \right) \hat{\boldsymbol{R}}^{-1} \right)^\dagger \begin{bmatrix} \hat{\boldsymbol{R}}^{-1} \\ \boldsymbol{0} \\ \boldsymbol{B}\hat{\boldsymbol{R}}^{-1} \end{bmatrix}^\top$$
$$= \begin{bmatrix} \boldsymbol{\Lambda}_1 \\ \boldsymbol{0} \\ \boldsymbol{\Lambda}_3\boldsymbol{B} \end{bmatrix} \left(\boldsymbol{\Lambda}_1 + \boldsymbol{B}^\top\boldsymbol{\Lambda}_3\boldsymbol{B} \right)^\dagger \left[\boldsymbol{I}_k, \boldsymbol{0}, \boldsymbol{B}^\top \right],$$

where we took out a factor of $\hat{\boldsymbol{R}}^{-1}$ and $\hat{\boldsymbol{R}}^{-\top}$ from the pseudo-inverse. This is possible because $(\boldsymbol{\Lambda}_1 + \boldsymbol{B}^\top\boldsymbol{\Lambda}_3\boldsymbol{B})$ is a non-singular $k \times k$ matrix if we condition on Ω_F. Now for shorthand, let

$\boldsymbol{Z} := \boldsymbol{\Lambda}_1 + \boldsymbol{B}^\top \boldsymbol{\Lambda}_3 \boldsymbol{B}$. Then,

$$\boldsymbol{I}_n - \boldsymbol{\mathcal{P}} = \begin{bmatrix} \boldsymbol{I}_k - \boldsymbol{\Lambda}_1 \boldsymbol{Z}^\dagger & \boldsymbol{0} & -\boldsymbol{\Lambda}_1 \boldsymbol{Z}^\dagger \boldsymbol{B}^\top \\ \boldsymbol{0} & \boldsymbol{I}_{(c_1-1)k} & \boldsymbol{0} \\ -\boldsymbol{\Lambda}_3 \boldsymbol{B} \boldsymbol{Z}^\dagger & \boldsymbol{0} & \boldsymbol{I}_{n-c_1 k} - \boldsymbol{\Lambda}_3 \boldsymbol{B} \boldsymbol{Z}^\dagger \boldsymbol{B}^\top \end{bmatrix}.$$

Therefore

$$\begin{aligned} \boldsymbol{V}^\top \boldsymbol{E} \boldsymbol{V} &= (\boldsymbol{I}_n - \boldsymbol{\mathcal{P}}) \begin{bmatrix} \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{\Lambda}_2 & \boldsymbol{0} \\ -\boldsymbol{\Lambda}_3 \boldsymbol{B} & \boldsymbol{0} & \boldsymbol{\Lambda}_3 \end{bmatrix} \\ &= \begin{bmatrix} \boldsymbol{\Lambda}_1 \boldsymbol{Z}^\dagger \boldsymbol{B}^\top \boldsymbol{\Lambda}_3 \boldsymbol{B} & \boldsymbol{0} & -\boldsymbol{\Lambda}_1 \boldsymbol{Z}^\dagger \boldsymbol{B}^\top \boldsymbol{\Lambda}_3 \\ \boldsymbol{0} & \boldsymbol{\Lambda}_2 & \boldsymbol{0} \\ -\boldsymbol{\Lambda}_3 \boldsymbol{B} + \boldsymbol{\Lambda}_3 \boldsymbol{B} \boldsymbol{Z}^\dagger \boldsymbol{B}^\top \boldsymbol{\Lambda}_3 \boldsymbol{B} & \boldsymbol{0} & \boldsymbol{\Lambda}_3 - \boldsymbol{\Lambda}_3 \boldsymbol{B} \boldsymbol{Z}^\dagger \boldsymbol{B}^\top \boldsymbol{\Lambda}_3 \end{bmatrix} \\ &= \begin{bmatrix} \boldsymbol{\Lambda}_1 \boldsymbol{Z}^\dagger \boldsymbol{B}^\top \boldsymbol{\Lambda}_3 \boldsymbol{B} & \boldsymbol{0} & -\boldsymbol{\Lambda}_1 \boldsymbol{Z}^\dagger \boldsymbol{B}^\top \boldsymbol{\Lambda}_3 \\ \boldsymbol{0} & \boldsymbol{\Lambda}_2 & \boldsymbol{0} \\ -\boldsymbol{\Lambda}_3 \boldsymbol{B} \boldsymbol{Z}^\dagger \boldsymbol{\Lambda}_1 & \boldsymbol{0} & \boldsymbol{\Lambda}_3 - \boldsymbol{\Lambda}_3 \boldsymbol{B} \boldsymbol{Z}^\dagger \boldsymbol{B}^\top \boldsymbol{\Lambda}_3 \end{bmatrix}, \end{aligned}$$

since $\boldsymbol{\Lambda}_3 \boldsymbol{B} \boldsymbol{Z}^\dagger \boldsymbol{B}^\top \boldsymbol{\Lambda}_3 \boldsymbol{B} = \boldsymbol{\Lambda}_3 \boldsymbol{B} \left(\boldsymbol{\Lambda}_1 + \boldsymbol{B}^\top \boldsymbol{\Lambda}_3 \boldsymbol{B} \right)^\dagger \boldsymbol{B}^\top \boldsymbol{\Lambda}_3 \boldsymbol{B} = \boldsymbol{\Lambda}_3 \boldsymbol{B} - \boldsymbol{\Lambda}_3 \boldsymbol{B} \boldsymbol{Z}^\dagger \boldsymbol{\Lambda}_1$. We now bound $\boldsymbol{E}$ in the nuclear norm. For shorthand, define the following error matrices,

$$\begin{aligned} \boldsymbol{E}_1 &= \boldsymbol{\Lambda}_1 \boldsymbol{Z}^\dagger \boldsymbol{B}^\top \boldsymbol{\Lambda}_3 \boldsymbol{B}, \\ \boldsymbol{E}_2 &= \boldsymbol{\Lambda}_1 \boldsymbol{Z}^\dagger \boldsymbol{B}^\top \boldsymbol{\Lambda}_3, \\ \boldsymbol{E}_3 &= \boldsymbol{\Lambda}_3 - \boldsymbol{\Lambda}_3 \boldsymbol{B} \boldsymbol{Z}^\dagger \boldsymbol{B}^\top \boldsymbol{\Lambda}_3. \end{aligned}$$

We then have

$$\|\boldsymbol{E}\|_* \leq \|\boldsymbol{\Lambda}_2\|_* + \|\boldsymbol{E}_1\|_* + 2\,\|\boldsymbol{E}_2\|_* + \|\boldsymbol{E}_3\|_*\,.$$

Let us note

$$\boldsymbol{\Lambda}_1\boldsymbol{Z}^\dagger = \boldsymbol{\Lambda}_1\left(\boldsymbol{\Lambda}_1 + \boldsymbol{B}^\top\boldsymbol{\Lambda}_3\boldsymbol{B}\right)^\dagger = \left(\boldsymbol{I}_k + \boldsymbol{B}^\top\boldsymbol{\Lambda}_3\boldsymbol{B}\boldsymbol{\Lambda}_1^{-1}\right)^\dagger,$$

and

$$\left\|\boldsymbol{B}^\top\boldsymbol{\Lambda}_3\boldsymbol{B}\right\|_{\mathrm{F}} = \left\|\boldsymbol{B}^\top|\boldsymbol{\Lambda}_3|^{1/2}\operatorname{sgn}(\boldsymbol{\Lambda}_3)|\boldsymbol{\Lambda}_3|^{1/2}\boldsymbol{B}\right\|_{\mathrm{F}} \leq \left\||\boldsymbol{\Lambda}_3|^{1/2}\boldsymbol{B}\right\|_{\mathrm{F}}^2,$$

where $|\boldsymbol{\Lambda}_3|$ and $\operatorname{sgn}(\boldsymbol{\Lambda}_3)$ are defined element-wise. Here, $\operatorname{sgn}(\cdot)$ is the sign function.

We now bound $\mathbb{E}[\|\boldsymbol{E}_1\|_*\,|\Omega_F]$, $\mathbb{E}[\|\boldsymbol{E}_2\|_*\,|\Omega_F]$ and $\mathbb{E}[\|\boldsymbol{E}_3\|_*\,|\Omega_F]$ using the second (Equation (4.5)) and the third (Lemma 4.4.1) key fact. We start with $\boldsymbol{E}_1$. Conditional on Ω_F, we have

$$\begin{aligned}
\|\boldsymbol{E}_1\|_* &\leq \sqrt{k}\left\|\boldsymbol{\Lambda}_1\boldsymbol{Z}^\dagger\boldsymbol{B}^\top\boldsymbol{\Lambda}_3\boldsymbol{B}\right\|_{\mathrm{F}} \\
&\leq \sqrt{k}\left\|\left(\boldsymbol{I}_k + \boldsymbol{B}^\top\boldsymbol{\Lambda}_3\boldsymbol{B}\boldsymbol{\Lambda}_1^{-1}\right)^\dagger\right\|_2\left\|\boldsymbol{B}^\top\boldsymbol{\Lambda}_3\boldsymbol{B}\right\|_{\mathrm{F}} \\
&\leq \sqrt{k}\frac{\left\|\boldsymbol{B}^\top\boldsymbol{\Lambda}_3\boldsymbol{B}\right\|_{\mathrm{F}}}{1 - \left\|\boldsymbol{B}^\top\boldsymbol{\Lambda}_3\boldsymbol{B}\right\|_{\mathrm{F}}\left\|\boldsymbol{\Lambda}_1^{-1}\right\|_2} \\
&\leq \frac{\sqrt{k}}{\left\|\boldsymbol{\Lambda}_1^{-1}\right\|_2}\frac{\left\||\boldsymbol{\Lambda}_3|^{1/2}\boldsymbol{B}\right\|_{\mathrm{F}}^2\left\|\boldsymbol{\Lambda}_1^{-1}\right\|_2}{1 - \||\boldsymbol{\Lambda}_3|^{1/2}\boldsymbol{B}\|_{\mathrm{F}}^2\left\|\boldsymbol{\Lambda}_1^{-1}\right\|_2} \\
&\leq \frac{\sqrt{k}}{\left\|\boldsymbol{\Lambda}_1^{-1}\right\|_2}\left(2\left\||\boldsymbol{\Lambda}_3|^{1/2}\boldsymbol{B}\right\|_{\mathrm{F}}^2\left\|\boldsymbol{\Lambda}_1^{-1}\right\|_2\right)
\end{aligned}$$

where the last inequality was obtained using the second fact with $\boldsymbol{Y} = \left\||\boldsymbol{\Lambda}_3|^{1/2}\boldsymbol{B}\right\|_{\mathrm{F}}^2\left\|\boldsymbol{\Lambda}_1^{-1}\right\|_2$, the event Ω_F, the interval $[0, 0.5]$, $f(x) = \frac{x}{1-x}$ which is convex on $[0, 0.5]$ and $g(x) = 2x$. Now

taking conditional expectation and using the third fact (Lemma 4.4.1) we get

$$\begin{aligned}\mathbb{E}\left[\left\|\boldsymbol{E}_1\right\|_* |\Omega_F\right] &\leq 2\sqrt{k}\mathbb{E}\left[\left\||\boldsymbol{\Lambda}_3|^{1/2}\boldsymbol{B}\right\|_{\mathrm{F}}^2\middle|\Omega_F\right] \\ &= 2\sqrt{k}b\left\||\boldsymbol{\Lambda}_3|^{1/2}\right\|_{\mathrm{F}}^2 \\ &= 2\sqrt{k}b\left\|\boldsymbol{\Lambda}_3\right\|_*.\end{aligned}$$

For $\boldsymbol{E}_2$, it is similar to $\boldsymbol{E}_1$. Conditional on Ω_F we have

$$\begin{aligned}\left\|\boldsymbol{E}_2\right\|_* &\leq \sqrt{k}\left\|\boldsymbol{\Lambda}_1(\boldsymbol{\Lambda}_1+\boldsymbol{B}^\top\boldsymbol{\Lambda}_3\boldsymbol{B})^\dagger\boldsymbol{B}^\top\boldsymbol{\Lambda}_3\right\|_{\mathrm{F}} \\ &\leq \sqrt{k}\left\|(\boldsymbol{I}_k+\boldsymbol{B}^\top\boldsymbol{\Lambda}_3\boldsymbol{B}\boldsymbol{\Lambda}_1^{-1})^\dagger\right\|_2\left\||\boldsymbol{\Lambda}_3|^{1/2}\boldsymbol{B}\right\|_{\mathrm{F}}\left\||\boldsymbol{\Lambda}_3|^{1/2}\right\|_{\mathrm{F}} \\ &\leq \frac{\sqrt{k}\sqrt{\left\|\boldsymbol{\Lambda}_3\right\|_*}}{\sqrt{\left\|\boldsymbol{\Lambda}_1^{-1}\right\|_2}}\frac{\left\||\boldsymbol{\Lambda}_3|^{1/2}\boldsymbol{B}\right\|_{\mathrm{F}}\sqrt{\left\|\boldsymbol{\Lambda}_1^{-1}\right\|_2}}{1-\left\||\boldsymbol{\Lambda}_3|^{1/2}\boldsymbol{B}\right\|_{\mathrm{F}}^2\left\|\boldsymbol{\Lambda}_1^{-1}\right\|_2} \\ &\leq \frac{\sqrt{k}\sqrt{\left\|\boldsymbol{\Lambda}_3\right\|_*}}{\sqrt{\left\|\boldsymbol{\Lambda}_1^{-1}\right\|_2}}\left(\sqrt{2}\left\||\boldsymbol{\Lambda}_3|^{1/2}\boldsymbol{B}\right\|_{\mathrm{F}}\sqrt{\left\|\boldsymbol{\Lambda}_1^{-1}\right\|_2}\right)\end{aligned}$$

where we used the second fact for the last inequality with $\boldsymbol{Y}=\left\||\boldsymbol{\Lambda}_3|^{1/2}\boldsymbol{B}\right\|_{\mathrm{F}}\sqrt{\left\|\boldsymbol{\Lambda}_1^{-1}\right\|_2}$, the interval $\left[0,\sqrt{0.5}\right]$, $f(x)=\frac{x}{1-x^2}$ and $g(x)=\sqrt{2}x$. Therefore we get

$$\begin{aligned}\mathbb{E}[\left\|\boldsymbol{E}_2\right\|_*|\Omega_F] &\leq \sqrt{2k}\sqrt{\left\|\boldsymbol{\Lambda}_3\right\|_*}\mathbb{E}\left[\left\||\boldsymbol{\Lambda}_3|^{1/2}\boldsymbol{B}\right\|_{\mathrm{F}}\middle|\Omega_F\right] \\ &\leq \sqrt{2k}\sqrt{\left\|\boldsymbol{\Lambda}_3\right\|_*}\sqrt{\mathbb{E}\left[\left\||\boldsymbol{\Lambda}_3|^{1/2}\boldsymbol{B}\right\|_{\mathrm{F}}^2\middle|\Omega_F\right]} \\ &\leq \sqrt{2kb}\left\|\boldsymbol{\Lambda}_3\right\|_*\end{aligned}$$

using Lemma 4.4.1 and Jensen's inequality for the second inequality.

Finally for $\boldsymbol{E}_3$, we get

$$\left\|\boldsymbol{E}_3\right\|_* \leq \left\|\boldsymbol{\Lambda}_3\right\|_* + \sqrt{k}\left\||\boldsymbol{\Lambda}_3|^{1/2}\right\|_2^2\left\||\boldsymbol{\Lambda}_3|^{1/2}\boldsymbol{B}\right\|_{\mathrm{F}}^2\left\|(\boldsymbol{\Lambda}_1+\boldsymbol{B}^\top\boldsymbol{\Lambda}_3\boldsymbol{B})^\dagger\right\|_2$$

in a similar manner, and conditional on Ω_F we have

$$\begin{aligned}\left\||\boldsymbol{\Lambda}_3|^{1/2}\boldsymbol{B}\right\|_{\mathrm{F}}^2\left\|(\boldsymbol{\Lambda}_1+\boldsymbol{B}^\top\boldsymbol{\Lambda}_3\boldsymbol{B})^\dagger\right\|_2 &\leq \left\||\boldsymbol{\Lambda}_3|^{1/2}\boldsymbol{B}\right\|_{\mathrm{F}}^2\left\|\boldsymbol{\Lambda}_1^{-1}\right\|_2\left\|\boldsymbol{\Lambda}_1(\boldsymbol{\Lambda}_1+\boldsymbol{B}^\top\boldsymbol{\Lambda}_3\boldsymbol{B})^\dagger\right\|_2\\ &\leq \frac{\left\||\boldsymbol{\Lambda}_3|^{1/2}\boldsymbol{B}\right\|_{\mathrm{F}}^2\left\|\boldsymbol{\Lambda}_1^{-1}\right\|_2}{1-\||\boldsymbol{\Lambda}_3|^{1/2}\boldsymbol{B}\|_{\mathrm{F}}^2\left\|\boldsymbol{\Lambda}_1^{-1}\right\|_2}\\ &\leq 2\left\||\boldsymbol{\Lambda}_3|^{1/2}\boldsymbol{B}\right\|_{\mathrm{F}}^2\left\|\boldsymbol{\Lambda}_1^{-1}\right\|_2\end{aligned}$$

using the second fact with the same values as the $\boldsymbol{E}_1$ case. Therefore,

$$\begin{aligned}\mathbb{E}[\|\boldsymbol{E}_3\|_*|\Omega_F] &\leq \|\boldsymbol{\Lambda}_3\|_* + 2\sqrt{k}\,\|\boldsymbol{\Lambda}_3\|_2\left\|\boldsymbol{\Lambda}_1^{-1}\right\|_2\mathbb{E}\left[\left\||\boldsymbol{\Lambda}_3|^{1/2}\boldsymbol{B}\right\|_{\mathrm{F}}^2\middle|\Omega_F\right]\\ &\leq \|\boldsymbol{\Lambda}_3\|_* + 2\sqrt{k}b\,\|\boldsymbol{\Lambda}_3\|_2\left\|\boldsymbol{\Lambda}_1^{-1}\right\|_2\|\boldsymbol{\Lambda}_3\|_*.\end{aligned}$$

Finally, combining everything together we get

$$\mathbb{E}\left[\|\boldsymbol{E}\|_*|\Omega_F\right] \leq \|\boldsymbol{\Lambda}_2\|_* + \|\boldsymbol{\Lambda}_3\|_* + 2b\sqrt{k}\left(1+\frac{|\lambda_{c_1k+1}(\boldsymbol{A})|}{|\lambda_k(\boldsymbol{A})|}+\sqrt{\frac{2}{b}}\right)\|\boldsymbol{\Lambda}_3\|_*.$$

Therefore

$$\mathbb{E}\left[\|\boldsymbol{E}\|_*|\Omega_F\right] \leq (1+\epsilon_{k,\boldsymbol{A}})\left(\|\boldsymbol{\Lambda}_2\|_* + \|\boldsymbol{\Lambda}_3\|_*\right) = (1+\epsilon_{k,\boldsymbol{A}})\,\|\boldsymbol{A}-[\![\boldsymbol{A}]\!]_k\|_*$$

with

$$\epsilon_{k,\boldsymbol{A}} = 2b\sqrt{k}\left(1+\sqrt{\frac{2}{b}}+\frac{|\lambda_{c_1k+1}(\boldsymbol{A})|}{|\lambda_k(\boldsymbol{A})|}\right)\frac{\|\boldsymbol{\Lambda}_3\|_*}{\|\boldsymbol{\Lambda}_2\|_*+\|\boldsymbol{\Lambda}_3\|_*}.$$

□

Remark 4.4.1.

1. *The relative-error nuclear norm bound is informative if $\epsilon_{k,\boldsymbol{A}}$ is small. Now since*

$b \approx (c_2 - c_1)^{-1} = \mathcal{O}(1)$, *we have*

$$\epsilon_{k,\boldsymbol{A}} = \mathcal{O}\left(\frac{\sqrt{k}\,\|\boldsymbol{\Lambda}_3\|_*}{\|\boldsymbol{\Lambda}_2\|_* + \|\boldsymbol{\Lambda}_3\|_*}\right). \tag{4.6}$$

Therefore the relative-error nuclear norm bound is good if

$$\sqrt{k}\sum_{j=c_1k+1}^{n}|\lambda_j(\boldsymbol{A})| = \sqrt{k}\,\|\boldsymbol{\Lambda}_3\|_* \lesssim \|\boldsymbol{\Lambda}_2\|_* = \sum_{j=k+1}^{c_1k}|\lambda_j(\boldsymbol{A})|.$$

2. *We can relax the condition* Ω_F *to* $\Omega_2 := \left\{\left\||\boldsymbol{\Lambda}_3|^{1/2}\boldsymbol{B}\right\|_2^2 \le 0.5|\lambda_k(\boldsymbol{A})|\right\}$ *at the cost of a slightly worse bound in Equation* (4.3)*. The bound in Equation* (4.3) *then changes to*

$$\mathbb{E}\left[\|\boldsymbol{E}\|_*\,|\Omega_2\right] \le (1+\sqrt{k}\epsilon_{k,\boldsymbol{A}})\,\|\boldsymbol{A} - [\![\boldsymbol{A}]\!]_k\|_*.$$

Probability of Ω_F. The probability of the event Ω_F happening can be computed by following the proof of Theorem 10.8 in [HMT11] using $p = (c_2 - c_1)k$, and Lemma 4.4.1. We get

$$\mathbb{P}\left(\left\||\boldsymbol{\Lambda}_3|^{1/2}\boldsymbol{B}\right\|_{\mathrm{F}} \le \sqrt{\|\boldsymbol{\Lambda}_3\|_*}\sqrt{\frac{3k}{(c_2-c_1)k+1}}t + \sqrt{\|\boldsymbol{\Lambda}_3\|_2}\cdot\frac{e\sqrt{(c_2-c_1+1)k}}{(c_2-c_1)k+1}tu\right) \ge 1 - 2t^{-(c_2-c_1)k} - e^{-u^2/2}$$

for $u, t > 0$. Now using $(x+y)^2 \le 2(x^2+y^2)$, we get

$$\left(\sqrt{\|\boldsymbol{\Lambda}_3\|_*}\sqrt{\frac{3k}{(c_2-c_1)k+1}}t + \sqrt{\|\boldsymbol{\Lambda_3}\|_2}\frac{e\sqrt{(c_2-c_1+1)k}}{(c_2-c_1)k+1}tu\right)^2 \le 2t^2\left(\|\boldsymbol{\Lambda}_3\|_*\frac{3k}{(c_2-c_1)k+1} + \|\boldsymbol{\Lambda}_3\|_2\frac{e^2(c_2-c_1+1)k}{((c_2-c_1)k+1)^2}u^2\right).$$

Therefore

$$\mathbb{P}\left(\Omega_F\right) = \mathbb{P}\left(\left\||\boldsymbol{\Lambda}_3|^{1/2}\boldsymbol{B}\right\|_{\mathrm{F}}^2 \leq 0.5|\lambda_k(\boldsymbol{A})|\right) \geq 1 - 2t^{-(c_2-c_1)k} - e^{-u^2/2},$$

if

$$0.5|\lambda_k(\boldsymbol{A})| \geq 2t^2\left(\|\boldsymbol{\Lambda}_3\|_* \frac{3k}{(c_2-c_1)k+1} + \|\boldsymbol{\Lambda}_3\|_2 \frac{e^2(c_2-c_1+1)k}{\left((c_2-c_1)k+1\right)^2}u^2\right),$$

i.e., Ω_F holds with high probability when the tail singular values of $\boldsymbol{A}$ decay rapidly. A similar result can also be derived for Ω_2 by following the same results in [HMT11].

Mixed norm bounds We can obtain mixed norm bounds for Theorem 4.4.1. The 2-norm version of Theorem 4.4.1 would give

$$\mathbb{E}\left[\|\boldsymbol{E}\|_2 \,|\Omega_F\right] \leq \|\boldsymbol{A} - [\![\boldsymbol{A}]\!]_k\|_2 + \frac{\epsilon_{k,\boldsymbol{A}}}{\sqrt{k}}\|\boldsymbol{A} - [\![\boldsymbol{A}]\!]_k\|_*, \tag{4.7}$$

and the Frobenius norm version would give

$$\mathbb{E}\left[\|\boldsymbol{E}\|_{\mathrm{F}} \,|\Omega_F\right] \leq \|\boldsymbol{A} - [\![\boldsymbol{A}]\!]_k\|_{\mathrm{F}} + \frac{\epsilon_{k,\boldsymbol{A}}}{\sqrt{k}}\|\boldsymbol{A} - [\![\boldsymbol{A}]\!]_k\|_* \tag{4.8}$$

where $\epsilon_{k,\boldsymbol{A}}$ is as in Theorem 4.4.1. This improves the constant in front of the best rank-k nuclear norm error to $\epsilon_{k,\boldsymbol{A}}/\sqrt{k} = \mathcal{O}(1)$ by the first remark in Remark 4.4.1 (Equation (4.6)). The proof for the two mixed norm bounds above can be obtained by following the proof of Theorem 4.4.1. More specifically, the proof for the mixed norm bounds stay the same until we bound $\boldsymbol{E}_1$, $\boldsymbol{E}_2$ and $\boldsymbol{E}_3$. To get the mixed norm bound, we use the appropriate norms to bound $\boldsymbol{E}_1$, $\boldsymbol{E}_2$ and $\boldsymbol{E}_3$. For example, to bound $\|\boldsymbol{E}_1\|_{\mathrm{F}}$, we start similarly as in

the nuclear norm case by conditioning on Ω_F to obtain

$$\begin{aligned}\|\boldsymbol{E}_1\|_{\mathrm{F}} &\leq \left\|\left(\boldsymbol{I}_k + \boldsymbol{B}^\top \boldsymbol{\Lambda}_3 \boldsymbol{B} \boldsymbol{\Lambda}_1^{-1}\right)^\dagger\right\|_2 \left\|\boldsymbol{B}^\top \boldsymbol{\Lambda}_3 \boldsymbol{B}\right\|_{\mathrm{F}} \\ &\leq \frac{\left\|\boldsymbol{B}^\top \boldsymbol{\Lambda}_3 \boldsymbol{B}\right\|_{\mathrm{F}}}{1 - \left\|\boldsymbol{B}^\top \boldsymbol{\Lambda}_3 \boldsymbol{B}\right\|_{\mathrm{F}} \left\|\boldsymbol{\Lambda}_1^{-1}\right\|_2} \\ &\leq \frac{1}{\left\|\boldsymbol{\Lambda}_1^{-1}\right\|_2} \left(2 \left\||\boldsymbol{\Lambda}_3|^{1/2} \boldsymbol{B}\right\|_{\mathrm{F}}^2 \left\|\boldsymbol{\Lambda}_1^{-1}\right\|_2\right).\end{aligned}$$

We then get

$$\mathbb{E}\left[\|\boldsymbol{E}_1\|_* \,|\Omega_F\right] \leq 2\mathbb{E}\left[\left\||\boldsymbol{\Lambda}_3|^{1/2} \boldsymbol{B}\right\|_{\mathrm{F}}^2 \Big| \Omega_F\right] = 2b \|\boldsymbol{\Lambda}_3\|_*.$$

The bound for $\|\boldsymbol{E}_2\|_{\mathrm{F}}, \|\boldsymbol{E}_3\|_{\mathrm{F}}, \|\boldsymbol{E}_1\|_2, \|\boldsymbol{E}_2\|_2$ and $\|\boldsymbol{E}_3\|_2$ follows similarly. The mixed norm bounds Equations (4.7) and (4.8) along with the relative-error nuclear norm bound in Theorem 4.4.1 are fairly consistent with the SPSD versions in Table 1 of [GM16].

Theorem 4.4.1 and its proof cannot simply be translated into an algorithm because the proof relies on the eigendecomposition of $\boldsymbol{A}$, which is usually too expensive to compute. However, the proof naturally suggests Algorithm 4. From the proof of Theorem 4.4.1, under the assumption that the matrix has a low-rank structure, as discussed in this section, for example in the paragraph following the statement of Theorem 4.4.1 and in Remark 4.4.1, a projection is desired in the core matrix. This projection eliminates the large 'unwanted' eigenvalues of $\boldsymbol{A}$, specifically those in $\boldsymbol{\Lambda}_2$. In the context of the Nyström method, a natural analogue is to truncate the smallest few singular values in the core matrix $\boldsymbol{M} = \boldsymbol{S}^\top \boldsymbol{A} \boldsymbol{S}$ to achieve the target rank k, as done in Algorithm 4. The theorem also suggests that the sketch size should be proportional to the target rank k, which we adopt in Algorithm 4. Despite Algorithm 4 lacking complete theory (even for the SPSD case), we recommend it because the algorithm does seem to work well empirically as we illustrate in Section 4.5.

4.5 Numerical illustration

We begin by illustrating Theorem 4.4.1 and Algorithm 4. In Figure 4.4, we present both the a priori (upper bound in Equation (4.3)) and a posteriori ($\|\boldsymbol{E}\|_*$ in Equation (4.3)) errors predicted by Theorem 4.4.1, as well as the empirical performance of Algorithm 4, using 1000×1000 symmetric indefinite matrices. In the left plot, the matrix $\boldsymbol{A}$ has eigenvalues that decay geometrically from 1 to 10^{-12} each assigned a random sign with equal probability. In the right plot, $\boldsymbol{A}$ has eigenvalues equal to ± 1 for the first 100 eigenvalues and $\pm 10^{-10}$ for the remaining 900 eigenvalues, again with random signs. This example is designed to test performance in the presence of a sharp drop in the singular values. For both plots, the eigenvectors are constructed in a 2×2 block diagonal form, $\mathrm{diag}(\boldsymbol{I}_{100}, \boldsymbol{U})$ where $\boldsymbol{I}_{100}$ is the 100×100 identity matrix and $\boldsymbol{U} \in \mathbb{R}^{900 \times 900}$ is a Haar-distributed orthogonal matrix. Both the algorithm and the theorem were constructed using the Gaussian sketch with the sketch size $1.5k$ for the algorithm, and $c_1 = 1.5$ and $c_2 = 2$ for the theorem.

In Figure 4.4, we observe that Ω_F holds whenever there is a rapid decay of eigenvalues, i.e., when $|\lambda_k| \gg |\lambda_{c_1 k+1}|$. But more importantly, we see that the bound holds when the event Ω_F occurs (circles) and often holds even if the event Ω_F did not occur (crosses). The theorem does extremely well when Ω_F has occurred. The algorithm consistently provides a robust low-rank approximation that is a modest factor worse than the best approximation given by the truncated SVD. Although the theorem appears to do better than the algorithm when Ω_F holds, it can produce unstable approximations when Ω_F fails to hold, as demonstrated in the right figure. This illustrates that the algorithm works well in practice.

4.5.1 Synthetic examples

We now compare some of the existing algorithms against Algorithm 4 using various kernel functions and synthetic datasets. We illustrate the following algorithms

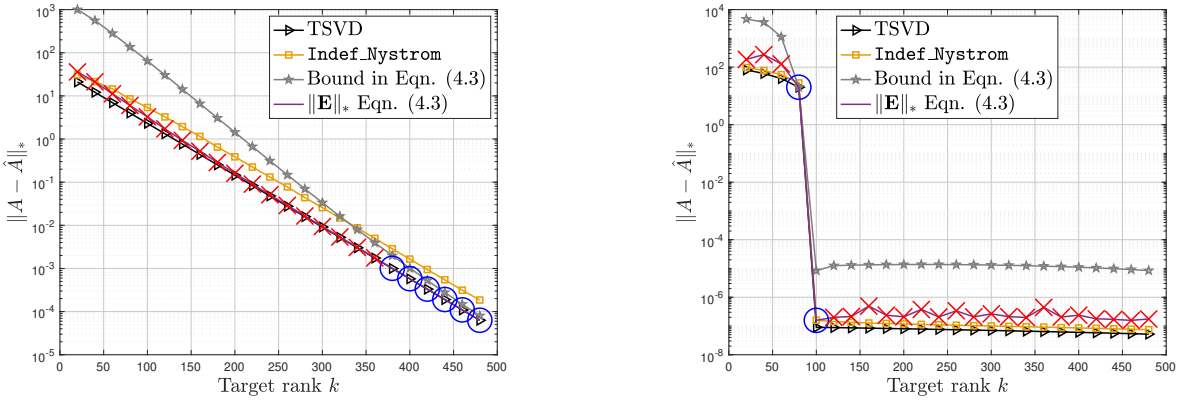


Figure 4.4: Two plots showing the empirical results for Theorem 4.4.1 and Algorithm 4. Algorithm 4 is robust with the approximation being a modest factor worse than the best approximation. Theorem 4.4.1 bound holds when Ω_F has occurred (circles on Theorem 4.4.1) and also frequently holds even if Ω_F has not occurred (crosses on Theorem 4.4.1). Theorem 4.4.1 does extremely well when Ω_F has occurred.

- Algorithm 4 with the SRTT sketch and the sketch size $s = 2k$,
- Algorithm 4 with uniform column sampling and the sketch size $s = 2k$,
- Algorithm 4 with leverage score column sampling and the sketch size $s = 2k$,
- Submatrix-Shifted (SMS) Nyström [RMMM22] with uniform column sampling and $s_1 = k$, $s_2 = 2k$ and $\alpha = 1.5$,
- Submatrix-Shifted (SMS) Nyström [RMMM22] with the Gaussian sketch and $s_1 = k$, $s_2 = 2k$ and $\alpha = 1.5$,
- Stabilised Nyström [CNX22] with the SRTT sketch, $s = k$ and $\epsilon = 10^{-14}$,

where k is the target rank and the parameters for SMS Nyström and Stabilised Nyström are as recommended in their original papers. For stabilised Nyström method, $s = k$ was chosen to ensure that all approximations in the experiment have rank at most k. For SMS Nyström method, the Gaussian sketch was not used in the original paper [RMMM22]. We use the following kernel functions

1. Epanechnikov kernel: $k_1(x, y) = \max\{1 - \|x - y\|^2, 0\}$

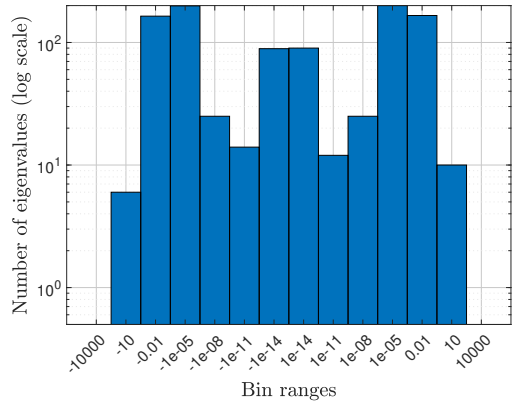
2. Multiquadric kernel: $k_2(x, y) = \sqrt{1 + \|x - y\|^2}$

3. Thin plate spline: $k_3(x, y) = \|x - y\|^2 \ln\left(\|x - y\|^2\right)$

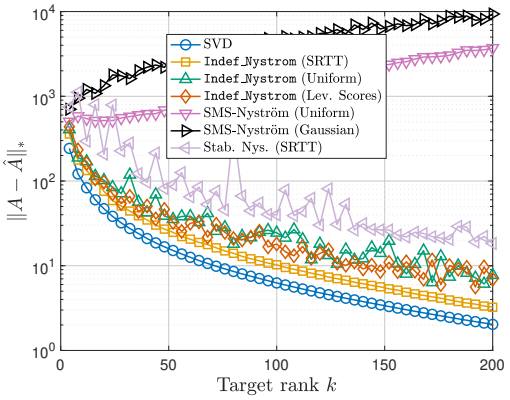
to generate the kernel matrices. The kernel matrices $\boldsymbol{K}^{(1)}, \boldsymbol{K}^{(2)}$ and $\boldsymbol{K}^{(3)}$ corresponding to the kernel functions k_1, k_2 and k_3 were generated by sampling 1000 random numbers $\{x_i\}_{i=1}^{1000}$ from the standard normal distribution, i.e., $\boldsymbol{K}_{ij}^{(\ell)} = k_\ell(x_i, x_j)$. All the kernel matrices are symmetric indefinite.

In Figure 4.5, we illustrate the results. The eigenvalue histogram is shown in the left plots. The right plots show the approximation. We see that SMS Nyström performs poorly in all 3 examples except the Gaussian case for the multiquadric kernel. This is possibly due to the presence of extreme eigenvalues with large magnitudes, where the resulting large shift degrades the quality of the low-rank approximation. The stabilised Nyström performs well for the multiquadric kernel and the thin plate spline. However, the approximation is very unstable for the Epanechnikov kernel. This instability may be attributed to the fact that, for the Epanechnikov kernel, the number of positive and negative eigenvalues are approximately equal and have similar magnitudes, which increases the likelihood of instability in the core matrix.[1] This also tells us that the truncation in the core matrix should not depend on the magnitude of the singular values of $\boldsymbol{M}$, but the truncation should always happen proportional to the target rank. Algorithm 4 using uniform column sampling and leverage score column sampling are both unstable for all 3 examples, which shows the unreliability of using column sampling matrices. On the other hand, Algorithm 4 with an SRTT works well in all cases.

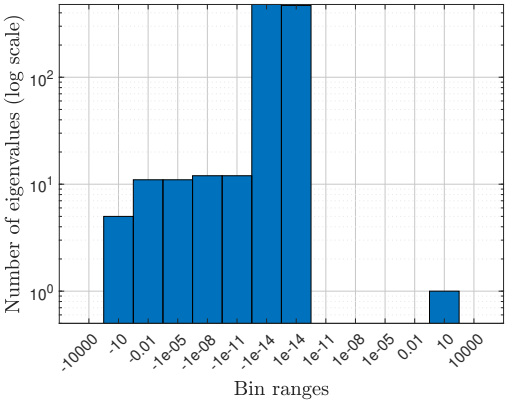
[1]To our knowledge, the numerical behaviour of stabilised Nyström method is an open problem; the stability analysis in [Nak20] applies only to the generalised Nyström method where $\boldsymbol{A}$ is sketched from both sides using independent sketches of different dimensions.



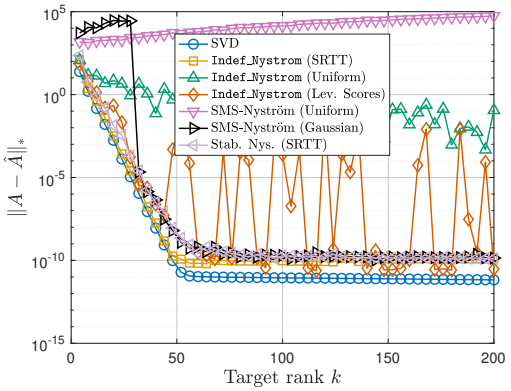
(a) Epanechnikov kernel



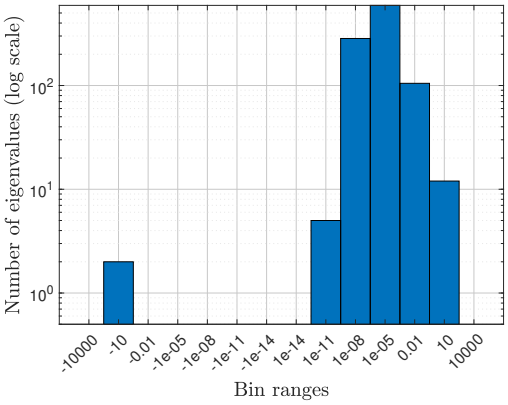
(b) Epanechnikov kernel



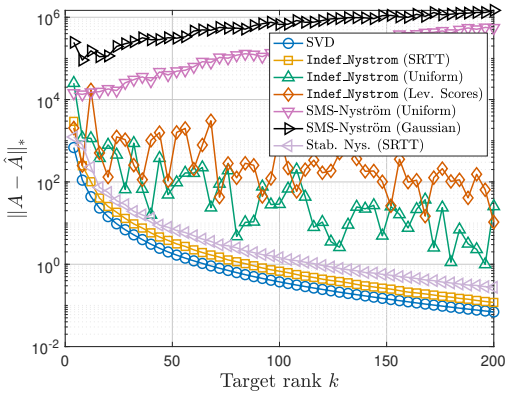
(c) Multiquadric kernel



(d) Multiquadric kernel



(e) Thin plate spline kernel



(f) Thin plate spline kernel

Figure 4.5: Comparison of different methods for symmetric indefinite matrices: SMS-Nyström [RMMM22], stabilised Nyström [CNX22] and Algorithm 4. The first two methods and Algorithm 4 using uniform column sampling and leverage score column sampling can fail on some kernels while Algorithm 4 using the SRTT sketch (random embedding) works well for all the kernels in the experiment.

4.5.2 Dataset examples

We now compare three different kernels using two different high-dimensional datasets, the Covertype and the Anuran Calls (MFCC) from the UC Irvine Machine Learning Repository [DG17]. We illustrate the following algorithms

- Algorithm 4 with an SRTT and the sketch size $s = 2k$,
- Algorithm 4 with k-means++ samples and the sketch size $s = 2k$,
- Algorithm 4 with uniform column sampling and the sketch size $s = 2k$,
- Stabilised Nyström with k-means++ samples, the sketch size $s = k$ and $\epsilon = 10^{-14}$,

where k is the target rank. We use the following kernel functions

1. Thin plate spline kernel: $\|x - y\|^2 \log\left(\|x - y\|^2\right)$,
2. Sigmoid kernel: $\tanh\left(1 + \|x - y\|^2\right)$,
3. Multiquadric kernel: $\sqrt{1 + \|x - y\|^2}$,

with the datasets

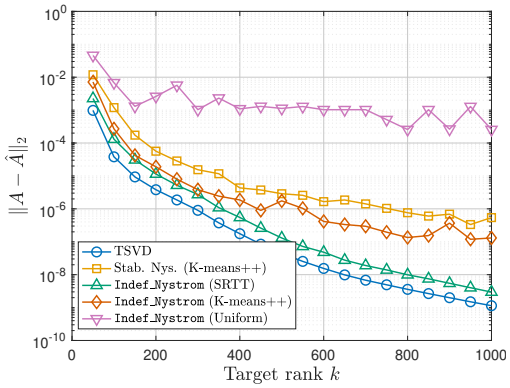
1. Covertype ($n = 581012$) with dimension $d = 54$,
2. Anuran Calls (MFCC) ($n = 7195$) with dimension $d = 22$.

For each dataset, we sample $n = 4000$ data uniformly at random and then centre the mean and normalize all features to have variance 1.

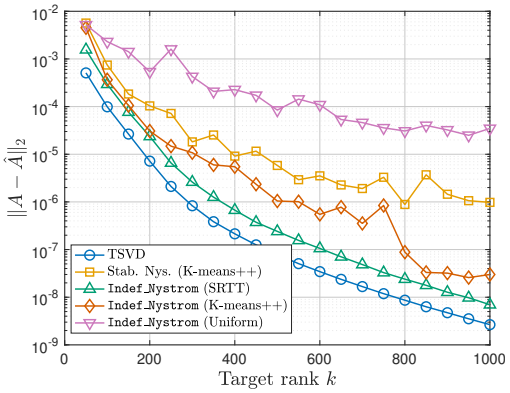
The results are illustrated in Figure 4.6. We observe that the stabilised Nyström method can still yield unstable approximations, emphasizing that the instability is not necessarily caused by very small singular values in the core matrix. Additionally, while Algorithm 4 using k-means++ sampling is generally more accurate than uniform sampling, neither approach

consistently produces robust low-rank approximations. This highlights the difficulty of designing a column sampling scheme that guarantees a stable Nyström approximation for symmetric indefinite matrices. In contrast, Algorithm 4 with an SRTT consistently delivers robust approximations throughout the experiment.

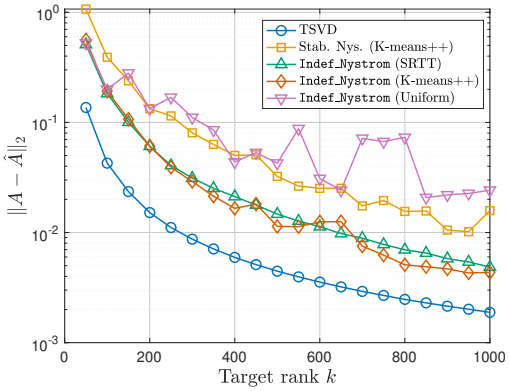
To conclude, the `Indef_Nystrom` algorithm (Algorithm 4) provides a robust Nyström approximation for symmetric indefinite matrices in practice. The algorithm relies on random embeddings. To our knowledge, it remains an open question whether column sampling matrices can also yield a similarly robust Nyström approximation.



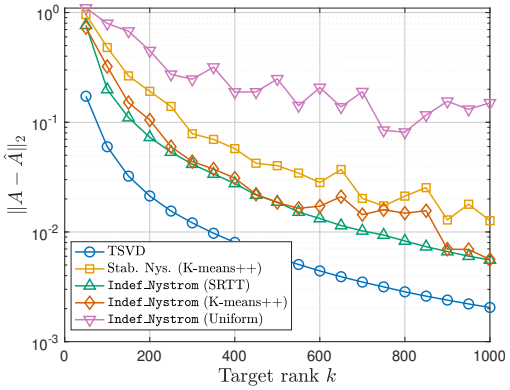
(a) Multiquadric kernel + Covertype dataset



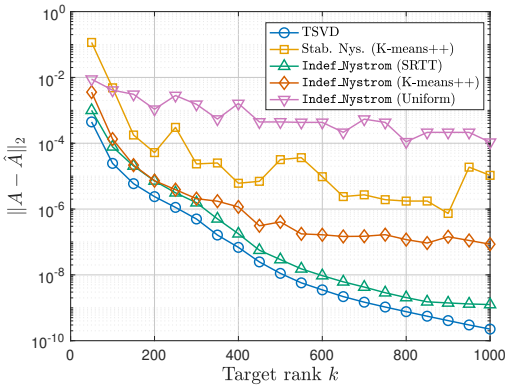
(b) Multiquadric kernel + MFCC dataset



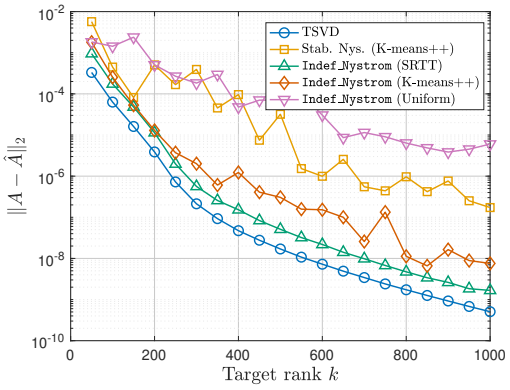
(c) Thin plate spline kernel + Covertype dataset



(d) Thin plate spline kernel + MFCC dataset



(e) Sigmoid kernel + Covertype dataset



(f) Sigmoid kernel + MFCC dataset

Figure 4.6: Comparison of stabilised Nyström method [CNX22] and Algorithm 4 for symmetric indefinite matrices using three different indefinite kernels and two different datasets. Stabilised Nyström method and Algorithm 4 using k-means++ samples and uniform column sampling can give unstable low-rank approximation while Algorithm 4 with an SRTT sketch gives robust approximation throughout the experiment.

5

Accuracy and stability of CUR decomposition

In preceding chapters, we have focused on algorithms based on random embeddings, as they tend to be more robust than column sampling matrices. In this chapter, we turn our attention to a more natural basis for low-rank approximation: the CUR decomposition, which selects a subset of rows and columns from a matrix [GTZ97, MD09, SE16, BW17]. The aim is to address the instability associated with the CUR with cross approximation (CUR-CA) formulation,

$$\boldsymbol{C}\boldsymbol{M}^{\dagger}\boldsymbol{R} = \boldsymbol{A}(:, J)\boldsymbol{A}(I, J)^{\dagger}\boldsymbol{A}(I, :),$$

where I and J denote the selected row and column indices, respectively. Indeed, the ill-conditioning of $\boldsymbol{A}(I, J)$ for CUR-CA becomes alarming when computing its matrix (pseudo)inverse, a concern that has been raised in various papers [SE16, MT20, DM23].

We show that CUR-CA can be implemented in a numerically stable manner using the ϵ-pseudoinverse in the presence of roundoff errors. This approach involves truncating the singular values of $\boldsymbol{A}(I, J)$ that are smaller than a given threshold ϵ, before computing its pseudoinverse. We denote the ϵ-pseudoinverse of the core matrix by $\boldsymbol{M}_\epsilon^\dagger$. We refer to this regularised version as the stabilised CUR-CA, abbreviated as SCUR-CA.[1] The use of ϵ-pseudoinverse is necessary in the analysis to ensure that the singular values of the core matrix, $\boldsymbol{M} = \boldsymbol{A}(I, J)$ are not too small, as such values can lead to numerical inaccuracies which cause the bound to blow up; see discussion at the end of the proof of Theorem 5.1.3.

The numerical stability of the stabilized CUR-CA, $\boldsymbol{A} \approx \boldsymbol{C}\boldsymbol{M}_\epsilon^\dagger\boldsymbol{R}$ has not been studied previously to our knowledge. However, there are some related works exploring the idea of ϵ-pseudoinverse in the core matrix. Firstly, Chiu and Demanet [CD13] study the ϵ-pseudoinverse in the context of the CUR-CA with uniform sampling and show that when the matrix is incoherent, the algorithm is observed to be empirically numerically stable. However, the paper does not include a stability analysis, and the incoherence condition imposed can be quite stringent. The authors in [CNX22, NP23] explore the ϵ-pseudoinverse in the context of the symmetric Nyström method, $\boldsymbol{A} \approx \boldsymbol{C}\boldsymbol{M}_\epsilon^\dagger\boldsymbol{C}^\top$, applied to symmetric indefinite matrices. However, the ϵ-pseudoinverse can deteriorate the accuracy of the symmetric Nyström method when applied to symmetric indefinite matrices; see Chapter 4. Lastly, Nakatsukasa [Nak20] studies the generalised Nyström method with ϵ-pseudoinverse and oversampling, which provides a numerically stable way of computing the generalised Nyström method and proves its stability. The paper also demonstrates that oversampling is necessary for a stable approximation in the generalised Nyström method.

In a related work, Hamm and Huang study the perturbation bounds of CUR decompositions in [HH21]. In [HH21], they derive perturbation bounds for CUR decompositions under the

[1]The stability analysis of the plain CUR-CA, $\boldsymbol{C}\boldsymbol{M}^\dagger\boldsymbol{R}$ is not covered in this thesis; we require the ϵ-pseudoinverse to carry out our analysis. However, we observe its numerical stability in practice; see Section 5.3.1.

influence of a noise matrix. They investigate several variants of the CUR decomposition including the ϵ-pseudoinverse variant, SCUR-CA, and provide perturbation bounds in terms of the noise matrix. This work is related, but different from our work as they investigate the accuracy of the CUR decomposition for the perturbed matrix $\tilde{\boldsymbol{A}} = \boldsymbol{A} + \boldsymbol{E}$, while we investigate the accuracy and stability for a numerical implementation of the CUR-CA in the presence of rounding errors.

5.1 Accuracy and stability of the stabilized CUR-CA

In this section, we study two topics related to the CUR-CA. We begin by analysing the accuracy of the CUR-CA,

$$\boldsymbol{A}_{IJ} = \boldsymbol{A}(:, J)\boldsymbol{A}(I, J)^{\dagger}\boldsymbol{A}(I, :) = \boldsymbol{A}\boldsymbol{\Pi}_J(\boldsymbol{\Pi}_I^{\top}\boldsymbol{A}\boldsymbol{\Pi}_J)^{\dagger}\boldsymbol{\Pi}_I^{\top}\boldsymbol{A} = \boldsymbol{C}\boldsymbol{M}^{\dagger}\boldsymbol{R},$$

and its ϵ-pseudoinverse variant, the SCUR-CA,

$$\boldsymbol{A}_{IJ}^{\epsilon} = \boldsymbol{A}(:, J)\boldsymbol{A}(I, J)_{\epsilon}^{\dagger}\boldsymbol{A}(I, :) = \boldsymbol{A}\boldsymbol{\Pi}_J(\boldsymbol{\Pi}_I^{\top}\boldsymbol{A}\boldsymbol{\Pi}_J)_{\epsilon}^{\dagger}\boldsymbol{\Pi}_I^{\top}\boldsymbol{A} = \boldsymbol{C}\boldsymbol{M}_{\epsilon}^{\dagger}\boldsymbol{R},$$

and show that the ϵ-truncation applied to the core matrix compromises the accuracy of the CUR-CA only by a factor of ϵ times its condition number; see Remark 5.1.1. We then turn to the numerical stability of SCUR-CA in the presence of roundoff errors, and demonstrate that it satisfies a similar bound in finite precision, indicating that SCUR-CA is numerically stable, provided the selected rows and columns approximate the dominant row and column spaces of $\boldsymbol{A}$ well; see Section 5.1.2.

We begin by outlining several standard assumptions from the literature [SE16, DM23], which will be used in the theorems that follow.

Specifically, the assumptions are:

Assumption 5.1.1.

1. $|I| = |J| = k \leq \operatorname{rank}(\boldsymbol{A})$ *where* k *is the target rank,*

2. $\boldsymbol{A}(I, J) \in \mathbb{R}^{k \times k}$ *is a non-singular matrix,*

3. $\boldsymbol{X}(:, J) \in \mathbb{R}^{k \times k}$ *has full row rank, where* $\boldsymbol{X}$ *is (any) row space approximator of* $\boldsymbol{A}$.[2]

We also note that since CUR-CA is connected to oblique projectors through

$$\boldsymbol{A}_{IJ} = \boldsymbol{A}\boldsymbol{\Pi}_J(\boldsymbol{\Pi}_I^\top \boldsymbol{A}\boldsymbol{\Pi}_J)^\dagger \boldsymbol{\Pi}_I^\top \boldsymbol{A} = \boldsymbol{\mathcal{P}}_{\boldsymbol{A}\boldsymbol{\Pi}_J, \boldsymbol{\Pi}_I} \boldsymbol{A} = \boldsymbol{A}\boldsymbol{\mathcal{P}}_{\boldsymbol{\Pi}_J, \boldsymbol{A}^\top \boldsymbol{\Pi}_I},$$

the properties of oblique projectors discussed in Section 2.3.1 are important. In particular, the norm of oblique projectors can be simplified as follows.

Lemma 5.1.1. *Let* $\boldsymbol{\mathcal{P}}_{\boldsymbol{X},\boldsymbol{Y}} \in \mathbb{R}^{n \times n}$ *be a projector where* $\boldsymbol{X} \in \mathbb{R}^{n \times k}$, $\boldsymbol{Y} \in \mathbb{R}^{n \times \ell}$ *and* $\boldsymbol{Y}^\top \boldsymbol{X} \in \mathbb{R}^{\ell \times k}$ *all have full column rank (so* $k \leq \ell$*). Then*

$$\|\boldsymbol{\mathcal{P}}_{\boldsymbol{X},\boldsymbol{Y}}\| = \left\|(\boldsymbol{Y}^\top \boldsymbol{Q}_{\boldsymbol{X}})^\dagger \boldsymbol{Y}^\top\right\| \tag{5.1}$$

for any unitarily invariant norm $\|\cdot\|$ *where* $\boldsymbol{Q}_{\boldsymbol{X}}$ *is an orthonormal matrix spanning the columns of* $\boldsymbol{X}$.

Proof. Let $\boldsymbol{X} = \boldsymbol{Q}_{\boldsymbol{X}} \boldsymbol{R}_{\boldsymbol{X}}$ be the thin QR decomposition of $\boldsymbol{X}$. Then

$$\|\boldsymbol{\mathcal{P}}_{\boldsymbol{X},\boldsymbol{Y}}\| = \left\|\boldsymbol{X}(\boldsymbol{Y}^\top \boldsymbol{X})^\dagger \boldsymbol{Y}^\top\right\| = \left\|\boldsymbol{Q}_{\boldsymbol{X}} \boldsymbol{R}_{\boldsymbol{X}} (\boldsymbol{Y}^\top \boldsymbol{Q}_{\boldsymbol{X}} \boldsymbol{R}_{\boldsymbol{X}})^\dagger \boldsymbol{Y}^\top\right\| = \left\|(\boldsymbol{Y}^\top \boldsymbol{Q}_{\boldsymbol{X}})^\dagger \boldsymbol{Y}^\top\right\|,$$

since $\boldsymbol{Y}^\top \boldsymbol{Q}_{\boldsymbol{X}} \in \mathbb{R}^{\ell \times k}$ has full column rank and $\boldsymbol{R}_{\boldsymbol{X}} \in \mathbb{R}^{k \times k}$ is nonsingular as $\boldsymbol{Y}^\top \boldsymbol{X}$ has full column rank so we have the relation $\left(\boldsymbol{Y}^\top \boldsymbol{Q}_{\boldsymbol{X}} \boldsymbol{R}_{\boldsymbol{X}}\right)^\dagger = \boldsymbol{R}_{\boldsymbol{X}}^{-1} \left(\boldsymbol{Y}^\top \boldsymbol{Q}_{\boldsymbol{X}}\right)^\dagger$ for the last equality. □

[2]The assumptions and the theorems in this section are stated in terms of row space approximators, but similar assumptions and theorems for column space approximators can be obtained, for example, by considering $\boldsymbol{A}^\top$ instead.

Since $\boldsymbol{A}(I, J)$ is assumed to be non-singular, $\boldsymbol{A}(:, J)$ and $\boldsymbol{A}(I, :)$ have full column and row rank, respectively. Under Assumption 5.1.1, by Lemma 5.1.1,

$$\begin{aligned}\left\|\boldsymbol{C}\boldsymbol{M}^{\dagger}\right\| &= \left\|\boldsymbol{C}(\boldsymbol{\Pi}_I^{\top}\boldsymbol{C})^{\dagger}\right\| = \left\|\boldsymbol{Q}_{\boldsymbol{C}}(I,:)^{\dagger}\right\|,\\ \left\|\boldsymbol{M}^{\dagger}\boldsymbol{R}\right\| &= \left\|(\boldsymbol{R}\boldsymbol{\Pi}_{\boldsymbol{J}})^{\dagger}\boldsymbol{R}\right\| = \left\|\boldsymbol{Q}_{\boldsymbol{R}}(J,:)^{\dagger}\right\|,\\ \left\|\boldsymbol{X}(J,:)^{\dagger}\boldsymbol{X}\right\| &= \left\|(\boldsymbol{X}\boldsymbol{\Pi}_{\boldsymbol{J}})^{\dagger}\boldsymbol{X}\right\| = \left\|\boldsymbol{Q}_{\boldsymbol{X}}(J,:)^{\dagger}\right\|\end{aligned}$$

for any unitarily invariant norm $\|\cdot\|$, where $\boldsymbol{Q}_{\boldsymbol{C}}$, $\boldsymbol{Q}_{\boldsymbol{R}}$ and $\boldsymbol{Q}_{\boldsymbol{X}}$ are the orthonormal matrices spanning the columns of $\boldsymbol{C}$, $\boldsymbol{R}^{\top}$ and $\boldsymbol{X}^{\top}$, respectively. We now prove the accuracy of the CUR-CA and its ϵ-pseudoinverse variant, the SCUR-CA.

5.1.1 Accuracy of CUR and its ϵ-pseudoinverse variant

We prove the accuracy of the SCUR-CA, $\boldsymbol{A}^{\epsilon}_{IJ}$ first. The accuracy for the CUR-CA, $\boldsymbol{A}_{IJ}$ follows by setting $\epsilon = 0$. The analysis presented in this section plays an essential role for the stability analysis as we show that the SCUR-CA satisfies a similar bound under roundoff errors, establishing its numerical stability.

Theorem 5.1.1. *Let $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ be a matrix, I and J be a set of row and column indices, respectively, with $|I| = |J| = k$, $\epsilon > 0$ and $\boldsymbol{X} \in \mathbb{R}^{k\times n}$ be any row space approximator of $\boldsymbol{A}$. Then under Assumption 5.1.1,*

$$\left\|\boldsymbol{A} - \boldsymbol{A}^{\epsilon}_{I\cup I_0,J}\right\| \leq \left\|\boldsymbol{Q}_{\boldsymbol{C}}(I\cup I_0,:)^{\dagger}\right\|_2 \left\|\boldsymbol{Q}_{\boldsymbol{X}}(J,:)^{-1}\right\|_2 \left(\left\|\boldsymbol{A} - \boldsymbol{A}\boldsymbol{X}^{\dagger}\boldsymbol{X}\right\| + \|\boldsymbol{E}\|\right) \tag{5.2}$$

for any unitarily invariant norm $\|\cdot\|$ where I_0 is a set of extra row indices distinct from I with $|I_0| = p$, and $\boldsymbol{E} \in \mathbb{R}^{k_\epsilon\times k_\epsilon}$ is a matrix satisfying $\|\boldsymbol{E}\|_2 \leq \epsilon$ where $k_\epsilon \leq k$ is the number of singular values of $\boldsymbol{A}(I\cup I_0, J)$ smaller than ϵ.

Proof. For shorthand, let $I_* := I \cup I_0$. Let the thin SVD of $\boldsymbol{A}(I_*, J) \in \mathbb{R}^{(k+p)\times k}$ be $\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^{\top} =$

$[\boldsymbol{U}_1, \boldsymbol{U}_2]\,\mathrm{diag}(\boldsymbol{\Sigma}_1, \boldsymbol{\Sigma}_2)[\boldsymbol{V}_1, \boldsymbol{V}_2]^\top$ where $\boldsymbol{\Sigma}_2$ contains the singular values of $\boldsymbol{A}(I_*, J)$ smaller than ϵ. Then

$$\begin{aligned}\boldsymbol{A}^\epsilon_{I_* J}\boldsymbol{\Pi}_J &= \boldsymbol{A}\boldsymbol{\Pi}_J(\boldsymbol{\Pi}_{I_*}^\top \boldsymbol{A}\boldsymbol{\Pi}_J)^\dagger_\epsilon \boldsymbol{\Pi}_{I_*}^\top \boldsymbol{A}\boldsymbol{\Pi}_J = \boldsymbol{A}\boldsymbol{\Pi}_J \boldsymbol{V}_1 \boldsymbol{\Sigma}_1^{-1}\boldsymbol{U}_1^\top \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top \\ &= \boldsymbol{A}\boldsymbol{\Pi}_J \boldsymbol{V}_1\boldsymbol{V}_1^\top = \boldsymbol{A}\boldsymbol{\Pi}_J - \boldsymbol{A}\boldsymbol{\Pi}_J\boldsymbol{V}_2\boldsymbol{V}_2^\top.\end{aligned}$$

Therefore, using the above equation, we obtain

$$\begin{aligned}\boldsymbol{A} - \boldsymbol{A}^\epsilon_{I_* J} &= \left(\boldsymbol{I} - \boldsymbol{A}\boldsymbol{\Pi}_J\left(\boldsymbol{\Pi}_{I_*}^\top \boldsymbol{A}\boldsymbol{\Pi}_J\right)^\dagger_\epsilon \boldsymbol{\Pi}_{I_*}^\top\right)\boldsymbol{A} \\ &= (\boldsymbol{I} - \boldsymbol{A}\boldsymbol{\Pi}_J(\boldsymbol{\Pi}_{I_*}^\top \boldsymbol{A}\boldsymbol{\Pi}_J)^\dagger_\epsilon \boldsymbol{\Pi}_{I_*}^\top)\boldsymbol{A}(\boldsymbol{I} - \boldsymbol{\Pi}_J(\boldsymbol{X}\boldsymbol{\Pi}_J)^\dagger \boldsymbol{X}) + \boldsymbol{A}\boldsymbol{\Pi}_J\boldsymbol{V}_2\boldsymbol{V}_2^\top(\boldsymbol{X}\boldsymbol{\Pi}_J)^\dagger\boldsymbol{X}.\end{aligned} \tag{5.3}$$

Note that $\boldsymbol{\mathcal{P}}^\epsilon_{\boldsymbol{A}\boldsymbol{\Pi}_J, \boldsymbol{\Pi}_{I_*}} := \boldsymbol{A}\boldsymbol{\Pi}_J(\boldsymbol{\Pi}_{I_*}^\top \boldsymbol{A}\boldsymbol{\Pi}_J)^\dagger_\epsilon \boldsymbol{\Pi}_{I_*}^\top$ is an oblique projector since

$$\begin{aligned}\left(\boldsymbol{\mathcal{P}}^\epsilon_{\boldsymbol{A}\boldsymbol{\Pi}_J, \boldsymbol{\Pi}_{I_*}}\right)^2 &= \boldsymbol{A}\boldsymbol{\Pi}_J\boldsymbol{V}_1\boldsymbol{\Sigma}_1^{-1}\boldsymbol{U}_1^\top\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top\boldsymbol{V}_1\boldsymbol{\Sigma}_1^{-1}\boldsymbol{U}_1^\top\boldsymbol{\Pi}_{I_*}^\top \\ &= \boldsymbol{A}\boldsymbol{\Pi}_J\boldsymbol{V}_1\boldsymbol{\Sigma}_1^{-1}\boldsymbol{U}_1^\top\boldsymbol{\Pi}_{I_*}^\top \\ &= \boldsymbol{\mathcal{P}}^\epsilon_{\boldsymbol{A}\boldsymbol{\Pi}_J, \boldsymbol{\Pi}_{I_*}}\end{aligned}$$

and similarly, $\boldsymbol{\mathcal{P}}_{\boldsymbol{\Pi}_J, \boldsymbol{X}^\top} = \boldsymbol{\Pi}_J(\boldsymbol{X}\boldsymbol{\Pi}_J)^\dagger\boldsymbol{X}$ is an oblique projector. Now bounding the first term of Equation (5.3) gives

$$\begin{aligned}\left\|(\boldsymbol{I} - \boldsymbol{\mathcal{P}}^\epsilon_{\boldsymbol{A}\boldsymbol{\Pi}_J, \boldsymbol{\Pi}_{I_*}})\boldsymbol{A}(\boldsymbol{I} - \boldsymbol{\mathcal{P}}_{\boldsymbol{\Pi}_J, \boldsymbol{X}^\top})\right\| &\le \left\|\boldsymbol{I} - \boldsymbol{\mathcal{P}}^\epsilon_{\boldsymbol{A}\boldsymbol{\Pi}_J, \boldsymbol{\Pi}_{I_*}}\right\|_2 \left\|\boldsymbol{A}(\boldsymbol{I} - \boldsymbol{\mathcal{P}}_{\boldsymbol{\Pi}_J, \boldsymbol{X}^\top})\right\| \\ &= \left\|\boldsymbol{\mathcal{P}}^\epsilon_{\boldsymbol{A}\boldsymbol{\Pi}_J, \boldsymbol{\Pi}_{I_*}}\right\|_2 \left\|\boldsymbol{A}(\boldsymbol{I} - \boldsymbol{X}^\dagger\boldsymbol{X})(\boldsymbol{I} - \boldsymbol{\mathcal{P}}_{\boldsymbol{\Pi}_J, \boldsymbol{X}^\top})\right\| \\ &\le \left\|\boldsymbol{\mathcal{P}}^\epsilon_{\boldsymbol{A}\boldsymbol{\Pi}_J, \boldsymbol{\Pi}_{I_*}}\right\|_2 \left\|\boldsymbol{A}(\boldsymbol{I} - \boldsymbol{X}^\dagger\boldsymbol{X})\right\| \left\|\boldsymbol{I} - \boldsymbol{\mathcal{P}}_{\boldsymbol{\Pi}_J, \boldsymbol{X}^\top}\right\|_2 \\ &= \left\|\boldsymbol{\mathcal{P}}^\epsilon_{\boldsymbol{A}\boldsymbol{\Pi}_J, \boldsymbol{\Pi}_{I_*}}\right\|_2 \left\|\boldsymbol{\mathcal{P}}_{\boldsymbol{\Pi}_J, \boldsymbol{X}^\top}\right\|_2 \left\|\boldsymbol{A}(\boldsymbol{I} - \boldsymbol{X}^\dagger\boldsymbol{X})\right\|\end{aligned}$$

where in the first equality we used $\boldsymbol{X}\boldsymbol{\mathcal{P}}_{\boldsymbol{\Pi}_J,\boldsymbol{X}^\top} = \boldsymbol{X}\boldsymbol{\Pi}_J(\boldsymbol{X}\boldsymbol{\Pi}_J)^\dagger\boldsymbol{X} = \boldsymbol{X}$, as $\boldsymbol{X}\boldsymbol{\Pi}_J = \boldsymbol{X}(:,J)$ has full row rank by Assumption 5.1.1. By letting $\boldsymbol{A}\boldsymbol{\Pi}_J = \boldsymbol{Q_C}\boldsymbol{R_C}$ be the thin QR decomposition and noting that $\boldsymbol{\Pi}_{I_*}^\top\boldsymbol{Q_C}$ has full column rank so $(\boldsymbol{\Pi}_{I_*}^\top\boldsymbol{Q_C})^\dagger(\boldsymbol{\Pi}_{I_*}^\top\boldsymbol{Q_C}) = \boldsymbol{I}_k$, the first term $\left\|\boldsymbol{\mathcal{P}}^{\epsilon}_{\boldsymbol{A}\boldsymbol{\Pi}_J,\boldsymbol{\Pi}_{I_*}}\right\|_2$ in the final expression can be bounded as

$$\begin{aligned}
\left\|\boldsymbol{\mathcal{P}}^{\epsilon}_{\boldsymbol{A}\boldsymbol{\Pi}_J,\boldsymbol{\Pi}_{I_*}}\right\|_2 &= \left\|\boldsymbol{Q_C}\boldsymbol{R_C}(\boldsymbol{\Pi}_{I_*}^\top\boldsymbol{A}\boldsymbol{\Pi}_J)^\dagger_\epsilon\right\|_2 = \left\|\boldsymbol{R_C}(\boldsymbol{\Pi}_{I_*}^\top\boldsymbol{A}\boldsymbol{\Pi}_J)^\dagger_\epsilon\right\|_2 \\
&= \left\|(\boldsymbol{\Pi}_{I_*}^\top\boldsymbol{Q_C})^\dagger\boldsymbol{\Pi}_{I_*}^\top\boldsymbol{Q_C}\boldsymbol{R_C}(\boldsymbol{\Pi}_{I_*}^\top\boldsymbol{A}\boldsymbol{\Pi}_J)^\dagger_\epsilon\right\|_2 \\
&\leq \left\|(\boldsymbol{\Pi}_{I_*}^\top\boldsymbol{Q_C})^\dagger\right\|_2\left\|\boldsymbol{\Pi}_{I_*}^\top\boldsymbol{A}\boldsymbol{\Pi}_J(\boldsymbol{\Pi}_{I_*}^\top\boldsymbol{A}\boldsymbol{\Pi}_J)^\dagger_\epsilon\right\|_2 \\
&\leq \left\|(\boldsymbol{\Pi}_{I_*}^\top\boldsymbol{Q_C})^\dagger\right\|_2.
\end{aligned}$$

Here, the final inequality is an equality as long as $(\boldsymbol{\Pi}_{I_*}^\top\boldsymbol{A}\boldsymbol{\Pi}_J)^\dagger_\epsilon \neq \boldsymbol{0}$. Therefore, using Lemma 5.1.1 on $\boldsymbol{\mathcal{P}}_{\boldsymbol{\Pi}_J,\boldsymbol{X}^\top}$, the first term in Equation (5.3) can be bounded as

$$\left\|(\boldsymbol{I} - \boldsymbol{\mathcal{P}}^{\epsilon}_{\boldsymbol{A}\boldsymbol{\Pi}_J,\boldsymbol{\Pi}_{I_*}})\boldsymbol{A}(\boldsymbol{I} - \boldsymbol{\mathcal{P}}_{\boldsymbol{\Pi}_J,\boldsymbol{X}^\top})\right\| \leq \left\|\boldsymbol{Q_C}(I_*,:)^\dagger\right\|_2\left\|\boldsymbol{Q_X}(J,:)^{-1}\right\|_2\left\|\boldsymbol{A}(\boldsymbol{I} - \boldsymbol{X}^\dagger\boldsymbol{X})\right\|.$$

The second term in Equation (5.3) can be bounded using a similar argument as

$$\begin{aligned}
\left\|\boldsymbol{A}\boldsymbol{\Pi}_J\boldsymbol{V}_2\boldsymbol{V}_2^\top(\boldsymbol{X}\boldsymbol{\Pi}_J)^\dagger\boldsymbol{X}\right\| &= \left\|\boldsymbol{Q_C}\boldsymbol{R_C}\boldsymbol{V}_2\boldsymbol{V}_2^\top(\boldsymbol{X}\boldsymbol{\Pi}_J)^\dagger\boldsymbol{X}\right\| = \left\|\boldsymbol{R_C}\boldsymbol{V}_2\boldsymbol{V}_2^\top(\boldsymbol{X}\boldsymbol{\Pi}_J)^\dagger\boldsymbol{X}\right\| \\
&= \left\|(\boldsymbol{\Pi}_{I_*}^\top\boldsymbol{Q_C})^\dagger\boldsymbol{\Pi}_{I_*}^\top\boldsymbol{Q_C}\boldsymbol{R_C}\boldsymbol{V}_2\boldsymbol{V}_2^\top(\boldsymbol{X}\boldsymbol{\Pi}_J)^\dagger\boldsymbol{X}\right\| \\
&= \left\|(\boldsymbol{\Pi}_{I_*}^\top\boldsymbol{Q_C})^\dagger\boldsymbol{U}_2\boldsymbol{\Sigma}_2\boldsymbol{V}_2^\top(\boldsymbol{X}\boldsymbol{\Pi}_J)^\dagger\boldsymbol{X}\right\| \\
&\leq \left\|(\boldsymbol{\Pi}_{I_*}^\top\boldsymbol{Q_C})^\dagger\right\|_2\left\|\boldsymbol{U}_2\boldsymbol{\Sigma}_2\boldsymbol{V}_2^\top\right\|\left\|(\boldsymbol{X}\boldsymbol{\Pi}_J)^\dagger\boldsymbol{X}\right\|_2 \\
&= \left\|\boldsymbol{Q_C}(I_*,:)^\dagger\right\|_2\left\|\boldsymbol{Q_X}(J,:)^{-1}\right\|_2\|\boldsymbol{\Sigma}_2\|.
\end{aligned}$$

Putting everything together and letting $\boldsymbol{E} = \boldsymbol{\Sigma}_2$, we get the desired result. □

As a corollary, we can obtain a bound for the standard CUR-CA. The corollary is stated below, which is simply a special case of Theorem 5.1.1 when $\epsilon = 0$.

Corollary 5.1.1. *Under the same assumptions as in Theorem 5.1.1,*

$$\|\boldsymbol{A} - \boldsymbol{A}_{I\cup I_0,J}\| \leq \left\|\boldsymbol{Q}_{\boldsymbol{C}}(I \cup I_0, :)^\dagger\right\|_2 \left\|\boldsymbol{Q}_{\boldsymbol{X}}(J, :)^{-1}\right\|_2 \left\|\boldsymbol{A} - \boldsymbol{A}\boldsymbol{X}^\dagger\boldsymbol{X}\right\|$$

for any unitarily invariant norm $\|\cdot\|$.

Proof. Set $\epsilon = 0$ in Theorem 5.1.1. □

Remark 5.1.1.

1. *The bound for the CUR-CA depends on the quantity*

$$C_{\boldsymbol{X},I\cup I_0,J} = \left\|\boldsymbol{Q}_{\boldsymbol{C}}(I \cup I_0, :)^\dagger\right\|_2 \left\|\boldsymbol{Q}_{\boldsymbol{X}}(J, :)^{-1}\right\|_2,$$

   as indicated by Equation (5.2)*. Theorem 5.1.1 tells us that the SCUR-CA,* $\boldsymbol{A}^{\epsilon}_{I_*J}$ *is worse than the CUR-CA by at most* $C_{\boldsymbol{X},I\cup I_0,J}\sqrt{k}\epsilon$ *in the Frobenius norm. Therefore, the bound for SCUR-CA is similar to the bound for CUR-CA for small enough* ϵ*.*

2. *The quantity defined in the first remark,* $C_{\boldsymbol{X},I\cup I_0,J}$*, depends on the choice of the row space approximator, and the column and row indices. Therefore, for a given sets of indices* $I \cup I_0$ *and* J*, there are cases when the optimal choice of* $\boldsymbol{X}$*, i.e.,* k*-dominant right singular vectors gives non-informative* $+\infty$ *bound, whereas a suboptimal choice of* $\boldsymbol{X}$ *may give an informative bound. For example, take a matrix with* $[\boldsymbol{I}_k, \boldsymbol{0}]^\top$ *as the* k *dominant right singular vectors with very slowly decaying singular values,* $\boldsymbol{X} = [\boldsymbol{0}, \boldsymbol{I}_k, \boldsymbol{0}]$ *with the identity matrix covering the* $(k+1)$*st to* $(2k)$*th column, and* $J = \{k+1, ..., 2k\}$*.*

3. *The bound in Theorem 5.1.1 and Corollary 5.1.1 has two factors involving the (pseudo)inverse, which is in contrast to the CUR with best approximation (CUR-BA),* $\boldsymbol{C}(\boldsymbol{C}^\dagger\boldsymbol{A}\boldsymbol{R}^\dagger)\boldsymbol{R}$ *having only one factor; see the analysis Lemma 4.2 of [SE16] or in Section 6.4. This makes the CUR-CA,* $\boldsymbol{A}_{IJ}$ *usually worse than the CUR-BA, which is expected as the*

CUR-CA is cheaper to compute. However, the CUR-CA can still be very accurate; see for example [ZO18, CK20], which establishes the existence of a rank-k CUR-CA that has error within a factor $1 + k$ *of the best rank-k approximation via the truncated SVD. The second multiplicative factor in Theorem 5.1.1 and Corollary 5.1.1 comes from the fact that* $\boldsymbol{A}_{IJ}$ *is associated with the oblique projector* $\boldsymbol{\mathcal{P}}_{\boldsymbol{A}\boldsymbol{\Pi}_J,\boldsymbol{\Pi}_I}$ *rather than the orthogonal projectors* $\boldsymbol{C}\boldsymbol{C}^\dagger$ *and* $\boldsymbol{R}^\dagger\boldsymbol{R}$ *for the CUR-BA. Nonetheless, the CUR-CA and the CUR-BA have comparable accuracy when employed with good row and column indices, with the CUR-CA being much more computationally efficient.*

4. *The row space approximator* $\boldsymbol{X} \in \mathbb{R}^{k\times n}$ *can be chosen in various ways, for example, the k-dominant right singular vectors of* $\boldsymbol{A}$ *or the row sketch of* $\boldsymbol{A}$, $\boldsymbol{S}^\top\boldsymbol{A}$. *The bounds in Theorem 5.1.1 and Corollary 5.1.1 can both be computed a posteriori when* $\boldsymbol{X}$ *can be computed easily. The k-dominant right singular vectors would make the right-most term,* $\left\|\boldsymbol{A} - \boldsymbol{A}\boldsymbol{X}^\dagger\boldsymbol{X}\right\|$ *in the bound of Theorem 5.1.1 and Corollary 5.1.1, optimal. However, the singular vectors are often too expensive to compute.*

Theorem 5.1.1 and Corollary 5.1.1 provide a bound for SCUR-CA and CUR-CA, respectively. The additional set of row indices I_0 serves as oversampling indices, making the total number of selected rows larger than the selected columns. With oversampling, the bound involves the pseudoinverse $\left\|\boldsymbol{Q}_{\boldsymbol{C}}(I \cup I_0, :)^\dagger\right\|_2$ rather than the matrix inverse $\|\boldsymbol{Q}_{\boldsymbol{C}}(I, :)^{-1}\|_2$, the former being always smaller. The concept of oversampling will be revisited in more detail in Chapter 6, where we demonstrate its advantages for both CUR-BA and CUR-CA.

Moreover, since $\left\|\boldsymbol{Q}_{\boldsymbol{C}}(I \cup I_0, :)^\dagger\right\|_2 = \left\|\boldsymbol{A}(:, J)\boldsymbol{A}(I_*, J)^\dagger\right\|_2$, and our aim is to minimise this quantity, the row indices I and I_0 should ideally be chosen based on the already-chosen columns of $\boldsymbol{A}$. This strategy has been proposed and applied in various works, including [VM17, DM23, Xia24, CY25]; see Algorithm 5 for example. One key benefit of this approach is that it typically results in a better-conditioned core matrix $\boldsymbol{A}(I_*, J)$, thereby improving

the accuracy of the CUR-CA. For example, consider the following 2×2 matrix,

$$\boldsymbol{A} = \begin{bmatrix} \epsilon & 1 \\ 1 & 0 \end{bmatrix}$$

where $0 < \epsilon < 1$. For a rank-1 approximation of $\boldsymbol{A}$, we need to choose a column and a row for the CUR-CA. If we choose a column and a row separately in the best possible way, we would choose the first row and the first column, giving us

$$\boldsymbol{A}_{1,1} = \begin{bmatrix} \epsilon \\ 1 \end{bmatrix} \epsilon^{-1} \begin{bmatrix} \epsilon & 1 \end{bmatrix} = \begin{bmatrix} \epsilon & 1 \\ 1 & 1/\epsilon \end{bmatrix},$$

which is a poor approximation as $\|\boldsymbol{A} - \boldsymbol{A}_{1,1}\|_{\mathrm{F}} = 1/\epsilon$ can be arbitrary large[3] as $\epsilon \to 0$. On the other hand, if we choose a column first and then a row, we choose the first column $[\epsilon, 1]^\top$ and the second row as $\epsilon < 1$, giving us

$$\boldsymbol{A}_{2,1} = \begin{bmatrix} \epsilon \\ 1 \end{bmatrix} 1^{-1} \begin{bmatrix} 1 & 0 \end{bmatrix} = \begin{bmatrix} \epsilon & 0 \\ 1 & 0 \end{bmatrix},$$

which is a reasonable approximation as $\|\boldsymbol{A} - \boldsymbol{A}_{2,1}\|_{\mathrm{F}} = 1 \approx \sigma_2(\boldsymbol{A})\,(\approx 1 - \epsilon/2)$.

The significance of controlling the $\|\boldsymbol{Q}_{\boldsymbol{C}}(I,:)^{-1}\|_2$ term will also be highlighted when we analyse the numerical stability of the CUR-CA in the subsequent section.

5.1.2 Numerical Stability of the CUR-CA

In the absence of rounding errors, the error for the CUR-CA can be bounded by Theorem 5.1.1 and Corollary 5.1.1. In this section, we derive an error bound that accounts for rounding errors.

[3]The accuracy of CUR-BA, by contrast, is good as long as the chosen columns and rows are good approximators for the range and co-range of $\boldsymbol{A}$ [DM23, Remark 1].

We use the standard model of floating-point arithmetic as in [Hig02, Section 2.2]:

$$fl(x \text{ op } y) = (x \text{ op } y)(1+\delta),\ |\delta| \le u$$

where op $\in \{+, -, *, /\}$ is the basic arithmetic operations and $u \ll 1$, the unit roundoff, is the precision at which the computations are performed. We use $fl(\cdot)$ and $\widehat{\cdot}$ to denote the computed value of the expression. We define $\gamma := p(m,n,k)u$ where $p(m,n,k)$ is a low-degree polynomial in m, n and k, and use γ to suppress any constant factors and terms related to the size of the matrix and the target rank, e.g., $\sqrt{m}, \sqrt{n}$ and k, but not $\sigma_i(\boldsymbol{A})$ or $1/\epsilon$. While this may appear as an oversimplication, this approach is standard practice in stability analysis; see e.g. [NH12, Nak20]. We denote the ith row of a matrix $\boldsymbol{B}$ as $[\boldsymbol{B}]_i$ and use oversampling for the row indices $I_* = I \cup I_0$ as in Theorem 5.1.1. Lastly, we assume that the truncation parameter ϵ satisfies $\|\boldsymbol{A}\|_2 \gg \epsilon > \gamma \|\boldsymbol{A}\|_2$.

We begin by stating two lemmas that will be used in the CUR-CA stability analysis. Lemma 5.1.2 proves a perturbation bound for the projector $\boldsymbol{C}\boldsymbol{M}_\epsilon^\dagger\boldsymbol{\Pi}_J$ when only $\boldsymbol{C}$ and $\boldsymbol{M}$ get perturbed and in Lemma 5.1.3, we prove that under small additive perturbation on $\boldsymbol{C}$ and $\boldsymbol{M}$, $(\boldsymbol{C}+\boldsymbol{\Delta C})(\boldsymbol{M}+\boldsymbol{\Delta M})_\epsilon^\dagger\boldsymbol{R}\boldsymbol{\Pi}_J$ approximately projects $\boldsymbol{C}$ onto itself. The proofs for the two lemmas can be found in Section 5.4.

Lemma 5.1.2. *Under Assumption 5.1.1, for any $\boldsymbol{\Delta C}$ and $\boldsymbol{\Delta M}$,*

$$\left\|(\boldsymbol{C}+\boldsymbol{\Delta C})(\boldsymbol{M}+\boldsymbol{\Delta M})_\epsilon^\dagger\right\|_2 \le \left\|\boldsymbol{Q}_{\boldsymbol{C}}(I_*,:)^\dagger\right\|_2 \left(1 + \frac{1}{\epsilon}\|\boldsymbol{\Delta M}\|_2\right) + \frac{1}{\epsilon}\|\boldsymbol{\Delta C}\|_2 .$$

Lemma 5.1.3. *Under Assumption 5.1.1, for any $\boldsymbol{\Delta C}$ and $\boldsymbol{\Delta M}$,*

$$(\boldsymbol{C}+\boldsymbol{\Delta C})(\boldsymbol{M}+\boldsymbol{\Delta M})_\epsilon^\dagger\boldsymbol{R}\boldsymbol{\Pi}_J = \boldsymbol{C} + \boldsymbol{E}_*$$

where

$$\|\boldsymbol{E}_*\|_2 \leq \left\|(\boldsymbol{Q}_{\boldsymbol{C}}(I_*,:)^\dagger\right\|_2 \left(\epsilon + 2\|\boldsymbol{\Delta M}\|_2 + \frac{1}{\epsilon}\|\boldsymbol{\Delta M}\|_2^2\right) + \|\boldsymbol{\Delta C}\|_2 \left(1 + \frac{\|\boldsymbol{\Delta M}\|_2}{\epsilon}\right).$$

We now begin with the stability analysis. Note that the stability depends on the specific implementation used. In the forthcoming analysis, we assume the SCUR-CA, $\boldsymbol{A}^{\epsilon}_{I_*J}$ is computed as follows.

1. Compute the factors $\hat{\boldsymbol{C}} = fl(\boldsymbol{A}(:,J)), \hat{\boldsymbol{M}} = fl(\boldsymbol{A}(I_*,J))$ and $\hat{\boldsymbol{R}} = fl(\boldsymbol{A}(I_*,:))$.

2. Solve the (rank-deficient) underdetermined linear systems,

$$\hat{\boldsymbol{s}}_i^{(1)} = fl\left(\left(\hat{\boldsymbol{M}}^\top\right)^\dagger_\epsilon \left[\hat{\boldsymbol{C}}\right]_i^\top\right) \in \mathbb{R}^{k+p}$$

   for all $i \in [m]$, where $\left[\hat{\boldsymbol{C}}\right]_i$ is the ith row of $\hat{\boldsymbol{C}}$.

3. Compute matrix-vector multiply $\hat{\boldsymbol{s}}_i^{(2)} = fl\left(\hat{\boldsymbol{R}}^\top \hat{\boldsymbol{s}}_i^{(1)}\right) \in \mathbb{R}^n$.

4. Let $\left(\hat{\boldsymbol{s}}_i^{(2)}\right)^\top$ be the ith row of the computed CUR decomposition $fl\left(\boldsymbol{A}^{\epsilon}_{I_*J}\right)$.

Note that the stability crucially depends on the implementation. Other implementations are possible, an obvious one being one that computes $\boldsymbol{C}(\boldsymbol{M}^\dagger\boldsymbol{R})$ rather than $(\boldsymbol{C}\boldsymbol{M}^\dagger)\boldsymbol{R}$ as done here. This is seen to work well too, although we do not have a proof.

We now analyse each step. The first step is matrix-matrix multiplications with orthonormal matrices, $\boldsymbol{\Pi}_{I_*}$ and $\boldsymbol{\Pi}_J$. Using the forward error bound[4] for matrix-matrix multiplication [Hig02, Section 3.5], we obtain the following:

- $\hat{\boldsymbol{C}} = \boldsymbol{C} + \boldsymbol{E}_{\boldsymbol{C}}$ where $\|\boldsymbol{E}_{\boldsymbol{C}}\|_2 \leq \gamma\|\boldsymbol{A}\|_2$,
- $\hat{\boldsymbol{M}} = \boldsymbol{M} + \boldsymbol{E}_{\boldsymbol{M}}$ where $\|\boldsymbol{E}_{\boldsymbol{M}}\|_2 \leq \gamma\|\boldsymbol{A}\|_2$,

[4]The error is termed the forward error as it indicates how close the computed version $\hat{\boldsymbol{C}}$ is to the exact version $\boldsymbol{C}$ [Hig02, Section 1.5].

- $\hat{\boldsymbol{R}} = \boldsymbol{R} + \boldsymbol{E}_{\boldsymbol{R}}$ where $\|\boldsymbol{E}_{\boldsymbol{R}}\|_2 \leq \gamma \|\boldsymbol{A}\|_2$.

In many cases, these error matrices $\boldsymbol{E}_{\boldsymbol{C}}, \boldsymbol{E}_{\boldsymbol{M}}$, and $\boldsymbol{E}_{\boldsymbol{R}}$ are the zero matrix as $\boldsymbol{C}, \boldsymbol{M}$, and $\boldsymbol{R}$ are simply submatrices of the original matrix $\boldsymbol{A}$.

In the second step, we solve the (rank-deficient) underdetermined linear systems row by row. The error analysis for the (rank-deficient) underdetermined linear systems can be summarized in the following theorem. The proof of Theorem 5.1.2 can be found in Section 5.5.

Theorem 5.1.2. *Consider the (rank-deficient) underdetermined linear system,*

$$\min_{\boldsymbol{x}'} \|\boldsymbol{B}_\epsilon \boldsymbol{x}' - \boldsymbol{b}\|_2 \tag{5.4}$$

where $\boldsymbol{B}_\epsilon \in \mathbb{R}^{m \times n}$ $(m \leq n)$ *is (possibly) rank-deficient* $(\text{rank}(\boldsymbol{B}_\epsilon) \leq m)$ *with singular vales larger than* ϵ *and* $\boldsymbol{b} \in \mathbb{R}^m$*. Then assuming* $\epsilon > \gamma \|\boldsymbol{B}_\epsilon\|_2$*, the minimum norm solution to Equation* (5.4) *can be computed in a backward stable manner, i.e., the computed solution* $\hat{\boldsymbol{s}}$ *satisfies*

$$\hat{\boldsymbol{s}} = (\boldsymbol{B}_\epsilon + \boldsymbol{E}_1)^\dagger (\boldsymbol{b} + \boldsymbol{E}_2)$$

where $\|\boldsymbol{E}_1\|_2 \leq \gamma \|\boldsymbol{B}_\epsilon\|_2$ *and* $\|\boldsymbol{E}_2\|_2 \leq \gamma \|\boldsymbol{b}\|_2$.

Theorem 5.1.2 tells us that the computed solution to a (rank-deficient) underdetermined linear system is the exact solution to a slightly perturbed problem. Now, using Theorem 5.1.2, for each $i \in [m]$ we obtain

$$\hat{\boldsymbol{s}}_i^{(1)} = \left(\left(\hat{\boldsymbol{M}}^\top\right)_\epsilon + \boldsymbol{E}_i^{(\boldsymbol{M})}\right)^\dagger \left([\hat{\boldsymbol{C}}]_i^\top + \boldsymbol{E}_i^{(\boldsymbol{C})}\right),$$

where $\left\|\boldsymbol{E}_i^{(\boldsymbol{M})}\right\|_2 \leq \gamma \|\boldsymbol{A}\|_2$ and $\left\|\boldsymbol{E}_i^{(\boldsymbol{C})}\right\|_2 \leq \gamma \|\boldsymbol{A}\|_2$. It is worth emphasizing that the backward errors $\boldsymbol{E}_i^{(\boldsymbol{C})} \in \mathbb{R}^{1 \times k}$, $\boldsymbol{E}_i^{(\boldsymbol{M})} \in \mathbb{R}^{k \times (k+p)}$ depend on i. Now since $\epsilon > \gamma \|\boldsymbol{A}\|_2$ by assumption and $\sigma_{\min}\left(\hat{\boldsymbol{M}}_\epsilon^\top + \boldsymbol{E}_i^{(\boldsymbol{M})}\right) \geq \epsilon - \gamma \|\boldsymbol{A}\|_2$ by Weyl's inequality, there exists a perturbation

$\boldsymbol{E}_i \in \mathbb{R}^{k\times(k+p)}$ with $\|\boldsymbol{E}_i\|_2 \leq \epsilon + \gamma \|\boldsymbol{A}\|_2$[5] such that

$$\hat{\boldsymbol{s}}_i^{(1)} = \left(\hat{\boldsymbol{U}}^\top + \boldsymbol{E}_i\right)^\dagger_{\epsilon-\gamma\|\boldsymbol{A}\|_2} \left([\hat{\boldsymbol{C}}]_i^\top + \boldsymbol{E}_i^{(\boldsymbol{C})}\right).$$

Let $\hat{\boldsymbol{S}}$ be a matrix with its ith row equal to $\left(\hat{\boldsymbol{s}}_i^{(1)}\right)^\top$.

In the third step, we compute a matrix-vector product [Hig02, Section 3.5] with $\hat{\boldsymbol{R}}^\top$, which gives us $\hat{\boldsymbol{s}}_i^{(2)} = fl\left(\hat{\boldsymbol{R}}^\top \hat{\boldsymbol{s}}_i^{(1)}\right) = \hat{\boldsymbol{R}}^\top \hat{\boldsymbol{s}}_i^{(1)} + \boldsymbol{E}_{\boldsymbol{s}_i}$ with

$$\begin{aligned}
\|\boldsymbol{E}_{\boldsymbol{s}_i}\|_2 &\leq \gamma \left\|\hat{\boldsymbol{R}}^\top\right\|_2 \left\|\hat{\boldsymbol{s}}_i^{(1)}\right\|_2 \\
&\leq \gamma \|\boldsymbol{A}\|_2 \left(\left\|\boldsymbol{Q}_{\boldsymbol{C}}(I_*,:)^\dagger\right\|_2 \left(1 + \frac{\|\boldsymbol{E}_i\|_2 + \|\boldsymbol{E}_{\boldsymbol{M}}\|_2}{\epsilon - \gamma\|\boldsymbol{A}\|_2}\right) + \frac{\left\|\boldsymbol{E}_i^{(\boldsymbol{C})}\right\|_2 + \|\boldsymbol{E}_{\boldsymbol{C}}\|_2}{\epsilon - \gamma\|\boldsymbol{A}\|_2} \right) \\
&\leq \gamma \|\boldsymbol{A}\|_2 \left(\left\|\boldsymbol{Q}_{\boldsymbol{C}}(I_*,:)^\dagger\right\|_2 \left(1 + \frac{(\epsilon + \gamma\|\boldsymbol{A}\|_2) + \gamma\|\boldsymbol{A}\|_2}{\epsilon - \gamma\|\boldsymbol{A}\|_2}\right) + \frac{\gamma\|\boldsymbol{A}\|_2 + \gamma\|\boldsymbol{A}\|_2}{\epsilon - \gamma\|\boldsymbol{A}\|_2} \right) \\
&\leq \gamma \|\boldsymbol{A}\|_2 \left\|\boldsymbol{Q}_{\boldsymbol{C}}(I_*,:)^\dagger\right\|_2, \qquad (5.5)
\end{aligned}$$

where Lemma 5.1.2 was used for the second inequality with $\boldsymbol{\Delta M} = \boldsymbol{E}_{\boldsymbol{M}} + \boldsymbol{E}_i^\top$ and $\boldsymbol{\Delta C} = \boldsymbol{E}_{\boldsymbol{C}} + \boldsymbol{e}_i \boldsymbol{E}_i^{(\boldsymbol{C})}$ where $\boldsymbol{e}_i \in \mathbb{R}^m$ is the ith canonical basis vector. In the last line of Equation (5.5), we used the fact that γ suppresses any low-degree polynomial in m, n and k, and $\epsilon > \gamma\|\boldsymbol{A}\|_2$.

The following shows the expression for $\hat{\boldsymbol{s}}_i^{(2)}$,

$$\begin{aligned}
\hat{\boldsymbol{s}}_i^{(2)} &= \hat{\boldsymbol{R}}^\top \hat{\boldsymbol{s}}_i^{(1)} + \boldsymbol{E}_{\boldsymbol{s}_i} = \hat{\boldsymbol{R}}^\top \left(\hat{\boldsymbol{M}}^\top + \boldsymbol{E}_i\right)^\dagger_{\epsilon-\gamma\|\boldsymbol{A}\|_2} \left([\hat{\boldsymbol{C}}]_i^\top + \boldsymbol{E}_i^{(\boldsymbol{C})}\right) + \boldsymbol{E}_{\boldsymbol{s}_i} \\
&= (\boldsymbol{R} + \boldsymbol{E}_{\boldsymbol{R}})^\top \left(\boldsymbol{M}^\top + \boldsymbol{E}_{\boldsymbol{M}}^\top + \boldsymbol{E}_i\right)^\dagger_{\epsilon-\gamma\|\boldsymbol{A}\|_2} \left([\boldsymbol{C} + \boldsymbol{E}_{\boldsymbol{C}}]_i^\top + \boldsymbol{E}_i^{(\boldsymbol{C})}\right) + \boldsymbol{E}_{\boldsymbol{s}_i}.
\end{aligned}$$

Finally, in the fourth step, we combine $\hat{\boldsymbol{s}}_i^{(2)} \in \mathbb{R}^n$ into a matrix to form $\widehat{\boldsymbol{A}^\epsilon_{I_*J}} = fl\left(\boldsymbol{A}^\epsilon_{I_*J}\right) \in$

[5] If $\hat{\boldsymbol{M}}^\top = \boldsymbol{U}_1\boldsymbol{\Sigma}_1\boldsymbol{V}_1^\top + \boldsymbol{U}_2\boldsymbol{\Sigma}_2\boldsymbol{V}_2^\top$ is the SVD of $\hat{\boldsymbol{M}}^\top$ where $\boldsymbol{\Sigma}_2$ contains the singular values of $\hat{\boldsymbol{M}}^\top$ smaller than ϵ, then we can take $\boldsymbol{E}_i = -\boldsymbol{U}_2\boldsymbol{\Sigma}_2\boldsymbol{V}_2^\top + \boldsymbol{E}_i^{(\boldsymbol{M})}$ for each i.

$\mathbb{R}^{m\times n}$ by setting the ith row of $\widehat{\boldsymbol{A}^{\epsilon}_{I_*J}}$ to be $\left(\hat{\boldsymbol{s}}_i^{(2)}\right)^\top$, giving us

$$fl\left(\boldsymbol{A}^{\epsilon}_{I_*J}\right) = \hat{\boldsymbol{S}}\hat{\boldsymbol{R}} + \boldsymbol{E} \tag{5.6}$$

where $\boldsymbol{E} \in \mathbb{R}^{m\times n}$ is a matrix with its ith row equal to $\boldsymbol{E}^\top_{\boldsymbol{s}_i}$ and satisfies $\|\boldsymbol{E}\|_2 \leq \gamma \|\boldsymbol{A}\|_2 \left\|\boldsymbol{Q}_{\boldsymbol{C}}(I_*,:)^\dagger\right\|_2$, since γ suppresses any low-degree polynomial in m.

We now state the main stability result for the CUR-CA with the ϵ-pseudoinverse, $\boldsymbol{A}^{\epsilon}_{I_*J}$.

Theorem 5.1.3. *Let $0 < \epsilon \ll 1$ be a truncation parameter for the pseudoinverse such that $\epsilon > \gamma \|\boldsymbol{A}\|_2$. Suppose that $\boldsymbol{A}^{\epsilon}_{I_*J} = \boldsymbol{A}(:,J)\boldsymbol{A}(I_*,J)^\dagger_\epsilon \boldsymbol{A}(I_*,:)$ is computed in the following order:*

1. *Compute $\boldsymbol{C} = \boldsymbol{A}(:,J)$, $\boldsymbol{M} = \boldsymbol{A}(I_*,J)$ and $\boldsymbol{R} = \boldsymbol{A}(I_*,:)$,*
2. *Compute each row of $\boldsymbol{C}\boldsymbol{M}^\dagger_\epsilon$ using a backward stable (rank-deficient) underdetermined linear solver,*
3. *Compute $\boldsymbol{C}\boldsymbol{M}^\dagger_\epsilon$ times $\boldsymbol{R}$. Let $fl\left(\boldsymbol{A}^{\epsilon}_{I_*J}\right)$ denote the output.*

Then under Assumption 5.1.1,

$$\begin{aligned}\left\|\boldsymbol{A} - fl\left(\boldsymbol{A}^{\epsilon}_{I_*J}\right)\right\|_{\mathrm{F}} \leq 4\sqrt{m}\left\|\boldsymbol{Q}_{\boldsymbol{C}}(I_*,:)^\dagger\right\|_2 \left\|\boldsymbol{Q}_{\boldsymbol{X}}(J,:)^\dagger\right\|_2 \left(\left\|\boldsymbol{A}(\boldsymbol{I} - \boldsymbol{X}^\dagger\boldsymbol{X})\right\|_{\mathrm{F}} + 2\epsilon\right) \\ + \gamma\|\boldsymbol{A}\|_2\left\|\boldsymbol{Q}_{\boldsymbol{C}}(I_*,:)^\dagger\right\|_2.\end{aligned}$$

Theorem 5.1.3 tells us that the bound for the computed version of the stabilized CUR-CA, $\widehat{\boldsymbol{A}^{\epsilon}_{IJ}}$ is at most a factor $\mathcal{O}(\sqrt{m})$ worse than its exact arithmetic counterpart $\boldsymbol{A}^{\epsilon}_{IJ}$ plus an error of $\gamma\|\boldsymbol{A}\|_2\left\|\boldsymbol{Q}_{\boldsymbol{C}}(I_*,:)^\dagger\right\|_2$. More specifically, we have

$$\left\|\boldsymbol{A} - fl(\boldsymbol{A}^{\epsilon}_{I_*J})\right\|_{\mathrm{F}} \leq 4\sqrt{m}\,(\text{Bound (5.2)}) + \gamma\|\boldsymbol{A}\|_2\left\|\boldsymbol{Q}_{\boldsymbol{C}}(I_*,:)^\dagger\right\|_2.$$

Roughly, this shows that the computed error is in the same order as the error in exact

arithmetic, up to a factor $\mathcal{O}(\sqrt{m})$. The factor $\mathcal{O}(\sqrt{m})$ is likely an artifact of the proof given below; in stability analysis it is common to see bounds with such overestimates, and also standard practice to expect to observe much better performance in practice. Throughout various parts of the proof, we loosely bound the 2-norm by the Frobenius norm, typically because we only have information about the norms of rows or columns of a matrix. For example in the proof of Theorem 5.1.3, we bound $\left\|\hat{\boldsymbol{S}}\right\|_2 \leq \left\|\hat{\boldsymbol{S}}\right\|_F$ in Equation (5.8), which is likely a pessimistic overestimate.

Proof of Theorem 5.1.3. From the above analysis, we have

$$fl\left(\boldsymbol{A}^{\epsilon}_{I_* J}\right) = \hat{\boldsymbol{S}}\hat{\boldsymbol{R}} + \boldsymbol{E} = \hat{\boldsymbol{S}}\boldsymbol{R} + \hat{\boldsymbol{S}}\boldsymbol{E}_{\boldsymbol{R}} + \boldsymbol{E}, \tag{5.6}$$

where $\|\boldsymbol{E}\|_2 \leq \gamma \|\boldsymbol{A}\|_2 \left\|\boldsymbol{Q}_{\boldsymbol{C}}(I_*, :)^\dagger\right\|_2$. Let us apply Lemma 5.1.3 to each row of $\hat{\boldsymbol{S}}\boldsymbol{R}$ to obtain

$$\hat{\boldsymbol{S}}_i \boldsymbol{R}\boldsymbol{\Pi}_J = \left([\boldsymbol{C} + \boldsymbol{E}_{\boldsymbol{C}}]_i + \left(\boldsymbol{E}_i^{(C)}\right)^\top\right)\left(\boldsymbol{M} + \boldsymbol{E}_{\boldsymbol{M}} + \boldsymbol{E}_i^\top\right)^\dagger_{\epsilon - \gamma\|\boldsymbol{A}\|_2} \boldsymbol{R}\boldsymbol{\Pi}_J = [\boldsymbol{C}]_i + \boldsymbol{E}_i^{(P)},$$

where $\left\|\boldsymbol{E}_i^{(P)}\right\|_2 \leq 8\epsilon \left\|\boldsymbol{Q}_{\boldsymbol{C}}(I_*, :)^\dagger\right\|_2$. Therefore, letting $\boldsymbol{E}^{(P)}$ be a matrix with $\boldsymbol{E}_i^{(P)}$ as its ith row, we get

$$\hat{\boldsymbol{S}}\boldsymbol{R}\boldsymbol{\Pi}_J = \boldsymbol{C} + \boldsymbol{E}^{(P)},$$

where $\left\|\boldsymbol{E}^{(P)}\right\|_{\mathrm{F}} \leq 8\epsilon\sqrt{m} \left\|\boldsymbol{Q}_{\boldsymbol{C}}(I_*, :)^\dagger\right\|_2$. Now proceeding similarly to the proof of Theorem 5.1.1, we obtain

$$\begin{aligned}
\boldsymbol{A} - \hat{\boldsymbol{S}}\boldsymbol{R} &= \left(\boldsymbol{I} - \hat{\boldsymbol{S}}\boldsymbol{\Pi}_{I_*}^\top\right)\boldsymbol{A} = \left(\boldsymbol{I} - \hat{\boldsymbol{S}}\boldsymbol{\Pi}_{I_*}^\top\right)\boldsymbol{A}(\boldsymbol{I} - \boldsymbol{\Pi}_J(\boldsymbol{X}\boldsymbol{\Pi}_J)^\dagger\boldsymbol{X}) - \boldsymbol{E}^{(P)}(\boldsymbol{X}\boldsymbol{\Pi}_J)^\dagger\boldsymbol{X} \\
&= \left(\boldsymbol{I} - \hat{\boldsymbol{S}}\boldsymbol{\Pi}_{I_*}^\top\right)\boldsymbol{A}(\boldsymbol{I} - \boldsymbol{X}^\dagger\boldsymbol{X})(\boldsymbol{I} - \boldsymbol{\Pi}_J(\boldsymbol{X}\boldsymbol{\Pi}_J)^\dagger\boldsymbol{X}) - \boldsymbol{E}^{(P)}(\boldsymbol{X}\boldsymbol{\Pi}_J)^\dagger\boldsymbol{X},
\end{aligned} \tag{5.7}$$

where $\boldsymbol{E}^{(P)}(\boldsymbol{X}\boldsymbol{\Pi}_J)^\dagger\boldsymbol{X}$ satisfies

$$\left\|\boldsymbol{E}^{(P)}(\boldsymbol{X}\boldsymbol{\Pi}_J)^\dagger\boldsymbol{X}\right\|_{\mathrm{F}} \leq \left\|\boldsymbol{E}^{(P)}\right\|_{\mathrm{F}} \left\|(\boldsymbol{X}\boldsymbol{\Pi}_J)^\dagger\boldsymbol{X}\right\|_2 \leq 8\epsilon\sqrt{m}\left\|\boldsymbol{Q}_{\boldsymbol{C}}(I_*,:)^\dagger\right\|_2 \left\|\boldsymbol{Q}_{\boldsymbol{X}}(J,:)^\dagger\right\|_2.$$

The first term in Equation (5.7) can be bound by

$$\begin{aligned}
&\left\|\boldsymbol{I} - \hat{\boldsymbol{S}}\boldsymbol{\Pi}_{I_*}^\top\right\|_2 \left\|\boldsymbol{A}(\boldsymbol{I} - \boldsymbol{X}^\dagger\boldsymbol{X})\right\|_{\mathrm{F}} \left\|\boldsymbol{I} - \boldsymbol{\Pi}_J(\boldsymbol{X}\boldsymbol{\Pi}_J)^\dagger\boldsymbol{X}\right\|_2 \\
&\qquad \leq \left(1 + \left\|\hat{\boldsymbol{S}}\right\|_2\right)\left\|\boldsymbol{A}(\boldsymbol{I} - \boldsymbol{X}^\dagger\boldsymbol{X})\right\|_{\mathrm{F}} \left\|\boldsymbol{\Pi}_J(\boldsymbol{X}\boldsymbol{\Pi}_J)^\dagger\boldsymbol{X}\right\|_2 \\
&\qquad \leq 4\sqrt{m}\left\|\boldsymbol{Q}_{\boldsymbol{C}}(I_*,:)^\dagger\right\|_2 \left\|\boldsymbol{Q}_{\boldsymbol{X}}(J,:)^\dagger\right\|_2 \left\|\boldsymbol{A}(\boldsymbol{I} - \boldsymbol{X}^\dagger\boldsymbol{X})\right\|_{\mathrm{F}}
\end{aligned}$$

where in the final line we use Equation (5.5) to deduce

$$\begin{aligned}
\left\|\hat{\boldsymbol{s}}_i^{(1)}\right\|_2 &\leq \left\|\boldsymbol{Q}_{\boldsymbol{C}}(I_*,:)^\dagger\right\|_2 \left(1 + \frac{(\epsilon + \gamma\left\|\boldsymbol{A}\right\|_2) + \gamma\left\|\boldsymbol{A}\right\|_2}{\epsilon - \gamma\left\|\boldsymbol{A}\right\|_2}\right) + \frac{\gamma\left\|\boldsymbol{A}\right\|_2 + \gamma\left\|\boldsymbol{A}\right\|_2}{\epsilon - \gamma\left\|\boldsymbol{A}\right\|_2} \\
&\leq \left\|\boldsymbol{Q}_{\boldsymbol{C}}(I_*,:)^\dagger\right\|_2 \left(1 + \frac{\epsilon + \gamma\left\|\boldsymbol{A}\right\|_2}{\epsilon - \gamma\left\|\boldsymbol{A}\right\|_2}\right) \\
&\leq 3\left\|\boldsymbol{Q}_{\boldsymbol{C}}(I_*,:)^\dagger\right\|_2,
\end{aligned}$$

and observe that

$$\left\|\hat{\boldsymbol{S}}\right\|_2 \leq \left\|\hat{\boldsymbol{S}}\right\|_{\mathrm{F}} \leq \sqrt{\sum_{i=1}^m \left\|\hat{\boldsymbol{s}}_i^{(1)}\right\|_2^2} \leq 3\sqrt{m}\left\|\boldsymbol{Q}_{\boldsymbol{C}}(I_*,:)^\dagger\right\|_2. \tag{5.8}$$

Using Equation (5.8), we can also bound

$$\left\|\hat{\boldsymbol{S}}\boldsymbol{E}_{\boldsymbol{R}}\right\|_{\mathrm{F}} \leq \left\|\hat{\boldsymbol{S}}\right\|_{\mathrm{F}} \left\|\boldsymbol{E}_{\boldsymbol{R}}\right\|_2 \leq \gamma\left\|\boldsymbol{A}\right\|_2 \left\|\boldsymbol{Q}_{\boldsymbol{C}}(I_*,:)^\dagger\right\|_2.$$

Finally putting everything together, we obtain

$$\begin{aligned}\left\|\boldsymbol{A}-fl(\boldsymbol{A}^{\epsilon}_{I_*J})\right\|_{\mathrm{F}} &= \left\|\boldsymbol{A}-\hat{\boldsymbol{S}}\boldsymbol{R}-\hat{\boldsymbol{S}}\boldsymbol{E}_{\boldsymbol{R}}-\boldsymbol{E}\right\|_{\mathrm{F}} \\ &\leq 4\sqrt{m}\left\|\boldsymbol{Q}_{\boldsymbol{C}}(I_*,:)^{\dagger}\right\|_2\left\|\boldsymbol{Q}_{\boldsymbol{X}}(J,:)^{\dagger}\right\|_2\left(\left\|\boldsymbol{A}(\boldsymbol{I}-\boldsymbol{X}^{\dagger}\boldsymbol{X})\right\|_{\mathrm{F}}+2\epsilon\right) \\ &\quad+\gamma\left\|\boldsymbol{A}\right\|_2\left\|\boldsymbol{Q}_{\boldsymbol{C}}(I_*,:)^{\dagger}\right\|_2.\end{aligned}$$

□

It is natural to wonder what could go wrong without the ϵ-pseudoinverse. Two problems may arise without the ϵ-pseudoinverse. First, the matrix $\boldsymbol{M}$ may not be numerically full rank, so Theorem 5.1.2 cannot be used as $\epsilon > \gamma\left\|\boldsymbol{A}\right\|_2$ may no longer hold. In addition, Lemmas 5.1.2 and 5.1.3 become meaningless as both require division by ϵ. Without the ϵ-pseudoinverse, the error bound can become uncontrollably large, causing issues in several places in the proof of Theorem 5.1.3. For example, the bound for the error $\left\|\boldsymbol{E}_{\boldsymbol{s}_i}\right\|_2$ in Equation (5.5) for computing the matrix-vector product and the bound for $\left\|\boldsymbol{E}_i^{(P)}\right\|_2$ in the proof of Theorem 5.1.3 may no longer hold as they can become arbitrarily large without the ϵ-pseudoinverse. Nevertheless, in practice we observe stability without the ϵ-truncation, so we recommend a careful implementation of the pseudoinverse (without the ϵ-truncation) for practical purposes. To be clear, implementing the ϵ-truncation does not increase the complexity and can be recommended for guaranteed stability. We have simply not observed instability without the ϵ-truncation. A similar observation, commenting on the role of the ϵ-pseudoinverse, has been mentioned in [Nak20, Section 4.2]. See Section 5.3.1 for numerical experiments.

In Theorem 5.1.3, one might question how large $\left\|\boldsymbol{Q}_{\boldsymbol{C}}(I_*,:)^{\dagger}\right\|_2$ can be, given that it is part of the added term, $\gamma\left\|\boldsymbol{A}\right\|_2\left\|\boldsymbol{Q}_{\boldsymbol{C}}(I_*,:)^{\dagger}\right\|_2$ in Theorem 5.1.3. This could pose a problem if $\left\|\boldsymbol{Q}_{\boldsymbol{C}}(I_*,:)^{\dagger}\right\|_2$ grows exponentially in m or n. However, Theorem 5.1.3 is enough to conclude that $\boldsymbol{A}^{\epsilon}_{IJ}$ is numerically stable when the indices are chosen reasonably. This means that we first choose the column indices J, and sensibly select the row indices I_* from the the selected

columns $\boldsymbol{A}(:, J)$ such that $\left\|\boldsymbol{A}(:, J)\boldsymbol{A}(I_*, J)^\dagger\right\|_2 = \left\|\boldsymbol{Q}_{\boldsymbol{C}}(I_*, :)^\dagger\right\|_2$ is bounded by a low-degree polynomial involving m, n and k; see Section 5.3.2. A similar approach is also employed in other works such as [VM17, DM23, Xia24, CY25]. For example, Gu-Eisenstat's strong rank-revealing QR factorization [GE96] can be used on the chosen columns $\boldsymbol{A}(:, J)$ to obtain $\|\boldsymbol{Q}_{\boldsymbol{C}}(I, :)^{-1}\|_2 \leq \sqrt{mk}$, which can be reduced further with oversampling. We explore the topic of oversampling in more detail in Chapter 6.

5.2 Numerical stability of CUR-BA

For the CUR-BA, $\boldsymbol{C}(\boldsymbol{C}^\dagger \boldsymbol{A}\boldsymbol{R}^\dagger \boldsymbol{R})$, there exist a numerically stable implementation given by the StableCUR algorithm in [ADM+15]. The algorithm computes the CUR-BA in MATLAB as follows:

$$[\mathtt{Q_C}, \sim] = \mathtt{qr}(\mathtt{A}(:, \mathtt{J}), 0),\ [\mathtt{Q_R}, \sim] = \mathtt{qr}(\mathtt{A}(\mathtt{I}, :)', 0),$$
$$\mathtt{A}_{\mathtt{IJ}}^{(\mathtt{BA})} = \mathtt{Q_C} * (\mathtt{Q_C}' * \mathtt{A} * \mathtt{Q_R}) * \mathtt{Q_R}'.$$

While StableCUR is numerically stable, it has the drawback that the resulting approximation is no longer expressed explicitly as a product of $\boldsymbol{C}$ and $\boldsymbol{R}$. This can be undesirable in applications where interpretability or explicit factorisation is important. In this thesis, we do not pursue further investigation into expressing CUR-BA stably using the factors $\boldsymbol{C}$ and $\boldsymbol{R}$. The accuracy bound of CUR-BA with oversampling will be analysed and discussed later in Section 6.4.

5.3 Numerical illustration

In this section, we illustrate the concepts discussed in the previous sections through numerical experiments. We first discuss implementation details of (S)CUR-CA in MATLAB. Then we show that, without loss of generality, after selecting the columns first, the rows should

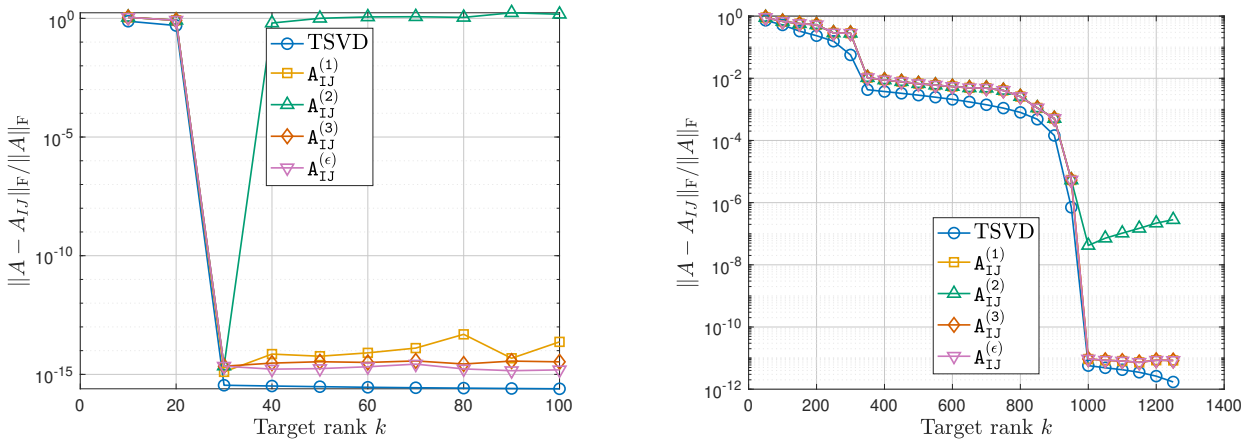
be chosen based on those columns, i.e., the rows and the columns for the CUR-CA should not be chosen independently. In all the experiments, the best rank-k approximation error using the truncated SVD (TSVD) is used as reference.

5.3.1 Implementation of (S)CUR-CA

The main concern in the computation of the CUR-CA is that (i) the representation of the CUR-CA typically involves three (highly) ill-conditioned matrix, and (ii) we take the pseudoinverse of a (highly) ill-conditioned matrix. When the original matrix $\boldsymbol{A}$ is low-rank and we have a good low-rank approximation of $\boldsymbol{A}$, then we expect $\boldsymbol{C}, \boldsymbol{M}$ and $\boldsymbol{R}$ to be all highly ill-conditioned. We test four possible implementations in MATLAB,

- $\mathtt{A}_{\mathtt{IJ}}^{(1)} = (\mathtt{A}(:,\mathtt{J})/\mathtt{A}(\mathtt{I},\mathtt{J})) * \mathtt{A}(\mathtt{I},:)$,
- $\mathtt{A}_{\mathtt{IJ}}^{(2)} = \mathtt{A}(:,\mathtt{J}) * (\mathtt{V}/\mathtt{S} * \mathtt{U}') * \mathtt{A}(\mathtt{I},:)$ where $[\mathtt{U},\mathtt{S},\mathtt{V}] = \mathtt{svd}(\mathtt{M},\text{`}\mathtt{econ}')$,
- $\mathtt{A}_{\mathtt{IJ}}^{(3)} = (\mathtt{A}(:,\mathtt{J}) * \mathtt{V}/\mathtt{S}) * (\mathtt{U}' * \mathtt{A}(\mathtt{I},:))$ where $[\mathtt{U},\mathtt{S},\mathtt{V}] = \mathtt{svd}(\mathtt{M},\text{`}\mathtt{econ}')$,
- $\mathtt{A}_{\mathtt{IJ}}^{(\epsilon)} = (\mathtt{A}(:,\mathtt{J}) * \mathtt{V}(:,\mathtt{II})/\mathtt{S}(\mathtt{II},\mathtt{II})) * (\mathtt{U}(:,\mathtt{II})' * \mathtt{A}(\mathtt{I},:))$
  where $[\mathtt{U},\mathtt{S},\mathtt{V}] = \mathtt{svd}(\mathtt{M},\text{`}\mathtt{econ}')$ and $\mathtt{II} = (\mathtt{diag}(\mathtt{S}) > \epsilon)$ with $\epsilon = 10^{-15}$.

The third implementation, $\mathtt{A}_{\mathtt{IJ}}^{(3)}$ is the suggested implementation of the CUR-CA. For guaranteed stability, we suggest the fourth implementation, $\mathtt{A}_{\mathtt{IJ}}^{(\epsilon)}$ with the ϵ-pseudoinverse. However we have not observed instability without the ϵ-pseudoinverse using the third implementation. We use two test matrices: (1) 1000×1000 test matrix generated using the MATLAB command, $\mathtt{randn}(1000, 30) * \mathtt{randn}(30, 1000)$, and (2) 1374×1374 matrix named `nnc1374` from the SuiteSparse Matrix Collection [DH11]. We choose the rows and columns by first choosing the column indices J using column pivoted QR (CPQR) on $\boldsymbol{A}$ and then using CPQR on the chosen columns $\boldsymbol{A}(:, J)$ to get the row indices I.



(a) $\texttt{randn}(1000, 30) * \texttt{randn}(30, 1000)$ **(b)** `nnc1374`

Figure 5.1: Tests for different implementations of the CUR-CA. The third and fourth implementation performs stably. We recommend the third implementation.

The results are depicted in Figure 5.1. First, we notice that the third implementation, $\mathtt{A}_{\mathtt{IJ}}^{(3)}$, which is the one we suggest, yields stable approximation throughout the experiment. In particular, the third implementation performs similarly to the fourth implementation with the ϵ-pseudoinverse. On the other hand, the second implementation, $\mathtt{A}_{\mathtt{IJ}}^{(2)}$ suffers from numerical errors as we lose accuracy when the singular values decay extremely rapidly. The reason is because we explicitly form the pseudoinverse of the core matrix $\boldsymbol{A}(I, J)$, which is highly ill-conditioned, an observation also noted in [Nak20]. Therefore, it is important to not form the pseudoinverse of $\boldsymbol{M} = \boldsymbol{A}(I, J)$ explicitly in the CUR-CA. Note also that the order in which the factors are multiplied is essential for numerical stability, as the second and third implementations only differ by the order in which the factors are computed. The first implementation, $\mathtt{A}_{\mathtt{IJ}}^{(1)}$ may lose 1 or 2 orders of magnitude in Figure 5.1a once the target rank becomes larger than the rank of the test matrix and appear to be less stable than the third implementation. The slash commands $(/, \backslash)$ in MATLAB should be used with caution for underdetermined problems, as its backslash command applied to numerically rank-deficient underdetermined problems output a sparse solution based on a pivoting strategy [ABB+99, § 2.4], which can differ significantly from the minimum-norm solution and may not satisfy the

assumptions in our analysis. Note that using the `pinv` command with tolerance parameter 0 for the CUR-CA in MATLAB, i.e., $\texttt{A}(:,\texttt{J}) * \texttt{pinv}(\texttt{A}(\texttt{I},\texttt{J}),0), *\texttt{A}(\texttt{I},:)$, is equivalent to the second implementation, which can be unstable. For the rest of the numerical experiments, we use the third implementation

$$(\texttt{A}(:,\texttt{J}) * \texttt{V}/\texttt{S}) * (\texttt{U}' * \texttt{A}(\texttt{I},:)) \text{ where } [\texttt{U},\texttt{S},\texttt{V}] = \texttt{svd}(\texttt{M},\text{`}\texttt{econ}').$$

The fourth implementation computes the stabilized CUR-CA and is implemented in such a way that the analysis in Section 5.1.2 hold. A less expensive alternative is to use a rank-revealing QR factorisation (or even column-pivoted QR factorisation) and truncate the bottom-right corner of the upper triangular factor and the relevant columns of the orthonormal factor corresponding to the diagonal elements less than ϵ. In the rank-revealing QR, the diagonal elements give a good approximation to the singular values; see for example [Cha87, GE96]. A possible workaround without the ϵ-pseudoinverse is also discussed in [Nak20] where we perturb the core matrix $\boldsymbol{A}(I,J)$ by a small noise matrix such that the singular values of $\boldsymbol{A}(I,J)$ are all larger than the unit roundoff.

5.3.2 Importance of choosing rows and columns dependently

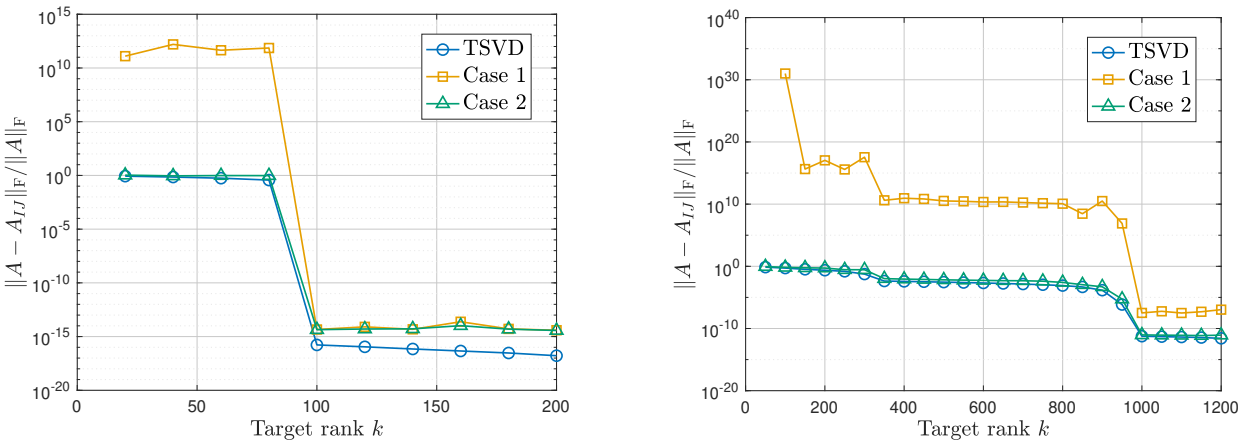
In this section, we demonstrate the importance of choosing the rows and columns that are dependent on one another in the CUR-CA. If the rows and columns are chosen independently of each other, the matrix $\boldsymbol{M} = \boldsymbol{A}(I,J)$ can be (nearly) singular. We use two test matrices: (1) synthetic matrix generated using

$$\boldsymbol{A} = \begin{bmatrix} 10^{-10} \cdot \texttt{randn}(50,50) & \texttt{randn}(50,950) \\ \texttt{randn}(950,50) & 0 \end{bmatrix} \in \mathbb{R}^{1000\times 1000}, \tag{5.9}$$

and (2) `nnc1374` matrix from the SuiteSparse Matrix Collection used in the previous section. We test the following two cases:

- **Case 1:** Choose columns and rows independently by applying CPQR on $\boldsymbol{A}$ and $\boldsymbol{A}^\top$ respectively,
- **Case 2:** Choose columns and rows dependently by applying CPQR on $\boldsymbol{A}$ and then applying CPQR on the chosen columns $\boldsymbol{A}(:, J)$ to obtain the rows,

where k is the target rank.



(a) Block random matrix (5.9) **(b)** `nnc1374`

Figure 5.2: Relationship between the rows and columns in the CUR-CA. In Case 1, the rows and columns are chosen independently from one another and in Case 2, we select the columns first and then the rows were computed from the selected columns. When the rows and columns are chosen independently (Case 1), the resulting approximation can be catastrophic.

In Figure 5.2, we show the importance of choosing the columns and rows in the CUR-CA dependently. When the columns and rows are chosen independently (Case 1), the resulting CUR-CA can be catastrophically poor. In Figure 5.2a, poor approximation happens because when we choose the columns and rows independently, we choose the first 50 rows and columns, as they are the most important, implying that the core matrix is the small $(1, 1)$-block of $\boldsymbol{A}$. Since the chosen rows and columns are $\mathcal{O}(1)$ and their intersection is $\mathcal{O}(10^{-10})$, the CUR-CA

becomes inaccurate. A similar issue also arises in Figure 5.2b where the approximation becomes noticeably unreliable throughout; see also the 2×2 example at the end of Section 5.1.1.

This issue is not limited to deterministic selection: randomised methods that sample rows and columns independently can suffer from the same failure mode. For instance, when $\boldsymbol{A} = \left[\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right]$ and the target rank is 1, the probability of sampling zero as the core matrix using leverage scores or column norms is 0.5 when the rows and columns are sampled independently. Independent sampling results in a poor CUR-CA 50% of the time. However, if we sample them dependently, we choose 1 as the core matrix with probability 1, leading to a sensible CUR-CA. Moreover, as demonstrated in Case 2, dependent selection of rows and columns lead to stable approximations. These observations underscore the importance of selecting rows and columns dependently in practice.

5.4 Proof of Lemmas 5.1.2 and 5.1.3

We give the proofs for the two lemmas, Lemmas 5.1.2 and 5.1.3 in this section.

Lemma 5.4.1 (Lemma 5.1.2)**.** *Under Assumption 5.1.1, for any* $\boldsymbol{\Delta C}$ *and* $\boldsymbol{\Delta M}$*,*

$$\left\|(\boldsymbol{C}+\boldsymbol{\Delta C})(\boldsymbol{M}+\boldsymbol{\Delta M})_\epsilon^\dagger\right\|_2 \leq \left\|\boldsymbol{Q_C}(I_*,:)^\dagger\right\|_2 \left(1+\frac{1}{\epsilon}\left\|\boldsymbol{\Delta M}\right\|_2\right)+\frac{1}{\epsilon}\left\|\boldsymbol{\Delta C}\right\|_2.$$

Proof. Let $\boldsymbol{C} = \boldsymbol{Q_C}\boldsymbol{R_C}$ be the thin QR factorization of $\boldsymbol{C}$. Then

$$\begin{aligned}
\left\|(\boldsymbol{C}+\boldsymbol{\Delta C})(\boldsymbol{M}+\boldsymbol{\Delta M})_\epsilon^\dagger\right\|_2 &\leq \left\|\boldsymbol{Q_C}\boldsymbol{R_C}(\boldsymbol{M}+\boldsymbol{\Delta M})_\epsilon^\dagger\right\|_2 + \left\|\boldsymbol{\Delta C}(\boldsymbol{M}+\boldsymbol{\Delta M})_\epsilon^\dagger\right\|_2 \\
&= \left\|\boldsymbol{R_C}(\boldsymbol{M}+\boldsymbol{\Delta M})_\epsilon^\dagger\right\|_2 + \left\|\boldsymbol{\Delta C}(\boldsymbol{M}+\boldsymbol{\Delta M})_\epsilon^\dagger\right\|_2 \\
&= \left\|(\boldsymbol{\Pi}_{I_*}^\top \boldsymbol{Q_C})^\dagger \boldsymbol{\Pi}_{I_*}^\top \boldsymbol{Q_C}\boldsymbol{R_C}(\boldsymbol{M}+\boldsymbol{\Delta M})_\epsilon^\dagger\right\|_2 \\
&\qquad + \left\|\boldsymbol{\Delta C}(\boldsymbol{M}+\boldsymbol{\Delta M})_\epsilon^\dagger\right\|_2 \\
&\leq \left\|(\boldsymbol{\Pi}_{I_*}^\top \boldsymbol{Q_C})^\dagger\right\|_2 \left\|\boldsymbol{M}(\boldsymbol{M}+\boldsymbol{\Delta M})_\epsilon^\dagger\right\|_2 + \frac{1}{\epsilon}\left\|\boldsymbol{\Delta C}\right\|_2
\end{aligned}$$

where we use $(\boldsymbol{\Pi}_{I_*}^\top \boldsymbol{Q_C})^\dagger \boldsymbol{\Pi}_{I_*}^\top \boldsymbol{Q_C} = \boldsymbol{I}$ for the penultimate line. The result follows by noting that $\left\|\boldsymbol{M}(\boldsymbol{M}+\boldsymbol{\Delta M})_\epsilon^\dagger\right\|_2$ simplifies to

$$\begin{aligned}\left\|\boldsymbol{M}(\boldsymbol{M}+\boldsymbol{\Delta M})_\epsilon^\dagger\right\|_2 &= \left\|(\boldsymbol{M}+\boldsymbol{\Delta M})(\boldsymbol{M}+\boldsymbol{\Delta M})_\epsilon^\dagger - \boldsymbol{\Delta M}(\boldsymbol{M}+\boldsymbol{\Delta M})_\epsilon^\dagger\right\|_2 \\ &\leq 1 + \frac{1}{\epsilon}\left\|\boldsymbol{\Delta M}\right\|_2.\end{aligned}$$

□

Lemma 5.4.2 (Lemma 5.1.3)**.** *Under Assumption 5.1.1, for any* $\boldsymbol{\Delta C}$ *and* $\boldsymbol{\Delta M}$*,*

$$(\boldsymbol{C}+\boldsymbol{\Delta C})(\boldsymbol{M}+\boldsymbol{\Delta M})_\epsilon^\dagger \boldsymbol{R}\boldsymbol{\Pi}_J = \boldsymbol{C} + \boldsymbol{E}_*$$

where

$$\left\|\boldsymbol{E}_*\right\|_2 \leq \left\|(\boldsymbol{Q_C}(I_*,:)^\dagger\right\|_2 \left(\epsilon + 2\left\|\boldsymbol{\Delta M}\right\|_2 + \frac{1}{\epsilon}\left\|\boldsymbol{\Delta M}\right\|_2^2\right) + \left\|\boldsymbol{\Delta C}\right\|_2 \left(1 + \frac{\left\|\boldsymbol{\Delta M}\right\|_2}{\epsilon}\right).$$

Proof. We divide the expression into two pieces:

$$(\boldsymbol{C}+\boldsymbol{\Delta C})(\boldsymbol{M}+\boldsymbol{\Delta M})_\epsilon^\dagger \boldsymbol{R}\boldsymbol{\Pi}_J = \underbrace{\boldsymbol{C}(\boldsymbol{M}+\boldsymbol{\Delta M})_\epsilon^\dagger \boldsymbol{R}\boldsymbol{\Pi}_J}_{(i)} + \underbrace{\boldsymbol{\Delta C}(\boldsymbol{M}+\boldsymbol{\Delta M})_\epsilon^\dagger \boldsymbol{R}\boldsymbol{\Pi}_J}_{(ii)}$$

and treat them separately. Let the thin SVD of $\boldsymbol{M}+\boldsymbol{\Delta M}$ be

$$\boldsymbol{M}+\boldsymbol{\Delta M} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top = [\boldsymbol{U}_1, \boldsymbol{U}_2]\begin{bmatrix}\boldsymbol{\Sigma}_1 & \\ & \boldsymbol{\Sigma}_2\end{bmatrix}[\boldsymbol{V}_1, \boldsymbol{V}_2]^\top$$

where $\boldsymbol{\Sigma}_2$ contains the singular values of $\boldsymbol{M}+\boldsymbol{\Delta M}$ smaller than ϵ, and $\boldsymbol{C} = \boldsymbol{Q_C}\boldsymbol{R_C}$ be the thin QR decomposition of $\boldsymbol{C}$.

We begin by examining the matrix (i):

$$\begin{aligned}\boldsymbol{C}(\boldsymbol{M}+\boldsymbol{\Delta M})_\epsilon^\dagger \boldsymbol{R}\boldsymbol{\Pi}_J &= \boldsymbol{C}(\boldsymbol{M}+\boldsymbol{\Delta M})_\epsilon^\dagger \boldsymbol{M} \\ &= \boldsymbol{C}(\boldsymbol{M}+\boldsymbol{\Delta M})_\epsilon^\dagger(\boldsymbol{M}+\boldsymbol{\Delta M}) - \boldsymbol{C}(\boldsymbol{M}+\boldsymbol{\Delta M})_\epsilon^\dagger \boldsymbol{\Delta M} \\ &= \boldsymbol{C}(\boldsymbol{I}_k - \boldsymbol{V}_2\boldsymbol{V}_2^\top) - \boldsymbol{C}(\boldsymbol{M}+\boldsymbol{\Delta M})_\epsilon^\dagger \boldsymbol{\Delta M} \\ &= \boldsymbol{C} + \boldsymbol{E}_1,\end{aligned}$$

where $\boldsymbol{E}_1 = -\boldsymbol{C}\boldsymbol{V}_2\boldsymbol{V}_2^\top - \boldsymbol{C}(\boldsymbol{M}+\boldsymbol{\Delta M})_\epsilon^\dagger \boldsymbol{\Delta M}$ satisfies

$$\begin{aligned}\|\boldsymbol{E}_1\|_2 &\le \left\|\boldsymbol{C}\boldsymbol{V}_2\boldsymbol{V}_2^\top\right\|_2 + \left\|\boldsymbol{C}(\boldsymbol{M}+\boldsymbol{\Delta M})_\epsilon^\dagger \boldsymbol{\Delta M}\right\|_2 \\ &\le \left\|\boldsymbol{C}\boldsymbol{V}_2\boldsymbol{V}_2^\top\right\|_2 + \left\|\boldsymbol{Q}_{\boldsymbol{C}}(I_*,:)^\dagger\right\|_2 \left(1 + \frac{1}{\epsilon}\|\boldsymbol{\Delta M}\|\right)\|\boldsymbol{\Delta M}\|_2\end{aligned}$$

by Lemma 5.1.2. We bound $\left\|\boldsymbol{C}\boldsymbol{V}_2\boldsymbol{V}_2^\top\right\|_2$ as in the final part of the proof of Theorem 5.1.1.

$$\begin{aligned}\left\|\boldsymbol{C}\boldsymbol{V}_2\boldsymbol{V}_2^\top\right\|_2 = \left\|\boldsymbol{R}_{\boldsymbol{C}}\boldsymbol{V}_2\boldsymbol{V}_2^\top\right\|_2 &= \left\|(\boldsymbol{\Pi}_{I_*}^\top \boldsymbol{Q}_{\boldsymbol{C}})^\dagger \boldsymbol{\Pi}_{I_*}^\top \boldsymbol{Q}_{\boldsymbol{C}}\boldsymbol{R}_{\boldsymbol{C}}\boldsymbol{V}_2\boldsymbol{V}_2^\top\right\|_2 \\ &= \left\|(\boldsymbol{\Pi}_{I_*}^\top \boldsymbol{Q}_{\boldsymbol{C}})^\dagger \boldsymbol{M}\boldsymbol{V}_2\boldsymbol{V}_2^\top\right\|_2 \\ &\le \left\|(\boldsymbol{\Pi}_{I_*}^\top \boldsymbol{Q}_{\boldsymbol{C}})^\dagger\right\|_2 \left\|(\boldsymbol{M}+\boldsymbol{\Delta M})\boldsymbol{V}_2\boldsymbol{V}_2^\top - \boldsymbol{\Delta M}\boldsymbol{V}_2\boldsymbol{V}_2^\top\right\|_2 \\ &\le \left\|(\boldsymbol{\Pi}_{I_*}^\top \boldsymbol{Q}_{\boldsymbol{C}})^\dagger\right\|_2 \left(\left\|\boldsymbol{U}_2\boldsymbol{\Sigma}_2\boldsymbol{V}_2^\top\right\|_2 + \|\boldsymbol{\Delta M}\|_2\right) \\ &\le \left\|(\boldsymbol{\Pi}_{I_*}^\top \boldsymbol{Q}_{\boldsymbol{C}})^\dagger\right\|_2 (\epsilon + \|\boldsymbol{\Delta M}\|_2)\,.\end{aligned}$$

Therefore, (i) can be bounded as

$$\boldsymbol{C}(\boldsymbol{M}+\boldsymbol{\Delta M})_\epsilon^\dagger \boldsymbol{R}\boldsymbol{\Pi}_J = \boldsymbol{C} + \boldsymbol{E}_1$$

where $\|\boldsymbol{E}_1\|_2 \le \left\|(\boldsymbol{Q}_{\boldsymbol{C}}(I_*,:)^\dagger\right\|_2 \left(\epsilon + 2\|\boldsymbol{\Delta M}\|_2 + \frac{1}{\epsilon}\|\boldsymbol{\Delta M}\|_2^2\right)$.

Next, we bound the matrix (ii). Let $\boldsymbol{E}_2 = \boldsymbol{\Delta C}(\boldsymbol{M} + \boldsymbol{\Delta M})_\epsilon^\dagger \boldsymbol{R}\boldsymbol{\Pi}_J$, then

$$\begin{aligned}
\|\boldsymbol{E}_2\|_2 &= \left\|\boldsymbol{\Delta C}(\boldsymbol{M} + \boldsymbol{\Delta M})_\epsilon^\dagger \boldsymbol{R}\boldsymbol{\Pi}_J\right\|_2 \\
&\leq \|\boldsymbol{\Delta C}\|_2 \left\|(\boldsymbol{M} + \boldsymbol{\Delta M})_\epsilon^\dagger \boldsymbol{M}\right\|_2 \\
&\leq \|\boldsymbol{\Delta C}\|_2 \left(\left\|(\boldsymbol{M} + \boldsymbol{\Delta M})_\epsilon^\dagger (\boldsymbol{M} + \boldsymbol{\Delta M})\right\|_2 + \left\|(\boldsymbol{M} + \boldsymbol{\Delta M})_\epsilon^\dagger \boldsymbol{\Delta M}\right\|_2\right) \\
&\leq \|\boldsymbol{\Delta C}\|_2 \left(1 + \frac{\|\boldsymbol{\Delta M}\|_2}{\epsilon}\right).
\end{aligned}$$

Putting everything together and setting $\boldsymbol{E}_* = \boldsymbol{E}_1 + \boldsymbol{E}_2$, we get the desired result:

$$(\boldsymbol{C} + \boldsymbol{\Delta C})(\boldsymbol{M} + \boldsymbol{\Delta M})_\epsilon^\dagger \boldsymbol{R}\boldsymbol{\Pi}_J = \boldsymbol{C} + \boldsymbol{E}_*$$

where

$$\|\boldsymbol{E}_*\|_2 \leq \left\|(\boldsymbol{Q}_{\boldsymbol{C}}(I_*, :)^\dagger\right\|_2 \left(\epsilon + 2\|\boldsymbol{\Delta M}\|_2 + \frac{1}{\epsilon}\|\boldsymbol{\Delta M}\|_2^2\right) + \|\boldsymbol{\Delta C}\|_2 \left(1 + \frac{\|\boldsymbol{\Delta M}\|_2}{\epsilon}\right).$$

□

5.5 Numerical stability of rank-deficient systems

In this section, we prove the stability result for solving rank-deficient underdetermined linear systems (Theorem 5.1.2). The result can also be extended to rank-deficient overdetermined problems; see Section 5.5.1. Theorem 5.1.2 was used to derive an expression for the computed solution to

$$\min_{\boldsymbol{x}} \left\|\left(\hat{\boldsymbol{M}}^\top\right)_\epsilon \boldsymbol{x} - [\hat{\boldsymbol{C}}]_i^\top\right\|_2$$

for each row of $\hat{\boldsymbol{C}}$. The statement and the proof is shown below. The proof follows a similar method outlined in [Nak20, Section 4.1].

Theorem 5.5.1 (Theorem 5.1.2)**.** *Consider the (rank-deficient) underdetermined linear system,*

$$\min_{\boldsymbol{x}'} \|\boldsymbol{B}_\epsilon \boldsymbol{x}' - \boldsymbol{b}\|_2 \tag{5.4}$$

where $\boldsymbol{B}_\epsilon \in \mathbb{R}^{m\times n}$ $(m \leq n)$ *is (possibly) rank-deficient* $(\text{rank}(\boldsymbol{B}_\epsilon) \leq m)$ *with singular vales larger than* ϵ *and* $\boldsymbol{b} \in \mathbb{R}^m$*. Then assuming* $\epsilon > \gamma \|\boldsymbol{B}_\epsilon\|_2$*, the minimum norm solution to Equation* (5.4) *can be computed in a backward stable manner, i.e., the computed solution* $\hat{\boldsymbol{s}}$ *satisfies*

$$\hat{\boldsymbol{s}} = (\boldsymbol{B}_\epsilon + \boldsymbol{E}_1)^\dagger (\boldsymbol{b} + \boldsymbol{E}_2)$$

where $\|\boldsymbol{E}_1\|_2 \leq \gamma \|\boldsymbol{B}_\epsilon\|_2$ *and* $\|\boldsymbol{E}_2\|_2 \leq \gamma \|\boldsymbol{b}\|_2$*.*

Proof. First, if $\boldsymbol{B}_\epsilon$ has full row-rank then the statement follows by the stability result in [Hig02, Theorem 21.4]. If $\boldsymbol{B}_\epsilon$ is rank-deficient, the stability result in [Hig02, Theorem 21.4] cannot be invoked as it is only applicable for underdetermined full-rank linear systems. So we project $\boldsymbol{B}_\epsilon$ onto its column space $\boldsymbol{Q}_\epsilon$ to make the problem an underdetermined full-rank linear system:

$$\min_{\boldsymbol{x}'} \left\|\boldsymbol{Q}_\epsilon^\top \boldsymbol{B}_\epsilon \boldsymbol{x}' - \boldsymbol{Q}_\epsilon^\top \boldsymbol{b}\right\|_2.$$

Let $\hat{\boldsymbol{Q}}_\epsilon$ be the computed column space of $\boldsymbol{B}_\epsilon$. Then $\hat{\boldsymbol{Q}}_\epsilon$ is the exact column space of $\boldsymbol{B}_\epsilon + \boldsymbol{\Delta B}$ where $\|\boldsymbol{\Delta B}\|_2 \leq \gamma \|\boldsymbol{B}_\epsilon\|_2$, $\hat{\boldsymbol{Q}}_\epsilon^\top \boldsymbol{B}_\epsilon$ is numerically full-rank with singular values larger than ϵ and $\hat{\boldsymbol{Q}}_\epsilon \hat{\boldsymbol{Q}}_\epsilon^\top \boldsymbol{B}_\epsilon = \boldsymbol{B}_\epsilon + \boldsymbol{E}$ where $\|\boldsymbol{E}\|_2 \leq \gamma \|\boldsymbol{B}_\epsilon\|_2$ [GVL13, Ch. 5.4.1]. Note the following from matrix-matrix (or matrix-vector) multiplication [Hig02, Section 3.5],

$$fl\left(\hat{\boldsymbol{Q}}_\epsilon^\top \boldsymbol{B}_\epsilon\right) = \hat{\boldsymbol{Q}}_\epsilon^\top \boldsymbol{B}_\epsilon + \boldsymbol{E}^{(1)}$$

where $\left\|\boldsymbol{E}^{(1)}\right\|_2 \leq \gamma \left\|\boldsymbol{B}_\epsilon\right\|_2$ and

$$fl\left(\hat{\boldsymbol{Q}}_\epsilon^\top \boldsymbol{b}\right) = \hat{\boldsymbol{Q}}_\epsilon^\top \boldsymbol{b} + \boldsymbol{E}^{(2)}$$

where $\left\|\boldsymbol{E}^{(2)}\right\|_2 \leq \gamma \left\|\boldsymbol{b}\right\|_2$. Since $\hat{\boldsymbol{Q}}_\epsilon^\top \boldsymbol{B}_\epsilon \in \mathbb{R}^{\mathrm{rank}(\boldsymbol{B}_\epsilon)\times n}$ is a fat rectangular matrix, we solve the following underdetermined full-rank linear system,

$$\min_{\boldsymbol{x}'} \left\|(\hat{\boldsymbol{Q}}_\epsilon^\top \boldsymbol{B}_\epsilon)\boldsymbol{x}' - \hat{\boldsymbol{Q}}_\epsilon^\top \boldsymbol{b}\right\|_2. \tag{5.10}$$

Assuming $\gamma\kappa_2(\boldsymbol{B}_\epsilon) \leq \gamma \left\|\boldsymbol{B}_\epsilon\right\|_2 / \epsilon < 1$ (where κ_2 denotes the 2-norm condition number), we obtain the computed solution of Equation (5.10), $\hat{\boldsymbol{s}}$, satisfying [Hig02, Theorem 21.4],

$$\begin{aligned}
\hat{\boldsymbol{s}} = fl\left(\left(\hat{\boldsymbol{Q}}_\epsilon^\top \boldsymbol{B}_\epsilon\right)^\dagger \hat{\boldsymbol{Q}}_\epsilon^\top \boldsymbol{b}\right) &= \left(fl(\hat{\boldsymbol{Q}}_\epsilon^\top \boldsymbol{B}_\epsilon) + \boldsymbol{E}^{(3)}\right)^\dagger fl(\hat{\boldsymbol{Q}}_\epsilon^\top \boldsymbol{b}) \\
&= \left(\hat{\boldsymbol{Q}}_\epsilon^\top \boldsymbol{B}_\epsilon + \boldsymbol{E}^{(1)} + \boldsymbol{E}^{(3)}\right)^\dagger \left(\hat{\boldsymbol{Q}}_\epsilon^\top \boldsymbol{b} + \boldsymbol{E}^{(2)}\right) \\
&= \left(\hat{\boldsymbol{Q}}_\epsilon \hat{\boldsymbol{Q}}_\epsilon^\top \boldsymbol{B}_\epsilon + \hat{\boldsymbol{Q}}_\epsilon \boldsymbol{E}^{(1)} + \hat{\boldsymbol{Q}}_\epsilon \boldsymbol{E}^{(3)}\right)^\dagger \left(\boldsymbol{b} + \hat{\boldsymbol{Q}}_\epsilon \boldsymbol{E}^{(2)}\right) \\
&= \left(\boldsymbol{B}_\epsilon + \boldsymbol{E} + \hat{\boldsymbol{Q}}_\epsilon \boldsymbol{E}^{(1)} + \hat{\boldsymbol{Q}}_\epsilon \boldsymbol{E}^{(3)}\right)^\dagger \left(\boldsymbol{b} + \hat{\boldsymbol{Q}}_\epsilon \boldsymbol{E}^{(2)}\right)
\end{aligned}$$

where $\left\|\boldsymbol{E}^{(3)}\right\|_2 \leq \gamma \left\|\boldsymbol{B}_\epsilon\right\|_2$ and we use $(\boldsymbol{QA})^\dagger \boldsymbol{b} = \boldsymbol{A}^\dagger \boldsymbol{Q}^\top \boldsymbol{b}$ for a tall orthonormal matrix $\boldsymbol{Q}$ for the third equality. Therefore, assuming that $\gamma \left\|\boldsymbol{B}_\epsilon\right\|_2 < \epsilon$, the computed solution of 5.4, $\hat{\boldsymbol{s}}$ satisfies

$$\hat{\boldsymbol{s}} = fl(\boldsymbol{B}_\epsilon^\dagger \boldsymbol{b}) = \left(\boldsymbol{B}_\epsilon + \boldsymbol{E}_1\right)^\dagger \left(\boldsymbol{b} + \boldsymbol{E}_2\right),$$

where $\left\|\boldsymbol{E}_1\right\|_2 \leq \gamma \left\|\boldsymbol{B}_\epsilon\right\|_2$ and $\left\|\boldsymbol{E}_2\right\|_2 \leq \gamma \left\|\boldsymbol{b}\right\|_2$. □

5.5.1 Extension to rank-deficient overdetermined problems

Theorem 5.1.2 can be extended to include rank-deficient overdetermined least-squares problems. Consider the rank-deficient overdetermined least-squares problem,

$$\min_{\boldsymbol{x}'} \|\boldsymbol{C}_\epsilon \boldsymbol{x}' - \boldsymbol{b}\|_2$$

where $\boldsymbol{C}_\epsilon \in \mathbb{R}^{m\times n}$ $(m \geq n)$ is a rank-deficient $(\text{rank}(\boldsymbol{C}_\epsilon) < n)$, tall matrix with singular vales larger than ϵ and $\boldsymbol{b} \in \mathbb{R}^m$. Then under the same assumption as Theorem 5.1.2, i.e., $\epsilon > \gamma \|\boldsymbol{C}_\epsilon\|_2$, the solution to the overdetermined least-squares problem can be computed in a backward stable manner.

The proof proceeds in the same way as Theorem 5.1.2, as once the tall matrix $\boldsymbol{C}_\epsilon$ is projected onto its column space , say $\boldsymbol{W}_\epsilon$, $\boldsymbol{W}_\epsilon^\top \boldsymbol{C}_\epsilon \in \mathbb{R}^{\text{rank}(\boldsymbol{C}_\epsilon)\times n}$ becomes a fat full-rank matrix, and we are now solving the underdetermined full-rank linear system: $\min_{\boldsymbol{x}'} \left\| \left(\hat{\boldsymbol{W}}_\epsilon^\top \boldsymbol{C}_\epsilon \right) \boldsymbol{x}' - \hat{\boldsymbol{W}}_\epsilon^\top \boldsymbol{b} \right\|_2$. The remainder of the proof proceeds in the same manner as that of Theorem 5.1.2.

6

Oversampling for CUR decomposition

The vast majority of CUR decompositions of $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ comes with a theoretical guarantee that involves the term

$$\left\|\boldsymbol{W}(J,:)^{-1}\right\|_2 = \frac{1}{\sigma_{\min}(\boldsymbol{W}(J,:))}, \tag{6.1}$$

where $\boldsymbol{W} \in \mathbb{R}^{n\times k}$ is the k (approximate) dominant singular vectors of $\boldsymbol{A}$ and $J \subset [n]$ with $|J| = k$ is a set of indices; see Theorems 2.4.1, 5.1.1 and 6.4.1. Equation (6.1) is usually the deciding factor for the accuracy of the CUR decomposition and therefore, a majority of the algorithms aim at choosing a set of indices J that would diminish the effect of Equation (6.1). A natural way of improving Equation (6.1) is to oversample, that is, obtain extra indices $J_0 \in \mathbb{R}^p$ distinct from J so that $\boldsymbol{W}(J \cup J_0, :) \in \mathbb{R}^{(k+p)\times k}$ becomes a rectangular matrix. By

appending more rows to $\boldsymbol{W}(J,:)$, $\boldsymbol{W}(J \cup J_0,:)$ has a larger minimum singular value by the Courant-Fischer min-max theorem, which improves the accuracy of the CUR decomposition. The topic of oversampling is not new, in particular, it is shown in [ADM$^+$15] that oversampling improves the accuracy of CUR-BA when the singular values decay rapidly.

In the context of sampling based methods, oversampling has been suggested for theoretical guarantees; see e.g. [MD09, CD13, BW17]. It is easy to oversample for sampling based methods as we can simply sample more than k indices from the given probability distribution. In contrast, it is often difficult to oversample for pivoting based methods. This is because we typically perform pivoting schemes on a smaller-sized surrogate $\boldsymbol{X} \in \mathbb{R}^{n\times k}$, for example, the sketch of $\boldsymbol{A}$, and the pivots beyond the first k indices carry little to no information. Therefore, for pivoting based methods, usually a different strategy is used for oversampling. In the context of DEIM, various ways of oversampling have been suggested [CFCA13, ZW16, MO18, PDG20, GH22]. Specifically, Zimmermann and Willcox [ZW16] show that oversampling can improve the condition number of oblique projections, and Donello et al. [DPN$^+$23] proves a bound for DEIM projectors with oversampling, which extends the proof without oversampling from [SE16]. The DEIM projector is given by $\boldsymbol{\mathcal{P}}_{\boldsymbol{Q},\boldsymbol{\Pi}_I} = \boldsymbol{Q}(\boldsymbol{\Pi}_I^\top \boldsymbol{Q})^\dagger \boldsymbol{\Pi}_I^\top$ where $\boldsymbol{Q} \in \mathbb{R}^{n\times k}$ is an orthonormal matrix approximating the range of the original matrix. A majority of the aforementioned literature focuses on the DEIM projector or the CUR-BA. However, oversampling can often be more effective for CUR with cross approximation (CUR-CA), as it not only improves the bound involving eq. (6.1) (see Theorem 5.1.1), but also transforms the core matrix $\boldsymbol{A}(I,J)$ into a rectangular matrix when either I or J (but not both) is oversampled. This generally improves the condition number of $\boldsymbol{A}(I,J)$ making the CUR-CA more accurate when computing its pseudoinverse. In contrast, the benefits of oversampling is less immediate for CUR with best approximation (CUR-BA); we explore this further in Section 6.4. Overall, we advocate oversampling when possible over the standard choice $|I| = |J|$.

In this chapter, we therefore investigate how to choose those extra indices. Concretely,

given initial index sets I and J of size $|I| = |J| = k$ where k is the target rank, we ask how best to augment one of them by p additional indices where we recommend p to be in proportion to the target rank; see also Section 6.3.2.[1] To answer this, we propose a deterministic way of oversampling for which the rationale for our idea can be explained by the cosine-sine (CS) decomposition, which identifies and appends those indices that most strengthen the smallest singular directions of the chosen submatrix. The construction could be of use in other contexts where (over)sampling from a subspace is desired; a common task that finds use, for example, in model reduction [CS10] and active learning [SCMW24].

6.1 Existing methods.

We review two existing algorithms for oversampling. First, Gidisu and Hochstenbach [GH22] oversample p extra indices by choosing the largest p leverage scores (row norms of (approximate) dominant singular vectors) out of the unchosen indices. The complexity is $\mathcal{O}(nk)$ for computing the leverage scores of an orthonormal matrix $\boldsymbol{W} \in \mathbb{R}^{n\times k}$. If we are only given an approximator that is not orthonormal then (approximate) orthonormalisation needs to be done, usually at a cost of $\mathcal{O}(nk^2)$. Second, Peherstorfer, Drmač and Gugercin [PDG20], iteratively select p extra indices for oversampling to maximize the minimum singular value of $\boldsymbol{W}(J,:)$ in a greedy fashion. This approach, which is called the `GappyPOD+E` algorithm, uses perturbation bounds on the eigenvalues given in [IN09] to find the next index that maximizes the lower bound for the minimum singular value of $\boldsymbol{W}(J,:)$. This approach is also a special case of [ZW16]. The algorithm runs with complexity $\mathcal{O}((k+p)^2k^2 + nk^2p)$ where p is the number of indices we oversample by. Again, if $\boldsymbol{W}$ is not orthonormal to begin with, then (approximate) orthonormalisation needs to be done usually at a cost of $\mathcal{O}(nk^2)$. The topic of approximate

[1]The recommendation that p is chosen to be proportion to the target rank is motivated by the generalised Nyström method. In the generalised Nyström method, the sketch size of the second random embedding is chosen to be proportionally larger than the first random embedding for stability [Nak20].

orthonormalisation are discussed in [AMT10, BG22].

6.2 Oversampling for the CUR-CA

In Chapter 5, we proved theoretical results involving the CUR-CA (Corollary 5.1.1), the stabilized CUR-CA (Theorem 5.1.1) and the stabilized CUR-CA in the presence of rounding errors (Theorem 5.1.3). All of the results involved an oversampling parameter p and an extra set of row indices I_0, and bounding $\left\|\boldsymbol{Q}_{\boldsymbol{C}}(I \cup I_0, :)^{\dagger}\right\|_2$ was important for the accuracy and stability of the CUR-CA. In this section, we discuss oversampling in the context of the CUR-CA and devise an algorithm that naturally arises from the discussion.

We first describe the setting. Suppose we have obtained the row indices I and the column indices J with $|I| = |J| = k$ by applying some algorithm, for example, the ones discussed in Section 2.4, on a row space approximator $\boldsymbol{X} \in \mathbb{R}^{k \times n}$ of $\boldsymbol{A} \in \mathbb{R}^{m \times n}$.[2] To have a concrete algorithm in mind for getting the set of indices I and J, we present pivoting on a random sketch [VM17, DG20, DM23] below in Algorithm 5.

Algorithm 5 Pivoting on a random sketch ([DM23, Algorithm 1])

Input: $\boldsymbol{A} \in \mathbb{R}^{m \times n}$, target rank k (typically $k \ll \min\{m, n\}$)
Output: Column indices J and row indices I with $|I| = |J| = k$

`function` $[I, J] =$ `Rand_Pivot`$(\boldsymbol{A}, k)$
1: Draw a random embedding $\boldsymbol{S} \in \mathbb{R}^{m \times k}$.
2: Set $\boldsymbol{X} = \boldsymbol{S}^{\top}\boldsymbol{A} \in \mathbb{R}^{k \times n}$, a row sketch of $\boldsymbol{A}$
3: Apply CPQR on $\boldsymbol{X}$. Let J be the k column pivots.
4: Apply CPQR on $\boldsymbol{A}(:, J)^{\top}$. Let I be the k row pivots.

Algorithm 5 is a version of Algorithm 1 from [DM23], which selects the column indices first by applying column pivoted QR (CPQR) on the row sketch $\boldsymbol{X} = \boldsymbol{S}^{\top}\boldsymbol{A}$ and then selects

[2] A similar version for column space approximator $\boldsymbol{Y}$ can also be devised by considering $\boldsymbol{A}^{\top}$ instead. In the case when $\boldsymbol{X}$ and $\boldsymbol{Y}$ are not available, for example, when the initial set of indices were obtained using uniform sampling, $\boldsymbol{R} = \boldsymbol{A}(I, :)$ and $\boldsymbol{C} = \boldsymbol{A}(:, J)$ can be used as the row space approximator and the column space approximator, respectively.

the row indices by applying CPQR on the chosen columns $\boldsymbol{A}(:, J)^\top$. LU with partial pivoting is also effective in practice [DM23]. Algorithm 5 is an example where we obtain the row indices from the already-chosen column indices, which was recommended in the previous chapter for the accuracy and stability of CUR-CA. Here, the row space approximator is the row sketch $\boldsymbol{X} = \boldsymbol{S}^\top \boldsymbol{A}$ and the column space approximator is the columns $\boldsymbol{A}(:, J)$. A bound for the CUR-CA (see Corollary 5.1.1) without oversampling is given by

$$\|\boldsymbol{A} - \boldsymbol{A}_{IJ}\|_{\mathrm{F}} \leq \left\|\boldsymbol{Q}_{\boldsymbol{C}}(I,:)^{-1}\right\|_2 \left\|\boldsymbol{Q}_{\boldsymbol{X}}(J,:)^{-1}\right\|_2 \left\|\boldsymbol{A} - \boldsymbol{A}\boldsymbol{X}^\dagger \boldsymbol{X}\right\|_{\mathrm{F}}, \tag{6.2}$$

where $\boldsymbol{A}_{IJ} = \boldsymbol{A}(:, J)\boldsymbol{A}(I, J)^\dagger \boldsymbol{A}(I, :)$ is the CUR-CA, and $\boldsymbol{Q}_{\boldsymbol{C}}$ and $\boldsymbol{Q}_{\boldsymbol{X}}$ are orthonormal matrices spanning the columns of $\boldsymbol{C}$ and $\boldsymbol{X}^\top$ respectively.

Many existing algorithms focus on minimizing the first two terms on the right-hand side of Equation (6.2) as they control the accuracy of the CUR-CA. In the CUR-CA and the other CUR decompositions such as the CUR-BA, $\boldsymbol{C}(\boldsymbol{C}^\dagger \boldsymbol{A}\boldsymbol{R}^\dagger)\boldsymbol{R}$, we often take $|I| = |J|$, however, without increasing the overall rank of the approximation, we can oversample either I or J. Oversampling both the row indices I and the column indices J independently is not recommended, which we discuss further in Section 6.3.2. Suppose we oversample the rows to $I_* := I \cup I_0$ where $|I_0| = p$ is an extra set of row indices for oversampling. Then the first term of the bound changes from $\|\boldsymbol{Q}_{\boldsymbol{C}}(I,:)^{-1}\|_2$ to $\left\|\boldsymbol{Q}_{\boldsymbol{C}}(I_*,:)^\dagger\right\|_2$ (see Theorem 5.1.1). By the Courant-Fischer min-max theorem $\sigma_{\min}(\boldsymbol{Q}_{\boldsymbol{C}}(I_*,:)) \geq \sigma_{\min}(\boldsymbol{Q}_{\boldsymbol{C}}(I,:))$, which improves the bound in Equation (6.2) as $\left\|\boldsymbol{Q}_{\boldsymbol{C}}(I_*,:)^\dagger\right\|_2 \leq \|\boldsymbol{Q}_{\boldsymbol{C}}(I,:)^{-1}\|_2$. Now, to maximize the effect of oversampling, we ought to find unchosen indices that enrich the trailing singular subspace of $\boldsymbol{Q}_{\boldsymbol{C}}(I,:)$, which in turn increases the minimum singular value(s) of $\boldsymbol{Q}_{\boldsymbol{C}}(I,:)$. It turns out that we can achieve this by projecting $\boldsymbol{Q}_{\boldsymbol{C}}([m] - I,:)$ onto the trailing singular subspace of $\boldsymbol{Q}_{\boldsymbol{C}}(I,:)$ and use a good row selection algorithm to choose p extra rows. Before stating the algorithm based on this observation, we first motivate our rationale using the cosine-sine (CS) decomposition.

Definition 6.2.1 (CS Decomposition; [PW94, GVL13])**.** *Let* $\boldsymbol{Q} \in \mathbb{R}^{n\times k}$ *be any orthonormal matrix with partition*

$$\boldsymbol{Q} = \begin{matrix} \\ \boldsymbol{Q}_1 \\ \boldsymbol{Q}_2 \end{matrix} \begin{matrix} k \\ \begin{bmatrix} \\ \\ \end{bmatrix} \\ \begin{matrix} n_1 \\ n_2 \end{matrix} \end{matrix}$$

where $n_1, n_2 \geq k$ *with* $n_1 + n_2 = n$*. Then there exists orthonormal matrices* $\boldsymbol{U}_1 \in \mathbb{R}^{n_1\times k}$, $\boldsymbol{U}_2 \in \mathbb{R}^{n_2\times k}, \boldsymbol{V} \in \mathbb{R}^{k\times k}$ *such that*

$$\boldsymbol{Q}_1 = \boldsymbol{U}_1\boldsymbol{C}_{\boldsymbol{Q}}\boldsymbol{V}^\top \text{ and } \boldsymbol{Q}_2 = \boldsymbol{U}_2\boldsymbol{S}_{\boldsymbol{Q}}\boldsymbol{V}^\top, \tag{6.3}$$

where $\boldsymbol{C}_{\boldsymbol{Q}} = \mathrm{diag}(c_1, c_2, ..., c_k), \boldsymbol{S}_{\boldsymbol{Q}} = \mathrm{diag}(s_1, s_2, ..., s_k)$ *are diagonal matrices satisfying* $c_i, s_i \geq 0$*, and* $c_i^2 + s_i^2 = 1$ *for all* i*.*

In our context, we can view $\boldsymbol{Q}_{\boldsymbol{C}}(I, :)$ as $\boldsymbol{Q}_1$, and $\boldsymbol{Q}_{\boldsymbol{C}}([m] - I, :)$ as $\boldsymbol{Q}_2$. Now assume, without loss of generality, that the c_i's are in non-increasing order so the s_i's are in non-decreasing order. Then in order to increase the minimum singular value of $\boldsymbol{Q}_1$, we could add the rows of $\boldsymbol{Q}_2$ that contribute the most to the trailing right singular subspace of $\boldsymbol{Q}_1$, i.e., $\boldsymbol{V}_{-p} = \boldsymbol{V}(:, k-p+1:k) \in \mathbb{R}^{k\times p}$. By Equation (6.3), $\boldsymbol{V}_{-p}$ is also the dominant right singular subspace of $\boldsymbol{Q}_2$ as $c_i^2 + s_i^2 = 1$. When we add the rows of $\boldsymbol{Q}_2$ that lie in the subspace spanned by $\boldsymbol{V}_{-p}$ to $\boldsymbol{Q}_1$, we can increase the minimum singular value(s) of $\boldsymbol{Q}_1$, i.e., increase c_k or the last few c_i's. Therefore we can apply any algorithm (such as those discussed in Section 2.4) that finds good row indices on $\boldsymbol{Q}_2\boldsymbol{V}_{-p} \in \mathbb{R}^{n_2\times p}$ and append them to $\boldsymbol{Q}_1$ to increase the minimum singular value of $\boldsymbol{Q}_1$.

If the algorithm requires the dominant left singular vectors of $\boldsymbol{Q}_2\boldsymbol{V}_{-p}$ such as in DEIM or leverage scores sampling, we can simply scale the columns of $\boldsymbol{Q}_2\boldsymbol{V}_{-p}$ using the singular

values of $\boldsymbol{Q}_1$, $\boldsymbol{C_Q} = \operatorname{diag}(c_1, ..., c_k)$ because

$$\boldsymbol{Q}_2 \boldsymbol{V}_{-p} = \boldsymbol{U}_{2,-p} \operatorname{diag}(s_{k-p+1}, ..., s_k) = \boldsymbol{U}_{2,-p} \sqrt{1 - \operatorname{diag}(c_{k-p+1}^2, ..., c_k^2)}$$

where $\boldsymbol{U}_{2,-p} = \boldsymbol{U}_2(:, k-p+1:k)$.

In light of the observation made above, we propose the following algorithm (Algorithm 6) for oversampling indices. For simplicity, we use column pivoted QR (CPQR) on $\boldsymbol{Q}_2 \boldsymbol{V}_{-p}$ to obtain a good set of oversampling indices. However, any algorithm that finds a good set of rows can replace line 4 of Algorithm 6.

Algorithm 6 Oversampling indices

Input: A full column rank matrix $\boldsymbol{B} \in \mathbb{R}^{n \times k}$ with $n \geq k$, an index set I with $|I| = k$ and an oversampling parameter $p \leq k$

Output: Extra indices I_0 with $|I_0| = p$

`function` $I_0 = \texttt{OS}(\boldsymbol{B}, I, p)$

1: $[\boldsymbol{Q_B}, \sim] = \texttt{qr}(\boldsymbol{B})$, ▷ Skip this step if $\boldsymbol{B}$ is already orthonormal.

2: $[\sim, \sim, \boldsymbol{V}] = \texttt{svd}(\boldsymbol{Q_B}(I, :))$,

3: Set $\boldsymbol{V}_{-p} = \boldsymbol{V}(:, k-p+1:k)$, the trailing p right singular vectors of $\boldsymbol{Q_B}(I, :)$.

4: Apply CPQR on $(\boldsymbol{Q_B}([m] - I, :)\boldsymbol{V}_{-p})^\top$. Let I_0 be p oversampled indices.

Given a full column rank matrix $\boldsymbol{B}$, an index set I, and an oversampling parameter $p(\leq k)$, Algorithm 6 finds an extra set of indices for oversampling by projecting $\boldsymbol{Q_B}$ onto the unchosen indices $[m] - I$ from the left and the trailing p right singular vectors of $\boldsymbol{Q_B}(I, :)$ from the right and performing CPQR to obtain the extra indices I_0 with $|I_0| = p$. When $p > k$, we can iterate Algorithm 6 to obtain at most k oversampling indices at each iteration. We can also devise a version of Algorithm 6 with a tolerance parameter $0 < \epsilon < 1$ rather than taking p as input; for example $\epsilon = \sqrt{k/n}$ may be a reasonable choice (given that singular values of a $\mathcal{O}(k) \times k$ submatrix of an $n \times n$ Haar distributed orthogonal matrix are of this order [Mec19]). This version uses projection onto the trailing right singular subspace of $\boldsymbol{Q_B}(I, :)$ corresponding to the singular values that are less than ϵ. While this version

does not guarantee $\sigma_{\min}\left(\boldsymbol{Q_B}(I \cup I_0, :)\right) \geq \epsilon$, it can be applied iteratively to achieve this. To guarantee $\sigma_{\min}\left(\boldsymbol{Q_B}(I \cup I_0, :)\right) \geq \epsilon$, one can alternatively use the `GappyPOD+E` oversampling algorithm [PDG20]. However, this approach comes with a higher computational cost. The complexity of Algorithm 6 is $\mathcal{O}(nk^2)$ where the dominant cost comes from taking the QR decomposition of B (line 1). If B were orthonormal to begin with, then the dominant cost comes from forming $\boldsymbol{Q_B}([m] - I, :)\boldsymbol{V}_{-p}$, which costs $\mathcal{O}(nkp)$.

In the analysis for the CUR-CA from the previous chapter (Section 5.1), oversampling played two key roles: (i) improving the accuracy of CUR-CA (see Theorem 5.1.1, Corollary 5.1.1, Theorem 5.1.3); and (ii) enhancing the stability of CUR-CA in the presence of roundoff errors. Note that the term $\|\boldsymbol{Q_C}(I, :)^{-1}\|_2$ also appears in front of the roundoff parameter γ in Theorem 5.1.3. Therefore, oversampling is crucial–particularly when the initially selected index sets are suboptimal, as illustrated in Figure 6.1. For instance, if the core matrix $\boldsymbol{A}(I, J)$ is (nearly) singular, oversampling helps mitigate the resulting instability.

Oversampling should be done in such a way that the term $\|\boldsymbol{Q_C}(I, :)^{-1}\|_2$ is reduced further by adding an extra set of indices I_0 that lie in the trailing singular subspace of $\boldsymbol{Q_C}(I, :)$. Algorithm 6 achieves this by picking good unchosen indices from the trailing singular subspace of $\boldsymbol{Q_C}$. We further highlight the significance of oversampling through numerical illustrations in the next section, Section 5.3.

6.3 Numerical illustrations

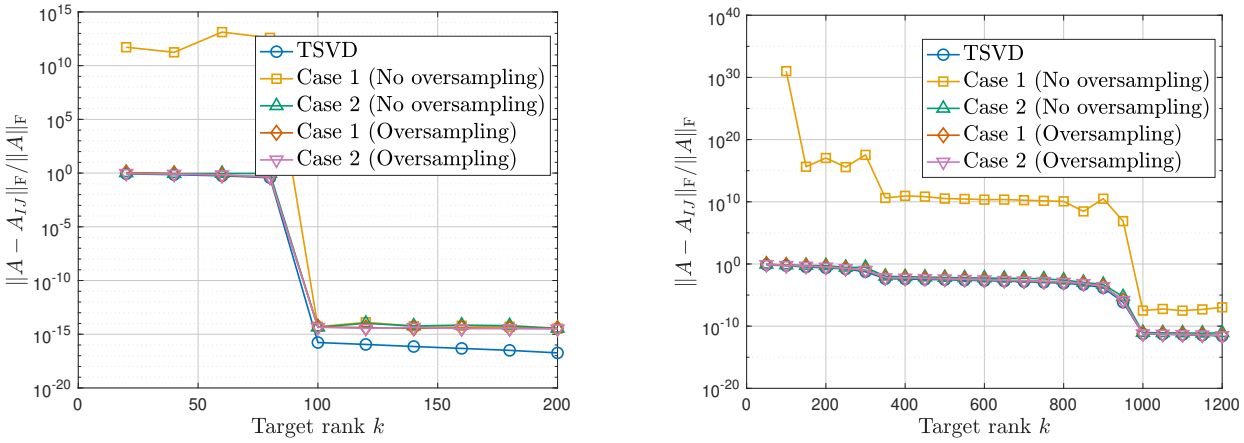
In this section, we illustrate the concepts discussed in the previous sections through numerical experiments. We first show that oversampling can improve the quality when the indices are poorly selected. In particular, we show that when oversampling is done, we can improve the accuracy of CUR-CA in Figure 5.2 when the row and column indices are suboptimal. Additionally, we also illustrate the effectiveness of the oversampling algorithm,

Algorithm 6, for the CUR-CA and show its competitiveness against some existing methods. In all the experiments, the best rank-k approximation error using the truncated SVD (TSVD) is used as reference.

6.3.1 Oversampling for suboptimal indices

We first show that oversampling can improve the quality when the indices are poorly selected. We run the same experiment as in Section 5.3.2 for Figure 5.2, but we now include oversampling.

In Figure 6.1, we repeat the experiment from Figure 5.2, this time also incorporating row oversampling with $p = k$. We observe that the poor approximation quality in Case 1 (no oversampling) can be improved for both plots through sufficient oversampling, as demonstrated in Case 1 (oversampling). This suggests that when a suboptimal set of indices is selected–or simply as a safeguard–it is advisable to perform adequate oversampling in the CUR-CA to ensure both accuracy and stability.



(a) Block random matrix (5.9) **(b)** `nnc1374`

Figure 6.1: Relationship between the rows and columns in the CUR-CA. In Case 1, the rows and columns are chosen independently from one another and in Case 2, we select the columns first and then the rows were computed from the selected columns. When the rows and columns are chosen independently (Case 1), the resulting approximation can be catastrophic. With sufficient row oversampling ($p = k$), the approximation quality is improved.

6.3.2 Oversampling algorithm comparison

In this section, we illustrate advantages of oversampling through numerical experiments. In all experiments, unless stated otherwise, we use pivoting on a random sketch (Algorithm 5) with column pivoted QR [VM17, DM23] and the Gaussian sketch to get the initial set of k row and column indices, where k is the target rank. We then obtain the oversampling indices using the following algorithms:

- `OS + P`: Algorithm 6,
- `OS + L`: Choose p extra indices corresponding to the largest p leverage scores out of the unchosen indices as in [GH22],
- `OS + E`: `GappyPOD + E` algorithm in [PDG20].

We set the number of oversampling indices to be $p = 0$, $p = 10$ and $p = 0.5k$.

We consider three different classes of test matrices, which are summarized below:

1. `CIFAR10`: The CIFAR-10 training set [Kri09] consists of 60000 images of size $32 \times 32 \times 3$. We choose 10000 random images, each flattened to a vector, and treat it as a 10000×3072 data matrix.
2. `YaleFace64x64`: Yale face is a full-rank dense matrix of size 165×4096 consisting of 165 face images each of size 64×64. The flattened image vectors are centered and normalized such that the mean is zero and the entries lie within $[-1, 1]$.
3. `SNN`: Random sparse non-negative matrices are test matrices used in [SE16, VM17] that is given by,

$$\texttt{SNN} = \sum_{j=1}^{r} s_j \boldsymbol{x}_j \boldsymbol{y}_j^\top,$$

where $s_1 \geq \cdots \geq s_r > 0$ and $\boldsymbol{x}_j \in \mathbb{R}^m, \boldsymbol{y}_j \in \mathbb{R}^n$ are random sparse vectors with non-negative entries. We take $m = 100000, n = 300$ with

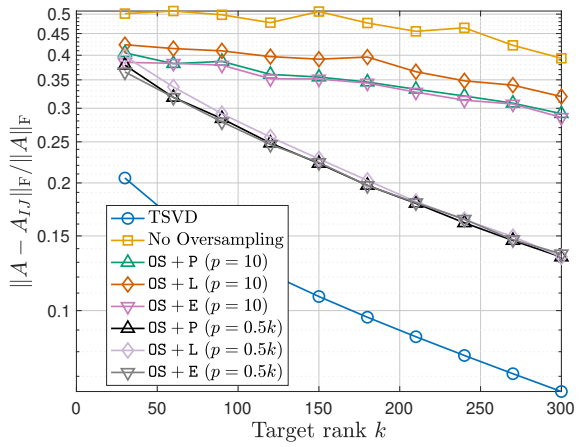
$$\sum_{j=1}^{50} \frac{2}{j} \boldsymbol{x}_j \boldsymbol{y}_j^\top + \sum_{j=51}^{300} \frac{1}{j} \boldsymbol{x}_j \boldsymbol{y}_j^\top,$$

where the sparse vectors $\boldsymbol{x}_j$'s and $\boldsymbol{y}_j$'s are computed in MATLAB using the command `sprand`$(m, 1, 0.025)$ and `sprand`$(n, 1, 0.025)$, respectively.

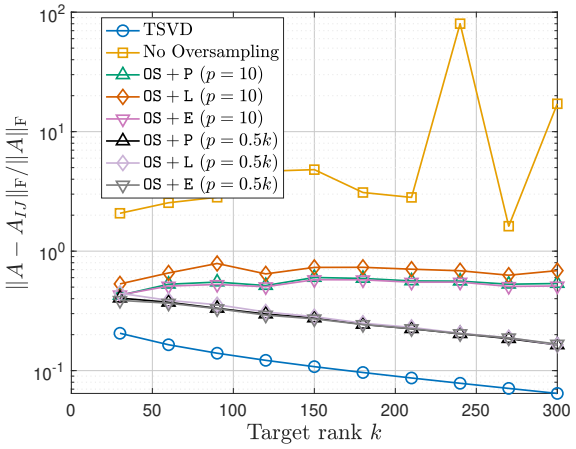
The results are depicted in Figure 6.2. We compare different oversampling algorithms for the three test matrices. We use pivoting on a random sketch (Algorithm 5) to obtain the initial set of indices except for Figure 6.2b where we use uniform sampling; pivoting on a random sketch usually results in a good set of indices, while uniform sampling can give a poor set of indices. Then we oversample the row indices using the three oversampling algorithms, `OS+P`, `OS+L` and `OS+E`. We also oversample column indices in Figure 6.2c. For Figure 6.2c, $p = 0.5k$ for row oversampling and $p = 10$ for column oversampling.

We begin with the top two plots involving the `CIFAR10` dataset, Figures 6.2a and 6.2b. We observe that as the oversampling parameter increases, the accuracy improves. The different oversampling techniques yield a similar result with `OS+L` being usually slightly worse. In Figures 6.2a and 6.2b, we observe that oversampling plays a big role for the accuracy of the CUR-CA. In Figure 6.2a, $p = 10$ improves the accuracy slightly, but when the oversampling parameter p becomes proportional to the target rank, we make further progress and we approximately capture the singular value decay rate. In Figure 6.2b, when the initial set of indices are worse, as it can be when we perform uniform sampling, we observe that oversampling makes the approximation more robust even when the CUR-CA without oversampling yields unstable results. Therefore, oversampling helps the CUR-CA yield more accurate and stable results.

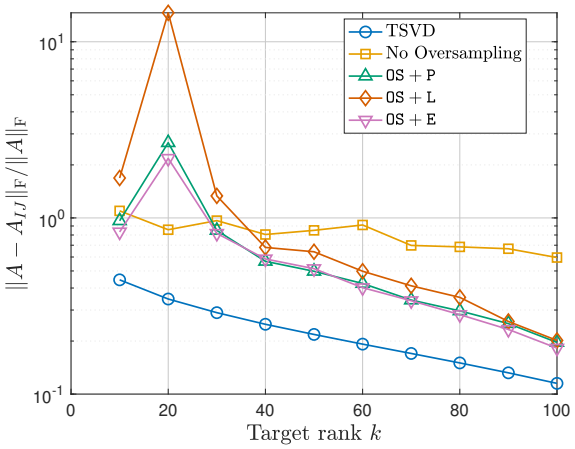
In Figure 6.2c, using the `YaleFace64x64` dataset, we show what happens when we



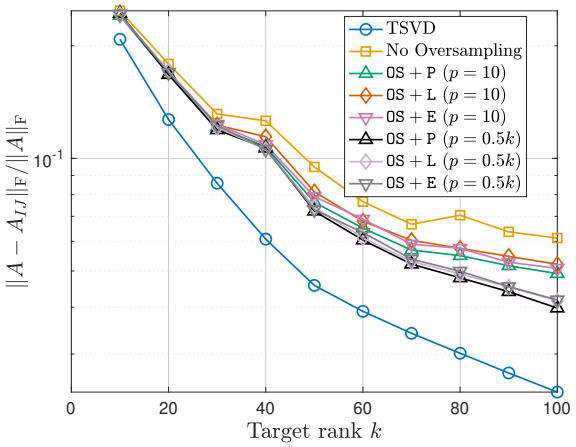
(a) `CIFAR10` using `Rand_Pivot` with row oversampling



(b) `CIFAR10` using uniform sampling



(c) `YaleFace64x64` using `Rand_Pivot` with row ($p = 0.5k$) and column ($p = 10$) oversampling



(d) `SNN` using `Rand_Pivot` with row oversampling

Figure 6.2: Comparison of different oversampling methods for various test matrices. We use pivoting on a random sketch to obtain the initial set of indices except for Figure 6.2b where we use uniform sampling. Then we oversample the row indices using the three oversampling algorithms, `OS+P`, `OS+L` and `OS+E`. We also oversample column indices in Figure 6.2c to demonstrate that oversampling both row and column indices can be harmful.

oversample both row and column indices. In this experiment, we oversample the row indices by $p = 0.5k$ and the column indices by $p = 10$. In Figure 6.2c, the CUR-CA can get worse with oversampling and the approximation may become unstable when both column and row indices are oversampled. This is caused by the core matrix $\boldsymbol{A}(I \cup I_0, J \cup J_0)$ underestimating the singular values of $\boldsymbol{A}$ as oversampling in only one of row or column indices improve the condition number of the core matrix, but when we oversample both row and column indices, condition number of the core matrix can become worse. This is similar to the phenomenon happening in Section 5.3.2, where the CUR-CA can yield poor accuracy when we choose the rows and columns independently. Therefore, we advocate oversampling in only one of row or column indices for the CUR-CA. Note that if we do not oversample column indices in Figure 6.2c, so that we only oversample rows for the `YaleFace64x64` dataset, the relative error decreases steadily, as observed in the other figures in Figure 6.2. In Figure 6.2d, we use the `SNN` dataset. The effect of oversampling is less immediate in this example, but the accuracy still improves and the decay rate is more in line with the spectral decay.

Lastly, in all the experiments in Figure 6.2, we observe that our algorithm for oversampling `OS + P` (Algorithm 6) is competitive with `OS + E` with `OS + L` usually being slightly worse. Therefore, Algorithm 6, which runs with complexity $\mathcal{O}(nk^2 + nkp)$ is competitive with `OS+E`, which run with complexity $\mathcal{O}(nk^2p + k^4)$.

6.4 Oversampling analysis of CUR-BA

In this section, we analyse CUR-BA (CUR with Best Approximation, Section 2.4),

$$\boldsymbol{A}_{IJ}^{(BA)} = \boldsymbol{A}(:, J) \left(\boldsymbol{A}(:, J)^{\dagger} \boldsymbol{A} \, \boldsymbol{A}(I, :)^{\dagger} \right) \boldsymbol{A}(I, :) = \boldsymbol{C}(\boldsymbol{C}^{\dagger} \boldsymbol{A} \boldsymbol{R}^{\dagger}) \boldsymbol{R},$$

with oversampling.

We first prove a relative norm bound for the CUR-BA with oversampling. We make the following standard assumptions, which is analogous to the CUR-CA counterpart in Section 5.1 (Assumption 5.1.1).

Assumption 6.4.1.

1. $|I| = |J| = k$ *where* k *is the target rank. The oversampling indices are* I_0 *with* $|I_0| = p_1$ *for the rows and* J_0 *with* $|J_0| = p_2$ *for the columns and they satisfy* $k + \max\{p_1, p_2\} \leq \operatorname{rank}(\boldsymbol{A})$.

2. $\boldsymbol{A}(:, J \cup J_0)$ *has full column rank and* $\boldsymbol{A}(I \cup I_0, :)$ *has full row rank.*

3. $\boldsymbol{X}(:, J \cup J_0) \in \mathbb{R}^{k \times (k+p_2)}$ *has full row rank, where* $\boldsymbol{X} \in \mathbb{R}^{k \times n}$ *is a row space approximator of* $\boldsymbol{A}$.

4. $\boldsymbol{Y}(I \cup I_0, :) \in \mathbb{R}^{(k+p_1) \times k}$ *has full column rank, where* $\boldsymbol{Y} \in \mathbb{R}^{m \times k}$ *is a column space approximator of* $\boldsymbol{A}$.

The assumption that $\boldsymbol{X}(:, J)$ and $\boldsymbol{A}(I, :)$ having a full row rank and $\boldsymbol{Y}(I, :)$ and $\boldsymbol{A}(:, J)$ having a full column rank is generic, since most methods that pick good row and column indices satisfy this assumption. If one of $\boldsymbol{X}$ or $\boldsymbol{Y}$ are not available then $\boldsymbol{A}(I, :)$ or $\boldsymbol{A}(:, J)$ can be used instead after obtaining the row or column indices from $\boldsymbol{Y}$ or $\boldsymbol{X}$, respectively.

We now prove the result for the CUR-BA with oversampling. The results shown below is a simple extension of [SE16, Lemma 4.2] and [DM23, Theorem 1]. Lemma 6.4.1 considers one-sided projection with oversampling where we project $\boldsymbol{A}$ onto the chosen columns of $\boldsymbol{A}$. In Theorem 6.4.1, we consider the CUR-BA, $\boldsymbol{C}(\boldsymbol{C}^\dagger \boldsymbol{A} \boldsymbol{R}^\dagger)\boldsymbol{R}$ where we project $\boldsymbol{A}$ onto the chosen rows and columns of $\boldsymbol{A}$.

Lemma 6.4.1. *Under Assumption 6.4.1,*

$$\left\|\boldsymbol{A} - \boldsymbol{C}\boldsymbol{C}^\dagger \boldsymbol{A}\right\| \leq \left\|\boldsymbol{Q}_{\boldsymbol{X}}(J \cup J_0, :)^\dagger\right\|_2 \left\|\boldsymbol{A} - \boldsymbol{A}\boldsymbol{X}^\dagger \boldsymbol{X}\right\|$$

where $\|\cdot\|$ is any unitarily invariant norm and $\boldsymbol{Q}_{\boldsymbol{X}} \in \mathbb{R}^{n\times k}$ is an orthonormal matrix spanning the columns of $\boldsymbol{X}^\top$.

Proof. For shorthand let $J_* = J \cup J_0$ and let $\boldsymbol{\Pi}_{J_*} = \boldsymbol{I}_n(:, J_*) \in \mathbb{R}^{n\times(k+p)}$ such that $\boldsymbol{C} = \boldsymbol{A}(:, J_*) = \boldsymbol{A}\boldsymbol{\Pi}_{J_*}$. We first define two oblique projectors

$$\boldsymbol{\mathcal{P}}_{\boldsymbol{X}} := \boldsymbol{\Pi}_{J_*}(\boldsymbol{X}\boldsymbol{\Pi}_{J_*})^\dagger \boldsymbol{X}, \quad \boldsymbol{\mathcal{P}}_{\boldsymbol{C}} := \boldsymbol{\Pi}_{J_*}\boldsymbol{C}^\dagger \boldsymbol{A} \in \mathbb{R}^{n\times n}.$$

Note that since $\boldsymbol{C}$ has full column rank, $\boldsymbol{C}^\dagger \boldsymbol{A}\boldsymbol{\Pi}_{J_*} = \boldsymbol{C}^\dagger \boldsymbol{C} = \boldsymbol{I}_{k+p}$, and

$$\boldsymbol{\mathcal{P}}_{\boldsymbol{C}}\boldsymbol{\mathcal{P}}_{\boldsymbol{X}} = \boldsymbol{\Pi}_{J_*}\boldsymbol{C}^\dagger \boldsymbol{A}\boldsymbol{\Pi}_{J_*}(\boldsymbol{X}\boldsymbol{\Pi}_{J_*})^\dagger \boldsymbol{X} = \boldsymbol{\mathcal{P}}_{\boldsymbol{X}}.$$

Therefore we get

$$\boldsymbol{A} - \boldsymbol{C}\boldsymbol{C}^\dagger \boldsymbol{A} = \boldsymbol{A}(\boldsymbol{I}_n - \boldsymbol{\mathcal{P}}_{\boldsymbol{C}}) = \boldsymbol{A}(\boldsymbol{I}_n - \boldsymbol{\mathcal{P}}_{\boldsymbol{C}})(\boldsymbol{I}_n - \boldsymbol{\mathcal{P}}_{\boldsymbol{X}}) = (\boldsymbol{I}_m - \boldsymbol{C}\boldsymbol{C}^\dagger)\boldsymbol{A}(\boldsymbol{I}_n - \boldsymbol{\mathcal{P}}_{\boldsymbol{X}}).$$

Since $\boldsymbol{X}\boldsymbol{\Pi}_{J_*} = \boldsymbol{X}(:, J_*)$ has full row rank,

$$\boldsymbol{X}\boldsymbol{\mathcal{P}}_{\boldsymbol{X}} = \boldsymbol{X}\boldsymbol{\Pi}_{J_*}\left(\boldsymbol{X}\boldsymbol{\Pi}_{J_*}\right)^\dagger \boldsymbol{X} = \boldsymbol{X}$$

and we obtain

$$(\boldsymbol{I}_n - \boldsymbol{\mathcal{P}}_{\boldsymbol{X}}) = (\boldsymbol{I}_n - \boldsymbol{X}^\dagger \boldsymbol{X})(\boldsymbol{I}_n - \boldsymbol{\mathcal{P}}_{\boldsymbol{X}}).$$

Now putting these together, we obtain

$$\begin{aligned}
\left\|\boldsymbol{A} - \boldsymbol{C}\boldsymbol{C}^\dagger \boldsymbol{A}\right\| &= \left\|(\boldsymbol{I}_m - \boldsymbol{C}\boldsymbol{C}^\dagger)\boldsymbol{A}(\boldsymbol{I}_n - \boldsymbol{X}^\dagger \boldsymbol{X})(\boldsymbol{I}_n - \boldsymbol{\mathcal{P}}_{\boldsymbol{X}})\right\| \\
&\leq \left\|\boldsymbol{I}_m - \boldsymbol{C}\boldsymbol{C}^\dagger\right\|_2 \left\|\boldsymbol{A}(\boldsymbol{I}_n - \boldsymbol{X}^\dagger \boldsymbol{X})\right\| \left\|\boldsymbol{I}_n - \boldsymbol{\mathcal{P}}_{\boldsymbol{X}}\right\|_2 \\
&= \left\|\boldsymbol{I}_n - \boldsymbol{\mathcal{P}}_{\boldsymbol{X}}\right\|_2 \left\|\boldsymbol{A}(\boldsymbol{I}_n - \boldsymbol{X}^\dagger \boldsymbol{X})\right\|,
\end{aligned}$$

since $\left\|\boldsymbol{I}_m - \boldsymbol{C}\boldsymbol{C}^\dagger\right\|_2 = 1$ as $\boldsymbol{C}\boldsymbol{C}^\dagger$ is an orthogonal projector. Now since $\boldsymbol{\mathcal{P}}_{\boldsymbol{X}}$ is an oblique projector [Szy06], $\|\boldsymbol{I}_n - \boldsymbol{\mathcal{P}}_{\boldsymbol{X}}\|_2 = \|\boldsymbol{\mathcal{P}}_{\boldsymbol{X}}\|_2$ so the result follows by noting that

$$\|\boldsymbol{\mathcal{P}}_{\boldsymbol{X}}\|_2 = \left\|(\boldsymbol{X}\boldsymbol{\Pi}_J)^\dagger \boldsymbol{X}\right\|_2 = \|\boldsymbol{Q}_{\boldsymbol{X}}(J_*, :)\|_2 \,.$$

□

Theorem 6.4.1. *Under Assumption 6.4.1,*

$$\begin{aligned}\left\|\boldsymbol{A} - \boldsymbol{A}^{(BA)}_{I\cup I_0, J\cup J_0}\right\| \leq \left\|\boldsymbol{Q}_{\boldsymbol{X}}(J\cup J_0, :)^\dagger\right\|_2 \left\|\boldsymbol{A} - \boldsymbol{A}\boldsymbol{X}^\dagger\boldsymbol{X}\right\| \\ + \left\|\boldsymbol{Q}_{\boldsymbol{Y}}(I\cup I_0, :)^\dagger\right\|_2 \left\|\boldsymbol{A} - \boldsymbol{Y}\boldsymbol{Y}^\dagger\boldsymbol{A}\right\|\end{aligned} \tag{6.4}$$

where $\|\cdot\|$ is any unitarily invariant norm and $\boldsymbol{Q}_{\boldsymbol{X}} \in \mathbb{R}^{n\times k}$ and $\boldsymbol{Q}_{\boldsymbol{Y}} \in \mathbb{R}^{m\times k}$ are the orthonormal matrices spanning the columns of $\boldsymbol{X}^\top$ and $\boldsymbol{Y}$, respectively.

Proof. The proof follows by Lemma 6.4.1 and noting that

$$\begin{aligned}\left\|\boldsymbol{A} - \boldsymbol{C}\boldsymbol{C}^\dagger\boldsymbol{A}\boldsymbol{R}^\dagger\boldsymbol{R}\right\| &\leq \left\|\boldsymbol{A} - \boldsymbol{C}\boldsymbol{C}^\dagger\boldsymbol{A}\right\| + \left\|\boldsymbol{C}\boldsymbol{C}^\dagger\boldsymbol{A} - \boldsymbol{C}\boldsymbol{C}^\dagger\boldsymbol{A}\boldsymbol{R}^\dagger\boldsymbol{R}\right\| \\ &\leq \left\|\boldsymbol{A} - \boldsymbol{C}\boldsymbol{C}^\dagger\boldsymbol{A}\right\| + \left\|\boldsymbol{C}\boldsymbol{C}^\dagger\right\|_2 \left\|\boldsymbol{A} - \boldsymbol{A}\boldsymbol{R}^\dagger\boldsymbol{R}\right\| \\ &= \left\|\boldsymbol{A} - \boldsymbol{C}\boldsymbol{C}^\dagger\boldsymbol{A}\right\| + \left\|\boldsymbol{A} - \boldsymbol{A}\boldsymbol{R}^\dagger\boldsymbol{R}\right\| \\ &\leq \left\|\boldsymbol{Q}_{\boldsymbol{X}}(J\cup J_0, :)^\dagger\right\|_2 \left\|\boldsymbol{A} - \boldsymbol{A}\boldsymbol{X}^\dagger\boldsymbol{X}\right\| + \left\|\boldsymbol{Q}_{\boldsymbol{Y}}(I\cup I_0, :)^\dagger\right\|_2 \left\|\boldsymbol{A} - \boldsymbol{Y}\boldsymbol{Y}^\dagger\boldsymbol{A}\right\|\end{aligned}$$

where the inequality for the second term $\left\|\boldsymbol{A} - \boldsymbol{A}\boldsymbol{R}^\dagger\boldsymbol{R}\right\|$ in the final line can be shown by considering $\boldsymbol{A}^\top$ in Lemma 6.4.1. □

Remark 6.4.1.

1. *The difference between Theorem 6.4.1 and its counterpart without oversampling, e.g. [DM23, Theorem 1], are the terms $\left\|\boldsymbol{Q}_{\boldsymbol{X}}(J\cup J_0, :)^\dagger\right\|_2$ and $\left\|\boldsymbol{Q}_{\boldsymbol{Y}}(I\cup I_0, :)^\dagger\right\|_2$, where*

instead of the matrix inverse, we have the pseudoinverse. This tightens the bound in Equation (6.4) *as*

$$\left\|\boldsymbol{Q}_{\boldsymbol{X}}(J \cup J_0, :)^{\dagger}\right\|_2 \leq \left\|\boldsymbol{Q}_{\boldsymbol{X}}(J, :)^{-1}\right\|_2,$$

and

$$\left\|\boldsymbol{Q}_{\boldsymbol{Y}}(I \cup I_0, :)^{\dagger}\right\|_2 \leq \left\|\boldsymbol{Q}_{\boldsymbol{Y}}(I, :)^{-1}\right\|_2,$$

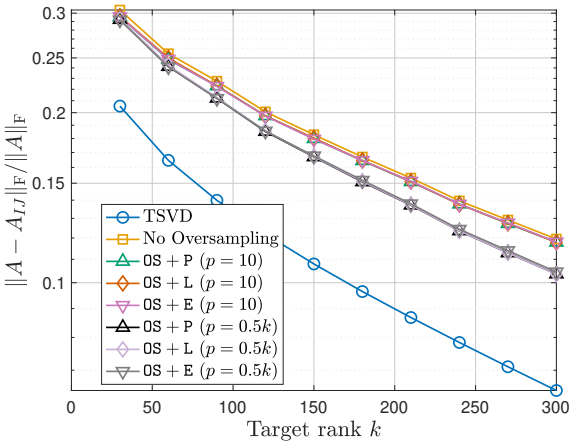
where I_0 and J_0 are the extra indices for oversampling.

2. *There are many possible choices for $\boldsymbol{X}$ and $\boldsymbol{Y}$. For example, if we choose them to be the dominant right and left singular vectors of $\boldsymbol{A}$, then we get the bound similar to the one in [SE16] involving the best rank-k approximation since $\left\|\boldsymbol{A} - \boldsymbol{A}\boldsymbol{X}^{\dagger}\boldsymbol{X}\right\| = \left\|\boldsymbol{A} - \boldsymbol{Y}\boldsymbol{Y}^{\dagger}\boldsymbol{A}\right\| = \|\boldsymbol{A} - [\![\boldsymbol{A}]\!]_k\|$ where $[\![\boldsymbol{A}]\!]_k$ is the best rank-k approximation to $\boldsymbol{A}$. If we choose $\boldsymbol{X}$ and $\boldsymbol{Y}$ to be the row sketch and the column sketch (see Algorithm 5) then we involve the randomised SVD error [HMT11]. Choosing $\boldsymbol{X}$ and $\boldsymbol{Y}$ to be the dominant singular vectors in Theorem 6.4.1 can be the optimal choice, but when $\boldsymbol{X}$ and $\boldsymbol{Y}$ are approximators, which can be computed easily, the bound* (6.4) *can be computed a posteriori.*

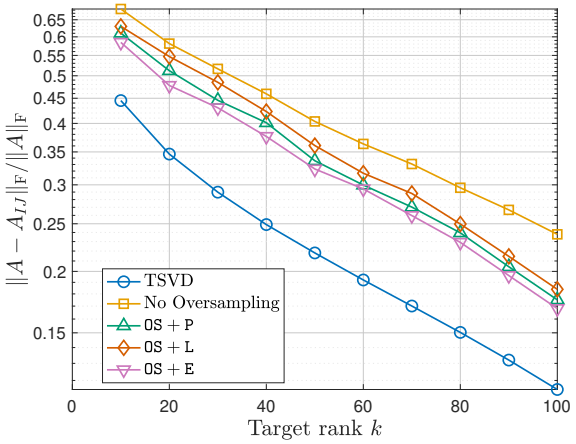
3. *When $k + \min\{p_1, p_2\} \geq \operatorname{rank}(\boldsymbol{A})$ and $\operatorname{rank}(\boldsymbol{C}) = \operatorname{rank}(\boldsymbol{R}) = \operatorname{rank}(\boldsymbol{A})$, we have $\boldsymbol{A} = \boldsymbol{C}\boldsymbol{C}^{\dagger}\boldsymbol{A}\boldsymbol{R}^{\dagger}\boldsymbol{R}$ [HH20].*

Theorem 6.4.1 suggests that oversampling helps to improve the accuracy of the CUR decomposition $\boldsymbol{A}_{IJ}^{(BA)}$. However, the numerical simulations shown below illustrate that the improvement is not too significant.

The numerical results are depicted in Figure 6.3, illustrating the effect of oversampling for the CUR-BA. We use pivoting on a random sketch (Algorithm 5) to obtain the initial set of indices. Then we oversample the row indices using the three oversampling algorithms, `OS+P`, `OS+L` and `OS+E` as in Section 6.3.2. We also oversample column indices in Figure 6.3b.



(a) `CIFAR10` using `Rand_Pivot` with row oversampling



(b) `YaleFace64x64` using `Rand_Pivot` with row ($p = 0.5k$) and column ($p = 10$) oversampling

Figure 6.3: The effect of oversampling for the CUR-BA. We use pivoting on a random sketch (Algorithm 5) to obtain the initial set of indices. Then we oversample the row indices using the three oversampling algorithms, `OS+P`, `OS+L` and `OS+E`. We also oversample column indices in Figure 6.3b.

We start with the `CIFAR10` dataset in Figure 6.3a. We first observe that as the oversampling parameter increases, the accuracy improves and the different oversampling techniques yield a similar result. However, there is no significant improvement when oversampling is used for the CUR-BA. This shows that the CUR-BA gives a robust approximation and oversampling only plays a slight role in its accuracy. In Figure 6.3b, we used the `YaleFace64x64` dataset to illustrate the effect of oversampling both rows and columns. We observe that when we oversample both rows and columns, we obtain a higher accuracy for the CUR-BA. This is expected as we are enlarging the subspace that we project onto $\boldsymbol{A}$, which increases the accuracy of the CUR-BA. This is contrary to the CUR-CA, as the CUR-CA can become more inaccurate and unstable when we oversample both rows and columns; see Section 6.3.2. However, simultaneously oversampling both rows and columns for CUR-BA increases the rank of the resulting approximation, potentially reducing its effectiveness as a low-rank representation.

To conclude, while oversampling improves the accuracy of the CUR-BA, it has less significant effect on its accuracy and stability.

7

Low-rank Approximation of Parameter-dependent Matrices via CUR

In this chapter, we apply the theory developed in Chapters 5 and 6 to study the low-rank approximation of parameter-dependent matrices $\boldsymbol{A}(t) \in \mathbb{R}^{m\times n}$ via CUR with cross approximation (CUR-CA). We assume throughout that $m \geq n$ and seek low-rank approximation for a finite number of parameter values $t \in \mathbb{R}$ in some compact domain $D \subset \mathbb{R}$. This task has appeared in several applications, for example, in compression of a series of images [NL08], dynamical systems [KL07] and Gaussian random fields [KLMU20].

When $\boldsymbol{A}(t) \equiv \boldsymbol{A}$ is a constant, using the truncated SVD attains the best low-rank approximation in any unitarily invariant norm. However, the cost of computing the truncated SVD becomes infeasible for large matrices. The situation is exacerbated when $\boldsymbol{A}(t)$ varies

with parameter t, requiring the evaluation of the truncated SVD at every parameter value t of interest. Specifically, we need to find a low-rank approximation to $\boldsymbol{A}(t_i)$ for $i = 1, 2, ..., q$. Consequently, numerous alternative approaches have been explored in recent years such as dynamical low-rank approximation [KL07, NL08], randomised SVD and generalised Nyström method [KL24] and the CUR decomposition [DPN$^+$23]. In this work, we use CUR-CA, $\boldsymbol{A} \approx \boldsymbol{C}\boldsymbol{M}^\dagger\boldsymbol{R}$, to find a low-rank approximation to a parameter-dependent matrix $\boldsymbol{A}(t)$.

The objective of this chapter is the following:

Objective 7.0.1. *Let $\boldsymbol{A}(t)$ be a parameter-dependent matrix. Then given a set of parameter values $t_1, t_2, ..., t_q$, and a tolerance ϵ, devise an algorithm that approximately achieves*

$$\left\|\boldsymbol{A}(t_i) - \left(\boldsymbol{A}(t_i)(:, J_i)\right)\left(\boldsymbol{A}(t_i)(I_i, J_i)^\dagger\right)\left(\boldsymbol{A}(t_i)(I_i, :)\right)\right\|_{\mathrm{F}} \leq \epsilon \left\|\boldsymbol{A}(t_i)\right\|_{\mathrm{F}} \tag{7.1}$$

for each i, where I_i and J_i are the row and column indices corresponding to t_i.

For parameter-dependent matrices $\boldsymbol{A}(t)$, we can naively recompute the row and the column indices for each parameter value of interest. However, this approach can be inefficient and wasteful, as the matrix may only undergo slight deviations when the parameter value changes. Consequently, the indices computed for one parameter value are likely to retain information relevant to nearby parameter values. The reason is because a matrix may only undergo slight deviations when the parameter value changes. More concretely, let $\boldsymbol{A}(t)$ be a parameter-dependent matrix and I and J be a set of row and column indices respectively. Then by setting $\boldsymbol{X} = \boldsymbol{V}_k(t)^\top$ in Corollary 5.1.1 with $I_0 = \emptyset$ where $\boldsymbol{V}_k(t)$ contains the k-dominant right singular vectors of $A(t)$,

$$\left\|\boldsymbol{A}(t) - \boldsymbol{C}(t)\boldsymbol{M}(t)^\dagger\boldsymbol{R}(t)\right\|_{\mathrm{F}} \leq \left\|\boldsymbol{Q}_{\boldsymbol{C}(t)}(I, :)^\dagger\right\|_2 \left\|\boldsymbol{Q}_{\boldsymbol{V}_k(t)}(J, :)^{-1}\right\|_2 \left\|\boldsymbol{A}(t) - [\![\boldsymbol{A}(t)]\!]_k\right\|_{\mathrm{F}} \tag{7.2}$$

where $\boldsymbol{C}(t)\boldsymbol{M}(t)^\dagger\boldsymbol{R}(t)$ is a rank-k CUR-CA for $\boldsymbol{A}(t)$. Now the rightmost term in Equation (7.2)

is optimal for any parameter value t, so we focus on comparing the first two terms. Suppose we have two different parameter values, say t_1 and t_2. If both $\left\|\boldsymbol{Q}_{\boldsymbol{C}(t_1)}(I,:)^\dagger\right\|_2 \left\|\boldsymbol{Q}_{\boldsymbol{V}_k(t_1)}(J,:)^{-1}\right\|_2$, and $\left\|\boldsymbol{Q}_{\boldsymbol{C}(t_2)}(I,:)^\dagger\right\|_2 \left\|\boldsymbol{Q}_{\boldsymbol{V}_k(t_2)}(J,:)^{-1}\right\|_2$ remain small with the same set of indices I and J, we can use I and J to approximate both $\boldsymbol{A}(t_1)$ and $\boldsymbol{A}(t_2)$.

In our setting, we allow $\|\boldsymbol{A}(t_1) - \boldsymbol{A}(t_2)\|_2$ to be large, say $\mathcal{O}(1)$, and therefore the perturbation bounds for the CUR decomposition in [HH21] is not enough to explain why the same indices can be reused for nearby parameter values. Yet, in practice, we often observe that for nearby parameter values, the identical set of indices I and J often capture the important rows and columns of both $\boldsymbol{A}(t_1)$ and $\boldsymbol{A}(t_2)$ even if $\|\boldsymbol{A}(t_1) - \boldsymbol{A}(t_2)\|_2$ is large; see Sections 7.4 and 7.5. This behaviour may be attributed to underlying regularity or structure present in many applications.

This observation motivates us to reuse the indices for various parameter values, potentially resulting in significant savings, often up to a factor k in complexity where k is the target rank. With this key idea in mind, the main goal of this chapter is to devise a rank-adaptive certified algorithm that reuses the indices as much as possible until the error becomes too large, at which point we recompute the indices. We use the term *certified* to denote approximation methods that are designed to control the approximation error.

To achieve this goal efficiently and reliably, we rely on a variety of efficient tools, many of which are borrowed from the randomised numerical linear algebra literature, which we discuss in Section 7.3. We assume throughout that for each parameter t, $\boldsymbol{A}(t)$ has decaying singular values so that a low-rank approximation is possible. This is clearly a required assumption for low-rank approximation techniques to be effective.

7.1 Brief outline of AdaCUR and FastAdaCUR

Before introducing the two algorithms, AdaCUR and FastAdaCUR, in greater detail in Section 7.4, we first provide a brief outline of their core ideas.

AdaCUR is aimed at efficiently computing an accurate low-rank approximation of parameter-dependent matrices. A high-level overview of AdaCUR is given in Figure 7.1 with the precise details outlined in Section 7.4.2. AdaCUR begins by computing an initial set of indices for $\boldsymbol{A}(t_1)$. For subsequent parameter values t_j, AdaCUR first computes the relative error for $\boldsymbol{A}(t_j)$ using the previous sets of indices. If the error is smaller than the tolerance ϵ, the algorithm retains the current sets of indices and proceeds to the next parameter value. If the error exceeds the tolerance, minor modifications are made to the indices, and the algorithm checks whether the tolerance is satisfied. If the modified indices meet the tolerance, they are retained; otherwise, the indices are recomputed from scratch. The algorithm can also perform multiple stages of minor modifications, which we discuss briefly at the end of Section 7.4.2.

FastAdaCUR is aimed at achieving even greater speed at the cost of possible reduction in accuracy. Instead of computing the error which can be expensive, FastAdaCUR keeps a small set of extra indices that helps with rank adaptivity. A high-level overview of FastAdaCUR is given in Figure 7.2 with the precise details outlined in Section 7.4.3. FastAdaCUR begins by computing an initial sets of indices I, J and a small set of additional indices I_s, J_s for $\boldsymbol{A}(t_1)$. For subsequent parameter values t_j, FastAdaCUR calculates the ϵ-rank of the core matrix $\boldsymbol{M}(t_j) = \boldsymbol{A}(I \cup I_s, J \cup J_s)$ as an efficient proxy for $\boldsymbol{A}$, using the previous sets of indices to determine whether the ϵ-rank exceeds the size of the index sets I and J. The ϵ-rank of a matrix $\boldsymbol{B}$, denoted by $\text{rank}_\epsilon(\boldsymbol{B})$, is defined as the number of singular values larger than ϵ. If the ϵ-rank has increased, indices are added to I and J from I_s and J_s. Conversely, if the ϵ-rank has decreased, indices are removed from I and J. After this adjustment, the algorithm proceeds to the next parameter value. Importantly, for FastAdaCUR, the entire

AdaCUR

Input: Parameter-dependent matrix $\boldsymbol{A}(t)$, parameters $t_1, ..., t_q$, error tol. ϵ.

Output: CUR factors $\boldsymbol{C}(t_j), \boldsymbol{M}(t_j), \boldsymbol{R}(t_j)$ such that $\boldsymbol{A}(t_j) \approx \boldsymbol{C}(t_j)\boldsymbol{M}(t_j)^\dagger \boldsymbol{R}(t_j)$.

1. Compute initial set of indices for $\boldsymbol{A}(t_1)$.

❂ For $j = 2, 3, ..., q$,

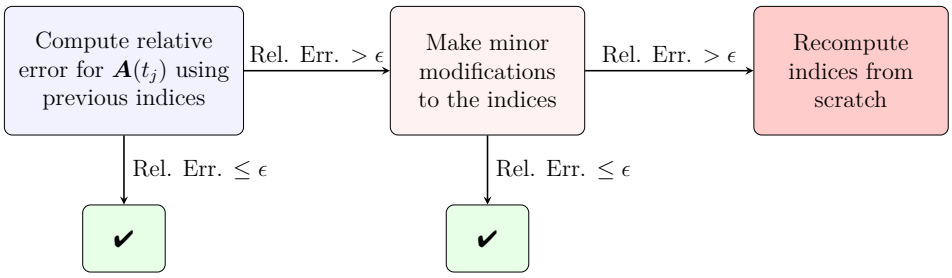


Figure 7.1: An overview of AdaCUR

matrix is not accessed beyond the first parameter value, which makes the algorithm efficient and suitable for settings where accessing every entry of future parameter values $\boldsymbol{A}(t_j)$ is not feasible. However, unlike AdaCUR, FastAdaCUR does not have error control mechanism, making it vulnerable to adversarial examples; see Section 7.5.4.

7.2 Existing methods.

There have been several different approaches for finding a low-rank approximation to parameter-dependent matrices $\boldsymbol{A}(t)$. We describe four different classes of methods. First, dynamical low-rank approximation is a differential-equation-based approach where the low-rank factors are obtained from projecting the time derivative of $\boldsymbol{A}(t)$, $\dot{\boldsymbol{A}}(t)$, onto the tangent space of the smooth manifold consisting of fixed-rank matrices; see [Lub14]. A number of variations from the original dynamical low-rank approximation [KL07] have been proposed to deal with various

FastAdaCUR

Input: Parameter-dependent matrix $\boldsymbol{A}(t)$, parameters $t_1, ..., t_q$, rank tol. ϵ

Output: CUR factors $\boldsymbol{C}(t_j), \boldsymbol{M}(t_j), \boldsymbol{R}(t_j)$ such that $\boldsymbol{A}(t_j) \approx \boldsymbol{C}(t_j)\boldsymbol{M}(t_j)^\dagger \boldsymbol{R}(t_j)$.

1. Compute initial set of indices I, J for $\boldsymbol{A}(t_1)$.
2. Compute a small set of extra indices I_s, J_s.

❁ For $j = 2, 3, ..., q$,

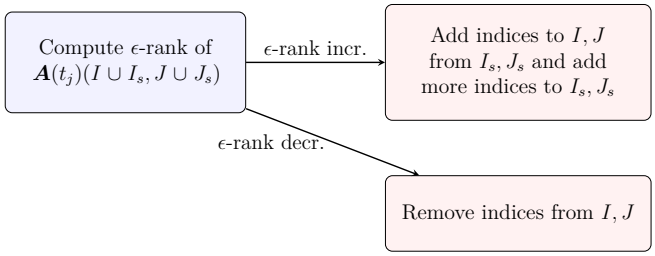


Figure 7.2: An overview of FastAdaCUR

issues such as the stiffness introduced by the curvature of the manifold [LO14, KLW16, CL22] and rank-adaptivity [CKL22, HPR22, HS23, HNS23]. The complexity of this approach is typically $\mathcal{O}(k_i T_{\boldsymbol{A}(t_i)})$ for the parameter t_i, where k_i is the target rank for $\boldsymbol{A}(t_i)$ and $T_{\boldsymbol{A}(t_i)}$ is the cost of matrix-vector product with $\boldsymbol{A}(t_i)$ or $\boldsymbol{A}(t_i)^\top$.

Secondly, Kressner and Lam [KL24] use randomised SVD [HMT11] and generalised Nyström [TYUC17b, Nak20], both based on random projections, to find low-rank approximations of parameter-dependent matrices $\boldsymbol{A}(t)$. They focus on matrices that admit an affine linear decomposition with respect to t. Notably, they use the same random embedding for all parameter values rather than generating a new random embedding for each parameter value. The complexity is also $\mathcal{O}(k_i T_{\boldsymbol{A}(t_i)})$ for the parameter value t_i.

Next, Donello et al. [DPN$^+$23] use the CUR decomposition to find a low-rank approximation to parameter-dependent matrices. At each iteration, a new set of column and row indices

for the CUR decomposition are computed by applying the discrete empirical interpolation method (DEIM) [CS10, DG16, SE16] to the singular vectors of the CUR decomposition from the previous iteration. While this approach benefits from not viewing the entire matrix, it may suffer from the same adversarial example as the fast algorithm, FastAdaCUR; see Algorithm 11 and Section 7.5.4. They allow oversampling of column and/or row indices to improve the accuracy of the CUR decomposition and propose rank-adaptivity where the rank is either increased or decreased by 1 if the smallest singular value of the current iterate is larger or smaller, respectively, than some threshold. Although their algorithm is based on the CUR decomposition, the low-rank factors in their approach are represented in the form of an SVD. This SVD is computed from the CUR decomposition at each iteration, given that orthonormal factors are necessary in DEIM. The complexity for this algorithm is $\mathcal{O}((m+n)k_i^2)$ for the parameter value t_i.

Lastly, in the special case when $\boldsymbol{A}(t)$ is symmetric positive definite, a parametric variant of adaptive cross approximation has been used in [KLMU20].

7.3 Preliminaries

In this section, we discuss the tools needed for our proposed algorithms introduced in Section 7.4. First, we use pivoting on a random sketch (Algorithm 5), introduced in the previous chapter, to efficiently obtain the indices required for the CUR-CA. The cost of Algorithm 7 is $\mathcal{O}(k \cdot T_{\boldsymbol{A}} + (m+n)k^2)$, where $k \cdot T_{\boldsymbol{A}}$ is the cost of forming the Gaussian row sketch $\boldsymbol{X}$ in line 2 where $T_{\boldsymbol{A}}$ is the cost of matrix-vector multiplication with $\boldsymbol{A}$ or $\boldsymbol{A}^\top$, and $\mathcal{O}(nk^2)$ and $\mathcal{O}(mk^2)$ are the cost of applying column-pivoted QR (CPQR) on $\boldsymbol{X}$ (line 3) and $\boldsymbol{A}(:,J)^\top$ (line 4) respectively. In the next sections, we review a fast randomised algorithm for computing the numerical rank of a matrix in Section 7.3.1 and in Section 7.3.2, we discuss an efficient method for computing the Frobenius norm of a matrix, which is

used for error estimation for AdaCUR in Section 7.4.

7.3.1 Randomised rank estimation

Randomised rank estimation [AN13, MN24] is an efficient tool for finding the ϵ-rank of a matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$. The ϵ-rank of a matrix is the number of singular values larger than ϵ. As we often require the target rank as input, for example in Algorithm 5, we would like an efficient method for computing the ϵ-rank of a matrix. This method also relies on sketching but, unlike pivoting on a random sketch, it involves sketching $\boldsymbol{S}_1 \in \mathbb{R}^{m \times s}, \boldsymbol{S}_2 \in \mathbb{R}^{n \times 2s}$ from both sides of the matrix. The idea is to approximate the ϵ-rank of $\boldsymbol{A}$ using

$$\hat{r}_\epsilon = \text{rank}_\epsilon(\boldsymbol{S}_1^\top \boldsymbol{A} \boldsymbol{S}_2) \tag{7.3}$$

where $\boldsymbol{S}_1^\top \boldsymbol{A} \boldsymbol{S}_2 \in \mathbb{R}^{s \times 2s}$ with $s \gtrsim \text{rank}_\epsilon(\boldsymbol{A})$. Here, $s \gtrsim \text{rank}_\epsilon(A)$ is achieved by gradually increasing s by appending to the sketches until $\sigma_{\min}(\boldsymbol{S}_1^\top \boldsymbol{A} \boldsymbol{S}_2) < \epsilon$; the details are in [MN24]. The overall cost of the algorithm is $\mathcal{O}(sT_{\boldsymbol{A}} + ns \log s + s^3)$ where $\mathcal{O}(sT_{\boldsymbol{A}})$ is the cost of forming the Gaussian sketch, $\mathcal{O}(ns \log s)$ is the cost of forming the SRTT sketch and $\mathcal{O}(s^3)$ is the cost of computing the singular values of $\boldsymbol{S}_1^\top \boldsymbol{A} \boldsymbol{S}_2$.

Combining pivoting on a random sketch and randomised rank estimation. Since we require the target rank as input for pivoting on a random sketch (Algorithm 5), randomised rank estimation and pivoting on a random sketch can be used together. Both algorithms need to form the row sketch $\boldsymbol{X} = \boldsymbol{S}^\top \boldsymbol{A}$ as part of their algorithm. Therefore, we can input the row sketch $\boldsymbol{X}$ obtained from randomised rank estimation into pivoting on a random sketch. This avoids the need to reform the row sketch making randomised rank estimation almost free when used with pivoting on a random sketch. The overall algorithm is outlined in Algorithm 7, giving the overall complexity of $\mathcal{O}(\hat{r}_\epsilon T_{\boldsymbol{A}} + (m+n)\hat{r}_\epsilon^2)$. By combining the two algorithms into

one, the number of matrix-vector products with $\boldsymbol{A}$ is halved, which is often the dominant cost.

Algorithm 7 Pivoting on a random sketch using randomised rank estimation

Input: $\boldsymbol{A} \in \mathbb{R}^{m\times n}$, tolerance ϵ

Output: Row indices I and Column indices J

`function` $[I, J] =$ `Rand_Pivot_RankEst`$(\boldsymbol{A}, \epsilon)$

1: $[\hat{r}_\epsilon, \boldsymbol{X}]$ = Estimation of ϵ-rank and row sketch $\boldsymbol{X} = \boldsymbol{S}_1^\top \boldsymbol{A}$ using Equation (7.3),

2: $[I, J] =$ `Rand_Pivot`$(\boldsymbol{A}, \boldsymbol{X}, \hat{r}_\epsilon)$ ▷ Take the row sketch $\boldsymbol{X}$ as input in Algorithm 5

7.3.2 Randomised norm estimation

In order to monitor how well our algorithm performs, we need an efficient way to estimate the norm of the error. Randomised norm estimation [GTP18, MT20] is an efficient method for finding an approximation to a norm of a matrix. In this work, we focus on the Frobenius norm as it is a natural choice for low-rank approximation and easier to estimate. Let $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ be a matrix and $\hat{\boldsymbol{A}}_k \in \mathbb{R}^{m\times n}$ be a rank-k approximation to $\boldsymbol{A}$. Then we would like to estimate $\left\|\boldsymbol{A} - \hat{\boldsymbol{A}}_k\right\|_{\mathrm{F}}$. As before, we can sketch to get an approximation to $\boldsymbol{A} - \hat{\boldsymbol{A}}_k$ and then compute its Frobenius norm. It turns out that the sketch size of $\mathcal{O}(1)$, say 5, suffices. More specifically, we have the following theorem from [GTP18].

Theorem 7.3.1 (Gratton & Titley-Peloquin 2018; [GTP18])**.** *Let $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ be a matrix, $\boldsymbol{S}_G \in \mathbb{R}^{m\times s}$ be a Gaussian embedding and set $\rho := \|\boldsymbol{A}\|_{\mathrm{F}}^2 / \|\boldsymbol{A}\|_2^2$. For any $\tau > 1$ and $s \leq m$,*

$$\mathbb{P}\left(\frac{\|\boldsymbol{A}\|_{\mathrm{F}}}{\tau} < \left\|\boldsymbol{S}_G^\top \boldsymbol{A}\right\|_{\mathrm{F}} \leq \tau \|\boldsymbol{A}\|_{\mathrm{F}}\right) \geq 1 - \exp\left(-\frac{s\rho}{2}(\tau-1)^2\right) - \exp\left(-\frac{s\rho}{4}\frac{(\tau^2-1)^2}{\tau^4}\right) \quad (7.4)$$

Setting $\|\boldsymbol{A}\|_{\mathrm{F}} = 5\|\boldsymbol{A}\|_2$, $\tau = 2$ and $s = 5$, the above theorem tells us

$$\mathbb{P}\left(\frac{\|\boldsymbol{A}\|_{\mathrm{F}}}{2} < \left\|\boldsymbol{S}_G^\top \boldsymbol{A}\right\|_{\mathrm{F}} \leq 2\|\boldsymbol{A}\|_{\mathrm{F}}\right) \geq 1 - 2.33 \cdot 10^{-8}, \quad (7.5)$$

i.e. with a small number of Gaussian vectors, we can well-approximate the Frobenius norm of a matrix. We note that the parameter ρ, which appears in the concentration bounds, plays only a minor role in practice. Since ρ depends solely on the spectral decay of $\boldsymbol{A}$, its effect is largely offset by the choice of s; even for moderate values of ρ, small values of s typically suffice to ensure strong probabilities guarantees. The algorithm specialised to CUR-CA is presented in Algorithm 8.

Algorithm 8 Randomised norm estimation for CUR-CA

Input: $\boldsymbol{A} \in \mathbb{R}^{m\times n}$, I row indices, J column indices, sample size s (rec. $s = 5$)
Output: E, an estimate for $\left\|\boldsymbol{A} - \boldsymbol{A}(:, J)\boldsymbol{A}(I, J)^{\dagger}\boldsymbol{A}(I, :)\right\|_{\mathrm{F}}$

`function` $E = \texttt{Norm\_Est}(\boldsymbol{A}, I, J, s)$
1: Draw a Gaussian embedding $\boldsymbol{S}_G \in \mathbb{R}^{m\times s}$,
2: Set $\boldsymbol{X} = \boldsymbol{S}_G^{\top}\boldsymbol{A} \in \mathbb{R}^{s\times n}$,
3: Set $\hat{\boldsymbol{X}} = \boldsymbol{X}(:, J)\boldsymbol{A}(I, J)^{\dagger}\boldsymbol{A}(I, :) \in \mathbb{R}^{s\times n}$, ▷ $\boldsymbol{X}(:, J) = \boldsymbol{S}_G^{\top}\boldsymbol{A}(:, J)$
4: $E = \left\|\boldsymbol{X} - \hat{\boldsymbol{X}}\right\|_{\mathrm{F}}$

The complexity of Algorithm 8 is $\mathcal{O}(sT_{\boldsymbol{A}} + (m + n)ks) = \mathcal{O}(T_{\boldsymbol{A}} + (m + n)k)$, where $k = \max\{|I|, |J|\}$, since $s = \mathcal{O}(1)$. This method is particularly useful when $\boldsymbol{A}$ can only be accessed through matrix-vector multiply.

Algorithm 8 can easily be used to estimate multiple low-rank approximations of $\boldsymbol{A}$. Suppose we are given two sets of indices I_1 and J_1, and I_2 and J_2. Then once the row sketch of $\boldsymbol{A}$, $\boldsymbol{X} = \boldsymbol{S}_G^{\top}\boldsymbol{A}$ has been computed, we only need to compute the row sketches of the low-rank approximations $\hat{\boldsymbol{X}}_1 = \boldsymbol{S}_G^{\top}\boldsymbol{A}_{I_1J_1} = \boldsymbol{X}(:, J_1)\boldsymbol{A}(I_1, J_1)^{\dagger}\boldsymbol{A}(I_1, :)$ and $\hat{\boldsymbol{X}}_2 = \boldsymbol{S}_G^{\top}\boldsymbol{A}_{I_2J_2} = \boldsymbol{X}(:, J_2)\boldsymbol{A}(I_2, J_2)^{\dagger}\boldsymbol{A}(I_2, :)$ to approximate the error. Here, $\boldsymbol{A}_{IJ} = \boldsymbol{A}(:, J)\boldsymbol{A}(I, J)^{\dagger}\boldsymbol{A}(I, :)$. Since forming $\boldsymbol{X}$ is typically the dominant cost in Algorithm 8, this allows us to efficiently test multiple low-rank approximations.

In addition to approximating the error using randomised norm estimation, we obtain, as

a by-product, a small sketch of the low-rank residual matrix since

$$\boldsymbol{X} - \hat{\boldsymbol{X}} = \boldsymbol{S}_G^\top(\boldsymbol{A} - \boldsymbol{A}(:, J)\boldsymbol{A}(I, J)^\dagger \boldsymbol{A}(I, :)). \tag{7.6}$$

This small sketch can be used to obtain an additional set of row and column indices by using pivoting on it, similarly to Algorithm 5. Specifically, we apply pivoting on the sketched residual $\boldsymbol{X} - \hat{\boldsymbol{X}}$ to get an extra set of s column indices, denoted J_s. Next, we can apply pivoting on the chosen columns of the residual matrix,

$$\boldsymbol{A}(:, J_s) - \boldsymbol{A}(:, J)\boldsymbol{A}(I, J)^\dagger \boldsymbol{A}(I, J_s)$$

to extract an extra set of s row indices I_s. It is important to note that I_s and J_s will not include indices already chosen in I and J because the residual matrix $\boldsymbol{A} - \boldsymbol{A}(:, J)\boldsymbol{A}(I, J)^\dagger \boldsymbol{A}(I, :)$ is zero in the rows and columns corresponding to I and J if $|I| = |J|$ [LAN24]. The procedure for obtaining s additional indices from the sketched residual is outlined in Algorithm 9.

Algorithm 9 Pivoting on the residual matrix

Input: $\boldsymbol{X}_{Res} \in \mathbb{R}^{s \times n}$ sketch of the residual matrix, I row indices, J column indices
Output: I_s and J_s, extra sets of indices

`function` $[I_s, J_s] =$ `Pivot_Residual`$(\boldsymbol{X}_{Res}, I, J)$
1: Apply CPQR on $\boldsymbol{X}_{Res}$. Let J_s be the s column pivots,
2: Set $\boldsymbol{C}_{Res} = \boldsymbol{A}(:, J_s) - \boldsymbol{A}(:, J)\boldsymbol{A}(I, J)^\dagger \boldsymbol{A}(I, J_s) \in \mathbb{R}^{m \times s}$
3: Apply CPQR on $\boldsymbol{C}_{Res}^\top$. Let I_s be the s row pivots.

The complexity of Algorithm 9 is $\mathcal{O}((m+n)s^2 + mks + k^3) = \mathcal{O}(n + mk + k^3)$, since s is set to be a constant. Here, $k = \max\{|I|, |J|\}$. The dominant cost lies in the computation of $\boldsymbol{A}(:, J)\boldsymbol{A}(I, J)^\dagger \boldsymbol{A}(I, J_s)$, which requires $\mathcal{O}(mk + k^3)$ operations. By adding these additional indices, we can improve the accuracy of the CUR-CA. Algorithm 9 will be applied later in AdaCUR to refine the approximation.

7.4 Proposed method

The goal of this chapter is to develop an efficient algorithm that maximally reuses the selected indices in the CUR-CA as we iterate through the parameter values. To illustrate how the same set of indices can be effective across varying parameters, we begin with a numerical example. We use the synthetic example from [CL22] given by

$$\boldsymbol{A}(t) = e^{t\boldsymbol{W}_1} e^{t} \boldsymbol{D} e^{t\boldsymbol{W}_2}, t \in [0, 1] \tag{7.7}$$

where $\boldsymbol{D} \in \mathbb{R}^{n\times n}$ is diagonal with entries $\boldsymbol{D}_{jj} = 2^{-j}$ and $\boldsymbol{W}_1, \boldsymbol{W}_2 \in \mathbb{R}^{n\times n}$ are randomly generated skew-symmetric matrices. Note that the matrix exponential of a skew-symmetric matrix is unitary, making the singular values of $\boldsymbol{A}(t)$ $e^t 2^{-j}$ for $j = 1, 2, ..., n$. We take $n = 500$ and take 101 equispaced points in the interval $[0, 1]$ for t. The matrix exponentials were computed using the `expm` command in MATLAB.

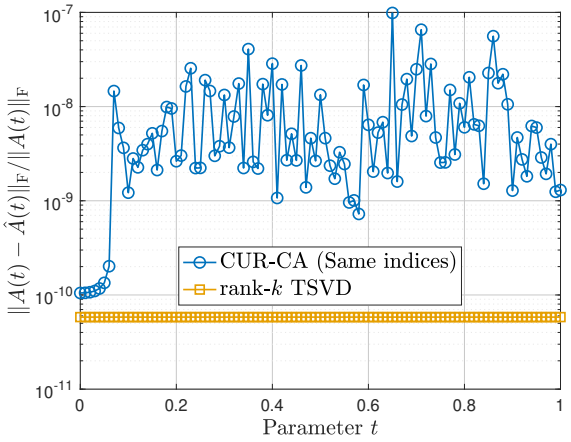


Figure 7.3: Relative error plot for the CUR-CA and the truncated SVD. We use the same set of indices for every parameter value. The target rank is $k = 33$ in this experiment.

The results are presented in Figure 7.3. In this experiment, we begin with an initial set of indices and reuse the same set of indices for all other parameter values. The target rank is set to 33. Despite using the same set of indices, we achieve a relative error of about 10^{-7}

throughout, only losing about 1–2 digits of accuracy compared to the initial accuracy.

This observation motivates a natural strategy: instead of recomputing the indices at every step, we first check whether the previously selected indices still yield a sufficiently accurate approximation. To safeguard against potential large errors, we incorporate error and rank estimation in our algorithms, which we discuss in detail in Sections 7.4.2 and 7.4.3.

As discussed in Chapter 6, oversampling can improve the accuracy of the CUR-CA. Given this, we adopt oversampling, and for definiteness choose to oversample rows in this work, that is, $|I_0| = p > 0$. The row oversampling is performed using Algorithm 6 from Chapter 6, which has computational complexity $\mathcal{O}(mk^2 + nkp)$.

7.4.1 Computing indices from scratch

In the next two sections, we discuss our algorithms, AdaCUR and FastAdaCUR. For both algorithms, we need to start with an index set for the CUR-CA of $\boldsymbol{A}(t_1)$. This can sometimes be done offline, for example, if we are solving a Matrix PDE, we can compute the initial set of indices for the CUR-CA from the initial condition matrix. In our algorithms, we use Algorithm 7 to get a set of indices from scratch. The procedure is to first approximate the $(\epsilon/\sqrt{n})$-rank of the initial matrix $\boldsymbol{A}(t_1)$ using randomised rank estimation and then apply pivoting on a random sketch with the estimated $(\epsilon/\sqrt{n})$-rank to get the initial set of indices. Here, we approximate the $(\epsilon/\sqrt{n})$-rank to ensure a relative error of ϵ for the CUR-CA in the Frobenius norm. More specifically, the $1/\sqrt{n}$ factor arises from the worst case scenario where the trailing singular values of $\boldsymbol{A}(t)$ are all equal. In such cases, the $(\epsilon/\sqrt{n})$-rank is required to guarantee a relative accuracy of ϵ in the Frobenius norm. If additional information about the spectrum of $\boldsymbol{A}(t)$ is available–for instance, if $\boldsymbol{A}(t)$ exhibits rapid singular value decay–then $\epsilon/\sqrt{n}$ can be replaced by $C\epsilon$, where $C < 1$ is a constant. However, to guarantee accuracy, we use $(\epsilon/\sqrt{n})$-rank throughout. The procedure for computing indices from scratch will be used for getting an initial set of indices for AdaCUR and FastAdaCUR, but also in AdaCUR when

the error becomes too large that we need to recompute the indices altogether from scratch.

7.4.2 AdaCUR algorithm

AdaCUR is an algorithm aimed at efficiently computing an accurate low-rank approximation of a parameter-dependent matrix with a given error tolerance. An overview of AdaCUR goes as follows. The algorithm starts by computing the initial set of indices for $\boldsymbol{A}(t_1)$ as discussed in Section 7.4.1. For subsequent parameter values, i.e., $\boldsymbol{A}(t_j)$ for $j = 2, ..., q$, we first verify whether the indices obtained from the previous parameter value are able to meet some given error tolerance, using an estimate of $\|\boldsymbol{A} - \boldsymbol{C}\boldsymbol{M}^{\dagger}\boldsymbol{R}\|_F$. If so, we continue to the next parameter value. Should the indices fail to meet the tolerance, we make low-cost minor modifications to the indices and test whether the adjustments satisfy the error tolerance. If this is still unsuccessful, we recompute the indices entirely from scratch and move on to the next parameter value. Minor modifications include adding a small fixed number of indices, reordering them based on their importance and removing some if necessary. The details are in the following paragraph. The addition and removal of indices, as well as recomputation of indices using rank estimation if necessary, make the algorithm rank-adaptive and computing the relative error at each instance make the algorithm certifiable. AdaCUR is presented in Algorithm 10.

AdaCUR (Algorithm 10) takes the parameter-dependent matrix $\boldsymbol{A}(t)$, evaluation points $t_1, ..., t_q \in D$, error tolerance ϵ, error sample size s, and oversampling parameter p as input. The algorithm produces the low-rank factors $\boldsymbol{C}(t_j), \boldsymbol{M}(t_j), \boldsymbol{R}(t_j)$ as output such that $\boldsymbol{A}(t_j) \approx \boldsymbol{C}(t_j)\boldsymbol{M}(t_j)^{\dagger}\boldsymbol{R}(t_j)$. If the entries of the parameter-dependent matrix can be computed quickly afterwards, the algorithm can be modified to output only row and column indices, saving storage. AdaCUR is quite involved, so we break it down into two core parts:

- **Initial error estimation**: Error estimation using previous indices (lines 5–8),
- **Low-cost minor modifications**: Minor updates for efficient fixes (lines 10–19).

The other lines involve computing the indices from scratch, which follow the same discussion as Section 7.4.1.

Algorithm 10 AdaCUR with certified accuracy

Input: Parameter-dependent matrix $\boldsymbol{A}(t) \in \mathbb{R}^{m\times n}$ $(m \geq n)$, evaluation points $t_1, ..., t_q$, error tolerance ϵ, error sample size s (rec. $s = 5$), oversampling parameter p.

Output: Matrices $\boldsymbol{C}(t_j), \boldsymbol{M}(t_j), \boldsymbol{R}(t_j)$ defining a subset of columns and rows of $\boldsymbol{A}(t_j)$ and their intersection for $j = 1, ..., q$.

1: $[I, J] = \texttt{Rand\_Pivot\_RankEst}(\boldsymbol{A}(t_1), \epsilon/\sqrt{n})$ ▷ Algorithm 7
2: $I_0 = \texttt{OS}(\boldsymbol{A}(t_1), I, J, p)$ ▷ Algorithm 6
3: Set $\boldsymbol{C}(t_1) = \boldsymbol{A}(t_1)(:, J)$, $\boldsymbol{R}(t_1) = \boldsymbol{A}(t_1)(I \cup I_0, :)$ and $\boldsymbol{M}(t_1) = \boldsymbol{A}(t_1)(I \cup I_0, J)$,
4: **for** $j = 2, ..., q$ **do**
5: Draw a Gaussian embedding $\boldsymbol{S}_G \in \mathbb{R}^{m\times s}$
6: Sketch $\boldsymbol{X}_s = \boldsymbol{S}_G^\top \boldsymbol{A}(t_j) \in \mathbb{R}^{s\times n}$
7: Set $\boldsymbol{E}_s = \boldsymbol{X}_s - \boldsymbol{S}_G^\top \boldsymbol{A}(t_j)(:, J)\boldsymbol{A}(t_j)(I \cup I_0, J)^\dagger \boldsymbol{A}(t_j)(I \cup I_0, :)$ ▷ Residual Sketch
8: Approximate relative error $E = \|\boldsymbol{E}_s\|_{\mathrm{F}} / \|\boldsymbol{X}_s\|_{\mathrm{F}}$
9: **if** $E > \epsilon$ **then**
10: $[I_1, J_1] = \texttt{Pivot\_Residual}(\boldsymbol{E}_s, I, J)$ ▷ Algorithm 9
11: Append indices $I \leftarrow I \cup I_1$ and $J \leftarrow J \cup J_1$
12: $[\sim, \sim, I_2] = \texttt{sRRQR}\left(\boldsymbol{A}(t_j)\,(I \cup I_0, J)^\top\right)$ ▷ strong Rank-Revealing QR [GE96]
13: $[\sim, \boldsymbol{R}^{(j)}, J_2] = \texttt{sRRQR}\left(\boldsymbol{A}(t_j)\,(I \cup I_0, J)\right)$
14: $k = \#\left|\left\{\boldsymbol{R}^{(j)}_{ii} : |\boldsymbol{R}^{(j)}_{ii}| > \epsilon/\sqrt{n} \cdot |\boldsymbol{R}^{(j)}_{11}|\right\}\right|$ ▷ Estimate relative $(\epsilon/\sqrt{n})$-rank
15: $I \leftarrow (I \cup I_0)\,(I_2(1:k))$ ▷ Select k important row indices
16: $I_0 \leftarrow (I \cup I_0)\,(I_2(k+1:k+p))$ ▷ Select the next p important row indices
17: $J \leftarrow J\,(J_2(1:k))$ ▷ Select k important column indices from J
18: Set $\boldsymbol{E}_s = \boldsymbol{X}_s - \boldsymbol{S}_G^\top \boldsymbol{A}(t_j)(:, J)\boldsymbol{A}(t_j)(I \cup I_0, J)^\dagger \boldsymbol{A}(t_j)(I \cup I_0, :)$
19: Recompute relative error $E = \|\boldsymbol{E}_s\|_{\mathrm{F}} / \|\boldsymbol{X}_s\|_{\mathrm{F}}$
20: **if** $E > \epsilon$ **then**
21: $[I, J] = \texttt{Rand\_Pivot\_RankEst}(\boldsymbol{A}(t_j), \epsilon/\sqrt{n})$
22: $I_0 = \texttt{OS}(\boldsymbol{A}(t_j), I, J, p)$ ▷ Algorithm 6
23: Set $C(t_j) = A(t_j)(:, J)$, $R(t_j) = A(t_j)(I \cup I_0, :)$ and $U(t_j) = A(t_j)(I \cup I_0, J)$.

Initial error estimation

In the for loop, the first task is to quantify how well the previous set of indices perform on the current parameter value. Error estimation is used for this task where we generate a Gaussian embedding $\boldsymbol{S}_G$ in line 5, sketch the matrix corresponding to the current parameter

value $\boldsymbol{X}_s = \boldsymbol{S}_G^\top \boldsymbol{A}(t_j)$ in line 6 and the residual error matrix for the CUR-CA in line 7, and finally, estimating the relative error E using the sketches in line 8. The sketch of the residual error matrix $\boldsymbol{E}_s$ is used for two tasks: approximating the relative error (lines 8, 19), and computing additional indices to reduce the relative error (line 10). If the relative error E is less than the tolerance ϵ, we use the previous set of indices and store the corresponding columns, rows and their intersection in line 23, after which we continue to the next parameter value t_j. On the other hand, if the relative error exceeds the tolerance, we make low-cost minor modifications in an attempt to reduce the relative error, as we describe next.

Low-cost minor modifications

The low-cost modifications involve trying to enlarge the index set using the sketched residual error matrix $\boldsymbol{E}_s$ (lines 10–11), reordering them by importance (lines 12–13), removing unimportant indices (lines 14–17), and recomputing the relative error (lines 18–19). More specifically, in line 10, randomised pivoting is used on the sketched residual matrix, $\boldsymbol{E}_s$ as in Algorithm 9 to obtain an additional set of s row and column indices. We append the extra set of s indices to the original sets of indices I and J in line 11, and order the indices in terms of their importance in lines 12–13. Here, Gu-Eisenstat's strong rank-revealing QR factorization [GE96] is used, but we can use other strong rank-revealing algorithms or more expensive algorithms with strong theoretical guarantees such as [CK20, Osi23], because we are working with a small $O(k) \times k$ matrix $\boldsymbol{A}(t_j)(I \cup I_0, J)$. In line 14, we approximate the $(\epsilon/\sqrt{n})$-rank of $\boldsymbol{A}(t_j)(I \cup I_0, J)$ by looking at the diagonal entries of the upper triangular factor in sRRQR as they give an excellent approximation to the singular values of the original matrix. Instead of approximating the $(\epsilon/\sqrt{n})$-rank of $\boldsymbol{A}(t_j)(I \cup I_0, J)$, we can set the rank tolerance slightly smaller, for example, $0.5\epsilon/\sqrt{n}$, to account for the approximation error and decrease the chance of recomputing the indices from scratch (lines 21–22) by keeping slightly more indices than necessary. In lines 15–17, we truncate based on the rank computed in line

14. The order of the indices is crucial to ensure that the important indices are preserved. We recompute the relative error in lines 18–19,[1] and if the relative error E is smaller than the tolerance, we store the new set of columns, rows of $\boldsymbol{A}(t_j)$ and their intersection in line 23 and continue to the next parameter value. If the relative error exceeds the tolerance, there are two recourses. AdaCUR outlines the first option in lines 21–22 where we recompute the indices altogether from scratch. In line 21, we perform both randomised rank estimation and randomised pivoting for two reasons. First, randomised rank estimation is extremely cheap when we have the row sketch already and second, when there is a sudden large change in rank, we can use randomised rank estimation to adjust the rank efficiently; note that when minor modifications did not help in lines 10–17, the matrix may have changed drastically.

An alternative to recomputing the indices. An alternative approach, not included in AdaCUR, is to increase the error sample size s gradually by incrementing its value, say $2s, 4s$, and so on, and updating the sketch accordingly by appending to it. As s increases, the number of additional important indices obtained from randomised pivoting in line 10 also increases, which will eventually reduce the relative error to less than the tolerance ϵ.

Role of the error sample size s. The value of s can be important for two reasons: error estimation and minor modifications. If we set s to be large, we obtain higher accuracy in our error estimation at the cost of more computation. However, we obtain a larger set of extra indices for the minor modification. This can help the algorithm to avoid the if statement in line 20, which recomputes the indices from scratch; see also Section 7.5.3. We can also create a version with increasing s if minor modifications fail to decrease the relative error below the tolerance, as described in the final part of the previous paragraph.

[1]For theoretical guarantee, the Gaussian embedding $\boldsymbol{S}_G$ needs to be independent from the modified indices [GTP18], however the extra set of indices is dependent on $\boldsymbol{S}_G$ as we apply pivoting on the sketched residual in line 10. Nonetheless, the dependency is rather weak so in practice, $\boldsymbol{S}_G$ can be reused without losing too much accuracy.

Complexity. Let k_j be the size of the index set for parameter t_j, i.e., the rank of the CUR-CA for $\boldsymbol{A}(t_j)$, Q_1 the set of parameter values for which we recompute the indices from scratch, i.e., invoked lines $1, 2, 21, 22$, and Q_2 the set of parameter values for which we do not need to recompute the indices from scratch. Note that $t_1 \in Q_1$, $Q_1 \cup Q_2 = \{t_1, ..., t_q\}$ and $Q_1 \cap Q_2 = \emptyset$. Then the complexity of AdaCUR is

$$\mathcal{O}\Big(\sum_{j \in Q_1} \big(k_j T_{\boldsymbol{A}(t_j)} + (m+n)k_j^2 + nk_j(k_j+p)\big) + \sum_{j \in Q_2} \big(sT_{\boldsymbol{A}(t_j)} + (m+n)k_j s + nsp\big) \Big) \quad (7.8)$$

where $T_{\boldsymbol{A}(t_j)}$ is the cost of matrix-vector multiply with $\boldsymbol{A}(t_j)$ or $\boldsymbol{A}(t_j)^\top$. The dominant cost, which can be quadratic with respect to m and n, comes from sketching, which cost $\mathcal{O}(k_j T_{\boldsymbol{A}(t_j)})$ in lines $1, 21$ and $\mathcal{O}(sT_{\boldsymbol{A}(t_j)})$ in line 6. The dominant linear costs come from pivoting and oversampling in lines $1, 2, 21, 22$, which cost $\mathcal{O}((m+n)k_j^2 + nk_j p)$ and computing the sketch of the residual error matrix in lines $7, 18$, which is $\mathcal{O}(mk_j s + ns(k_j + p))$. All the other costs are smaller linear costs or sublinear with respect to m and n, for example, the SRTT cost in randomised rank estimation in lines $1, 21$ is $\mathcal{O}(nk_j \log k_j)$ and the cost of sRRQR in lines $12, 13$ is $\mathcal{O}((k_j + p + s)(k_j + s)^2)$. When $s, p = \mathcal{O}(1)$ the complexity simplifies to

$$\mathcal{O}\left(\sum_{j \in Q_1} \big(k_j T_{\boldsymbol{A}(t_j)} + (m+n)k_j^2\big) + \sum_{j \in Q_2} \big(T_{\boldsymbol{A}(t_j)} + (m+n)k_j\big) \right). \quad (7.9)$$

It turns out that in many applications, the cardinality of the set Q_1 is small when we choose the oversampling parameter p and the error sample size s appropriately, making the algorithm closer to $\mathcal{O}(T_{\boldsymbol{A}(t_j)})$ for the jth parameter value; see Sections 7.5.2 and 7.5.3. This makes AdaCUR usually faster than other existing methods such as dynamical low-rank approximation [LO14], and randomised SVD and generalised Nyström method [KL24]; see Section 7.5.5.

7.4.3 FastAdaCUR algorithm

The bottleneck in AdaCUR is usually in the sketching, which is crucial for reliably computing the set of row and column indices for the CUR-CA and for error estimation, ensuring the algorithm's robustness. Without sketching, our algorithm would have linear complexity with respect to m and n; see Equation (7.9). In this section, we propose FastAdaCUR that achieves linear complexity after the initial phase of computing the initial set of indices for $\boldsymbol{A}(t_1)$. FastAdaCUR offers the advantage that the number of rows and columns required to be seen for each parameter value is approximately its target rank. Therefore, there is no need to read the whole matrix, which is advantageous in settings where the entries are expensive to compute. Unfortunately, the fast version does suffer from adversarial examples as FastAdaCUR does not look at the entire matrix; see Section 7.5.4. Nevertheless, the algorithm seems to perform well in practice for easier problems as illustrated in Section 7.5.

An overview of FastAdaCUR goes as follows. The algorithm starts by computing the initial set of indices for $\boldsymbol{A}(t_1)$ as discussed in Section 7.4.1. Then for each subsequent parameter value, we first form the core matrix $\boldsymbol{M}_j = \boldsymbol{A}(t_j)(I, J)$ where I and J include extra indices from the buffer space and oversampling from the previous step $j-1$. Here, the buffer space is the extra indices kept to quickly detect a possible change in rank or the important indices as we iterate through the parameter values. We then order the indices by importance and compute the $(\epsilon/\sqrt{n})$-rank of the core matrix $\boldsymbol{M}_j$. We compute the $(\epsilon/\sqrt{n})$-rank of the core matrix to aim for a relative low-rank approximation error of ϵ in the Frobenius norm. As in AdaCUR, we can set the rank tolerance slightly smaller, say $0.5\epsilon/\sqrt{n}$ to account for the approximation error. The order of indices is important here as we explain in (ii) below. If the rank increased from the previous iteration, we add extra indices from the buffer space and replenish it by invoking the oversampling algorithm using $\texttt{OS}(\boldsymbol{A}, I, J, \delta_k)$ (Algorithm 6) where δ_k is the rank increase. Conversely, if the rank decreases, we simply remove some indices.

Finally, we store the corresponding rows and columns of $\boldsymbol{A}(t_j)$ and their intersection and move to the next parameter value. FastAdaCUR is presented in Algorithm 11.

Algorithm 11 FastAdaCUR for speed

Input: Parameter-dependent matrix $\boldsymbol{A}(t) \in \mathbb{R}^{m\times n}$ $(m \geq n)$, evaluation points $t_1, ..., t_q$, buffer space b, oversampling parameter p, rank tolerance ϵ.

Output: Matrices $\boldsymbol{C}(t_j), \boldsymbol{M}(t_j), \boldsymbol{R}(t_j)$ defining a subset of columns and rows of $\boldsymbol{A}(t_j)$ and their intersection for $j = 1, ..., q$.

$[I, J] = \texttt{Rand\_Pivot\_RankEst}(\boldsymbol{A}(t_1), \epsilon/\sqrt{n})$ ▷ Algorithm 7
Set $k \leftarrow |I|$
$I_0 = \texttt{OS}(\boldsymbol{A}(t_1), I, J, p + b)$ ▷ Algorithm 6; includes buffer space and oversampling
$J_0 = \texttt{OS}(\boldsymbol{A}(t_1)^\top, J, I, b)$ ▷ Algorithm 6; includes buffer space
Set $I \leftarrow I \cup I_0$ and $J \leftarrow J \cup J_0$
Set $\boldsymbol{C}(t_1) = \boldsymbol{A}(t_1)(:, J(1:k)), \boldsymbol{R}(t_1) = \boldsymbol{A}(t_1)(I(1:k+p), :)$,
    $\boldsymbol{M}(t_1) = \boldsymbol{A}(t_1)(I(1:k+p), J(1:k))$
for $j = 2, ..., q$ **do**
    Set $\boldsymbol{M} = \boldsymbol{A}(t_j)(I, J) \in \mathbb{R}^{(k+b+p)\times(k+b)}$
    $[\sim, \sim, I_1] = \texttt{sRRQR}\left(\boldsymbol{M}^\top\right)$
    $[\sim, \boldsymbol{R}^{(j)}, J_1] = \texttt{sRRQR}\left(\boldsymbol{M}\right)$
    $k_0 = \#\left|\left\{\boldsymbol{R}^{(j)}_{ii} : |\boldsymbol{R}^{(j)}_{ii}| > \epsilon/\sqrt{n} \cdot |\boldsymbol{R}^{(j)}_{11}|\right\}\right|$ ▷ Estimate relative $(\epsilon/\sqrt{n})$-rank
    if $r_0 \leq r$ **then**
        Set $I \leftarrow I\left(I_1(1:k_0+b+p)\right)$ ▷ Reorder row indices and truncate
        Set $J \leftarrow J\left(J_1(1:k_0+b)\right)$ ▷ Reorder column indices and truncate
    else
        $I_2 = \texttt{OS}(\boldsymbol{A}(t_j), I, J, k_0 - k)$ ▷ Replenish row indices
        $J_2 = \texttt{OS}(\boldsymbol{A}(t_j)^\top, J, I, k_0 - k)$ ▷ Replenish column indices
        Set $I \leftarrow I(I_1) \cup I_2$ and $J \leftarrow J(J_1) \cup J_2$ ▷ Reorder and add indices
    Set $k \leftarrow k_0$ ▷ Update $(\epsilon/\sqrt{n})$-rank
    Set $\boldsymbol{C}(t_j) = \boldsymbol{A}(t_j)(:, J(1:k)), \boldsymbol{R}(t_j) = \boldsymbol{A}(t_j)(I(1:k+p), :)$,
        $\boldsymbol{M}(t_j) = \boldsymbol{A}(t_j)(I(1:k+p), J(1:k))$

The main distinction between AdaCUR and FastAdaCUR lies in the existence of error estimation. In AdaCUR, an error estimate is computed for each parameter value, which ensures that the row and the column indices are guaranteed to be good with high probability. This is absent in FastAdaCUR. Consequently, if there is a sudden change in rank or the previous set of indices are unimportant for the current parameter value, FastAdaCUR may deliver poor results. To mitigate this disadvantage, FastAdaCUR incorporates a buffer space

of size b (i.e., additional indices in I, J) as input to assist with detecting changes in rank. FastAdaCUR involves many heuristics, so we break it down to discuss and justify each part of the algorithm. We divide FastAdaCUR into three parts:

- **Rank estimation using the core matrix**: lines 8–11,
- **Rank decrease via truncation**: lines 12–14,
- **Rank increase via oversampling**: lines 15–18.

The initial set of indices are constructed from scratch as in Section 7.4.1, with the indices for the buffer space and oversampling obtained using the oversampling algorithm from Algorithm 6.

Rank estimation using the core matrix

Upon entering the for loop in line 7, we first compute the core matrix $\boldsymbol{M}(t_j) = \boldsymbol{A}(t_j)(I, J) \in \mathbb{R}^{(k+b+p)\times(k+b)}$ (line 8) for the current parameter value, which includes extra indices from the buffer space and oversampling. As FastAdaCUR prioritizes efficiency over accuracy, the core matrix $\boldsymbol{M}(t_j)$ is a sensible surrogate to approximate the singular values of $\boldsymbol{A}(t_j)$ without looking at the entire matrix. The extra indices from the buffer space and oversampling will assist with the approximation as the buffer space allows the core matrix to approximate more singular values, helping to detect a potential $(\epsilon/\sqrt{n})$-rank change. The additional set of row indices from oversampling improves the accuracy of the estimated singular values of $\boldsymbol{A}(t_j)$ (line 10) by the Courant-Fischer min-max theorem, which is used for approximating the relative $(\epsilon/\sqrt{n})$-rank in line 11. In addition, oversampling increases the singular values of the core matrix, thereby allowing FastAdaCUR to increase the rank when appropriate and reduce the error. However, we see in the experiments (Section 7.5.2) that the role that oversampling plays in FastAdaCUR is rather complicated. We use sRRQR to order the columns and rows of the core matrix by importance in lines 9 and 10, and subsequently

estimate the $(\epsilon/\sqrt{n})$-rank of the core matrix $\boldsymbol{M}$ using the diagonal entries of the upper triangular factor in sRRQR in line 11. We use the $(\epsilon/\sqrt{n})$-rank to aim for a tolerance of ϵ in the low-rank approximation. If the computed rank, k_0 is smaller than the previous rank, k, then we decrease the index set via truncation. Otherwise, we increase the index set via oversampling. The details are discussed below.

Rank decrease via truncation

After estimating the $(\epsilon/\sqrt{n})$-rank, k_0, if k_0 has not increased from the previous iteration, we adjust the number of indices (either decrease or stay the same) in lines 13 and 14 by applying the order of importance computed using sRRQR in lines 9 and 10, and truncating, if necessary. Specifically, we truncate the trailing $k - k_0$ row and column indices. At the end of this procedure, the algorithm ensures $|I| = k_0 + b + p$ and $|J| = k_0 + b$.

Rank increase via oversampling

If the rank has increased, i.e. $k_0 > k$, we refill the extra indices by the amount the rank has increased by, i.e., $k_0 - k$. Unlike in AdaCUR, for which we have the sketched residual matrix, we do not have a good proxy to obtain a good set of extra indices. Therefore, we use the already-selected columns and rows to obtain an extra set of column and row indices.[2] We use the oversampling algorithm (Algorithm 6) to achieve this as it only requires the selected rows and columns as input. Adding extra indices using oversampling has been suggested before in [DPN$^+$23] to make small rank increases in the context of dynamical low-rank approximation. We get the extra indices from oversampling in lines 16 and 17 and append to the original set of indices in line 18. At the end of this procedure, the algorithm ensures $|I| = k_0 + b + p$ and $|J| = k_0 + b$.

[2]The only recourse to obtaining a reliable set of extra indices is to view the entire matrix; see [Mar19, Ch. 17.5] for a related discussion. However, this puts us in a similar setting as AdaCUR, which we recommend when ensuring accuracy is the priority rather than speed.

When will FastAdaCUR work well?

FastAdaCUR relies on heuristics to make sensible decisions for estimating the $(\epsilon/\sqrt{n})$-rank, selecting the important indices, and adding indices as we describe above. Despite FastAdaCUR being susceptible to adversarial examples, as we see later in Section 7.5.4, it should work on easier problems. For example, if the singular vectors of $\boldsymbol{A}(t)$ are incoherent [MT20, Ch. 9.6] or (approximately) Haar-distributed, FastAdaCUR will perform well because uniformly sampled columns and rows approximate $\boldsymbol{A}(t)$ well [CD13]. In such cases, rank adaption is also expected to work well, as the submatrix of $\boldsymbol{A}(t)$ with oversampling behaves similarly to the two-sided sketch used for randomised rank estimation [MN24] (Section 7.3.1). Additionally, with the aid of buffer space, FastAdaCUR is expected to detect rank changes occurring in the problem and efficiently adapt to the rank changes using oversampling or truncation. Therefore, for problems where $\boldsymbol{A}(t)$ is incoherent for the parameter values of our interest, FastAdaCUR is expected to perform effectively.

Role of the buffer space b. The main role of the buffer space b is to identify rank changes while iterating through the parameter values. A larger value of b enhances the algorithm's ability to accurately detect rank changes at an expense of increased complexity. Furthermore, a larger buffer space enables the algorithm to accommodate larger rank changes, as the maximum possible rank change at any point in the algorithm is b; see Section 7.5.3 for numerical results. If prior knowledge about the rank change is available, we recommend adaptively setting the buffer space to match (or slightly larger than) the expected rank change between parameter values. If such information is not available, we suggest using either the maximum possible rank change or a modest constant, such as $b = 5$ or $b = 10$.

Complexity. Let k_j be the size of the index set for parameter t_j, i.e., the rank of the CUR-CA for $\boldsymbol{A}(t_j)$. The dominant cost $\mathcal{O}\left(k_1 T_{\boldsymbol{A}(t_1)}\right)$ in FastAdaCUR comes from line 1 for computing

the initial set of indices for $\boldsymbol{A}(t_1)$ excluding the buffer space and oversampling. Besides the computation of initial indices, the complexity is at most linear in m and n. For $j = 2, ..., q$, the complexity of FastAdaCUR for the jth parameter value is $\mathcal{O}((m+n)(k_j+p+b)^2)$ if the rank has increased from t_{j-1} to t_j. The cost involves the usage of oversampling to replenish extra indices. If the rank has either decreased or stayed the same then the complexity is $\mathcal{O}((k_j+p+b)(k_j+b)^2)$ for using sRRQR. Hence, excluding the first line, FastAdaCUR runs at most linearly in m and n for each parameter value, making it remarkably fast as demonstrated in Section 7.5.5.

7.5 Numerical illustration

In this section, we illustrate AdaCUR and FastAdaCUR using numerical experiments. The code for the strong rank-revealing QR algorithm [GE96] is taken from [Xin24].

In the following sections, we perform different numerical simulations with varying input parameters to test AdaCUR and FastAdaCUR. These simulations allow us to test the following:

1. The accuracy of the approximation obtained from the two algorithms by varying the tolerance ϵ,
2. The role that the oversampling parameter p plays in our algorithms,
3. The role that the error sample size s plays in AdaCUR and the buffer size b in FastAdaCUR,
4. How our algorithms perform on adversarial problems,
5. The speed of our algorithms against existing methods in [LO14, KL24].

It is worth pointing out that in the first three examples $\min_j \|\boldsymbol{A}(t_j) - \boldsymbol{A}(t_{j-1})\|_2 > 0.03$ for all j, indicating that the performance of AdaCUR and FastAdaCUR is not explained by the fact that the matrices are undergoing small perturbations as we iterate through the parameters.

In the experiments, we measure how many times AdaCUR needs to enter some of the 'if' statements, as the heavier computations are usually done inside them. This happens when the current set of indices do not meet the given tolerance in AdaCUR. We define the following

- h_1 is the number of times AdaCUR has invoked lines 10–18, but not lines 20–21. This happens when only minor modifications were needed to meet the tolerance. The complexity is $\mathcal{O}(T_{\boldsymbol{A}(t_j)} + (m+n)k_j^2)$ for the jth parameter value.

- h_2 is the number of times AdaCUR has invoked lines 20–21. This happens when minor modifications were not enough to meet the tolerance and the row and the column indices had to be recomputed from scratch. The complexity is $\mathcal{O}(k_j T_{\boldsymbol{A}(t_j)} + (m+n)k_j^2)$ for the jth parameter value.

In the plots that depict the $(\epsilon/\sqrt{n})$-rank of the low-rank approximation, a large dot is used to indicate the parameter value for which the indices were recomputed from scratch in AdaCUR. The number of large dots is equal to h_2. In FastAdaCUR, besides the computation of initial sets of indices, the heaviest computation is done when the rank has increased; i.e. in lines 16 and 17 when oversampling is invoked. However, this is heavily dependent on the given problem, so we do not track this. For reference, in FastAdaCUR, a large dot is used to indicate the parameter value where oversampling was applied when the rank increased.

7.5.1 Varying the tolerance ϵ

In this part of the section, we test the accuracy of our algorithms by varying the tolerance parameter ϵ. We use the same synthetic example [CL22] used for Figure 7.3, which is given by

$$\boldsymbol{A}(t) = e^{t\boldsymbol{W}_1} e^{t} \boldsymbol{D} e^{t\boldsymbol{W}_2}, t \in [0,1]. \tag{7.7}$$

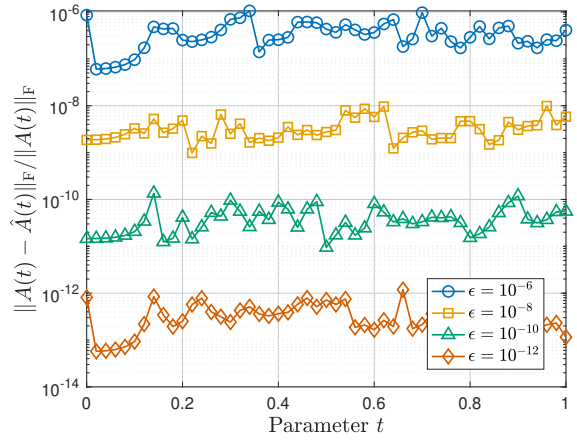
This example tests robustness against small singular values.

The results are depicted in Figure 7.4 and Table 7.1. In Figures 7.4a and 7.4b, we run AdaCUR with varying tolerance ϵ on the synthetic problem (Equation (7.7)). We set the oversampling parameter p to 5 and the error sample size s to 5 in this experiment. We first observe in Figure 7.4a that the relative error for AdaCUR meets the tolerance ϵ within a small constant factor. This is expected, given that the randomised methods in the algorithm typically yield errors within a small constant factor of the optimal solution; therefore one can simply take a smaller ϵ to ensure the actual error is bounded by the desired accuracy. Furthermore, the performance of AdaCUR is reflected in Figure 7.4b where the algorithm attempts to efficiently find the low-rank approximation of the parameter-dependent matrix by matching the $(\epsilon/\sqrt{n})$-rank at each parameter value. Regarding the algorithm's complexity, as shown in Table 7.1, we observe that h_1 and h_2 generally increases as the tolerance ϵ decreases. This increase is anticipated since a higher accuracy requirement generally equates to heavier computations.

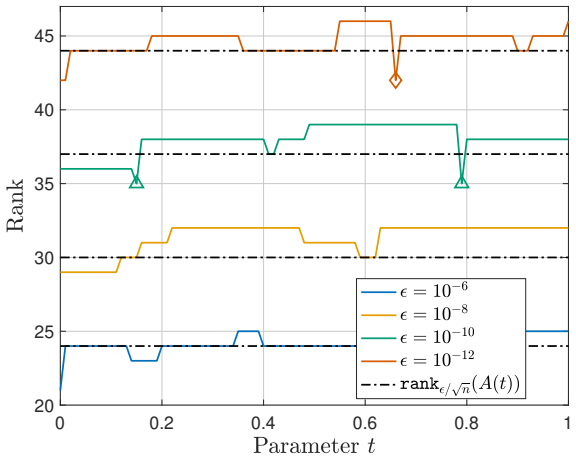
In Figures 7.4c and 7.4d, we run FastAdaCUR with varying tolerance ϵ on the synthetic problem (Equation (7.7)). We set the oversampling parameter p to 5 and the buffer space b to 5. We notice in Figure 7.4c that FastAdaCUR performs well by meeting the tolerance ϵ within a small constant factor. This performance is also reflected in Figure 7.4d, where the algorithm tries to match the $(\epsilon/\sqrt{n})$-rank at each parameter value. However, as shown in the numerical examples that follow, FastAdaCUR must be used with caution because as the problem becomes more difficult, the error may continue to grow as we iterate through the parameter values.

Table 7.1: The number of instances of heavier computations performed out of 100 parameter values t by AdaCUR in the experiment corresponding to Figure 7.4.

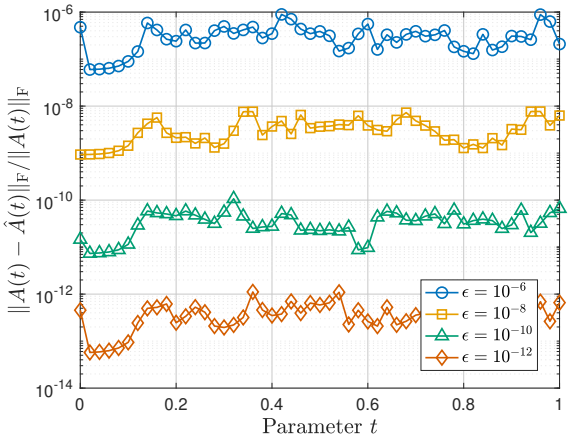
Tolerance ϵ	10^{-6}	10^{-8}	10^{-10}	10^{-12}
Minor mod. only h_1	8	8	10	14
Recomput. Indices h_2	0	0	2	1



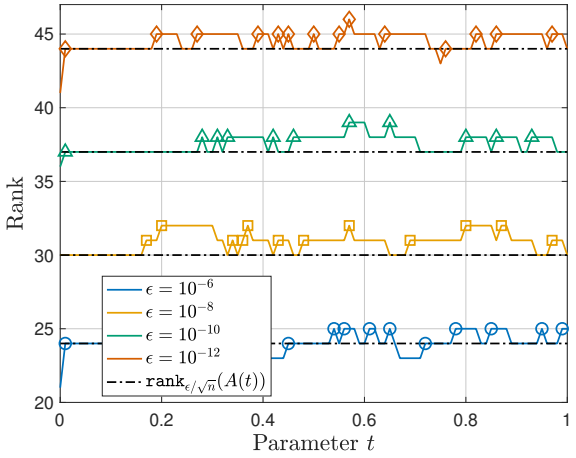
(a) AdaCUR with $p = 5$, $s = 5$



(b) Rank changes for AdaCUR



(c) FastAdaCUR with $p = 5$, $b = 5$



(d) Rank changes for FastAdaCUR

Figure 7.4: Testing tolerance parameter ϵ for AdaCUR and FastAdaCUR using the synthetic problem (Equation (7.7)).

7.5.2 Varying the oversampling parameter p

In this part of the section, we test the role that the oversampling parameter p plays in our algorithms. As shown in Chapters 5 and 6, oversampling can help with the accuracy and the numerical stability of the CUR-CA. We use a problem based on a PDE [KT11, CGV23] whose aim is to solve a heat equation simultaneously with several heat coefficients. This problem is

also known as the parametric cookie problem. In matrix form, the problem is given by

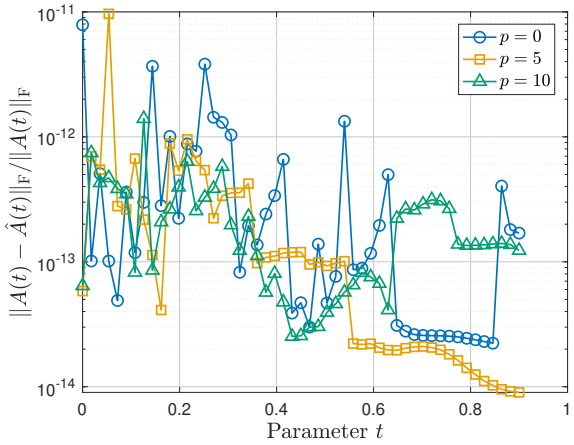
$$\dot{\boldsymbol{A}}(t) = -\boldsymbol{B}_0\boldsymbol{A}(t) - \boldsymbol{B}_1\boldsymbol{A}(t)\boldsymbol{C} + \boldsymbol{b}\boldsymbol{1}^\top,\ \boldsymbol{A}(t_0) = \boldsymbol{A}_0 \in \mathbb{R}^{1580\times 101} \tag{7.10}$$

where $\boldsymbol{B}_0, \boldsymbol{B}_1 \in \mathbb{R}^{1580\times 1580}$ are the mass and stiffness matrices, respectively, $\boldsymbol{b} \in \mathbb{R}^{1580}$ is the discretized inhomogeneity and $\boldsymbol{C} = \text{diag}(0, 1, ..., 100)$ is a diagonal matrix containing the parameter samples on its diagonal. We follow [KL24] by setting $t_0 = -0.01$ and $\boldsymbol{A}_0$ as the zero matrix and compute the solution at $t = 0$ exactly, after which we discretize the interval $[0, 0.9]$ with 101 uniform points and obtain $\boldsymbol{A}(t)$ using the `ode45` command in MATLAB with tolerance parameters $\{$'RelTol',$1e-12$,'AbsTol',$1e-12\}$ on each subinterval.

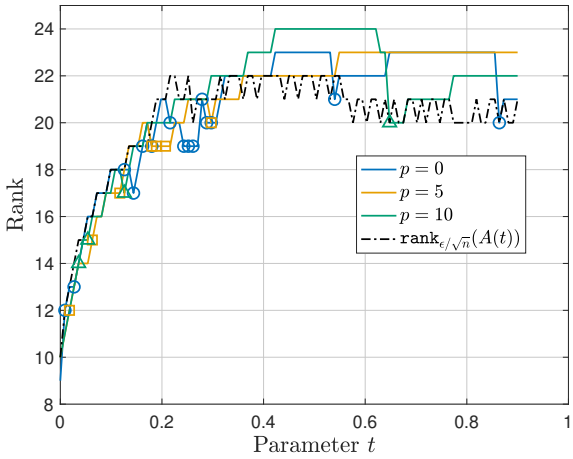
The results are presented in Figure 7.5 and Table 7.2. In Figures 7.5a and 7.5b, AdaCUR was used with varying oversampling parameter p on the parametric cookie problem (Equation (7.10)). We set the tolerance ϵ to 10^{-12} and the error sample size s to 1 in this experiment. In Figure 7.5a, the algorithm meets the tolerance level up to a modest constant factor while approximately matching the $(\epsilon/\sqrt{n})$-rank in Figure 7.5b. The benefits of oversampling is demonstrated in Table 7.2 where the number of instances the algorithm recomputes the indices (which forms the dominant cost), h_2, decreases as the oversampling parameter p increases. This demonstrates that oversampling assists AdaCUR in improving its complexity by reducing the frequency of heavier computations.

In Figures 7.5c and 7.5d, we use FastAdaCUR. We vary the oversampling parameter p on the parametric cookie problem with the tolerance ϵ equal to 10^{-12} and the buffer size b set to 5. In Figure 7.5c, the algorithm fails to meet the tolerance $\epsilon = 10^{-12}$ for certain parameter values, but still meets the tolerance within 2 orders of magnitude. By oversampling we are increasing the singular values of the core matrix $\boldsymbol{M}(t_i) = \boldsymbol{A}(t_i)(I, J) \in \mathbb{R}^{(k_i+b+p)\times(k_i+b)}$; see Section 7.4.3. This, in turn, should advocate a rank increase in FastAdaCUR, which the algorithm uses to reduce the error. However, as the algorithm is based on heuristics to prioritize speed, the

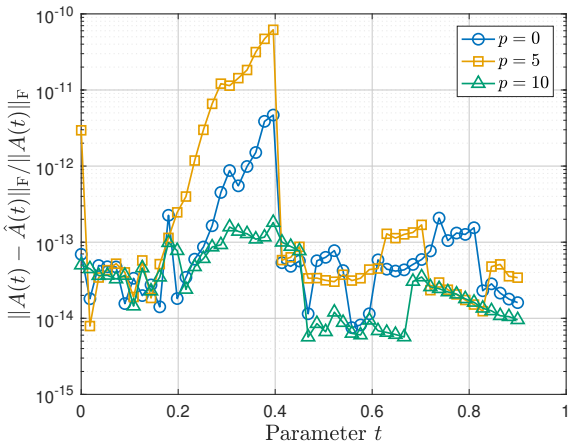
impact that the parameters have on the algorithm is rather complicated and inconclusive. Running FastAdaCUR on the parametric cookie problem with the same parameter values as in Figure 7.5c several times yield different results with varying performance between different oversampling values, making the impact that oversampling has for the accuracy inconclusive. Nevertheless, for the accuracy and stability of the CUR-CA, we recommend at least a little bit of oversampling as in Chapters 5 and 6.



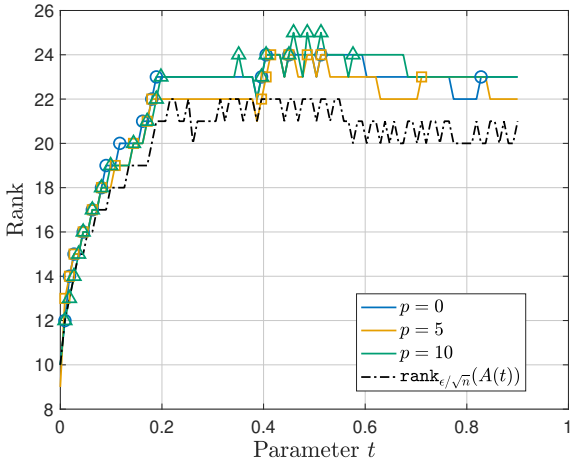
(a) AdaCUR with $\epsilon = 10^{-12}$, $s = 1$



(b) Rank changes for AdaCUR



(c) FastAdaCUR with $\epsilon = 10^{-12}$, $b = 5$



(d) Rank changes for FastAdaCUR

Figure 7.5: Testing the oversampling parameter p for AdaCUR and FastAdaCUR using the parametric cookie problem (Equation (7.10)).

Table 7.2: The number of instances of heavier computations performed out of 100 parameter values t by AdaCUR in the experiment corresponding to Figure 7.5.

Oversampling p	0	5	10
Minor mod. only h_1	31	21	17
Recomput. Indices h_2	15	7	4

7.5.3 Varying the error sample size s and the buffer size b

Here we test the role that the error sample size s plays in AdaCUR and the buffer size b in FastAdaCUR. We use the discrete Schrödinger equation in imaginary time for this experiment. The example is taken from [CL22] and is given by

$$\dot{\boldsymbol{A}}(t) = \frac{1}{2}(\boldsymbol{D}\,\boldsymbol{A}(t) + \boldsymbol{A}(t)\,\boldsymbol{D}) - \boldsymbol{V}_{\cos}\,\boldsymbol{A}(t)\,\boldsymbol{V}_{\cos},\ \ \boldsymbol{A}(0) = \boldsymbol{A}_0, \tag{7.11}$$

where $\boldsymbol{D} = \text{tridiag}(-1, 2, -1)$ is the discrete 1D Laplacian and $\boldsymbol{V}_{\cos}$ is the diagonal matrix with entries $1-\cos(2j\pi/n)$ for $j = -n/2, ..., n/2-1$. We take $n = 512$ and the initial condition $\boldsymbol{A}_0$ is a randomly generated matrix with singular values 10^{-i} for $i = 1, ..., 512$. We obtain $\boldsymbol{A}(t)$ by discretizing the interval $[0, 0.1]$ with 101 uniform points and using the `ode45` command in MATLAB with tolerance parameters {'RelTol',$1e-12$,'AbsTol',$1e-12$} on each subinterval.

The results are shown in Figure 7.6 and Table 7.3. In Figures 7.6a and 7.6b, we execute AdaCUR with varying error sample size s on the Schrödinger equation (Equation (7.11)). The oversampling parameter was set to $p = 10$ and the tolerance $\epsilon = 10^{-12}$. The algorithm satisfies the tolerance ϵ up to a modest constant factor as demonstrated in Figure 7.6a, while approximately aligning with the $(\epsilon/\sqrt{n})$-rank, as depicted in Figure 7.6b. As described in the role of the error sample size s in Section 7.4.2, it makes the minor modifications in AdaCUR more effective. This is demonstrated in Table 7.3 where the number of times the algorithm recomputed the indices, h_2, decreases as the error sample size s increases. The experiment demonstrates that with an increase in s, the algorithm becomes better at lowering the relative error using the low-cost minor modifications only. Therefore, the error sample size s should

be set sensibly to allow the algorithm to resolve the inaccuracy using minor modifications only so that the algorithm does not recompute indices until it becomes necessary. Note that higher s increases the complexity of AdaCUR, so s should not be too large.

In Figures 7.6c and 7.6d, we execute FastAdaCUR. We vary the buffer size b on the Schrödinger equation with the tolerance set to $\epsilon = 10^{-12}$ and the oversampling parameter set to $p = 10$. In Figure 7.6c, the algorithm performs well by meeting the tolerance except for the large spike happening in the first few parameter values. The cause of this spike is explained in Figure 7.6d where the problem experiences a substantial rank increase in the initial parameter values. Since the algorithm is only able to make a maximum rank increase of b at each iteration, the algorithm struggles to keep up with the large rank increase in the initial parameter values. This observation is evident in Figure 7.6c, where the initial spike in the graph diminishes as b increases. Therefore, if we expect a large rank increase in the parameter-dependent matrix, the buffer size b should be set to a higher value for those parameter values.

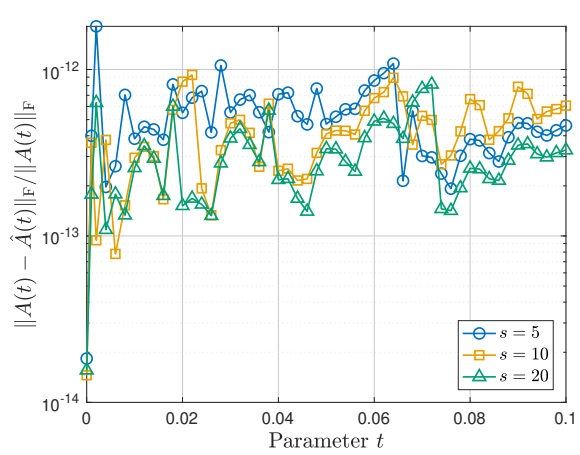
Table 7.3: The number of instances of heavier computations performed out of 100 parameter values t by AdaCUR in the experiment corresponding to Figure 7.6.

Error sample size s	5	10	20
Minor mod. only h_1	13	7	6
Recomput. Indices h_2	3	1	0

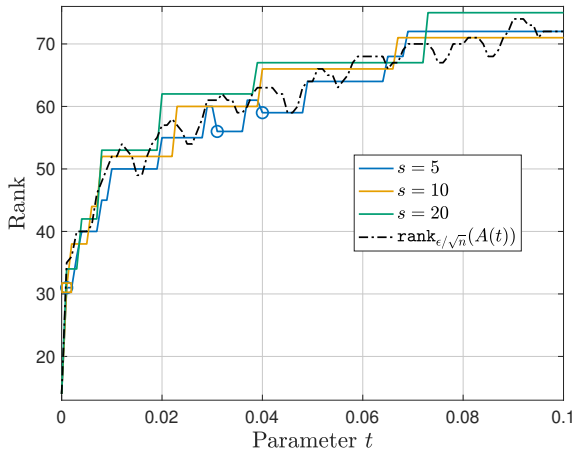
7.5.4 Adversarial example for FastAdaCUR

In this section, we create an adversarial example for FastAdaCUR to show that the algorithm can fail. We propose the following block diagonal matrix

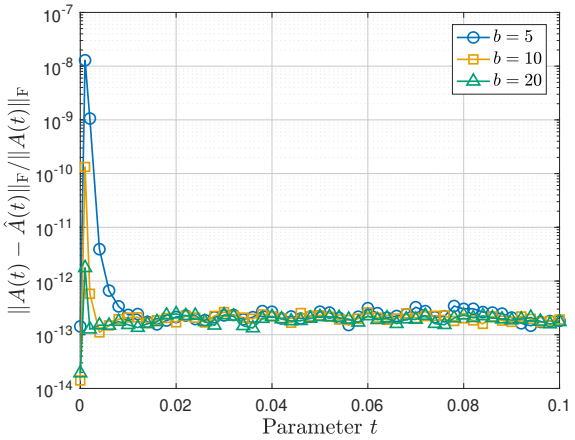
$$\boldsymbol{A}(t) = \begin{bmatrix} \boldsymbol{A}_1 & 0 & 0 \\ 0 & 0 & c(t)t\boldsymbol{A}_2 \end{bmatrix} \in \mathbb{R}^{300\times 100}, t \in [0,1] \tag{7.12}$$



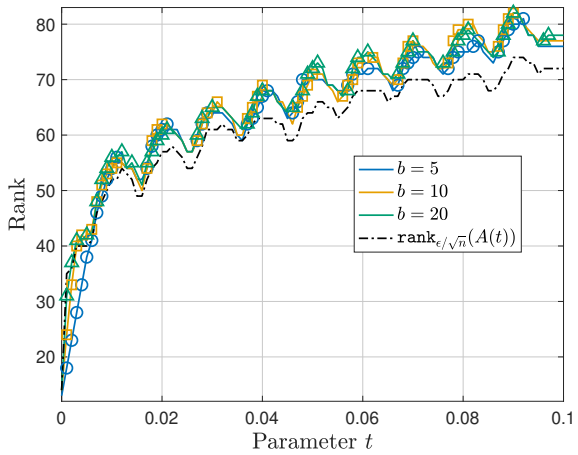
(a) AdaCUR with $\epsilon = 10^{-12}$, $p = 10$



(b) Rank changes for AdaCUR



(c) FastAdaCUR with $\epsilon = 10^{-12}$, $p = 10$



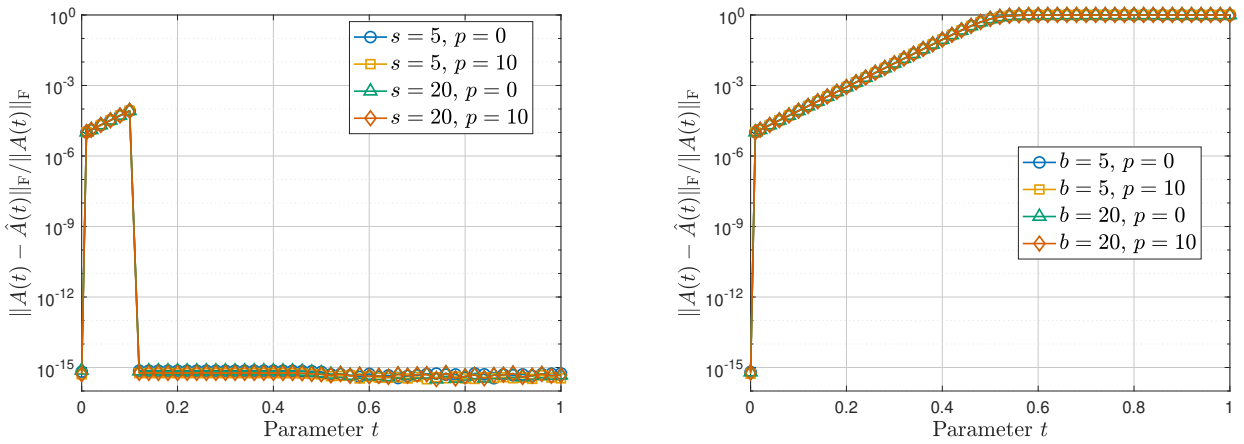
(d) Rank changes for FastAdaCUR

Figure 7.6: Testing the error sample size s for AdaCUR and the buffer size b for FastAdaCUR using the Schrödinger equation (Equation (7.11)).

where $\boldsymbol{A}_1 = \texttt{randn}(100, 20)$, $\boldsymbol{A}_2 = \texttt{randn}(200, 10)$ and $c(t) = 10^{-5+10t}$ using the `randn` command in MATLAB. The parameter-dependent matrix $A(t)$ starts with a large component in the $\boldsymbol{A}_1$ block, since $c(t)t\boldsymbol{A}_2$ is the zero matrix at $t = 0$. As t increases, the dominant block becomes $c(t)t\boldsymbol{A}_2$.

The results are presented in Figure 7.7. We execute FastAdaCUR in Figure 7.7b with tolerance $\epsilon = 10^{-4}$ and various values of buffer size b and oversampling parameter p. FastAdaCUR fails as the error continues to grow as we iterate through the parameter

values. This can be explained by the nature of the algorithm. Since it only considers a submatrix of the original matrix for each parameter value, the $c(t)t\boldsymbol{A}_2$ block remains hidden as t increases. Consequently, the algorithm is never able to capture the $c(t)t\boldsymbol{A}_2$ block, making the approximation poor. This type of counterexample is present in essentially all algorithms that do not view the entire matrix. For example, the algorithm in [DPN+23] may suffer from the same adversarial problem as it does not view the entire matrix at each parameter value. See [Mar19, Ch. 17.5] for a related discussion. On the other hand, as illustrated in Figure 7.7a, AdaCUR remains successful for the adversarial problem. AdaCUR is able to satisfy the tolerance of 10^{-4}, and when the $c(t)t\boldsymbol{A}_2$ block starts to become dominant, minor modifications or recomputation of indices is employed to capture the $c(t)t\boldsymbol{A}_2$ block, making the algorithm successful.



(a) AdaCUR with $\epsilon = 10^{-4}$ **(b)** FastAdaCUR with $\epsilon = 10^{-4}$

Figure 7.7: Adversarial example for FastAdaCUR with tolerance $\epsilon = 10^{-4}$. The adversarial example causes FastAdaCUR to fail, whereas AdaCUR remains successful.

7.5.5 Speed test

In this section, we compare the speed of our algorithms against other methods. Specifically, we benchmark AdaCUR and FastAdaCUR against the randomised SVD and the generalised Nyström method from [KL24], as well as an algorithm in the dynamical low-rank approximation

literature proposed by Lubich and Oseledets [LO14]. Additionally, we test a variant of FastAdaCUR that uses uniform sampling to select row and column indices, replacing the pivoting-based steps in lines $1, 3, 4, 16$, and 17 of Algorithm 11, making the algorithm's complexity sublinear, $\mathcal{O}((k_j + p + b)(k_j + b)^2)$.

When the target rank k becomes sufficiently large, we anticipate that our algorithms, which run with $\mathcal{O}(T_{\boldsymbol{A}(t)})$ and $\mathcal{O}((m + n)k^2)$ complexity, will outperform many existing methods, which run with $\mathcal{O}(kT_{\boldsymbol{A}(t)})$ complexity. We demonstrate this through experiments using the following artificial problem. Let $\boldsymbol{A}$ be a low-rank matrix with $\text{rank}(\boldsymbol{A}) = 500$ given by $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top \in \mathbb{R}^{50000\times 5000}$ where $\boldsymbol{U} \in \mathbb{R}^{50000\times 500}$ and $\boldsymbol{V} \in \mathbb{R}^{5000\times 500}$ are Haar-distributed orthonormal matrices and $\boldsymbol{\Sigma} \in \mathbb{R}^{500\times 500}$ is a diagonal matrix with entries that decay geometrically from 1 to 10^{-8}. The parameter-dependent matrix $\boldsymbol{A}(t)$ is then defined by

$$\boldsymbol{A}(i) = \boldsymbol{A} + \delta \sum_{j=1}^{i-1} \boldsymbol{X}_j \tag{7.13}$$

where $i \in \{1, 2, ..., 101\}$, $\delta = 10^{-12}$ and $\boldsymbol{X}_j$'s are sparse random matrices generated using $\texttt{sprandn}(\texttt{50000}, \texttt{5000}, \texttt{10}^{-\texttt{5}})$ command in MATLAB. The parameter-dependent matrix $\boldsymbol{A}(t)$ starts as a low-rank matrix at $i = 1$, with sparse noise introduced at discrete time intervals. The input parameters of AdaCUR, FastAdaCUR, and FastAdaCUR with uniform sampling were $\epsilon = 10^{-6}$ for tolerance, $s, b = 10$ for the error sample size and the buffer size, and the oversampling parameter was set to $p = 10$. This problem was constructed such that uniform sampling works well, which means that FastAdaCUR with uniform sampling should be both fast and accurate. The target rank for the randomised SVD, generalised Nyström method and the algorithm by Lubich and Osedelets was set to the rank of $\boldsymbol{A}$, which is 500. The second sketch size of the generalised Nyström was set to 750. See their respective papers for details [LO14, KL24].

The results are illustrated in Figure 7.8. For AdaCUR, FastAdaCUR, and FastAdaCUR

with uniform sampling, we notice a slight rise at the beginning, stemming from the initial computation of indices. However, the low-rank approximation of the subsequent parameter values is computed very quickly, making the slope of the cumulative runtime graph relatively flat as we iterate through the parameter values. The graph is notably flat for FastAdaCUR and even more so for FastAdaCUR with uniform sampling, as it runs at most linearly in m and n at each iteration. On the other hand, the three existing methods display a steeper slope due to the higher computational cost associated with each parameter value. This higher cost stems from the increased number of matrix-vector products with $\boldsymbol{A}(t)$ or its transpose. AdaCUR is approximately 4–7 times faster than the three existing algorithms, while FastAdaCUR achieves a speedup of roughly 9–15 times. When FastAdaCUR is used with uniform sampling, it is even more efficient, outperforming other algorithms by a factor of approximately 17–31.

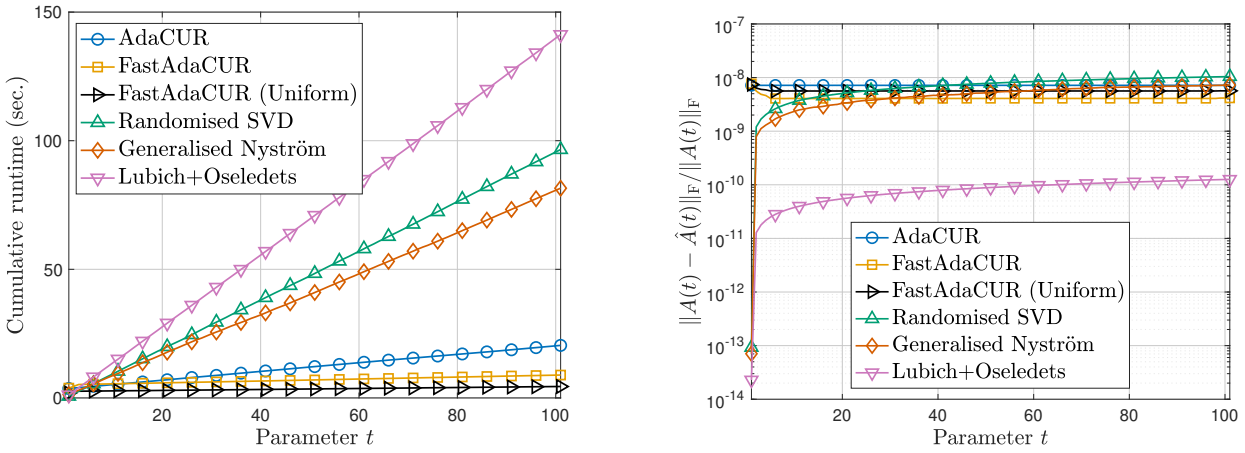


Figure 7.8: Runtime test for AdaCUR and FastAdaCUR against the three existing algorithms: randomised SVD, generalised Nyström method [KL24] and the algorithm by Lubich and Osedelets [LO14].

8

Conclusion

This thesis explored various topics in randomised numerical linear algebra with a focus on low-rank approximation methods. The foundational tools and key themes in randomised numerical linear algebra were introduced in Chapter 2.

In Chapter 3, we investigated a randomised algorithm based on the sketch-and-solve method for approximating the trailing right singular vectors of a tall matrix. Using multiplicative perturbation theory, we analysed the accuracy of the resulting approximation and demonstrated that it depends on the relative gap $\sigma_{n-k+1}/\sigma_{n-k}$. When this ratio is small, our method yields accurate approximations of the k trailing right singular vectors. We applied this technique to the total least squares problem and the AAA algorithm for rational approximation. For the latter, we also developed an efficient method to update the sketch when rows or columns are added or removed from the original matrix, improving the performance of the

sketch-and-solve strategy. This chapter provided theoretical and practical insight into when the sketch-and-solve method can effectively approximate the trailing singular subspace.

Chapter 4 addressed the application of the Nyström method to symmetric indefinite matrices. We showed a key issue in the naive application of the Nyström method: underestimation of the singular values in the core matrix, which can lead to instability. We proposed an algorithm based on this observation which truncates the core matrix proportional to the target rank. Numerical experiments demonstrated the robustness of this approach, and we provided theoretical analysis for a close variant. To the best of our knowledge, this algorithm is the first Nyström method designed specifically for stability in the symmetric indefinite setting.

The primary objective of Chapter 5 was to analyse the accuracy and stability of CUR-CA, whose stability had been questioned in other works. We provided an accuracy and numerical stability result for CUR-CA, and recommended an implementation that is observed to be stable in practice:

$$\mathtt{A_{IJ}} = (\mathtt{A}(:,\mathtt{J}) * \mathtt{V}/\mathtt{S}) * (\mathtt{U}' * \mathtt{A}(\mathtt{I},:)) \text{ where } [\mathtt{U},\mathtt{S},\mathtt{V}] = \mathtt{svd}(\mathtt{M},\text{`}\mathtt{econ}')$$

or the analogous implementation using the QR factorisation. Based on the theory developed in this section, we showed that the row and column indices should be selected dependently—one set of indices should be selected based on the choice of the other—to achieve better accuracy.

In Chapter 6, we studied the role of oversampling in improving the accuracy and stability of CUR-CA. We demonstrated how oversampling should be done and showed that it mitigates the effects of poorly chosen indices by increasing the smallest singular value of a square submatrix within an orthonormal matrix. Oversampling also improves numerical stability by making the core matrix rectangular, which is generally more well-conditioned than a square matrix. Based on these findings, we recommend oversampling either the row or the column indices—but not both—and suggest setting the oversampling parameter to be

proportional to the target rank whenever feasible.

Finally, in Chapter 7, we devised two efficient, rank-adaptive algorithms for computing low-rank approximations of parameter-dependent matrices using CUR-CA: AdaCUR and FastAdaCUR. The key idea behind these algorithms is to try to reuse the row and the column indices from the previous iterate as much as possible. AdaCUR offers many favourable properties such as rank-adaptivity, error-control and a typical complexity of $\mathcal{O}(T_{A(t)})$. In contrast, FastAdaCUR is faster—scaling linearly in m and n—but lacks error control, making it vulnerable to adversarial inputs. Nonetheless, FastAdaCUR is effective for simpler problems, such as those where the parameter-dependent matrices that are coherent at each parameter value.

Randomised algorithms in numerical linear algebra are powerful tools, which are constantly being developed and improved to enable scalable and robust solutions to problems that are otherwise intractable. This thesis opens several directions for future research, and we highlight two of them.

First, in Chapter 4, we introduced a variant of the Nyström method for symmetric indefinite matrices based on random embeddings. However, column sampling methods are more commonly used in practice due to their scalability. While many Nyström approaches rely on column sampling for this reason, it remains, to the best of our knowledge, an open question whether a column sampling scheme can be designed to provide robust approximation guarantees for symmetric indefinite matrices.

Second, in Chapter 7, a challenge with FastAdaCUR is the absence of an efficient mechanism for detecting large approximation errors without viewing the full matrix. Although, adversarial examples always exist if we do not view the entire matrix, developing a probabilistic method that allows error control with high probability would significantly enhance the reliability and utility of FastAdaCUR.

Bibliography

[ABB+99] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, et al. *LAPACK Users' guide*. SIAM, 1999. 10, 122

[AC09] N. Ailon and B. Chazelle. The fast Johnson–Lindenstrauss transform and approximate nearest neighbors. *SIAM J. Comput.*, 39(1):302–322, 2009. 21

[ADM+15] D. Anderson, S. Du, M. Mahoney, C. Melgaard, K. Wu, and M. Gu. Spectral gap error bounds for improving CUR matrix decomposition and the Nyström method. In *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics*, volume 38, pages 19–27. PMLR, 2015. 34, 120, 133

[AMT10] H. Avron, P. Maymounkov, and S. Toledo. Blendenpik: Supercharging LAPACK's least-squares solver. *SIAM J. Sci. Comput.*, 32(3):1217–1236, 2010. 135

[AN13] A. Andoni and H. L. Nguyễn. Eigenvalues of a matrix in the streaming model. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1729–1737. 2013. 77, 157

[ARS21] H. Al Daas, T. Rees, and J. Scott. Two-level Nyström–Schur preconditioner for sparse symmetric positive definite matrices. *SIAM J. Sci. Comput.*, 43(6):A3837–A3861, 2021. 11

[BDMI14] C. Boutsidis, P. Drineas, and M. Magdon-Ismail. Near-optimal column-based matrix reconstruction. *SIAM J. Comput.*, 43(2):687–717, 2014. 33

[Beb00] M. Bebendorf. Approximation of boundary element matrices. *Numer. Math.*, 86(4):565–589, 2000. 34

[BG13] C. Boutsidis and A. Gittens. Improved matrix algorithms via the subsampled randomized Hadamard transform. *SIAM J. Matrix Anal. Appl.*, 34(3):1301–1340, 2013. 22, 49, 81

[BG22] O. Balabanov and L. Grigori. Randomized Gram–Schmidt process with application to GMRES. *SIAM J. Sci. Comput.*, 44(3):A1450–A1474, 2022. 11, 135

[BMD09] C. Boutsidis, M. W. Mahoney, and P. Drineas. An improved approximation algorithm for the column subset selection problem. In *Proceedings of the 2009 Annual ACM-SIAM SODA*, pages 968–977, 2009. 32

[BRN10] L. Balzano, B. Recht, and R. Nowak. High-dimensional matched subspace detection when data are missing. In *2010 IEEE International Symposium on Information Theory*, pages 1638–1642, 2010. 32

[BSS12] J. Batson, D. A. Spielman, and N. Srivastava. Twice-ramanujan sparsifiers. *SIAM J. Comput.*, 41(6):1704–1721, 2012. 33

[BW17] C. Boutsidis and D. P. Woodruff. Optimal CUR matrix decompositions. *SIAM J. Comput.*, 46(2):543–589, 2017. 102, 133

[CD13] J. Chiu and L. Demanet. Sublinear randomized algorithms for skeleton decompositions. *SIAM J. Matrix Anal. Appl.*, 34(3):1361–1383, 2013. 33, 103, 133, 172

[CFCA13] K. Carlberg, C. Farhat, J. Cortial, and D. Amsallem. The GNAT method for nonlinear model reduction: Effective implementation and application to computational fluid dynamics and turbulent flows. *J. Comput. Phys.*, 242:623–647, 2013. 133

[CFS21] C. Cartis, J. Fiala, and Z. Shao. Hashing embeddings of optimal dimension, with applications to linear least squares. *arXiv preprint arXiv:2105.11815*, 2021. 23

[CGMR05] H. Cheng, Z. Gimbutas, P. G. Martinsson, and V. Rokhlin. On the compression of low rank matrices. *SIAM J. Sci. Comput.*, 26(4):1389–1404, 2005. 24

[CGV23] B. Carrel, M. J. Gander, and B. Vandereycken. Low-rank parareal: a low-rank parallel-in-time integrator. *BIT*, 63(1):13, 2023. 176

[Cha87] T. F. Chan. Rank revealing QR factorizations. *Linear Algebra Appl.*, 88-89:67–82, 1987. 37, 123

[CHHN21] H. Cai, K. Hamm, L. Huang, and D. Needell. Robust CUR decomposition: Theory and imaging applications. *SIAM J. Imaging Sci.*, 14(4):1472–1503, 2021. 31

[CK20] A. Cortinovis and D. Kressner. Low-rank approximation in the Frobenius norm by column and row subset selection. *SIAM J. Matrix Anal. Appl.*, 41(4):1651–1673, 2020. 33, 34, 73, 110, 165

[CK24] A. Cortinovis and D. Kressner. Adaptive randomized pivoting for column subset selection, DEIM, and low-rank approximation. *arXiv preprint arXiv:2412.13992*, 2024. 33

[CKL22] G. Ceruti, J. Kusch, and C. Lubich. A rank-adaptive robust integrator for dynamical low-rank approximation. *BIT*, 62(4):1149–1174, 2022. 155

[CL22] G. Ceruti and C. Lubich. An unconventional robust integrator for dynamical low-rank approximation. *BIT*, 62:23–44, March 2022. 155, 161, 174, 179

[CLMW11] E. J. Candès, X. Li, Y. Ma, and J. Wright. Robust principal component analysis? *J. ACM*, 58(3), June 2011. 24

[CNX22] D. Cai, J. Nagy, and Y. Xi. Fast deterministic approximation of symmetric indefinite kernel matrices with high dimensional datasets. *SIAM J. Matrix Anal. Appl.*, 43(2):1003–1028, 2022. 74, 77, 80, 96, 98, 101, 103

[Coh16] M. B. Cohen. Nearly tight oblivious subspace embeddings by trace inequalities. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 278–287. 2016. 24

[CS10] S. Chaturantabut and D. C. Sorensen. Nonlinear model reduction via discrete empirical interpolation. *SIAM J. Sci. Comput.*, 32(5):2737–2764, 2010. 33, 134, 156

[CT06] E. J. Candès and T. Tao. Near-optimal signal recovery from random projections: Universal encoding strategies? *IEEE Trans. Inf. Theory*, 52(12):5406–5425, 2006. 18

[CW09] K. L. Clarkson and D. P. Woodruff. Numerical linear algebra in the streaming model. In *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*, page 205–214. ACM, 2009. 11, 60

[CW17] K. L. Clarkson and D. P. Woodruff. Low-rank approximation and regression in input sparsity time. *J. ACM*, 63(6):1–45, 2017. 24

[CY25] A. Cortinovis and L. Ying. A sublinear-time randomized algorithm for column and row subset selection based on strong rank-revealing QR factorizations. *SIAM J. Matrix Anal. Appl.*, 46(1):22–44, 2025. 110, 120

[DCLT19] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the NAACL-HLT, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics, 2019. 70

[Dem97] J. W. Demmel. *Applied Numerical Linear Algebra*. SIAM, 1997. 10

[DG16] Z. Drmač and S. Gugercin. A new selection operator for the discrete empirical interpolation method—improved a priori error bound and extensions. *SIAM J. Sci. Comput.*, 38(2):A631–A648, 2016. 33, 156

[DG17] D. Dua and C. Graff. UCI machine learning repository, 2017. 99

[DG20] J. A. Duersch and M. Gu. Randomized projection for rank-revealing matrix factorizations and low-rank approximations. *SIAM Rev.*, 62(3):661–682, 2020. 33, 135

[DGHL12] J. Demmel, L. Grigori, M. Hoemmen, and J. Langou. Communication-optimal parallel and sequential QR and LU factorizations. *SIAM J. Sci. Comput.*, 34(1):A206–A239, 2012. 37

[DH11] T. A. Davis and Y. Hu. The University of Florida sparse matrix collection. *ACM Trans. Math. Softw.*, 38(1), 2011. 121

[DK70] C. Davis and W. M. Kahan. The rotation of eigenvectors by a perturbation. III. *SIAM J. Numer. Anal.*, 7(1):1–46, 1970. 41, 43

[DKM20] M. Dereziński, R. Khanna, and M. W. Mahoney. Improved guarantees and a multiple-descent curve for column subset selection and the Nyström method. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20. Curran Associates Inc., 2020. 32

[DKR02] P. Drineas, I. Kerenidis, and P. Raghavan. Competitive recommendation systems. In *Proceedings of the Thiry-Fourth Annual ACM Symposium on Theory of Computing*, STOC '02, page 82–90. ACM, 2002. 31

[DM05] P. Drineas and M. W. Mahoney. On the Nyström method for approximating a Gram matrix for improved kernel-based learning. *J. Mach. Learn. Res.*, 6(72):2153–2175, 2005. 72

[DM21] M. Dereziński and M. Mahoney. Determinantal point processes in randomized numerical linear algebra. *Notices Amer. Math. Soc.*, 60(01):1, 2021. 33

[DM23] Y. Dong and P.-G. Martinsson. Simpler is better: a comparative study of randomized pivoting algorithms for CUR and interpolative decompositions. *Adv. Comput. Math.*, 49(66), 2023. 24, 32, 33, 34, 102, 104, 110, 111, 120, 135, 136, 141, 145, 147

[DMIMW12] P. Drineas, M. Magdon-Ismail, M. W. Mahoney, and D. P. Woodruff. Fast approximation of matrix coherence and statistical leverage. *J. Mach. Learn. Res.*, 13(111):3475–3506, 2012. 18, 22, 33

[DMM08] P. Drineas, M. W. Mahoney, and S. Muthukrishnan. Relative-error CUR matrix decompositions. *SIAM J. Matrix Anal. Appl.*, 30(2):844–881, 2008. 31, 33

[DMMS11] P. Drineas, M. W. Mahoney, S. Muthukrishnan, and T. Sarlós. Faster least squares approximation. *Numer. Math.*, 117(2):219–249, 2011. 38

[DPN+23] M. Donello, G. Palkar, M. H. Naderi, D. C. Del Rey Fernández, and H. Babaee. Oblique projection for scalable rank-adaptive reduced-order modelling of nonlinear stochastic partial differential equations with time-dependent bases. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 479(2278):20230320, 2023. 133, 151, 155, 171, 182

[DPP23] N. Derevianko, G. Plonka, and M. Petz. From ESPRIT to ESPIRA: estimation of signal parameters by iterative rational approximation. *IMA J. Numer. Anal.*, 43(2):789–827, 2023. 64

[DR10] A. Deshpande and L. Rademacher. Efficient volume sampling for row/column subset selection. In *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, pages 329–338, 2010. 33

[DRVW06] A. Deshpande, L. Rademacher, S. S. Vempala, and G. Wang. Matrix approximation and projective clustering via volume sampling. *Theory Comput.*, 2(12):225–247, 2006. 33

[DS01] K. R. Davidson and S. J. Szarek. Local operator theory, random matrices and Banach spaces. In W. Johnson and J. Lindenstrauss, editors, *Handbook of the Geometry of Banach Spaces*, volume 1, pages 317–366. Elsevier, 2001. 18

[EI95] S. C. Eisenstat and I. C. F. Ipsen. Relative perturbation techniques for singular value problems. *SIAM J. Numer. Anal.*, 32(6):1972–1988, 1995. 19, 20

[EP94] L. Eldén and H. Park. Block downdating of least squares solutions. *SIAM J. Matrix Anal. Appl.*, 15(3):1018–1034, 1994. 67

[Epp] E. N. Epperly. Low-rank approximation toolbox: The Gram correspondence. `https://www.ethanepperly.com/index.php/2024/12/07/low-rank-approximation-toolbox-the-gram-correspondence/`. Accessed: 2025-04-19. 30

[EY36] C. Eckart and G. Young. The approximation of one matrix by another of lower rank. *Psychometrika*, 1(3):211–218, 1936. 16, 25

[FTU23] Z. Frangella, J. A. Tropp, and M. Udell. Randomized Nyström preconditioning. *SIAM J. Matrix Anal. Appl.*, 44(2):718–752, 2023. 11, 32, 71

[GE96] M. Gu and S. C. Eisenstat. Efficient algorithms for computing a strong rank-revealing QR factorization. *SIAM J. Sci. Comput.*, 17(4):848–869, 1996. 33, 37, 120, 123, 164, 165, 173

[GE03] I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *J. Mach. Learn. Res.*, 3:1157–1182, 2003. 32

[GG20] I. V. Gosea and S. Gugercin. The AAA framework for modeling linear dynamical systems with quadratic output. *arXiv preprint arXiv:2005.10316*, 2020. 64

[GH22] P. Y. Gidisu and M. E. Hochstenbach. A hybrid DEIM and leverage scores based method for CUR index selection. In *Progress in Industrial Mathematics at ECMI 2021*, pages 147–153. Springer International Publishing, 2022. 33, 133, 134, 141

[Git11] A. Gittens. The spectral norm error of the naïve Nyström extension. *arXiv preprint arXiv:1110.5305*, 2011. 30, 71

[GM16] A. Gittens and M. W. Mahoney. Revisiting the Nyström method for improved large-scale machine learning. *J. Mach. Learn. Res.*, 17(1):3977–4041, 2016. 30, 70, 71, 72, 94

[GOS+10] S. A. Goreinov, I. V. Oseledets, D. V. Savostyanov, E. E. Tyrtyshnikov, and N. L. Zamarashkin. *How to Find a Good Submatrix*, pages 247–256. 2010. 34

[GPW12] A. C. Gilbert, J. Y. Park, and M. B. Wakin. Sketched SVD: Recovering spectral features from compressive measurements. *arXiv preprint arXiv:1211.0361*, 2012. 12, 37, 41, 43

[GS15] A. Gisbrecht and F.-M. Schleif. Metric and non-metric proximity transformations at linear costs. *Neurocomputing*, 167:643–657, 2015. 70, 75

[GT19] A. Gopal and L. N. Trefethen. Representation of conformal maps by rational functions. *Numer. Math.*, 142:359–382, 2019. 64

[GTP18] S. Gratton and D. Titley-Peloquin. Improved bounds for small-sample estimation. *SIAM J. Matrix Anal. Appl.*, 39(2):922–931, 2018. 158, 166

[GTZ97] S. Goreinov, E. Tyrtyshnikov, and N. Zamarashkin. A theory of pseudoskeleton approximations. *Linear Algebra Appl.*, 261(1):1–21, 1997. 102

[GVL80] G. H. Golub and C. F. Van Loan. An analysis of the total least squares problem. *SIAM J. Numer. Anal.*, 17(6):883–893, 1980. 36, 56, 57

[GVL13] G. H. Golub and C. F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, 4 edition, 2013. 10, 16, 24, 33, 129, 137

[Han87] P. C. Hansen. The truncated SVD as a method for regularization. *BIT*, 27(4):534–553, 1987. 24

[HH20] K. Hamm and L. Huang. Perspectives on CUR decompositions. *Appl. Comput. Harmon. Anal.*, 48(3):1088–1099, 2020. 148

[HH21] K. Hamm and L. Huang. Perturbations of CUR decompositions. *SIAM J. Matrix Anal. Appl.*, 42(1):351–375, 2021. 103, 152

[Hig02] N. J. Higham. *Accuracy and Stability of Numerical Algorithms*. SIAM, second edition, 2002. 112, 113, 115, 129, 130

[HMT11] N. Halko, P.-G. Martinsson, and J. A. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Rev.*, 53(2):217–288, 2011. 10, 11, 21, 22, 24, 27, 28, 49, 75, 85, 86, 92, 93, 148, 155

[HNS23] M. Hochbruck, M. Neher, and S. Schrammer. Rank-adaptive dynamical low-rank integrators for first-order and second-order matrix differential equations. *BIT*, 63(1):9, 2023. 155

[Hoc17] A. Hochman. FastAAA: A fast rational-function fitter. In *26th Conference on Electrical Performance of Electronic Packaging and Systems*, pages 1–3. 2017. 67

[HP92] Y. P. Hong and C.-T. Pan. Rank-revealing QR factorizations and the singular value decomposition. *Math. Comp.*, 58(197):213–232, 1992. 33, 37

[HPR22] J. S. Hesthaven, C. Pagliantini, and N. Ripamonti. Rank-adaptive structure-preserving model order reduction of hamiltonian systems. *ESAIM: M2AN*, 56(2):617–650, 2022. 155

[HPS12] H. Harbrecht, M. Peters, and R. Schneider. On the low-rank approximation by the pivoted cholesky decomposition. *Appl. Numer. Math.*, 62(4):428–440, 2012. Third Chilean Workshop on Numerical Analysis of Partial Differential Equations (WONAPDE 2010). 34

[HS23] C. D. Hauck and S. Schnake. A predictor-corrector strategy for adaptivity in dynamical low-rank approximations. *SIAM J. Matrix Anal. Appl.*, 44(3):971–1005, 2023. 155

[IN09] I. C. F. Ipsen and B. Nadler. Refined perturbation bounds for eigenvalues of Hermitian and non-Hermitian matrices. *SIAM J. Matrix Anal. Appl.*, 31(1):40–53, 2009. 134

[JL84] W. B. Johnson and J. Lindenstrauss. Extensions of Lipschitz mappings into a Hilbert space. In *Conference in modern analysis and probability*, volume 26 of *Contemp. Math.*, pages 189–206. Amer. Math. Soc., 1984. 18

[KL07] O. Koch and C. Lubich. Dynamical low-rank approximation. *SIAM J. Matrix Anal. Appl.*, 29(2):434–454, 2007. 150, 151, 154

[KL24] D. Kressner and H. Y. Lam. Randomized low-rank approximation of parameter-dependent matrices. *Numer. Linear Algebra Appl.*, 31(6):e2576, 2024. 151, 155, 167, 173, 177, 182, 183, 184

[KLMU20] D. Kressner, J. Latz, S. Massei, and E. Ullmann. Certified and fast computations with shallow covariance kernels. *Foundations of Data Science*, 2(4):487–512, 2020. 150, 156

[KLW16] E. Kieri, C. Lubich, and H. Walach. Discretized dynamical low-rank approximation in the presence of small singular values. *SIAM J. Numer. Anal.*, 54(2):1020–1038, 2016. 155

[KN14] D. M. Kane and J. Nelson. Sparser Johnson-Lindenstrauss transforms. *J. ACM*, 61(1), January 2014. 23

[Kri09] A. Krizhevsky. Learning multiple layers of features from tiny images. 2009. 141

[KT11] D. Kressner and C. Tobler. Low-rank tensor Krylov subspace methods for parametrized linear systems. *SIAM J. Matrix Anal. Appl.*, 32(4):1288–1316, 2011. 176

[LAN24] L. Lazzarino, H. Al Daas, and Y. Nakatsukasa. Matrix perturbation analysis of methods for extracting singular values from approximate singular subspaces. *arXiv preprint arXiv:2409.09187*, 2024. 160

[LBKL15] M. Li, W. Bi, J. T. Kwok, and B.-L. Lu. Large-scale Nyström kernel matrix approximation using randomized SVD. *IEEE Trans. Neural Netw. Learn. Syst.*, 26(1):152–164, 2015. 72

[Li98a] R.-C. Li. Relative perturbation theory: I. eigenvalue and singular value variations. *SIAM J. Matrix Anal. Appl.*, 19(4):956–982, 1998. 44

[Li98b] R.-C. Li. Relative perturbation theory: II. eigenspace and singular subspace variations. *SIAM J. Matrix Anal. Appl.*, 20(2):471–492, 1998. 37, 41, 43, 44

[LLS+17] H. Li, G. C. Linderman, A. Szlam, K. P. Stanton, Y. Kluger, and M. Tygert. Algorithm 971: An implementation of a randomized algorithm for principal component analysis. *ACM Trans. Math. Softw.*, 43(3):28:1–28:14, 2017. 77

[LMPV22] P. Lietaert, K. Meerbergen, J. Pérez, and B. Vandereycken. Automatic rational approximation and linearization of nonlinear eigenvalue problems. *IMA J. Numer. Anal.*, 42(2):1087–1115, 2022. 64

[LO14] C. Lubich and I. V. Oseledets. A projector-splitting integrator for dynamical low-rank approximation. *BIT*, 54(1):171–188, 2014. 155, 167, 173, 183, 184

[Lub14] C. Lubich. *Low-Rank Dynamics*, pages 381–396. Springer International Publishing, Cham, 2014. 154

[Mar19] P.-G. Martinsson. Randomized methods for matrix computations. *The Mathematics of Data*, 25(4):187–231, 2019. 171, 182

[Mas22] S. Massei. Some algorithms for maximum volume and cross approximation of symmetric semidefinite matrices. *BIT*, 62(1):195–220, 2022. 33

[MD09] M. W. Mahoney and P. Drineas. CUR matrix decompositions for improved data analysis. *Proc. Natl. Acad. Sci.*, 106(3):697–702, 2009. 18, 31, 33, 102, 133

[MDM+23] R. Murray, J. Demmel, M. W. Mahoney, N. B. Erichson, M. Melnichenko, O. A. Malik, L. Grigori, P. Luszczek, M. Dereziński, M. E. Lopes, T. Liang, H. Luo, and J. Dongarra. Randomized numerical linear algebra : A perspective on the field with an eye to software. *arXiv preprint arXiv:2302.11474*, 2023. 11

[Mec19] E. S. Meckes. *The random matrix theory of the classical compact groups*, volume 218. Cambridge University Press, 2019. 18, 49, 138

[Mir60] L. Mirsky. Symmetric gauge functions and unitarily invariant norms. *Q. J. Math.*, 11(1):50–59, 01 1960. 25, 44

[MN24] M. Meier and Y. Nakatsukasa. Fast randomized numerical rank estimation for numerically low-rank matrices. *Linear Algebra Appl.*, 686:1–32, 2024. 157, 172

[MO18] A. Mikhalev and I. Oseledets. Rectangular maximum-volume submatrices and their applications. *Linear Algebra Appl.*, 538:187–211, 2018. 133

[MP67] V. A. Marčenko and L. A. Pastur. Distribution of eigenvalues for some sets of random matrices. *Mathematics of the USSR-Sbornik*, 1(4):457–483, 1967. 18

[MSM14] X. Meng, M. A. Saunders, and M. W. Mahoney. LSRN: a parallel iterative solver for strongly over- or underdetermined systems. *SIAM J. Sci. Comput.*, 36(2):C95–C118, 2014. 11

[MT20] P.-G. Martinsson and J. A. Tropp. Randomized numerical linear algebra: Foundations and algorithms. *Acta Numer.*, 29:403–572, 2020. 10, 17, 21, 23, 32, 34, 63, 71, 72, 102, 158, 172

[MW17] C. Musco and D. P. Woodruff. Sublinear time low-rank approximation of positive semidefinite matrices. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 672–683, 2017. 71

[Nak20] Y. Nakatsukasa. Fast and stable randomized low-rank matrix approximation. *arXiv preprint arXiv:2009.11392*, 2020. 29, 74, 75, 76, 84, 85, 97, 103, 112, 119, 122, 123, 129, 134, 155

[NH12] Y. Nakatsukasa and N. J. Higham. Backward stability of iterations for computing the polar decomposition. *SIAM J. Matrix Anal. Appl.*, 33(2):460–479, 2012. 112

[NL08] A. Nonnenmacher and C. Lubich. Dynamical low-rank approximation: applications and numerical experiments. *Math. Comput. Simulation*, 79(4):1346–1357, 2008. 150, 151

[NN13] J. Nelson and H. L. Nguyên. OSNAP: Faster numerical linear algebra algorithms via sparser subspace embeddings. In *Proc. IEEE 54th Annu. Symp. Found. Comput. Sci.*, pages 117–126, 2013. 23, 24

[NP23] Y. Nakatsukasa and T. Park. Randomized low-rank approximation for symmetric indefinite matrices. *SIAM J. Matrix Anal. Appl.*, 44(3):1370–1392, 2023. 13, 103

[NST18] Y. Nakatsukasa, O. Sète, and L. N. Trefethen. The AAA algorithm for rational approximation. *SIAM J. Sci. Comput.*, 40(3):A1494–A1522, 2018. 36, 59, 64, 65, 66

[NT24] Y. Nakatsukasa and J. A. Tropp. Fast and accurate randomized algorithms for linear systems and eigenvalue problems. *SIAM J. Matrix Anal. Appl.*, 45(2):1183–1214, 2024. 11, 22

[Nys30] E. J. Nyström. Über die praktische auflösung von integralgleichungen mit anwendungen auf randwertaufgaben. *Acta Math.*, 54:185 – 204, 1930. 30, 70

[OG17] D. Oglic and T. Gärtner. Nyström method with kernel k-means++ samples as landmarks. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 2652–2660. PMLR, 06–11 Aug 2017. 33

[OG19] D. Oglic and T. Gärtner. Scalable learning in reproducing kernel Kreĭn spaces. In *International Conference on Machine Learning*, pages 4912–4921. PMLR, 2019. 70, 75

[Osi23] A. Osinsky. Close to optimal column approximations with a single SVD. *arXiv preprint arXiv:2308.09068*, 2023. 33, 165

[Ost59] A. M. Ostrowski. A quantitative formulation of Sylvester's law of inertia. *Proc. Natl. Acad. Sci. USA*, 45(5):740–744, 1959. 19

[PABW18] F. Pourkamali-Anaraki, S. Becker, and M. Wakin. Randomized clustered Nyström for large-scale kernel machines. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1):3960–3967, Apr. 2018. 72

[Pan00] C.-T. Pan. On the existence and computation of rank-revealing LU factorizations. *Linear Algebra Appl.*, 316(1):199–222, 2000. 33

[PDG20] B. Peherstorfer, Z. Drmač, and S. Gugercin. Stability of discrete empirical interpolation and gappy proper orthogonal decomposition with randomized and deterministic sampling points. *SIAM J. Sci. Comput.*, 42(5):A2837–A2864, 2020. 133, 134, 139, 141

[PKB14] D. Papailiopoulos, A. Kyrillidis, and C. Boutsidis. Provable deterministic leverage score sampling. In *Proceedings of the 20th ACM SIGKDD*, page 997–1006. ACM, 2014. 33

[PN23] T. Park and Y. Nakatsukasa. A fast randomized algorithm for computing an approximate null space. *BIT*, 63(2):36, 2023. 12

[PN25a] T. Park and Y. Nakatsukasa. Accuracy and stability of CUR decompositions with oversampling. *SIAM J. Matrix Anal. Appl.*, 46(1):780–810, 2025. 13

[PN25b] T. Park and Y. Nakatsukasa. Low-rank approximation of parameter-dependent matrices via cur decomposition. *SIAM J. Sci. Comput.*, 47(3):A1858–A1887, 2025. 14

[PR14] B. Piccoli and F. Rossi. Generalized Wasserstein Distance and its Application to Transport Equations with Source. *Arch. Ration. Mech. Anal.*, 211(1):335–358, 2014. 70

[PW94] C. Paige and M. Wei. History and generality of the CS decomposition. *Linear Algebra Appl.*, 208-209:303–326, 1994. 137

[RMMM22] A. Ray, N. Monath, A. McCallum, and C. Musco. Sublinear time approximation of text similarity matrices. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(7):8072–8080, 2022. 74, 96, 98

[RT08] V. Rokhlin and M. Tygert. A fast randomized algorithm for overdetermined linear least-squares regression. *Proc. Natl. Acad. Sci. USA*, 105(36):13212–13217, 2008. 11

[Sar06] T. Sarlós. Improved approximation algorithms for large matrices via random projections. In *Proc. IEEE 47th Annu. Symp. Found. Comput. Sci.*, page 143–152, 2006. 11, 17, 38

[SCMW24] A. Shimizu, X. Cheng, C. Musco, and J. Weare. Improved active learning via dependent leverage score sampling. *The Twelfth International Conference on Learning Representations*, 2024. 134

[SE16] D. C. Sorensen and M. Embree. A DEIM induced CUR factorization. *SIAM J. Sci. Comp.*, 38(3):A1454–A1482, 2016. 32, 33, 34, 102, 104, 109, 133, 141, 145, 148, 156

[Shi21] Y. Shitov. Column subset selection is NP-complete. *Linear Algebra Appl.*, 610:52–58, 2021. 32

[SS90] G. W. Stewart and J.-g. Sun. *Matrix Perturbation Theory*. Academic Press, 1990. 42

[Ste98] G. W. Stewart. *Matrix Algorithms: Volume 1: Basic Decompositions*. SIAM, 1998. 37

[SW02] A. Stathopoulos and K. Wu. A block orthogonalization procedure with constant synchronization requirements. *SIAM J. Sci. Comput.*, 23(6):2165–2182, 2002. 37

[Szy06] D. B. Szyld. The many proofs of an identity on the norm of oblique projections. *Numer. Algorithms*, 42(3):309–323, 2006. 26, 27, 147

[TB97] L. N. Trefethen and D. Bau. *Numerical Linear Algebra*. SIAM, 1997. 33

[Tre19] L. N. Trefethen. *Approximation Theory and Approximation Practice, Extended Edition*. SIAM, Philadelphia, PA, 2019. 65

[Tro11] J. A. Tropp. Improved analysis of the subsampled randomized Hadamard transform. *Advances in Adaptive Data Analysis*, 03(01n02):115–126, 2011. 22, 49

[TW23] J. A. Tropp and R. J. Webber. Randomized algorithms for low-rank matrix approximation: Design, analysis, and applications. *arXiv preprint arXiv:2306.12418*, 2023. 28

[TYUC17a] J. A. Tropp, A. Yurtsever, M. Udell, and V. Cevher. Fixed-rank approximation of a positive-semidefinite matrix from streaming data. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, page 1225–1234, 2017. 23, 72, 77

[TYUC17b] J. A. Tropp, A. Yurtsever, M. Udell, and V. Cevher. Practical sketching algorithms for low-rank matrix approximation. *SIAM J. Matrix Anal. Appl.*, 38(4):1454–1485, 2017. 29, 75, 155

[TYUC19] J. A. Tropp, A. Yurtsever, M. Udell, and V. Cevher. Streaming low-rank matrix approximation with an application to scientific simulation. *SIAM J. Sci. Comput.*, 41(4):A2430–A2463, 2019. 11, 24, 60

[UT19] M. Udell and A. Townsend. Why are big data matrices approximately low rank? *SIAM J. Math. Data Sci.*, 1(1):144–160, 2019. 27

[VHV91] S. Van Huffel and J. Vandewalle. *The total least squares problem: computational aspects and analysis*. SIAM, 1991. 36, 57

[VM17] S. Voronin and P.-G. Martinsson. Efficient algorithms for CUR and interpolative matrix decompositions. *Adv. Comput. Math.*, 43(3):495–516, 2017. 33, 110, 120, 135, 141

[Wed72] P.-Å. Wedin. Perturbation bounds in connection with singular value decomposition. *BIT*, 12(1):99–111, 1972. 41, 43

[WGM19] S. Wang, A. Gittens, and M. W. Mahoney. Scalable kernel k-means clustering with Nyström approximation: Relative-error bounds. *J. Mach. Learn. Res.*, 20(1):431–479, jan 2019. 72

[WLRT08] F. Woolfe, E. Liberty, V. Rokhlin, and M. Tygert. A fast randomized algorithm for the approximation of matrices. *Appl. Comput. Harmon. Anal.*, 25(3):335–366, 2008. 10, 22

[WLZ14] S. Wang, L. Luo, and Z. Zhang. SPSD matrix approximation vis column selection: Theories, algorithms, and extensions. *J. Mach. Learn. Res.*, 17:49:1–49:49, 2014. 75

[Woo14] D. Woodruff. *Sketching as a Tool for Numerical Linear Algebra*. Foundations and Trends® in Theoretical Computer Science Series. Now Publishers, 2014. 10, 17, 23

[WS00] C. Williams and M. Seeger. Using the Nyström method to speed up kernel machines. In T. Leen, T. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems*, volume 13. MIT Press, 2000. 11, 30, 70

[WS17] Y. Wang and A. Singh. Provably correct algorithms for matrix column subset selection with selectively sampled data. *J. Mach. Learn. Res.*, 18(1):5699–5740, 2017. 31

[Xia24] J. Xia. Making the Nyström method highly accurate for low-rank approximations. *SIAM J. Sci. Comput.*, 46(2):A1076–A1101, 2024. 110, 120

[Xin24] X. Xing. Strong rank revealing QR decomposition, 2024. Retrieved December 24, 2023. 173

[YMM16] J. Yang, X. Meng, and M. W. Mahoney. Implementing randomized matrix algorithms in parallel and distributed environments. *Proceedings of the IEEE*, 104(1):58–92, 2016. 11

[ZO18] N. L. Zamarashkin and A. I. Osinsky. On the existence of a nearly optimal skeleton approximation of a matrix in the Frobenius norm. *Dokl. Math.*, 97(2):164–166, 2018. 33, 110

[ZW16] R. Zimmermann and K. Willcox. An accelerated greedy missing point estimation procedure. *SIAM J. Sci. Comput.*, 38(5):A2827–A2850, 2016. 133, 134