Federated Linear Dimensionality Reduction

Andreas A. Grammenos

Department of Computer Science and Technology
University of Cambridge

This dissertation is submitted for the degree of
Doctor of Philosophy

Wolfson College April 2021

This thesis is dedicated to my loving parents Antonis & Margarita, my brother Marios, and my wife Vasiliki for their infinite support . . .

Declaration

Andreas A. Grammenos
April 2021

Acknowledgements

Firstly, I would like to thank my supervisors Cecilia Mascolo and Jon Crowcroft. If not for them, I would not be studying for this degree, nor writing this Thesis and had it not been for a random email sent in the middle of the night, probably would have followed a completely different path. During these years both patiently monitored and directed my work, facilitating the expression and development of the good qualities that I may hold. Both acted as mentors and instilled different traits which proved invaluable while navigating the rough landscape of academia. In one hand, Cecilia taught me how to put structure into chaos and narrow my focus to the things that matter. On the other hand, Jon showed me that pub defined science exists and can be successful. I consider them to be my academic parents and have treated me as such. From the bottom of my heart, thank you both for your unreserved support that helped me make the impossible possible.

Subsequently, I would like to extend my heartfelt gratitude to many people that tagged along for the journey. More concretely, I want to thank my collaborators without whom I would not be able to finish this thesis. The first in line, is Gareth T., who apart from his cunning and elaborate humour allowed me, with a publication, to cross an item out of my bucket-list. As promised, it will always remain at the top of my "highlighted works". Next is Rodrigo M., who was an instrumental collaborator of mine over the years and continues to be. Moreover, I would like to thank Andrea F. for the awesome times we had chatting about magnets and interference. It was amazing to quantify how much a Black Sabbath record can affect an earable. From my Cambridge based group, I would like to extend my thanks to Dimitris S., Lorena Q., and Alessandro M. for both the science as well as the banter. Moreover, Eva K. and Themis C. were excellent collaborators but also constant pillars of support. I would also like to acknowledge the support received from the Alan Turing Institute for not only fully funding my studies but also providing me with a place I can call an academic home. The amazing interactions there with Valerio G., Shouto Y., Luca M., Sebastian B., Aris P., Lina G., Abhinav M., Alex B., Petros D., and Dave B. were just the icing on the cake. Finally, I would also like to thank my academic siblings Anastasios N. and Andrius A. for their support throughout the years.

Lastly, I owe my sincere and earnest gratitude to my wife, brother, and parents for their love and unconditional support through this journey, without which any of my achievements would not have been possible.

Abstract

In recent years, the explosive rate of dataset expansion has offered the ability for researchers to access an unprecedented amount of information. Moreover, in addition to actual dataset size increases, the types and locations of data generators are more heterogeneous than ever - ranging from traditional servers to a myriad of IoT devices. These facts, coupled with recent emphasis on privacy and data-ownership, led to the creation of *federated* datasets. Such datasets are characterised by their massive size and are usually scattered across decentralised edge devices, each holding their local data samples. As exciting as these federated datasets might be, they introduce an astounding challenge: *how to efficiently process federated data at scale?* Naturally, given these constraints, centralisation of such datasets is often intractable, thus making traditional analytical methods inapplicable.

This thesis introduces a suite of mathematical advancements that makes traditional learning algorithms applicable to the federated setting, by summarising the massive amounts of information into succinct dataset-specific representations. Concretely, we focus primarily on linear dimensionality reduction and, in particular, on Principal Component Analysis (PCA) due to its pervasiveness, along with its ability to process unstructured data. The first advancement we introduce is a novel algorithm to perform streaming and memory-limited dimesionality reduction at the edge that uses a generalisation of incremental Singular Value Decomposition (SVD). Further, we provide a rank-adaptive SVD extension able to account for distribution shifts over-time. Subsequently, building upon previous constructions, we present an (ε, δ)-differentially private federated algorithm for PCA. To achieve federation, we put forth a lightweight merging algorithm that unlocks the ability to process each subproblem locally at the edge which, in turn, through merging is propagated accordingly. We are able to guarantee differential privacy via an input-perturbation scheme in which the covariance matrix of a dataset is perturbed with a non-symmetric random Gaussian matrix.

To evaluate the practicality of our innovations, we describe an algorithm able to perform task scheduling on federated data centres. The scheduler enables each decentralised node to incrementally compute its local model and independently execute scheduling decisions on whether to accept an incoming job based on the workload seen thus far. Finally, we complement our findings with an evaluation on synthetic as well as real-world datasets including sensor node measurements, hard-written images, and wine quality readings, considering a wide range of data modalities and dimensionalities.

Table of contents

Nomenclature

Acronyms/Abbreviations

i.i.d.	Independent and identically distributed
w.r.t.	with respect to
cf.	confer
DASM	Distributed Agglomerative Summary Model
DP	Differentially Private
e.g.	exempli gratia
etc.	et cetera
i.e.	id est
iff	if and only if
LHS	left hand side
PCA	Principal Components Analysis
PD	Positive Definite matrix
PL	Privacy Loss
PSD	Positive Semidefinite matrix
RHS	right hand side
s.t.	such that
SGD	Stochastic Gradient Descent
SVD	Singular Value Decomposition

Greek Symbols

$\mathcal{O}$	Upper bound of complexity
Ω	Lower bound of complexity
Θ	Asymptotically tight upper and lower bound of complexity

Distributions

$\mathcal{N}$	Normal distribution

Chapter 1

Introduction

Large-scale computation is an instrumental component of modern society, providing the means and data to make pivotal decisions affecting billions. Useful data representations can form patterns that can then be exploited to aid in the discovery of underlying disease causes, detect fraud, and offer user-tailored recommender systems that power tools and services we take for granted such as personalised music or product recommendations. To do so scientists rely on vast computing resources that can be used to tap and exploit the wealth of stored information. One of the main purposes of processing "big-data" is to extract useful and representative trends or features and has been an active research topic in the scientific literature spanning decades [54, 129, 39, 176].

However, modern datasets expand at an alarming rate much quicker than the hardware advancements can keep up with and have long surpassed Moore's law. Further, the sources of ever-generating information are more heterogeneous than ever ranging from traditional servers to a myriad of connected Internet of Things (IoT). These sources are able to provide multiple continuous streams of data that, in turn, need to be stored and processed. More importantly, such devices can be resource-restricted, in the sense that they could be located in places of poor or intermittent connectivity, have limited bandwidth, scarce computing capabilities or a combination thereof. Inherently, this heterogeneity not only increases the volume of the actual datasets but imposes severe constraints on how to process vasts quantities of data that can also be scattered across the globe. Naturally, working on these datasets without any assumptions is an intractable task; thankfully, has have been a number of intuitive observations that paved the way to tractable processing of arbitrarily large datasets.

One of the most crucial, and perhaps initially confusing, observations is that that most real-world datasets are *massively redundant* - meaning that their intrinsic dimension is considerably lower than their actual one. Moreover, working in high-dimensions not only can be computationally intractable, but also incurs various anomalies and phenomena not present in lower dimensions due to the "curse-of-dimensionality" [172, 145]. More importantly, and again perhaps initially counter-intuitively, this phenomenon is even more evident as the dimensionality of the dataset increases; this boils down to the empirical observation that the actual rank-frequency distribution of the dataset presents itself as an inverse relation. Mathematically, this could potentially be explained because, as dimensionality increases, the volume of topological space the data-points reside in grows so large that the data contained in that space become sparse. Notably, to date, no concrete proof for this conjecture has been provided; however, mounting empirical evidence has shown this conjecture to be valid; one such example is Zipf's law [200] that states many types of data studied in physical and social sciences exhibit this behaviour [3, 62]. In a nutshell, this observation allows us to guarantee that the minimum dimensionality required to retain most of the characteristics and features of the original massive dataset can be summarised into one that has a dimension significantly lower than the original one. However, while we do know that in such datasets the intrinsic dimension is small, we do not have a closed form solution in order to find its exact value for all cases. This begs the question: *given this observation, how could we produce summarisations of massive-datasets in a scalable and efficient manner?*

At the same time, a large fraction of the most interesting datasets stems from privacy-sensitive domains with notable examples being health and personalised user data. Complicating things further, there has been a number of growing concerns on how to access these datasets while respecting user privacy [186, 196]. These very issues was what led some countries to introduce specific legislation trying to address them, such as the recent European Union's General Data Protection Regulation (GDPR) [4]. This is because it has been shown numerous times that even after processing of the actual data the resulting models can reveal user identifiable information which could then be exploited for nefarious reasons [52]. To that end, the summarisations produced should be able *by construction*, if required, to provide formal guarantees of statistical privacy. That property would, in turn, enable data analysts to tap into the knowledge offered within these datasets but without the risk of revealing individual user information. To tackle this problem, several statistical frameworks have been proposed to achieve this goal. However, the predominant technique that is able to offer such guarantees is *differential-privacy* [50]. Essentially, the key idea behind differential privacy is that provided the effect of an arbitrary substitution in a dataset is sufficiently small, then the query result cannot be used to infer personally identifiable information about a single individual and therefore is able to guarantee privacy. However, two of the major drawbacks of differential privacy that prevented its widespread use were its steep computational costs and more importantly, its substantial sample complexity requirements [174]. Notably, recent advancements have improved the computational aspects of differential privacy thus improving its applicability. Unfortunately, the sample complexity to guarantee differential privacy is by construction significant; meaning that with traditional, smaller, datasets its applicability remains limited. On the other hand, in the case of massive datasets, the sample complexity ends up being an opportunity rather than an actual issue, hence, making such use-cases ideal applications of differential privacy.

The aforementioned problem of dataset summarisation can also be thought, a bit more formally, as *dimensionality reduction*. There have been many frameworks put forth to achieve this but by far the most ubiquitous ones are arguably subspace tracking [11] and Principal Component Analysis (PCA) [100]. Their pervasiveness in most scientific domains makes them indispensable tools for detecting structure in collected data [6, 110, 169, 181] with minimal assumptions. More importantly, the best-fit subspaces and computed principal components for a given dataset, not only can be used for dimensionality reduction, but also provide critical insights for performing signal estimation, noise filtering, or anomaly detection [159]. This is because assuming a dataset admits an exploitable low-dimensional structure, these methods then enable otherwise infeasible tasks, such as large scale inference or parametric studies to be performed.

Intuitively, the central idea behind PCA is simple: given an arbitrary dataset consisting of a large number of *interrelated variables*, reduce its dimensionality by transforming them into a new set of fewer *uncorrelated variables* - the Principal Components (PCs) - while retaining the *maximal* variance out of the original ones. Notably, PCA can be computed using a number of different ways; though traditionally, there have been two predominant methods for its derivation. These can be broadly separated if the computation of PCA requires the expansion of specific matrices, such as the covariance or correlation matrices of the dataset, or not. However, such expansions become prohibitively expensive as datasets scale and thus can quickly become an infeasible option. Necessitated by the scale of datasets considered, the most efficient methods, resource-wise, are incremental and exploit various techniques in order to avoid the expensive covariance/correlation expansions.

This dissertation studies the problem of dimensionality reduction on federated datasets through the lens of subspace estimation with a particular focus on PCA, as is one of the most ubiquitous tools when performing data analysis. However, its computation is normally prohibitively expensive and not trivial if the dataset size is large enough. One of the great uses of a federated algorithm is that analysts could be able to tap into dataset sizes previously unavailable, whether it is due to the infeasibility of materialising the whole dataset, its processing - or more commonly both. The applications at hand are numerous, but to give some concrete examples directly relevant to this work is: analytics. Relating to the work presented herein, we use the federated metrics to assign tasks appropriately while maximising the utilisation. Taking this one step further, one could similarly exploit these metrics to produce representative trends that could be tracked over time. This could potentially help practitioners of large decentralised systems to monitor the performance of their computation networks providing valuable insights. Another interesting application is anomaly and outlier detection - a problem that has been long been a great candidate for approximate sketching or PCA algorithms [27, 150, 93, 90]. Further, PCA has been a widely-used tool in genomics and statistical genetics, employed to infer cryptic population structure from genome-wide data [1, 2, 35]. However, these methods lack the quality guarantees our methods provide and are not as scalable to modern large-scale, potentially decentralised/federated, datasets. In turn, such problems could be directly solved by employing our methods faster, more effectively, and at scale. To this end, this thesis provides a number of advances to the state-of-the-art with several theoretical contributions which are unified in a practical mathematical framework that makes PCA applicable to federated datasets while offering attractive properties. Additionally, we exploit a number of the ideas presented and introduce a novel federated task scheduler that is able to operate and schedule tasks on large-scale data-centre topologies allowing for increased resource utilisation. In the following two sections, firstly in Section 1.1 we present the aims of this dissertation and, secondly, in Section 1.2 we outline its contributions paired with a chapter outline.

1.1 Thesis and its substantiation

As previously discussed, the recent proliferation of dataset size along with data source heterogeneity introduced unique and intricate challenges in their analysis. This offered unique opportunities for researchers to tackle and unlock widespread dissemination and processing of massive datasets. In this context, this dissertation seeks to unlock the ability of performing PCA on federated datasets while offering variety of attractive properties, namely: incremental computation, requiring limited resources, differential privacy guarantees, as well as introducing a novel adaptive intrinsic rank estimation method.

Consequently, the **goal of this dissertation** is to *to provide a unifying mathematical framework for processing and performing dimensionality reduction upon federated datasets of arbitrary dimensionality while using limited resources, offering differential privacy, and being able to adaptively estimate their intrinsic dimension.*

This statement is substantiated with three main threads of research. The first, is the development of the mathematical primitives in order to perform dimensionality reduction using PCA with the desired properties at the edge. Concretely, we introduce the algorithmic frameworks required to enable incremental local model updates at the edge while using limited memory, guaranteeing differential privacy, and allow adaptive intrinsic dimension estimation. Further, in order to scale these innovations to be applicable in the context of federated datasets we provide a novel merging algorithm along with the required properties to ensure the result remains (ε, δ)-differentially private. To do so, the merging algorithm preserves the embeddings and is able to hierarchically propagate the intermediate results while also preserving the guarantees provided at the edge. This means that the aggregated results remain differentially private and rank adjusting. The final thread of research serves as a practical application that exploits parts of contributions presented herein. Elaborating, we put forward a novel data-centre scale task scheduler that is able to allocate incoming jobs when used in both traditional and, more importantly, federated data-centres of the future.

1.2 Chapters and contributions

This dissertation introduces several theoretical contributions that are presented as parts of a unified mathematical framework for performing dimensionality reduction in federated datasets. We begin by introducing Chapter 2 which provides the necessary background and preliminaries required to follow the rest of the dissertation material. The contributions of this dissertation, in the form of technical chapters, are outlined below:

- In Chapter 3, we built upon traditional Singular Value Decomposition (SVD) methods and propose a novel Memory-limited Online Subspace Estimation Scheme for streaming dimensionality reduction. The key intuition that led to the materialisation of this work was that we could generalise the traditional incremental SVD to update its current subspace estimate with every incoming *thin* block of data, rather than with every incoming vector. This difference between incremental SVD and our method is what enables us to complement our method with a comprehensive statistical analysis that was previously not available for incremental SVD methods. We consider the important case where the incoming data vectors are drawn from a zero-mean normal distribution. This stochastic setup is a powerful generalisation of the popular *spiked covariance* model, common in statistical signal processing [99]. Further, we prove that our scheme nearly matches the performance of "offline" truncated SVD, which assumes unlimited memory and computing resources, provided that the corresponding covariance matrix is well-conditioned and has a small residual. Unlike prior art, such as in [13, 25], we are able to perform both subspace tracking as well as reduce the dimensionality of the incoming data *at the same time* using only a single pass. Moreover, we concretely interpret our algorithm as an approximate solver for the underlying non-convex PCA optimisation program. Through our empirical evaluation we also find that our method exhibits state-of-the-art performance in our numerical experiments with both synthetic and real-world datasets.

- In Chapter 4, we build on the foundations of the previous chapter and we introduce the first federated, asynchronous, and (ε, δ)-differentially private algorithm for PCA in the memory-limited setting. Our algorithm incrementally computes local model updates using a streaming procedure and adaptively estimates its r leading principal components when only $\mathcal{O}(dr)$ memory is available with d being the dimensionality of the data. We guarantee differential privacy via an input-perturbation scheme in which the covariance matrix of a dataset $\mathbf{X} \in \mathbb{R}^{d\times n}$ is perturbed with a non-symmetric random Gaussian matrix with variance in $\mathcal{O}\left(\left(\frac{d}{n}\right)^2 \log d\right)$, thus improving upon the state-of-the-art [32, 33, 18]. Furthermore, contrary to previous federated or distributed algorithms for PCA and in the absence of perturbation masks, our algorithm is also invariant to permutations in the incoming data, which provides robustness against straggler or failed nodes. Through numerical simulations we show that, while using limited-memory, our algorithm exhibits performance that closely matches or outperforms traditional non-federated algorithms, and in the absence of communication latency, it exhibits attractive horizontal scalability.

- In Chapter 5 by using parts of the building blocks introduced in the previous chapters we present a federated, asynchronous, memory-limited algorithm for online task scheduling. Our proposed scheme is able to handle large scale job allocations across networks comprising with hundreds of workers. This is achieved through the ability to incrementally compute local model updates by exploiting Federated-PCA, previously presented in Chapter 4. This unlocks the execution of scheduling decisions within each node, independently, on whether to accept an incoming job based on the workload seen thus far. Further, the global "view" of the system can be aggregated, as needed, in order to produce a holistic perspective of the system which could reflect its overall responsiveness. Through our empirical evaluation which uses a large-scale real-world dataset of traces gathered from a production data-centre we show that, while using limited-memory, our algorithm exhibits state-of-the-art performance. Specifically, it is able to predict changes in the system responsiveness ahead of time based on industry standard metrics and, in turn, can lead to better scheduling decisions and overall utilisation of the available resources.

In the last chapter of this dissertation, namely Chapter 6, we provide a summary of the findings and identify directions for future work that could generalise or improve our federated PCA scheme.

Summarising, the contributions presented in this thesis introduce several innovations that help advance theoretical and applied computer science. Starting with the contributions of Chapter 3, MOSES to the best of our knowledge, is the first method that is able to guarantee quality on all SVD outputs, namely $\mathbf{U}$, Σ, and $\mathbf{V}^T$ while providing a *deterministic* bound on their quality. It is also able to offer significant performance improvements as it is able to update its estimates using block sizes in order of the target rank-r, rather than the ambient dimension compared to seminal prior art [130, 131]. We hope that this provides both practical benefits in applications, as well as, a starting point for researchers to built upon our constructions to further advance the field. Moreover, Chapter 4 tackled the issue of scaling out MOSES to be applicable to federated datasets in the form of Federated-PCA. This unlocks the ability to process massive decentralised datasets while also providing the same attractive quality guarantees. Notably, since Federated-PCA is built upon the foundations provided by MOSES, it is a significant advancement due to its ability to provide a deterministic result (in the absence of DP). This is contrary to popular SGD-based methods, as they suffer from several drawbacks, which our work seeks to address. In addition to the above, we bring large scale federated PCA computation with $(\varepsilon, \delta)-$DP guarantees, if the application requires it. To the best of our knowledge it is the first method to introduce federated computation of PCA with DP guarantees unlocking the potential to compute PCA in large-scale privacy constrained environments.

Notably, the contributions presented in the thesis are tied directly to improvements in performance of statistical inference and prediction tasks. Namely, since PCA is a tool used commonly in such tasks to extract meaningful embeddings. The contributions in this thesis help to perform these tasks better, faster, and at scale which unlocks numerous possibilities. This is key, as based on the quality of the resulting subspace containing the principal components along with its associated singular values determines how everything exploiting them performs. Practically speaking what our contributions achieve, is a way to extract these important components as close to their offline counterparts as possible, which are assumed to have infinite resources for their computation. Some potential applications of immediate benefit of these improvements, follow. Firstly, these can directly improve regression performance due to the quality guarantees of the produced subspace, especially if the input dataset is large and/or decentralised. Secondly, a by-product of a high quality resulting subspace and its associated singular values is the ability to produce the underlying covariance matrix easily. Note, that the ability to return the associated singular values is key, as normally they are not returned by similar prior art, making the covariance estimation and tasks that require the singular values much harder or impossible. Another commonly encountered use-case, is that provided with full rank data we are able to generate the projected data with ease or even better exploit the underlying subspace to perform data imputation.

Concluding, we note that the main code repositories pertaining to the contributions of this dissertation, namely MOSES[1] and Federated-PCA[2] are publicly accessible.

1.3 Publication list

During my studies I have been fortunate enough to be involved into many fruitful and interdisciplinary collaborations that have yielded 10 published papers & 6 pre-prints that span the areas of Machine Learning, Mobile Systems, Data Centres, Control Theory, Ubiquitous computing, Social Networks, and Sound Analysis. These works reflect my efforts and technical contributions made in the last few years; however, none of these would have been possible without the support of my collaborators. Concretely, related to the context of this dissertation, Chapter 3 is based on work performed in [76, 56], Chapter 4 is based on [75, 80, 79], and finally Chapter 5 is based on [74].

Papers & Pre-prints related to this dissertation

- [76] Online pattern discovery in distributed, high-dimensional, streaming data under the YOLO principle. Grammenos, Andreas and Mascolo, Cecilia and Crowcroft, Jon. EuroSys 2018 Doctoral Workshop, 2018.

[1] https://github.com/andylamp/moses

[2] https://github.com/andylamp/federated_pca

- [75] Efficient, privacy aware federated model sharing. Grammenos, Andreas and Mascolo, Cecilia and Crowcroft, Jon. First UK Mobile, Wearable and Ubiquitous Systems Research Symposium, 2018.
- [78] On Device Federated PCA & Subspace Tracking. Grammenos, Andreas and Mascolo, Cecilia and Crowcroft, Jon. Second UK Mobile, Wearable and Ubiquitous Systems Research Symposium, 2019.
- [56] MOSES: A streaming algorithm for linear dimensionality reduction. Eftekhari, Armin and Hauser, Raphael A and Grammenos, Andreas. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2019.
- [80] Federated PCA with Adaptive Rank Estimation. Grammenos, Andreas and Mendoza-Smith, Rodrigo and Mascolo, Cecilia and Crowcroft, Jon. arXiv preprint arXiv:1907.08059v1, 2019.
- [79] Federated Principal Component Analysis. Grammenos, Andreas and Mendoza-Smith, Rodrigo and Mascolo, Cecilia and Crowcroft, Jon. 34th Conference on Neural Information Processing Systems (NeurIPS), 2020.
- [74] Pronto: Federated Task Scheduling. Grammenos, Andreas and Kalyvianaki, Evangelia and Pietzuch, Peter. *Currently in review*, arXiv:2104.13429, 2021.

Other works during PhD study

- [77] You are sensing, but are you biased? A user unaided sensor calibration approach for mobile sensing. Grammenos, Andreas and Mascolo, Cecilia and Crowcroft, Jon. Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies, 2018.
- [81] Dissecting the Workload of a Major Adult Video Portal. Grammenos, Andreas and Raman, Aravindh and Böttger, Timm and Gilani, Zafar and Tyson, Gareth. International Conference on Passive and Active Network Measurement, 2020.
- [28] Exploring Automatic Diagnosis of COVID-19 from Crowdsourced Respiratory Sound Data. Brown, Chloë and Chauhan, Jagmohan and Grammenos, Andreas and Han, Jing and Hasthanasombat, Apinan and Spathis, Dimitris and Xia, Tong and Cicuta, Pietro and Mascolo, Cecilia. Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2020.
- [82] Exploring Automatic COVID-19 Diagnosis via voice and symptoms from Crowdsourced Data. Han, Jing and Brown, Chloë and Chauhan, Jagmohan and Grammenos,

Andreas and Hasthanasombat, Apinan and Spathis, Dimitris and Xia, Tong and Cicuta, Pietro and Mascolo, Cecilia. ICASSP, 2021.

- [60] Enabling In-Ear Magnetic Sensing: Automatic and User Transparent Magnetometer Calibration. Ferlini, Andrea and Montanari, Alessandro and Grammenos, Andreas and Harle, Robert and Mascolo, Cecilia. IEEE International Conference on Pervasive Computing and Communications (PerCom), 2021.
- [73] CPU Scheduling in Data Centers Using Asynchronous Finite-Time Distributed Coordination Mechanisms. Grammenos, Andreas and Charalambous, Themistoklis and Kalyvianaki, Evangelia. *Currently in review*, arXiv preprint arXiv:2101.06139, 2021.
- [98] An Asynchronous Approximate Distributed Alternating Direction Method of Multipliers in Digraphs. Wei, Jiang and Grammenos, Andreas and Charalambous, Themistoklis and Kalyvianaki, Evangelia. 60th IEEE Conference on Decision and Control, 2021.
- [149] Optimal CPU Scheduling in Data Centers via a Finite-Time Distributed Quantized Coordination Mechanism. Rikos I., Apostolos and Grammenos, Andreas and Charalambous, Themistoklis and Kalyvianaki, Evangelia, and Johansson H., Karl and Hadjicostis N., Christoforos. 60th IEEE Conference on Decision and Control, 2021.
- [83] Sounds of COVID-19: exploring realistic performance of audio-based digital testing. Han, Jing and Spathis, Dimitris and Xia, Tong and Bondareva, Erika and Brown, Chloë and Chauhan, Jagmohan and Grammenos, Andreas and Hasthanasombat, Apinan and Cicuta, Pietro and Mascolo, Cecilia. Nature npj Digital Medicine, 2021.
- [191] COVID-19 Sounds: A Large-Scale Audio Dataset for Digital COVID-19 Detection. Xia, Tong and Spathis, Dimitris and Brown, Chloë and Chauhan, Jagmohan and Grammenos, Andreas and Han, Jing and Hasthanasombat, Apinan and Bondareva, Erika and Ting Dang and Andres Floto and Cicuta, Pietro and Mascolo, Cecilia. 35th Conference on Neural Information Processing Systems (NeurIPS), 2021.
- [42] COVID-19 Disease Progression Prediction via AudioSignals: A Longitudinal Study. Dang, Ting and Han, Jing and Xia, Tong and Spathis, Dimitris and Bondareva, Erika and Brown, Chloë and Chauhan, Jagmohan and Grammenos, Andreas and Hasthanasombat, Apinan and Cicuta, Pietro and Mascolo, Cecilia. *Currently in review.*

Chapter 2

Background & Preliminaries

This chapter aims to provide the reader with the required background and preliminaries for the thesis. We set the context by providing an introduction to the linear algebra concepts and tools used throughout. Further, we provide brief introductions to Singular Value Decomposition (SVD), Principal Component Analysis (PCA). We also present the various differential privacy concepts used as our tools for the analysis presented in Chapter 4 as well as a description of federated computation which we also exploit. Note that, we use lowercase letters c for scalars, bold lowercase letters $\mathbf{v}$ for vectors, bold capitals $\mathbf{A}$ for matrices, $\mathbf{A}^T$ denotes the transpose, and calligraphic capitals $\mathcal{U}$ for subspaces.

2.1 Linear algebra preliminaries

We will start by revisiting basic linear algebra concepts which lead up to the importance of SVD and PCA. In most applications the collected data forms a matrix, let that matrix be $\mathbf{A} \in \mathbb{R}^{n \times d}$ where d are the features and n the number of samples contained in the matrix. Such an example would be a mote sensor providing values of temperature, humidity, and light over time. In this case, its columns would reflect the individual measurements - namely, temperature, humidity, and light; whereas, each row would represent an individual sample. Now, a fundamental task that is performed in data science and analytical tasks is to *fit* a model to the data. One of the most deceivingly simple, yet difficult family of fitting problems is the following:

$$\mathbf{A}\mathbf{x} = \mathbf{b} \tag{2.1}$$

Essentially, given $\mathbf{A} \in \mathbb{R}^{n \times d}$ try to find vector $\mathbf{x} \in \mathbb{R}^d$ such as $\mathbf{A}\mathbf{x}$ equals vector $\mathbf{b} \in \mathbb{R}^n$. Unfortunately, in many real-world cases this leads to *unsolvable* linear equations. Obviously, in such cases we cannot solve this as-is nor change its equations without changing the problem itself. However, for many such instances we still need a solution even if it is an approximate one. By relaxing the problem in eq. (2.1), we are able to reduce the problem to another widely studied case: *Least Squares* - which is part of the *minimisation* family of problems. The least squares method formulation of the problem is shown below:

$$\min \|\mathbf{b} - \mathbf{A}\widehat{\mathbf{x}}\|^2 \tag{2.2}$$

This formulation tries to choose an approximate solution $\widehat{\mathbf{x}}$ such that the quantity $\|\mathbf{b} - \mathbf{A}\widehat{\mathbf{x}}\|^2$ is as small as possible. Consequently, by minimising that quantity means that its derivatives are also zero, which are the *normal* equations $\mathbf{A}^T\mathbf{A}\widehat{\mathbf{x}} = \mathbf{A}^T\mathbf{b}$. Note, that while this formulation of least squares is most commonly used, it is also frequent in certain domains (e.g. [11, 130, 58]) that the features of $\mathbf{A}$ correspond to its rows and the number of samples to its columns. This leads to a slightly different least squares formulation, but in principle the underlying problem to solve remains the same. For clarity and to avoid confusion for the remainder of this section we will use the formulation of (2.2), where the columns of the matrix are its features and the rows the number of samples.

Another equally important example is the case where we continuously add rows to $\mathbf{A}$ as new measurements arrive. This leads to $\mathbf{A}$ gradually becoming highly triangular and eventually large enough, that we cannot store it in memory or perform analytical tasks using traditional methods. Thus, in order to alleviate this, we need to *decompose* matrix $\mathbf{A}$ into special matrices that are able to *reconstruct* $\mathbf{A}$ exactly or as close as possible, yet take much less space in memory or offer other useful properties. These family of problems is called *factorisation*, an example of which can be formulated as follows:

$$\mathbf{A} = \mathbf{C}\mathbf{R} \tag{2.3}$$

This problem requires to find two special matrices $\mathbf{C}$ and $\mathbf{R}$ such that $\mathbf{C}$ is made of *independent columns* of $\mathbf{A}$ and a particular matrix $\mathbf{R}$ with certain properties. The interesting part is that the rank of $\mathbf{C}$ has to be equal to the rank of $\mathbf{A}$ and if the rank is *much less* than the dimension of $\mathbf{A}$ then these matrices end up taking less space to store. Consequently, the more redundancy $\mathbf{A}$ has, the lower its rank is; hence, the more space savings we get. Note that this factorisation is rank-revealing, meaning that the rank of the matrix $\mathbf{A}$ is computed (*i.e.* "revealed") during the factorisation process. However, it is worth mentioning that when using this factorisation technique there is no explicit scoring of the columns and rows within matrices $\mathbf{C}$ and $\mathbf{R}$ respectively based on their contribution, which can be a significant limitation.

Matrix properties revisited

Let us now recap a few properties about matrices, as they will become invaluable as we go along. Recall our fitting problem $\mathbf{A}\mathbf{x} = \mathbf{b}$ in eq. (2.1); the fundamental observation here is that we can see $\mathbf{A}\mathbf{x}$ as a *linear combination* of the columns of $\mathbf{A}$. Expanding on this, let us illustrate it with an example - first, we use row-wise multiplication:

$$\mathbf{Ax} = \begin{bmatrix} 1 & 6 \\ 2 & 3 \\ 3 & 7 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} x_1 & + & 6x_2 \\ 2x_1 & + & 3x_2 \\ 3x_1 & + & 7x_2 \end{bmatrix} \tag{2.4}$$

This leads to a familiar form, requiring the computation of dot products. Now, let us perform the same operation but this time using column-wise multiplication instead:

$$\mathbf{Ax} = \begin{bmatrix} 1 & 6 \\ 2 & 3 \\ 3 & 7 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = x_1 \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} + x_2 \begin{bmatrix} 6 \\ 3 \\ 7 \end{bmatrix} \tag{2.5}$$

Naturally, both ways produce the exact same end result - however, we see that eq. (2.4) produces three dot products which is used to compute the elements of $\mathbf{b}$ but its notation is hard to understand and does not provide the intuitions we seek. On the other hand, column-wise multiplication presents us with a remarkable finding: that we can *express* $\mathbf{Ax}$ as a *linear combination* of columns from $\mathbf{A}$. Now, if we set the columns of $\mathbf{A}$ as $\mathbf{c}_1 = [1\ 2\ 3]^T$ and $\mathbf{c}_2 = [6\ 3\ 7]^T$ then we can see that,

$$\mathbf{Ax} = x_1\mathbf{c}_1 + x_2\mathbf{c}_2 \tag{2.6}$$

This formulation, not only forms a more succinct representation of the solution but also provides insights for its geometric implications. Intuitively, we can see that even if the columns of $\mathbf{A}$ are in $\mathbb{R}^3$ all combinations of vectors $\mathbf{c}_1$ and $\mathbf{c}_2$ which form the feasible solution set for our example lie on *plane* instead. This plane is formed by $\mathbf{c_1}$ and $\mathbf{c_2}$ vectors which also includes the origin $(0, 0, 0)$ materialising when x_1 and x_2 are equal to zero. Such spaces are called *vector spaces* and we define them below,

Definition. *A vector space is defined as the set V which contains r vectors of $\mathbb{R}^m$, with $r, m \geq 0$.*

In fact, particular vector space we previously mentioned has a very specific name, it is the **column space** of $\mathbf{A}$. Armed with these facts we can now provide our first definition,

Definition (Column space of a matrix)**.** *The column space of a matrix $\mathbf{A}$ is defined as the set of all possible linear combinations of its column vectors.*

Usually, we denote the column space of a matrix $\mathbf{A}$ as $\mathbf{col}(\mathbf{A})$ or depending on context just $\mathbf{C}$. This implies that all vectors $\mathbf{b}$ that satisfy eq. (2.1) have to be in the column space of $\mathbf{A}$, meaning they are within the feasible solution set.

Another key fact is that we mentioned previously that the $\mathbf{A} = \mathbf{CR}$ factorisation is rank-revealing, however we did not specify exactly what the rank of $\mathbf{A}$ is. To discuss the rank, we first need to address *independence*. Recall, that we said that matrix $\mathbf{C}$ would be comprised out of *independent* columns of **A**, meaning that we do not want to include columns that can be produced using a linear combination of existing ones in the set. This property allows us to create a *basis* and is defined as follows:

Definition (Subspace basis)**.** *The basis of a subspace is defined as a full set of independent vectors, which are called basis vectors.*

The statement above implies that *all* vectors in a space (e.g. matrix **A**) can be generated by using linear combinations of its basis vectors (e.g. matrix **C**). Let us illustrate this with an example, suppose that we have a matrix **A** along with its column space **C** defined as shown below:

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 6 \\ 2 & 4 & 3 \\ 3 & 6 & 7 \end{bmatrix}, \quad \mathbf{C} = \begin{bmatrix} 1 & 6 \\ 2 & 3 \\ 3 & 7 \end{bmatrix} \tag{2.7}$$

We see that the column space **C** only contains the first and third rows of **A**. This is because we can see that the second column of **A** can be expressed as a *linear combination* of the first row. As such, this column violates the basis definition provided; hence, cannot be a basis vector and is excluded from the set. The number of vectors in the column space tells us the *rank* of the matrix. Note that we started putting columns in **C** from left to right, we could have easily started this process from right to left. This would have resulted in a different basis for the column space of **A** but without changing the number of columns within the set. In other words, a basis is *not* unique but the amount of columns in the set always stays the same. Consequently, the *rank* of the matrix is equal to the dimension of the column space which is equal to the number of independent columns within the set.

Definition (Column space dimension)**.** *The dimension of the column space is defined to be the number of individual columns within the set.*

Given the column space dimension, we are now able to provide a definition for the matrix rank as follows:

Definition (Matrix rank)**.** *The rank of a matrix* **A** *is defined as the dimension of its column space.*

In some instances the rank of the matrix can also be known as its intrinsic dimension, in this dissertation these terms are used interchangeably. Notice, that from the rank definition it follows that the column space rank is equal to the matrix rank - this is very important. Referring back to eq. (2.3), we can immediately relate the column space to matrix **C** but what about matrix **R**? We can reconstruct **A** using linear combinations of vectors from its column space **C** but we need a map to indicate these combinations - that map is provided by matrix **R**. Let us illustrate that with an example using the previous matrices **A** and **C** from eq. (2.7),

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 6 \\ 2 & 4 & 3 \\ 3 & 6 & 7 \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & 6 \\ 2 & 3 \\ 3 & 7 \end{bmatrix}}_{\mathbf{C}} \underbrace{\begin{bmatrix} 1 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\mathbf{R}} = \mathbf{CR} \tag{2.8}$$

The shapes of the matrices are $\mathbf{A} \in \mathbb{R}^{m \times n}$, $\mathbf{C} \in \mathbb{R}^{m \times r}$, and $\mathbf{R} \in \mathbb{R}^{r \times n}$, where m the rows, n the columns, and r the rank. In fact, matrix $\mathbf{R}$ is instrumental in linear algebra and has a specific name, it is the row-reduced echelon form of $\mathbf{A}$.

So far, we have only been reasoning about the columns - what about the rows? Now, as we can see from eq. (2.8) we can get every row of $\mathbf{A}$ through the r rows contained within $\mathbf{R}$. This can be achieved by using $\mathbf{C}$ as the map of the linear combinations of the r rows contained in $\mathbf{R}$ to reconstruct $\mathbf{A}$. We can also see that these two rows are independent and thus form a basis. Meaning, that these rows form a basis for the row space of $\mathbf{A}$. Notably, both the column and row spaces have the same rank r which is equal to the matrix rank - each formed by r basis vectors. The column space is constructed using r basis column vectors whereas the row space is constructed using r basis row vectors. To illustrate the row independence let us revisit the $\mathbf{R}$ matrix from eq. (2.8),

$$\mathbf{R} = \begin{bmatrix} 1 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.9}$$

The rows that contain ones and zeros within $\mathbf{R}$ indicate that these are not a linear combination of each other. This is the case, because there no linear combination that can produce $\mathbf{r}_2 = [0\ 0\ 1]$ from $\mathbf{r}_1 = [1\ 2\ 0]$ and vice versa. We can easily validate this by computing their dot product which is zero. Having a zero dot product implies these vectors are perpendicular to each other and thus no such linear combination exists - thus, are independent.

Matrix as a sum of rank-one matrices

Let us now focus on how we can reconstruct $\mathbf{A}$ using rank-one matrices, as this is one of the fundamental building blocks used throughout. First, suppose that we have decomposed $\mathbf{A} \in \mathbb{R}^{m \times n}$ into two matrices, let these matrices be $\mathbf{C} \in \mathbb{R}^{m \times p}$ and $\mathbf{R} \in \mathbb{R}^{p \times n}$, where m the rows, n the columns, and p the rank. These matrices have the following shapes,

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}, \mathbf{C} = \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1p} \\ c_{21} & c_{22} & \cdots & c_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m1} & c_{m2} & \cdots & c_{mp} \end{bmatrix}, \mathbf{R} = \begin{bmatrix} r_{11} & r_{12} & \cdots & r_{1n} \\ r_{21} & r_{22} & \cdots & r_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ r_{p1} & r_{p2} & \cdots & r_{pn} \end{bmatrix} \tag{2.10}$$

Traditionally, as we said in the previous section, we can multiply **C** and **R** using the dot products as such,

$$a_{ij} = c_{i1}r_{1j} + c_{i2}r_{2j} + \cdots + c_{ip}r_{pj} = \sum_{k=1}^{p} c_{ik}r_{kj} \tag{2.11}$$

However, we can get the same result by multiplying the columns of **C** times the rows of **R**. Let us illustrate this with an example - suppose we have the following rank-one matrix $\mathbf{A} \in \mathbb{R}^{3\times3}$:

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 4 \\ 2 & 4 & 8 \\ 3 & 6 & 12 \end{bmatrix} \tag{2.12}$$

We can clearly see all columns are exact multiples of the first one. Thus the column space of **A** is the a single vector $\mathbf{C} = [1\ 2\ 3]^T$ and $\mathbf{R} = [1\ 2\ 4]$. Having these components, then **A** can be decomposed as:

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 4 \\ 2 & 4 & 8 \\ 3 & 6 & 12 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \begin{bmatrix} 1 & 2 & 4 \end{bmatrix} = \mathbf{CR} \tag{2.13}$$

This operation is called the outer product and produces a matrix; but what happens if the rank of **A** is greater than one? Then matrix **A** is the summation of *all* rank-one matrices produced by taking the outer product for each column of **C** with its respective row of **R**. To illustrate this, let us reformulate the $\mathbf{A} = \mathbf{CR}$ from eq. (2.10) as the sum of multiple rank-one matrices as follows,

$$\mathbf{A} = \underbrace{\begin{bmatrix} | & & | \\ c_1 & \cdots & c_p \\ | & & | \end{bmatrix}}_{\mathbf{C}} \underbrace{\begin{bmatrix} \text{———} & r_1 & \text{———} \\ & \vdots & \\ \text{———} & r_p & \text{———} \end{bmatrix}}_{\mathbf{R}} = c_1r_1 + \cdots + c_pr_p = \sum_{k=1}^{p} c_kr_k \tag{2.14}$$

Notably, both the end result and the computational complexity are the same for both methods. However, there is a very important difference and that is its *representation*. To elaborate, if we are using dot products to compute the multiplication then each result computed represents its *final* value. On the other hand, when we are using the summation of rank-one matrices each intermediate result is *additive*. These matrices can be seen as a building blocks for **A**. Each containing a portion of **A**, that when added, can reconstruct the final matrix.

Taking this one step further, let us reformulate eq. (2.13) to use vector notation instead since we know it is a rank-one matrix as follows,

$$\mathbf{A} = \underbrace{\begin{bmatrix}1\\2\\3\end{bmatrix}}_{\mathbf{u}} \underbrace{\begin{bmatrix}1 & 2 & 4\end{bmatrix}}_{\mathbf{v}^T} = \begin{bmatrix}1 & 2 & 4\\2 & 4 & 8\\3 & 6 & 12\end{bmatrix} \tag{2.15}$$

Now we know that $\mathbf{A} = \mathbf{u}\mathbf{v}^T$, however we can evidently see that each *column* of $\mathbf{A}$ is a multiple of $\mathbf{u}$. Similarly, we can also notice that each *row* of $\mathbf{A}$ is a multiple of $\mathbf{v}^T$. Interestingly, the column space of $\mathbf{A}$, and by extension $\mathbf{u}\mathbf{v}^T$ as in eq. (2.15), has only one column and thus its rank is one. To explain this geometrically, if the column space of a matrix is one dimensional then it is a line. In our case that line is the one that goes through vector $\mathbf{u}$. However, let us now discuss $\mathbf{A}^T$ and how its column space looks like. Using the same notation as in eq. (2.15) we see that $\mathbf{A}^T$ can be decomposed as follows,

$$\mathbf{A}^T = \underbrace{\begin{bmatrix}1\\2\\4\end{bmatrix}}_{\mathbf{v}} \underbrace{\begin{bmatrix}1 & 2 & 3\end{bmatrix}}_{\mathbf{u}^T} = \begin{bmatrix}1 & 2 & 3\\2 & 4 & 6\\4 & 6 & 12\end{bmatrix} \tag{2.16}$$

Notice, that the column space of $\mathbf{A}^T$ is the row space of $\mathbf{A}$ transposed and it lies on the line that goes through the vector $\mathbf{v}$. Generalising, we can state that *all* rank-one matrices have a column space that passes through a line, more formally,

Definition (Rank-one matrix from two vectors)**.** *Given two non-zero column vectors $\mathbf{u} \in \mathbb{R}^m$ and $\mathbf{v} \in \mathbb{R}^n$, then their outer product $\mathbf{u}\mathbf{v}^T$ forms a rank-one matrix that lies in $\mathbb{R}^{m\times n}$.*

Further given $\mathbf{A}$, then the column space of $\mathbf{A}^T$ is the transpose of the row space of $\mathbf{A}$. In fact, there is a deeper connection here as famously introduced by Strang [164, 166] which is known as the four fundamental subspaces of a matrix. Specifically, provided a matrix $\mathbf{A}$ that has m rows and n columns then it yields into four fundamental spaces that two of them lie in $\mathbb{R}^m$ and two in $\mathbb{R}^n$. So far we have seen and defined the column space of a matrix $\mathbf{A}$, denoted as $\mathbf{col}(\mathbf{A})$ which a set of vectors that lies $\mathbb{R}^n$ and can express as a linear combination all of the columns of $\mathbf{A}$. We shall now define properly these four spaces as follows,

Definition (Fundamental subspaces of a matrix [164–166])**.** *Suppose we have a matrix $\mathbf{A}$ with m rows and n columns then,*

- *Column space of $\mathbf{A}$ and denoted as $\mathbf{col}(\mathbf{A})$, is defined as the subspace of $\mathbb{R}^m$ that contains the set of all possible linear combinations of its column vectors.*
- *Row space of $\mathbf{A}$ and denoted as $\mathbf{row}(\mathbf{A})$, is defined as the subspace of $\mathbb{R}^n$ that contains the set of all possible linear combinations of its row vectors.*
- *(Right) Nullspace of $\mathbf{A}$ and denoted as $\mathbf{null}(\mathbf{A})$, is defined as the subspace of $\mathbb{R}^m$ which contains all the $\mathbf{x}$'s in $\mathbb{R}^n$ that satisfy $\mathbf{A}\mathbf{x} = 0$.*

- *Left Nullspace of* $\mathbf{A}$ *and denoted as* $\mathbf{null}(\mathbf{A}^T)$*, is defined as the subspace of* $\mathbb{R}^n$ *which contains all the* $\mathbf{y}$*'s in* $\mathbb{R}^m$ *that satisfy* $\mathbf{A}^T\mathbf{y} = 0$*.*

Interestingly, there is a beautiful symmetry in the above statements. The columnspace of $\mathbf{A}$ is the rowspace of its transpose; equally, the rowspace of $\mathbf{A}^T$ is the columnspace of $\mathbf{A}$. The same property holds for the left and right nullspaces of $\mathbf{A}$ and $\mathbf{A}^T$. This is a very powerful property as it provides a map from its columns that lie in $\mathbb{R}^m$ to its rows that lie in $\mathbb{R}^n$ and vice-versa for any matrix $\mathbf{A}$. The rank of the matrix dictates the dimension of these subspaces, which in turn sets the dimension of their respective nullspace. That is notable as it shows is when the solution to our linear problem, namely $\mathbf{Ax}$ or $\mathbf{Ay}$, is zero.

Common matrix types

Before we continue it is worth introducing commonly used types of matrices and their properties. They become our second nature as we discuss various, more advanced, decompositions and their properties. Let us start by discussing orthogonality in subspaces - in fact orthogonality is one of the most commonly used terms in linear algebra and it means *perpendicular*. The simplest form of that is the dot product between two vectors, where if zero means that these vectors are perpendicular to each other - in other words, orthogonal. There is also a stricter form of orthogonality, which to enforce that their length is equal to one - this is denoted as orthonormal. These properties can be extrapolated for subspaces as well as matrices. We start by providing the definition of an orthogonal basis,

Definition (Orthogonal subspace basis)**.** *An orthogonal basis for a subspace* $\mathcal{U}$ *means that every pair of basis vectors within the set, let these vectors be* $\mathbf{u}_i \in \mathbb{R}^m$ *and* $\mathbf{u}_j \in \mathbb{R}^m$ *then it satisfies* $\mathbf{u}_i^T\mathbf{u}_j = 0$ *when* $\mathbf{i} \neq \mathbf{j}$ *else* $\|\mathbf{u}_i\|$*.*

As we previously mentioned, the main difference between orthogonality and orthonormality is the added constraint with respect to the length of each basis vector; thus the definition for an orthonormal subspace basis is as follows,

Definition (Orthonormal subspace basis)**.** *An orthonormal basis for a subspace* $\mathcal{U}$ *means that basis vectors within the set, let these vectors be* $\mathbf{u}_i \in \mathbb{R}^m$ *and* $\mathbf{u}_j \in \mathbb{R}^m$ *then it satisfies* $\mathbf{u}_i^T\mathbf{u}_j = 0$ *when* $\mathbf{i} \neq \mathbf{j}$ *else* $\|\mathbf{u}\|$ *and have a length of* $\|\mathbf{u}_i\| = 1$*.*

Converting an orthogonal subspace basis to be orthonormal is easy and requires us to divide each basis vector $\mathbf{u}_i$ by its length $\|\mathbf{u}_i\|$. Now provided two subspaces that are orthogonal, let these be $\mathcal{U}$ and $\mathcal{V}$ then for this to hold each vector within $\mathcal{U}$ must be perpendicular to each vector within $\mathcal{V}$. Formalising the previous statement we can define this as follows,

Definition (Orthogonal subspaces)**.** *Let* $\mathcal{U}$ *and* $\mathcal{V}$ *be two subspaces, these are orthogonal if and only if each vector contained in* $\mathcal{U}$ *is orthogonal to each of the vectors contained in* $\mathcal{V}$ *and vice-versa.*

So far we talked about subspaces which are a special kind of matrices, what about regular matrices? Things are a little bit different when dealing with generic matrices rather than just subspaces and thus warrants a bit more explanation. In general, matrices are *not* orthogonal if not square - meaning that the number of rows equal that of the columns. However, if they are *tall and thin* while having orthonormal *columns* then it can be proved that $\mathbf{A}^T\mathbf{A} = \mathbf{I}$. If so, then for any vector $\mathbf{v}$ if multiplied by $\mathbf{A}$ then its length will not change. Concretely this means the following,

$$\|\mathbf{A}\mathbf{v}\| = \|\mathbf{v}\| \tag{2.17}$$

This follows because,

$$(\mathbf{A}\mathbf{v})^T(\mathbf{A}\mathbf{v}) = \mathbf{v}^T \underbrace{\mathbf{A}^T\mathbf{A}}_{\mathbf{I}} \mathbf{v} = \mathbf{v}^T\mathbf{I}\mathbf{v} = \mathbf{v}^T\mathbf{v} = \|\mathbf{v}\| \tag{2.18}$$

However, tall and thin matrices *do not* satisfy $\mathbf{A}\mathbf{A}^T = \mathbf{I}$. This is because if rows are greater than the columns they cannot be orthogonal.

On the other hand, if the matrix is square then it can be orthogonal, meaning that it can satisfy both $\mathbf{A}^T\mathbf{A} = \mathbf{I}$ and $\mathbf{A}\mathbf{A}^T = \mathbf{I}$. Provided this, it follows that $\mathbf{A}^T = \mathbf{A}^{-1}$, which indicates that the transpose is equal to the inverse of the matrix. All of the previous statements regarding orthogonal matrices can be formally packed into the following definition,

Definition (Orthogonal matrix)**.** *An orthogonal matrix* $\mathbf{A}$ *is a square matrix with real values having both orthonormal columns and rows. It satisfies both* $\mathbf{A}\mathbf{A}^T = \mathbf{A}^T\mathbf{A} = \mathbf{I}$ *and* $\mathbf{A}^T = \mathbf{A}^{-1}$*, where* $\mathbf{A}^T$ *is its transpose,* $\mathbf{A}^{-1}$ *its inverse, and* $\mathbf{I}$ *the identity.*

Moving on, let us now introduce symmetric matrices. This family of matrices satisfies a single, yet crucial property; that is $\mathbf{A} = \mathbf{A}^T$.

Definition (Symmetric matrix)**.** *A matrix is symmetrical, if and only if it is equal to its transpose; meaning provided matrix* $\mathbf{A}$ *then it holds that* $\mathbf{A} = \mathbf{A}^T$*.*

One handy property that we must prove is that given *any* matrix $\mathbf{A}$, then $\mathbf{A}^T\mathbf{A}$ is *always* symmetric. Now let $\mathbf{B} = \mathbf{A}^T\mathbf{A}$ thus we have,

$$\mathbf{B}^T = (\mathbf{A}^T\mathbf{A})^T = \mathbf{A}^T(\mathbf{A}^T)^T = \mathbf{A}^T\mathbf{A} = \mathbf{B} \tag{2.19}$$

The above holds by using the actual definition of symmetric matrices and by the fact that $(\mathbf{A}^T)^T = \mathbf{A}$, however it is not immediately evident and is immensely useful. Relating to the above, it has to be noted that for any matrix $\mathbf{A}$ the symmetric matrix $\mathbf{B} = \mathbf{A}^T\mathbf{A}$ has the same rank, meaning that $\text{rank}(\mathbf{A}) = \text{rank}(\mathbf{B})$. To prove that, it is sufficient to show that both programs $\mathbf{A}\mathbf{x} = 0$ and $\mathbf{A}^T\mathbf{A}\mathbf{x} = 0$ share the same solution $\mathbf{x}$. Suppose that there exists an $\mathbf{x}$ in the nullspace of $\mathbf{A}$, that is $\mathbf{x} \in \mathbf{null}(\mathbf{A})$ such that $\mathbf{A}\mathbf{x} = 0$, then it follows that,

$$\mathbf{A}\mathbf{x} = 0 \Rightarrow \tag{2.20}$$

$$\mathbf{A}^T(\mathbf{A}\mathbf{x}) = \mathbf{A}^T 0 \Rightarrow \tag{2.21}$$

$$\mathbf{A}^T(\mathbf{A}\mathbf{x}) = 0, \quad \text{since } \mathbf{x} \in \mathbf{null}(\mathbf{A}) \tag{2.22}$$

The last part is crucial, as it holds since $\mathbf{x}$ lies in the nullspace of $\mathbf{A}$ and thus the left side of the equation is also zero. It allows us also to claim that the nullspace of $\mathbf{A}$ is a subset or equal to the nullspace of $\mathbf{A}^T\mathbf{A}$, or more formally,

$$\mathbf{null}(\mathbf{A}) \subseteq \mathbf{null}(\mathbf{A}^T\mathbf{A}) \tag{2.23}$$

To complete the proof, we must now prove that $\mathbf{A}^T\mathbf{A}\mathbf{x} = 0$ holds as well. Our proof starts by making the same assumption as previously, namely that $\mathbf{x}$ lies in the nullspace of $\mathbf{A}$, hence we have,

$$\mathbf{A}^T\mathbf{A}\mathbf{x} = 0 \Rightarrow \tag{2.24}$$

$$\mathbf{x}^T\mathbf{A}^T\mathbf{A}\mathbf{x} = \mathbf{x}^T 0 \Rightarrow \tag{2.25}$$

$$(\mathbf{x}^T\mathbf{A}^T)(\mathbf{A}\mathbf{x}) = 0 \tag{2.26}$$

The use of parentheses here is key, as it allows us to perform a crucial transformation. We start by picking up from Equation (2.26),

$$(\mathbf{x}^T\mathbf{A}^T)(\mathbf{A}\mathbf{x}) = (\mathbf{A}\mathbf{x})^T\mathbf{A}\mathbf{x} = \underbrace{(\mathbf{A}\mathbf{x})^T}_{\mathbf{v}^T}\underbrace{\mathbf{A}\mathbf{x}}_{\mathbf{v}} = 0 \tag{2.27}$$

The result holds as the dot product of the same vectors is always equal to zero, thus by letting $\mathbf{v} = \mathbf{A}\mathbf{x}$ we have $\mathbf{v}^T\mathbf{v} = 0$; hence arrive at the desired solution. Note, this allows us to claim that the nullspace of $\mathbf{A}^T\mathbf{A}$ is a subset or equal to the nullspace of $\mathbf{A}$, or more concretely,

$$\mathbf{null}(\mathbf{A}^T\mathbf{A}) \subseteq \mathbf{null}(\mathbf{A}) \tag{2.28}$$

More importantly, in light of equations 2.23 and 2.28 we can deduce that the nullspaces of $\mathbf{A}$ and $\mathbf{A}^T\mathbf{A}$ are in fact equal. Thus we can claim the following,

$$\mathbf{null}(\mathbf{A}^T\mathbf{A}) = \mathbf{null}(\mathbf{A}), \text{ see eq. 2.23 and 2.28} \tag{2.29}$$

Moreover, we know that the dimension of a matrix is the summation of its rank r and the dimensionality of its nullspace. Provided their nullspaces are equal as we just showed and that the dimension of $\mathbf{A} \in \mathbb{R}^{n \times d}$ and $\mathbf{A}^T\mathbf{A} \in \mathbb{R}^{n \times n}$ is n, we can deduce that their rank is equal and thus conclude our proof. Note that the same result holds for $\mathbf{A}\mathbf{A}^T$ due to the result of eq. (2.19), meaning that $\mathbf{A}\mathbf{A}^T = (\mathbf{A}\mathbf{A}^T)^T$. However, normally matrices $\mathbf{A}\mathbf{A}^T$ and $\mathbf{A}^T\mathbf{A}$ are different, hence $\mathbf{A}\mathbf{A}^T \neq \mathbf{A}^T\mathbf{A}$ unless $\mathbf{A}$ is a symmetric matrix.

Another commonly encountered family of matrices is diagonal ones. For a matrix to be diagonal it needs to satisfy the requirement that all of its entries that are *outside* of its diagonal are zero. Formally, we can define this family of matrices as follows,

Definition (Diagonal matrix)**.** *A matrix is diagonal, if and only if has non-zero entries across its diagonal and zeros elsewhere.*

To provide a concrete example, the identity matrix $\mathbf{I}$ is a special case of both a diagonal and a symmetric matrix as it satisfies the requirements for both families. Interestingly, looking a bit more carefully, we can also see that $\mathbf{I}$ is also *orthogonal* - so it is indeed a very special matrix.

Matrix factorisation

At the heart of this dissertation lies matrix factorisation and its implications. Before coming full circle and discuss Singular Value Decomposition, we need to address why factorisation of matrices is indeed important. In the previous section we introduced how rank-one matrices can be added in order to produce the whole matrix. These are essentially outer products of vectors and can be seen as the building blocks for matrix composition - this is key. Factorisation can be thought as the reverse of multiplication and is more computationally intensive. More importantly, factorisation can reveal information about the matrix that otherwise is not immediately visible. For example, when factoring a matrix we can discover its rank. This in turns tells us how much *redundant* information is contained within that matrix - how can we exploit that? One of the obvious examples is compression. If the rank is low then the matrix as we saw before can be decomposed into factors that can take much less space to store. Concretely, a rank one matrix requires exactly two vectors to be stored in order to be expressed in full. Meaning if its dimensions were $m \times n$ then it would require $m + n$ entries to be stored instead of mn entries. In other instances these factors can reveal patterns that are not readily available when looking at the entirety of the matrix. The most commonly encountered factorisation techniques, along with a brief description for each are outlined below,

- $\mathbf{A} = \mathbf{L}\mathbf{U}$ is is called the Lower-Upper factorisation which splits the matrix $\mathbf{A}$ into two triangular matrices. The first one $\mathbf{L}$ is lower triangular while the $\mathbf{U}$ is upper triangular. Common applications for this factorisation technique are elimination when solving linear systems and inverting a matrix [15].

- $\mathbf{A} = \mathbf{Q}\mathbf{R}$ is an orthogonalisation decomposition which results in an orthogonal matrix $\mathbf{Q}$ and an upper triangular matrix $\mathbf{R}$. It is closely related to the traditional Gram-Schmidt process but it is more expensive in terms of resources but does not suffer from numerical stability issues plaguing traditional Gram-Schmidt [70].

- $\mathbf{A} = \mathbf{CUR}$ this is a somewhat related factorisation to the toy one we used previously in the chapter eq. (2.8). It was introduced by Mahoney et al.[122] and more recently Boutsidis et al. [22] discussed its performance with respect to its optimality. The main difference when compared to our version of $\mathbf{A} = \mathbf{CR}$ is that now the matrix $\mathbf{R}$ is also comprised out of *actual* rows from the original matrix rather than just $\mathbf{C}$. Recall, that our $\mathbf{R}$ was in row-reduced echelon form - this requirement is now dropped. However, by doing so the map has to be stored elsewhere as it now does not have the correct form. This is where the "mixing" matrix $\mathbf{U}$ comes into play. In this decomposition the mapping matrix is defined to be the pseudo inverse of the intersection of matrices $\mathbf{C}$ and $\mathbf{R}$. While we lose the guarantee of optimality that other decomposition can offer, this has the advantage of being faster to compute and can be is easier to interpret. This is because the columns and rows come directly from $\mathbf{A}$ rather than being projected into an arbitrary space as is with other decompositions.

- $\mathbf{S} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^T$ which is able to decomposes only a *symmetric* matrix $\mathbf{A}$ into a basis $\mathbf{Q}$ and diagonal $\mathbf{\Lambda}$, it is commonly known as the spectral factorisation. In this case $\mathbf{Q}$ is composed out of the eigenvectors of $\mathbf{A}$ and $\mathbf{\Lambda}$ is a diagonal matrix holding the eigenvalues of $\mathbf{A}$ along its diagonal [101]. One of the major drawbacks of this decomposition is that it cannot be used if the provided matrix to factorise is not *symmetric*.

- $\mathbf{A} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^{-1}$ this diagonalisation of the matrix $\mathbf{A}$ but is more commonly known as the eigendecomposition [165]. By performing this decomposition we are able to factorise a square matrix $\mathbf{A}$ into $\mathbf{V}$, $\mathbf{\Lambda}$, and $\mathbf{V}^{-1}$. The matrix $\mathbf{V}$ is comprised out of the eigenvectors of $\mathbf{A}$ with $\mathbf{V}^{-1}$ being its inverse, whereas matrix $\mathbf{\Lambda}$ holds the eigenvalues of $\mathbf{A}$.

- $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$ is the celebrated Singular Value Decomposition (SVD) [54, 164]. It is able to factorise any matrix, regardless if it is square or not, into three special matrices. The $\mathbf{U}$ and $\mathbf{V}$ are comprised out of the left and right singular vectors respectively. Further, $\mathbf{\Sigma}$ is a diagonal matrix which holds the singular values of $\mathbf{A}$ in decreasing order across its diagonal.

Interestingly enough, so far we only discussed how we add *all* of these rank one matrices to reconstruct the full matrix. However, what if we could pick the ones that are able to capture *most* but not all of the information contained? Surely, in that case since we do not have all of the rank one matrices required we cannot possibly reconstruct $\mathbf{A}$ in full, but how close can we get without losing too much of $\mathbf{A}$? The answer to this question is one of the goals of dimensionality reduction, and in our humble opinion, the most important one.

2.2 Eigendecomposition

One last step to do before we delve into Singular Value Decomposition, is to discuss one of its spiritual relatives which is the *Eigendecomposition.* As we mentioned in the previous section, it is a factorisation method that is able given a square matrix $\mathbf{A}$ to decompose it into $\mathbf{A} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^{-1}$. It is also known in certain textbooks or references as matrix diagonalisation. However, this can be misleading since it actually produces *three* matrices only one of which, namely $\mathbf{\Lambda}$, is diagonal and contains the eigenvalues of $\mathbf{A}$. But first, let us introduce briefly what eigenvalues and eigenvectors are. To do so, we need to introduce a new program to solve,

$$\mathbf{A}\mathbf{v} = \lambda\mathbf{v} \tag{2.30}$$

Where $\mathbf{A}$ is our matrix, λ a scalar, and $\mathbf{v}$ a vector we want to find that can solve this equation. As we can see, this is again a *linear transformation* since on the RHS of eq. (2.30) we can see that the result is the summation columns of $\mathbf{A}$ multiplied by scalar values in each respective row of vector $\mathbf{v}$. On the LHS we see that the same holds as $\mathbf{v}$ is scaled by a scalar value λ. The $\mathbf{v}$ is an eigenvector of $\mathbf{A}$ which has a special property that if multiplied by $\mathbf{A}$ it does not change its direction. What can change, is its scale by a factor of λ. Which is the implication of applying that transformation to $\mathbf{v}$ and is scaled by the magnitude dictated by the eigenvalue λ. Generally speaking λ can be any value and if, for example, is negative then the direction is can be reversed by the means of scaling. We are now ready to provide a definition for the eigenvectors,

Definition (Eigenvector of a square matrix)**.** *An eigenvector of a square matrix* $\mathbf{A}$ *is a vector* $\mathbf{v}$ *in* $\mathbb{R}^m$*, where* n *the ambient dimension of* $\mathbf{A}$ *such that* $\mathbf{A}\mathbf{v} = \lambda\mathbf{v}$ *holds for some scalar* λ*.*

Adding to the definition of the eigenvector, we can now also provide one for the eigenvalues as follows,

Definition (Eigenvalue of a square matrix)**.** *An eigenvalue of a square matrix* $\mathbf{A}$ *with ambient dimension* n *is a scalar* λ*, such that there is a non-zero vector* $\mathbf{v}$ *in* $\mathbb{R}^n$ *providing* $\mathbf{A}\mathbf{v} = \lambda\mathbf{v}$ *with a non-trivial solution.*

Normally, most square matrices with ambient dimension n have also n different eigenvectors, let these be $\mathbf{v}_1, ...\mathbf{v}_n$ and an equal number of different eigenvalues $\lambda_1, ..., \lambda_n$. Furthermore, a matrix can have complex numbers as its eigenvalues, a good example of which is orthogonal matrices. Now, we can see how we can express every n-dimensional vector $\mathbf{u}$ as a linear combination of the eigenvectors; it follows that,

$$\mathbf{u} = \mathbf{c_1}\mathbf{v}_1 + ... + \mathbf{c_n}\mathbf{v}_n \tag{2.31}$$

Suppose that we multiply $\mathbf{u}$ with a square matrix $\mathbf{A}$ having the same ambient dimension n, then it follows that,

$$\mathbf{A}^k\mathbf{u} = \mathbf{c_1}\lambda_1^k\mathbf{v}_1 + ... + \mathbf{c_n}\lambda_n^k\mathbf{v}_n \tag{2.32}$$

Where k is the power that $\mathbf{A}$ is raised. This is because the direction of the $\mathbf{u}$ does not change, but is only scaled based on the eigenvalues of $\mathbf{A}$ as we previously discussed. Now, we can observe that as eigenvalues grow, so is the significance of that component. Let us now illustrate the diagnonalisation process of a dense square matrix $\mathbf{A}$ that has an ambient dimension n and an equal number of independent eigenvectors. Suppose, we have these eigenvectors $\mathbf{v}_i$ where $i \in [1, ..n]$ and they form an invertible matrix $\mathbf{V}$. Then by multiplying that matrix with $\mathbf{A}$ and using the original equality of eigendecomposition eq. (2.30) we get,

$$\mathbf{A}\underbrace{\begin{bmatrix} \\ \mathbf{v}_1 & \cdots & \mathbf{v}_n \\ \\ \end{bmatrix}}_{\mathbf{V}} = \begin{bmatrix} \\ \mathbf{A}\mathbf{v}_1 & \cdots & \mathbf{A}\mathbf{v}_n \\ \\ \end{bmatrix} = \begin{bmatrix} \\ \lambda_1\mathbf{v}_1 & \cdots & \lambda_1\mathbf{v}_n \\ \\ \end{bmatrix} \tag{2.33}$$

Thus, by using the result of eq. (2.33) we can reformulate as follows,

$$\begin{bmatrix} \\ \lambda_1\mathbf{v}_1 & \cdots & \lambda_1\mathbf{v}_n \\ \\ \end{bmatrix} = \underbrace{\begin{bmatrix} \\ \mathbf{v}_1 & \cdots & \mathbf{v}_n \\ \\ \end{bmatrix}}_{\mathbf{V}} \underbrace{\begin{bmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_n \end{bmatrix}}_{\mathbf{\Lambda}} \tag{2.34}$$

Thus we arrived at the conclusion that the following holds,

$$\mathbf{AV} = \mathbf{V\Lambda} \tag{2.35}$$

but because we said from the beginning that matrix $\mathbf{V}$ is indeed invertible we arrive at the final factorisation form,

$$\mathbf{A} = \mathbf{V\Lambda V}^{-1} \tag{2.36}$$

However, provided some further assumptions about the matrix $\mathbf{A}$ we can arrive at a stronger factorisation. Let us consider the case in which our matrix is symmetric, meaning that $\mathbf{A} = \mathbf{A}^T$. Now, it is well known that symmetric matrices have all of their eigenvalues are real and additionally their eigenvectors can indeed be orthogonal to each other [91]. In fact, this a special case of the eigendecomposition in case the provided matrix is symmetric and is called the spectral theorem or spectral decomposition. What it says is simple and follows from the definition of symmetric matrices,

$$\mathbf{A} = \mathbf{Q\Lambda Q}^T \tag{2.37}$$

This is because the matrix $\mathbf{Q}$ is orthonormal and thus it means that $\mathbf{Q}^T = \mathbf{Q}^{-1}$ as well as $\mathbf{Q}^T\mathbf{Q} = \mathbf{I}$. Out of these matrices, there is an even more special family of matrices that is a subset of symmetric matrices. As we previously discussed, symmetric matrices have *real* eigenvalues; this implies although they are non-conjugate they can be both positive *and* negative. These particular matrices are split into two categories, the positive definite and positive semidefinite; they both share the property that their eigenvalues are positive but on semidefinite case zero eigenvalues are allowed as well. Formally, the positive definite matrices are defined as,

Definition (Symmetric positive definite matrix)**.** *A matrix* $\mathbf{S}$ *is symmetric positive definite if the quantity* $\mathbf{v}^T\mathbf{S}\mathbf{v}$ *is strictly greater than zero for all non-zero vectors* $\mathbf{v}$ *in* $\mathbb{R}^n$ *where* n *the dimension of* $\mathbf{A}$*.*

whereas the positive semidefinite matrices can be defined as follows,

Definition (Symmetric positive semidefinite)**.** *A matrix* $\mathbf{S}$ *is symmetric semi-positive definite if the quantity* $\mathbf{v}^T\mathbf{S}\mathbf{v}$ *is greater or equal to zero for all vectors* $\mathbf{v}$ *in* $\mathbb{R}^n$ *where* n *the dimension of* $\mathbf{A}$*.*

The quantity $\mathbf{v}^T\mathbf{S}\mathbf{v}$ is also known as *energy* [70]. Connecting this with symmetric matrices and why the above definitions hold we can do the following,

$$\mathbf{S}\mathbf{v} = \lambda\mathbf{v} \Rightarrow \mathbf{v}^T\mathbf{S}\mathbf{v} = \mathbf{v}^T\lambda\mathbf{v} \Rightarrow \mathbf{v}^T\mathbf{S}\mathbf{v} = \lambda\mathbf{v}^T\mathbf{v} \geq 0, \quad \text{provided } \lambda \geq 0 \tag{2.38}$$

Then based on the value of the eigenvalues we can see that this dictates if the energy greater or equal to zero. Notably, one extremely important property is the following,

$$\mathbf{S} = \mathbf{A}^T\mathbf{A} \tag{2.39}$$

This claims that for any matrix $\mathbf{A}$ if we take $\mathbf{A}^T\mathbf{A}$ then it results into a symmetric matrix $\mathbf{S}$. Further, the matrix $\mathbf{S}$ is positive definite in the case that $\mathbf{A}$ has *independent* columns and positive semidefinite if it has *dependent* columns. Let us formalise this statement,

$$\mathbf{v}^T\mathbf{S}\mathbf{v} = \mathbf{v}^T\mathbf{A}^T\mathbf{A}\mathbf{v} = (\mathbf{A}\mathbf{v})^T(\mathbf{A}\mathbf{v}) = \|\mathbf{A}\mathbf{v}\|^2 \tag{2.40}$$

$$\mathbf{S} = \mathbf{A}^T\mathbf{A} = \begin{bmatrix} 1 & 2 \\ 1 & 2 \\ 2 & 4 \end{bmatrix} \begin{bmatrix} 1 & 1 & 2 \\ 1 & 2 & 4 \end{bmatrix} = \begin{bmatrix} 5 & 5 & 10 \\ 5 & 5 & 10 \\ 10 & 10 & 20 \end{bmatrix} \tag{2.41}$$

We can see that the columns of the above matrix are dependent as the first column is equal to the second while the third column is an exact multiple of the first. The eigenvalues for this matrix are $\lambda_1 = 0$, $\lambda_2 = 0$, and $\lambda_3 = 30$. Since it has *at least one* eigenvalue equal to zero by eq. (2.38) it has zero energy and thus is positive semidefinite. Concluding this section, even if eigendecomposition is helpful tool the strict requirement of only being applicable to square matrices ends up rendering the method impractical for many real world applications where matrices are not normally symmetric. We would like to have something that can be applied to *all* matrices, regardless of their shape.

2.3 Singular Value Decomposition

As we said in the previous section due to the limitations of eigendecomposition we would like to have a factorisation that is able to cater for all matrices rather than ones that are square or admit special structure. The Singular Value Decomposition, in my opinion, is the perfect method to fill this gap. To start however, we need to introduce a different program to solve,

$$\mathbf{A}\mathbf{v} = \sigma\mathbf{u} \tag{2.42}$$

Recall that in eq. (2.30) we had scalars, the λ's but the vectors in both sides of the equation were the same. Instead, in this program we have two *different* set of vectors, namely $\mathbf{v}$'s and $\mathbf{u}$'s which we will explain shortly. Intuitively, having two sets of vectors makes sense as by relaxing the square requirement of the matrix we break the symmetry. More specifically, for a given matrix $\mathbf{A}$ that has m rows and n columns then as per eq. (2.42), the $\mathbf{v}$'s and $\mathbf{u}$'s that satisfy it would have to reside in $\mathbb{R}^n$ and and $\mathbb{R}^d$ respectively. Realising this, we cannot possibly apply eigendecomposition because of the different spaces these vectors lie. Further, note that since we are not dealing with eigenvectors and eigenvalues, the notation changes; these are now called using singular vectors and singular values respectively. Concretely, referencing eq. (2.42) $\mathbf{u}$ contains the *left* singular vector whereas $\mathbf{v}$ contains the *right* singular vector, and σ the singular value. Of course, regardless of the decomposition, the intrinsic dimension of the matrix does not change. This fact, as we will see later on, can be used to our advantage. Formally, we claim for any matrix $\mathbf{A}$ with rank r, d rows, and n columns that,

$$\mathbf{A}\mathbf{v}_i = \sigma_i\mathbf{u}_i, \text{ for } i = [1, r] \tag{2.43}$$

As we will prove later, this decomposition has a nice property; that every combination of $\sigma_i\mathbf{u}_i$ with an i higher than the matrix rank r is equal to zero. This is because its singular value is zero and thus the whole product of $\sigma_i\mathbf{u}_i = 0$ for $i > r$, where r the matrix rank. Not only that, but the singular values are ordered in descending order based on their significance; meaning that $\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_n$. Further, since we said that the singular values higher than the rank r are zero we can claim that,

$$\sigma_1 \geq \sigma_1 \geq ... \geq \sigma_r > 0 \tag{2.44}$$

Thus the following holds,

$$\mathbf{A}\mathbf{v}_i = \sigma_i \mathbf{u}_i, \; \textit{for } i \in [1..r] \text{ and } 0 \text{ otherwise.} \tag{2.45}$$

Provided the above, we can formulate this program using matrices as follows,

$$\mathbf{A} \underbrace{\begin{bmatrix} | & & | \\ \mathbf{v}_1 & \cdots & \mathbf{v}_n \\ | & & | \end{bmatrix}}_{\mathbf{V}} = \underbrace{\begin{bmatrix} | & & | \\ \mathbf{u}_1 & \cdots & \mathbf{u}_d \\ | & & | \end{bmatrix}}_{\mathbf{U}} \underbrace{\left[\begin{array}{ccc|c} \sigma_1 & & & \\ & \ddots & & 0 \\ & & \sigma_r & \\ \hline & 0 & & 0 \end{array}\right]}_{\mathbf{\Sigma}} \tag{2.46}$$

One of the key properties of that the d left and n right singular vectors that comprise matrices $\mathbf{U}$ and $\mathbf{V}$ matrices respectively are *orthogonal*. However, it has to be noted that $\mathbf{U}$ is orthogonal in $\mathbb{R}^d$ whereas $\mathbf{V}$ is orthogonal in $\mathbb{R}^n$. This is because the as we say previously, eigendecomposition required matrices $\mathbf{U}$ and $\mathbf{V}$ to be the same and thus orthogonal in the same space. Breaking that requirement is essentially what enables SVD to be applicable to non-square matrices. Additionally, note that the singular values that have index greater than the matrix rank r within $\mathbf{\Sigma}$ are equal to zero. Now, we can pack the full SVD shown in eq. (2.46), as follows,

$$\mathbf{AV} = \mathbf{U\Sigma} \tag{2.47}$$

Note that eq. (2.47) is not yet a complete formula. In order to have a proper factorisation scheme we need the matrix $\mathbf{A}$ to be in the LHS of the equation on its own. At first glance, it is not immediately evident how we can bring $\mathbf{V}$ over to the other side. However, both $\mathbf{V}$ and $\mathbf{U}$ are special, they are *orthogonal*. This implies, by definition, that they are equal to both their transpose as well as their inverse. Thus, by multiplying both sides by $\mathbf{V}^{-1} = \mathbf{V}^T$ we can get,

$$\mathbf{A} = \mathbf{U\Sigma V}^T \tag{2.48}$$

The statement above holds under the assumption that $\mathbf{A} \in \mathbb{R}^{d\times n}$ which in turn indicates that $\mathbf{U} \in \mathbb{R}^{d\times d}$, $\mathbf{\Sigma} \in \mathbb{R}^{d\times n}$, and $\mathbf{V} \in \mathbb{R}^{n\times n}$. $\mathbf{\Sigma}$ only up to rank $r \leq d$ if rank $r < d$ then it has $d - r$ zeros padded in the diagonal after the r-th singular value. Let us now quickly prove eq. (2.48) in way that, while not optimal for computing the SVD in a real-world application, is able to provide us with an intuitive proof [165, 166] [1].

Recall, that our goal is to compute the left and right singular vectors of our provided matrix. To do so, we will use a trick we introduced from the previous section, symmetric matrices. One way to form a symmetric matrix is to multiply it with its transpose and that holds for *any* matrix $\mathbf{A}$. Let us now start with $\mathbf{A}^T\mathbf{A}$ first,

[1]Note that the SVD holds also for conjugate matrices and such a proof can be found in [170]; however, in this dissertation we only deal with real matrices.

$$\mathbf{A}^T\mathbf{A} = (\mathbf{U\Sigma V}^T)^T(\mathbf{U\Sigma V}^T) = (\mathbf{V\Sigma}^T\mathbf{U}^T)(\mathbf{U\Sigma V}^T) = \mathbf{V}\underbrace{\mathbf{\Sigma}^T\mathbf{\Sigma}}_{\mathbf{\Lambda}}\mathbf{V}^T \tag{2.49}$$

We are able to arrive at the desired result because $\mathbf{U}$ is orthogonal and thus by definition $\mathbf{U}^T\mathbf{U} = \mathbf{I}$. Now doing the same process for $\mathbf{AA}^T$ we have the following,

$$\mathbf{AA}^T = (\mathbf{U\Sigma V}^T)(\mathbf{U\Sigma V}^T)^T = (\mathbf{U\Sigma V}^T)(\mathbf{V\Sigma}^T\mathbf{U}^T) = \mathbf{U}\underbrace{\mathbf{\Sigma}^T\mathbf{\Sigma}}_{\mathbf{\Lambda}}\mathbf{U}^T \tag{2.50}$$

Similarly to eq. (2.49) we can see that $\mathbf{V}$ is orthogonal and thus again by definition $\mathbf{V}^T\mathbf{V} = \mathbf{I}$. Observing closely, these should look a bit familiar with eq. (2.37) for both eq. (2.50) and eq. (2.49). Knowing that, both $\mathbf{AA}^T$ and $\mathbf{A}^T\mathbf{A}$ are *symmetric* and thus can be factorised into $\mathbf{Q\Lambda Q}^T$. In this case, the $\mathbf{Q}$ for the decomposition is equal to $\mathbf{U}$ and $\mathbf{V}$ for eq. (2.50) and eq. (2.49) respectively. In other words, $\mathbf{V}$ holds the eigenvectors of $\mathbf{A}^T\mathbf{A}$ whereas $\mathbf{U}$ holds the eigenvectors of $\mathbf{AA}^T$ both of which are orthonormal. As far as the eigenvalues ($\mathbf{\Lambda}$) for both $\mathbf{A}^T\mathbf{A}$ and $\mathbf{AA}^T$ they have the *same* non-zero eigenvalues and are equal from $\lambda_1 = \sigma_1^2$ to $\lambda_r = \sigma_r^2$ where r the rank of the matrix. If the matrix rank d is less than its ambient dimension d, then additional $d - r$ eigenvalues equal to zero are padded to the diagonal as necessary.

Notably, the squared eigenvalues or the possibility of having equal non-zero ones poses a problem. They are also the reason why SVD requires two different sets of vectors to work properly. Recall from our definition that $\mathbf{Av}_i = \sigma_k\mathbf{u}_i$ and needs to hold for all i up to the matrix rank r. This equation can be seen as a map for each of the *right* singular vectors ($\mathbf{v}_i$) to the *left* singular vectors ($\mathbf{u}_i$) for $i \in [1, ..., r]$. In the case of a squared eigenvalue this it has the implication that given a symmetric matrix $\mathbf{S}$ when we have a positive eigenvalue ($\lambda > 0$) then $\mathbf{Su} = \lambda\mathbf{u}$ while in the case of a negative one ($\lambda < 0$) we have $\mathbf{S}(-\mathbf{u}) = \lambda(-\mathbf{u})$. On the other hand, if there are double eigenvalues then the solutions lie in a plane.

Let us start by picking a "right" set of orthonormal eigenvectors $\mathbf{v}_1, ...\mathbf{v}_r$ from $\mathbf{A}^T\mathbf{A}$ which, as we previously said form matrix $\mathbf{V}$. This means that based on the eigendecomposition of $\mathbf{A}^T\mathbf{A}$ we can do,

$$\mathbf{A}^T\mathbf{Av}_i = \lambda_i\mathbf{v}_i = \sigma_i^2\mathbf{v}_i, \text{ for } i = [1, ..., r] \tag{2.51}$$

Meaning that $\mathbf{v}_i$'s are eigenvectors of $\mathbf{A}^T\mathbf{A}$. In order for the SVD to hold, we need to pick certain vectors $\mathbf{u}_i$ for $i = [1, ..., r]$ such that $\mathbf{Au}_i = \sigma_i\mathbf{u}_i$ for $i = [1, ..., r]$. For ease of exposition, we can rewrite the set of desired vectors $\mathbf{u}_i$ as follows,

$$\mathbf{u}_i = \frac{\mathbf{Av}_i}{\sigma_i}, \text{ for } i = [1, ..., r] \tag{2.52}$$

Following that step, we need to show that the $\mathbf{u}_i$'s are eigenvectors of $\mathbf{A}\mathbf{A}^T$ as follows,

$$\mathbf{A}\mathbf{A}^T\mathbf{u}_i = \mathbf{A}\mathbf{A}^T\left(\frac{\mathbf{A}\mathbf{v}_i}{\sigma_i}\right) = \mathbf{A}\left(\frac{\overbrace{(\mathbf{A}^T\mathbf{A}\mathbf{v}_i)}^{\sigma_i^2\mathbf{v}_i}}{\sigma_i}\right) = \mathbf{A}\left(\frac{\sigma_i^2\mathbf{v}_i}{\sigma_i}\right) \tag{2.53}$$

From the equation above and in order to arrive at our desired form we need to exploit the result of eq. (2.52) as follows,

$$\mathbf{A}\left(\frac{\sigma_i^2\mathbf{v}_i}{\sigma_i}\right) = \sigma_i^2\left(\frac{\mathbf{A}\mathbf{v}_i}{\sigma_i}\right) = \sigma_i^2\mathbf{u}_i, \text{ for } i = [1, ..., r] \tag{2.54}$$

To complete the proof, we need to show that our selected vectors $\mathbf{u}$ are indeed orthonormal. All that is required, is to prove that the quantity $\mathbf{u}_i^T\mathbf{u}_j$ is equal to one when $i = j$ and zero otherwise. To do so, we start by expanding $\mathbf{u}_i^T\mathbf{u}_j$ as follows,

$$\mathbf{u}_j^T\mathbf{u}_i = \left(\frac{\mathbf{A}\mathbf{v}_j}{\sigma_j}\right)^T\left(\frac{\mathbf{A}\mathbf{v}_i}{\sigma_i}\right) = \frac{(\mathbf{v}_j^T\mathbf{A}^T)\mathbf{A}\mathbf{v}_i}{\sigma_j\sigma_i} = \frac{\mathbf{v}_j^T(\mathbf{A}^T\mathbf{A}\mathbf{v}_i)}{\sigma_j\sigma_i} = \frac{\mathbf{v}_j^T\sigma_i^2\mathbf{v}_i}{\sigma_j\sigma_i} = \frac{\sigma_i}{\sigma_j}\mathbf{v}_j^T\mathbf{v}_i \tag{2.55}$$

We are certain that the above statement holds as σ_i/σ_j can be thought as a non-zero scalar, while the $\mathbf{v}$'s where picked from the eigenvectors of $\mathbf{A}^T\mathbf{A}$ and thus are orthonormal by construction.

However, we are not yet finished. The reason being, what happens if the matrix rank r is *less* than either the columns or rows of the initial matrix $\mathbf{A}$? Thankfully, to solve this issue is trivial. This is because these vectors, by definition, have eigenvalues equal to zero hence they lie in the nullspace of either row-space or column-space. Concretely, assuming that our matrix $\mathbf{A}$ has d rows and n columns; then we need to pick $d - r$ vectors $\mathbf{u}_{r+1}, ..., \mathbf{u}_d$ from the column nullspace and $n - r$ vectors $\mathbf{v}_{r+1}, ..., \mathbf{v}_n$ from the row nullspace. Provided with this fact, we can pick any orthonormal basis for these nullspaces which will automatically be orthogonal to the $\mathbf{u}$'s and $\mathbf{v}$'s that are already in the respective sets. This concludes our proof and the shape of SVD can be seen from eq. (2.56) that follows,

$$\mathbf{A}\in\mathbb{R}^{d\times n} = \underbrace{\left[\begin{array}{cccccc} \multicolumn{3}{c}{\overbrace{\qquad\qquad}^{\textbf{col}(\mathbf{A})}} & \multicolumn{3}{c}{\overbrace{\qquad\qquad}^{\textbf{null}(\mathbf{A})}} \\ | & & | & | & & | \\ \mathbf{u}_1 & \cdots & \mathbf{u}_r & \mathbf{u}_{r+1} & \cdots & \mathbf{u}_d \\ | & & | & | & & | \end{array}\right]}_{\mathbf{U}\in\mathbb{R}^{d\times d}} \underbrace{\left[\begin{array}{ccc|c} \sigma_1 & & & \\ & \ddots & & 0 \\ & & \sigma_r & \\ \hline & 0 & & 0 \end{array}\right]}_{\mathbf{\Sigma}\in\mathbb{R}^{d\times n}} \underbrace{\left[\begin{array}{ccc} — & \mathbf{v}_1 & — \\ & \vdots & \\ — & \mathbf{v}_r & — \\ — & \mathbf{v}_{r+1} & — \\ & \vdots & \\ — & \mathbf{v}_n & — \end{array}\right]}_{\mathbf{V}^T\in\mathbb{R}^{n\times n}} \begin{array}{l} \left.\vphantom{\begin{array}{c}a\\b\\c\end{array}}\right\}\textbf{row}(\mathbf{A}) \\ \left.\vphantom{\begin{array}{c}a\\b\\c\end{array}}\right\}\textbf{null}(\mathbf{A}^T) \end{array} \tag{2.56}$$

Note that as $\mathbf{A}$ is broken down to its respective pieces through the SVD we can see where in each matrices the actual information and the nullspaces lie. Essentially what this shows is that *all* of the important information is help up to the matrix rank r. As we saw in the previous section, each matrix can be broken into a summation of rank-one matrices. This property also holds for the SVD and has the following form,

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T = \sum_{i=1}^{r} \mathbf{u}_i \sigma_i \mathbf{v}_i^T \tag{2.57}$$

More importantly, and perhaps one of its most significant properties, is that these rank-one matrices are *ordered.* They are ordered by their significance indicated by their singular value. In other words, they are ordered by the amount contributed to the total summation of eq. (2.57). However, these extra rows and columns of the nullspaces they require a significant amount of space to be stored thus making the decomposition factors taking more space than they actually require. As we saw from the eq. (2.57), anything above the matrix rank does not contribute anything; surely we can do better. This realisation was the key idea behind a "thin" or "reduced" representation of SVD which has the following form,

$$\mathbf{A} = \underbrace{\begin{bmatrix} | & & | \\ \mathbf{u}_1 & \cdots & \mathbf{u}_r \\ | & & | \end{bmatrix}}_{\mathbf{U}} \underbrace{\begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_r \end{bmatrix}}_{\mathbf{\Sigma}} \underbrace{\begin{bmatrix} — & \mathbf{v}_1 & — \\ & \vdots & \\ — & \mathbf{v}_r & — \end{bmatrix}}_{\mathbf{V}^T} \tag{2.58}$$

The representation shown above only stored the information that is indeed required and can contribute something to the reconstruction of $\mathbf{A}$ as per eq. (2.57). The savings can be *significant* especially as the gap between the rank r of the matrix and its other dimensions increases.

Optimality of SVD

Let us now briefly address the question of how good actually is SVD. In fact, SVD is so good that (so far) we *cannot* find a better representation of equal rank that the SVD can produce. Furthermore, when compared to other factorisation techniques we mentioned previously such as **QR**, **LU**, and **CUR** the SVD offers another extremely important property. That the produced rank-one matrices are *ordered by their significance*, and if summed, can provide the best rank-k approximation of $\mathbf{A}$ where k is from 1 up to r. Of course, if $k = r$ then we are able to reconstruct the matrix $\mathbf{A}$, in full. This celebrated theorem was proved by Eckart et al [54] and was also later independently rediscovered by Mirsky et al [129]. However, we need to state by which metric we measure the quality of the reconstruction. Normally, such errors are measured with respect to a norm. In this instance, we will formulate the theorem using the Frobenius norm as follows,

Theorem 2.3.1 (Best rank-k matrix approximation [54, 129])**.** *Given two matrices* $\mathbf{A}$ *and* $\mathbf{B}$ *if* $\mathbf{B}$ *has rank-k then* $\|\mathbf{A} - \mathbf{B}\|_F \geq \|\mathbf{A} - \mathbf{A}_k\|_F$.

It has to be noted, that there are proofs for this theorem that hold for Spectral (ℓ_2), Frobenious (ℓ_F), and Manhattan (ℓ_1) norms just to name a few [135, 165, 126].

Geometry of SVD

Before concluding our discussion about the Singular Value Decomposition, it is also worth exploring what this factorisation does geometrically and how. Intuitively, we can say that SVD factorises the matrix into three different matrices; two of which are orthogonal and one diagonal. Geometrically, an orthogonal matrix implies either a rotation or a reflection while diagonal matrices are always stretches. Suppose that we had a matrix $\mathbf{A}$ that was 2×2 and $\mathbf{x}$ that was comprised out of unit vectors, then we could reformulate the transformation $\mathbf{Ax}$ as follows,

$$\mathbf{Ax} = \mathbf{U\Sigma V}^T\mathbf{x}. \tag{2.59}$$

The broad picture can be seen from Figure 2.1 inspired by [164], in which we see how each of the factorisation components of matrix $\mathbf{A}$ affect $\mathbf{x}$ along with their reverse operations. Concretely, matrices $\mathbf{U}$ and $\mathbf{V}$ can be seen as rotations or reflections; in our case, and for ease of exposition we use rotation. Moreover, matrix $\mathbf{\Sigma}$ indicates a stretch by the amount dictated by the respective singular value. Note, that since the singular values in SVD are *always* greater or equal to zero we do not have a stretch that can reverse the directions of vectors.

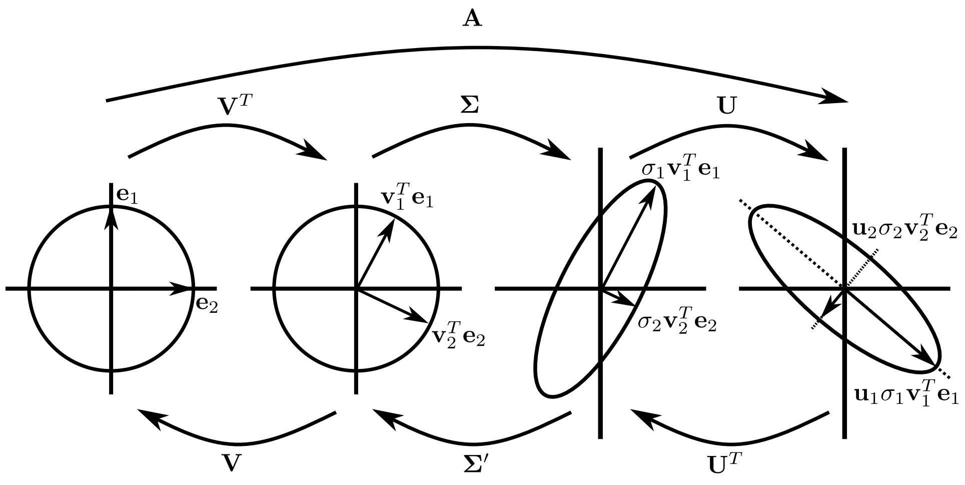

$\mathbf{A}$
$\mathbf{V}^T$
$\mathbf{\Sigma}$
$\mathbf{U}$
$\mathbf{e}_1$
$\mathbf{e}_2$
$\mathbf{v}_1^T\mathbf{e}_1$
$\mathbf{v}_2^T\mathbf{e}_2$
$\sigma_1\mathbf{v}_1^T\mathbf{e}_1$
$\sigma_2\mathbf{v}_2^T\mathbf{e}_2$
$\mathbf{u}_2\sigma_2\mathbf{v}_2^T\mathbf{e}_2$
$\mathbf{u}_1\sigma_1\mathbf{v}_1^T\mathbf{e}_1$
$\mathbf{V}$
$\mathbf{\Sigma}'$
$\mathbf{U}^T$

Fig. 2.1 Anatomy of Singular Value Decomposition for a 2×2 matrix $\mathbf{A}$ and $\mathbf{x}$, which contains the two unit vectors; namely $\mathbf{e}_1$ and $\mathbf{e}_2$ and is depicted in the far left circle within the figure. Then we show each of the transformation steps for $\mathbf{Ax}$ as each of the SVD pieces is applied to $\mathbf{x}$.

Firstly, $\mathbf{V}^T$ rotates $\mathbf{x}$ to the correct plane. Then $\mathbf{\Sigma}$ stretches each vector according to the magnitude of the respective singular value and by construction we have that $\sigma_1 \geq \sigma_2$. Finally, $\mathbf{U}$ performs another rotation to bring everything into the correct alignment. The bottom part of the figure show how these operations can be *reversed*. The orthogonal parts are straightforward, since we only need to use their transpose. On the other hand, to reverse the stretch we need to take the inverted singular values from the original $\mathbf{\Sigma}$ which forms a new matrix $\mathbf{\Sigma}'$. This matrix $\mathbf{\Sigma}'$ has across its diagonal $1/\sigma_i$ for i up to r where r the matrix rank, which in our example is equal to 2.

2.4 Principal Component Analysis

Principal Component Analysis (PCA) is arguably the oldest and most ubiquitous technique to perform multivariate analysis. It was introduced first by Pearson [143] and also rediscovered independently by [92]. Due to its complexity to calculate, like most multivariate analysis tools, only became popular with the advent of computers. This digital revolution was what enabled the practical application of said techniques to real datasets. Furthermore, it has now become the "de facto" tool of choice and most statistical packages and libraries come with at least one implementation of PCA. The pivotal idea behind PCA is to exploit the inherent data redundancy that most datasets have [200, 16]. It does so by transforming the dataset into a smaller representation that explains *most* of the variance present within. This transformation is *linear* and is achieved by taking its datapoints that are perhaps *interrelated* and converts them into a new set of variables, called the Principal Components (PC's), which are *uncorrelated*. Further, PC's are *ordered* by their significance - this is *significant*. Meaning that the PC that contributes the *maximal* variance is *always* the first while the others must contribute *less or equal* follow. Note that in most real-world datasets we expect the total variance to be concentrated in the first few components due to data-redundancy, as previously discussed.

In fact, this should all sound fairly familiar as it very much looks like something that the SVD, we previously introduced, would be used to perform - and it does. However, there is a catch: that for the SVD of a matrix to be equal its PCA then that matrix has to be centered. Concretely, the PC's of a matrix $\mathbf{A}$ are its singular vectors and are contained in the orthogonal matrices $\mathbf{U}$ and $\mathbf{V}$ produced by the SVD. In this context, PCA uses the singular values produced by the SVD and their corresponding vectors $\mathbf{u}$ and $\mathbf{v}$ in order to understand the underlying structure of the data contained within a matrix. Recall, that through the SVD we can decompose any matrix $\mathbf{A}$ into $\sum_{i=1}^{r} \mathbf{u}_i \sigma_i \mathbf{v}_i^T$ and through Theorem 2.3.1 we know that this is the best we can do. In statistics we are looking for a given matrix $\mathbf{A}$ its pieces that express most of its contained variance, which is exactly what SVD does.

In the context of machine learning literature, PCA can be classified as a form of unsupervised learning. Practically, it implies that the SVD does not use any auxiliary information such as labels or ground truth data to verify its result, but rather just applies the underlying transformation. This is conversely with supervised methods such as deep learning which try to model a non-linear function by using huge datasets as training data. One of the key benefits of PCA through SVD is that it is a "hands-off" approach requiring no tuning to provide its guarantees while also being is much more interpretable than other "black-box" based approaches [156].

Let us now examine the statistical properties of PCA in a bit more detail. We can compute PCA using a variety of techniques. However, one of the most prominent ones is by either forming a correlation or a covariance matrix and applying the SVD on it. The main difference between these two methods usually is that covariance matrix is employed when the variable scales are similar whereas correlation matrix is used when the variables are on different scales. In other words, using the correlation matrix is equivalent as performing standardisation of each of the studied variables. These formulations can produce different results, especially when the scales of the variables are different.

When performing PCA we are interested to explain the maximal variance from the dataset at hand. However, an arbitrary dataset has two degrees of freedom w.r.t. to where its datapoints reside, namely its *mean* and *variance*. The mean is defined as the average sum of each row of our matrix, whereas the variance is defined as the sums of the squared distances from the mean.

Definition (Empirical covariance)**.** *The empirical (or sample) covariance of a given matrix* $\mathbf{A}$ *of* d *rows and* n *columns is defined as the symmetric positive semidefinite matrix,*

$$\operatorname{cov}(\mathbf{A}) = \frac{(\mathbf{A}-\mu)(\mathbf{A}-\mu)^T}{n-1} \tag{2.60}$$

Now, if the data is centered that implies $\mu = 0$*, hence the above transforms to the following,*

$$\operatorname{cov}(\mathbf{A}) = \frac{\mathbf{A}\mathbf{A}^T}{n-1} \tag{2.61}$$

By construction $\operatorname{cov}(\mathbf{A})$ for matrix $\mathbf{A} \in \mathbb{R}^{d\times n}$ is a square and symmetric positive semidefinite matrix in $\mathbb{R}^{d\times d}$. The variances of $\mathbf{A}$ are located across the diagonal entries of $\operatorname{cov}(\mathbf{A})$ whereas the covariances are located in its off-diagonal entries. From the definition of the empirical covariance we note that if the data are not centered, then we would not be able to get the PC's that correspond to the true covariance matrix. Since SVD can be applied to *any* matrix, naturally it will applicable to non-centered data. However, the directions produced might differ when compared to the ones dictated by its application to the true covariance matrix. Moreover, apart from the significant resources required to construct the covariance matrix of $\mathbf{A}$ when dealing with large matrices, it can also result in numerical stability issues [112]. Fortunately, we can do better. Since the covariance matrix is a square and positive semidefinite matrix, then we can use the eigendecomposition to factorise it as follows,

$$\operatorname{cov}(\mathbf{A}) = \frac{\mathbf{A}\mathbf{A}^T}{n-1} = \frac{\mathbf{U}\mathbf{\Lambda}\mathbf{U}^T}{n-1} = \frac{\mathbf{U}\mathbf{\Lambda}^2\mathbf{U}^T}{n-1} \tag{2.62}$$

Recall, that from eq. (2.50), that the eigendecomposition of $\mathbf{A}\mathbf{A}^T$ contains the *left singular* vectors within $\mathbf{U}$. This is exactly what we want PCA to compute for us. If we apply SVD to the regular matrix $\mathbf{A}$ we can get both its left singular vectors as well as its singular values. Thus the problem is reduced to applying the SVD to the actual matrix $\mathbf{A}$ rather than using its covariance matrix. Which not only is more computationally efficient, but is also much more robust when dealing with large matrices thus reducing numerical stability issues.

Let us now examine a little closer the problem that traditional PCA tries to solve. Recall, from the beginning of this chapter the least squares fit problem, namely $\mathbf{A}\mathbf{x} = \mathbf{b}$ which tries to minimise the quantity $\|\mathbf{A}\mathbf{x} - \mathbf{b}\|^2$. Geometrically, it tries find the best line which minimises the distances from the given set of points. However, PCA is different; as it can be translated into applying the SVD to the covariance matrix. This problem is known as *orthogonal regression* or *perpendicular least squares*. What PCA essentially does, is rather than performing a fit it applies a linear transformation. Further, it guarantees that the line (or subspace) dictated by the first PC minimises the sum of squared distances between the points and that line (or subspace). To verify this, we can see this as it is a corollary of theorem 2.3.1. More concretely, let $\|\mathbf{A}\|_{2,1}$ be the sum of the Euclidean norms of the matrix columns and $\mathbf{a}_i$ the i-th column then we have,

$$\|\mathbf{A}\|_{2,1} = \sum_{i=1}^{n} \|\mathbf{a}_i\|^2 = \sum_{j=1}^{r} \left(\sum_{i=1}^{n} |\mathbf{a}_i^T \mathbf{u}_j|^2 \right) = \sum_{j=1}^{r} \mathbf{u}_j^T \mathbf{A}\mathbf{A}^T \mathbf{u}_j \tag{2.63}$$

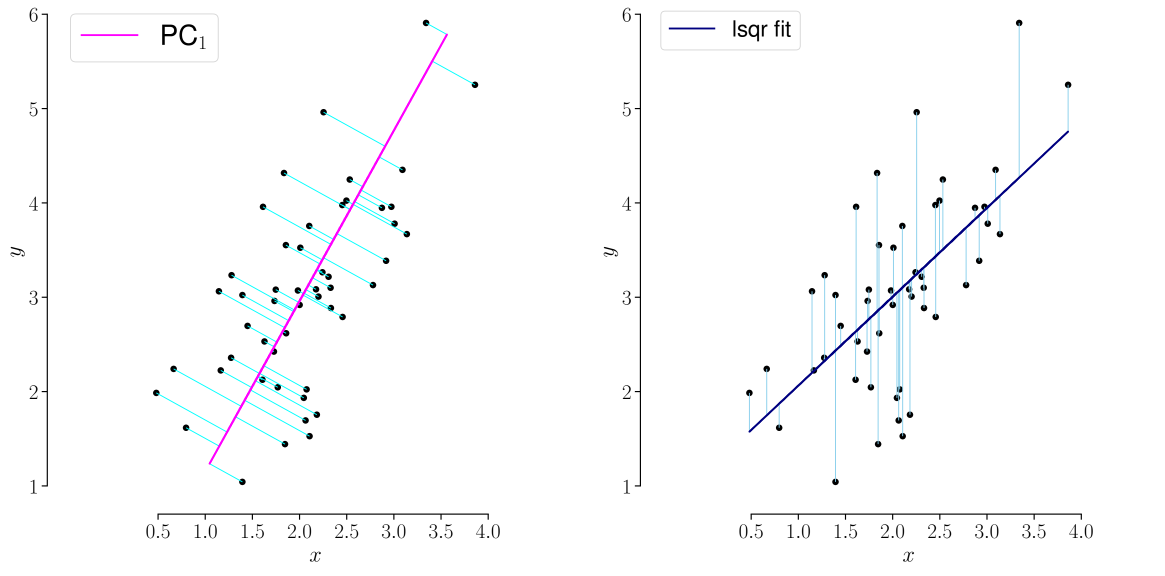
Now if we take the inner term, namely the dot product of $|\mathbf{a}_i^T\mathbf{u}_j|^2$ and expand it we get $(\mathbf{a}_i^T\mathbf{u}_j)^T\mathbf{a}_i^T\mathbf{u}_j = \mathbf{u}_j^T\mathbf{a}_i\mathbf{a}_i^T\mathbf{u}_j$. Summing over n it adds up to $\mathbf{u}_j^T\mathbf{A}\mathbf{A}^T\mathbf{u}_j$ which provides the final form in eq. (2.63). This is important as if we take the first principal component we maximise the respective summation and hence, by theorem 2.3.1 the following holds,

$$\mathbf{u}_1^T\mathbf{A}\mathbf{A}^T\mathbf{u}_1 \geq \mathbf{u}_2^T\mathbf{A}\mathbf{A}^T\mathbf{u}_2 \geq \cdots \geq \mathbf{u}_r^T\mathbf{A}\mathbf{A}^T\mathbf{u}_r \tag{2.64}$$

Note, that the equality only holds if equal singular values exist, otherwise it is a strict inequality.

Let us now visualise what PCA does with an example. We start, by comparing the results of PCA and ordinary least squares against a matrix $\mathbf{A} \in \mathbb{R}^{2\times 50}$, without any preprocessing, as shown in Figure 2.2.

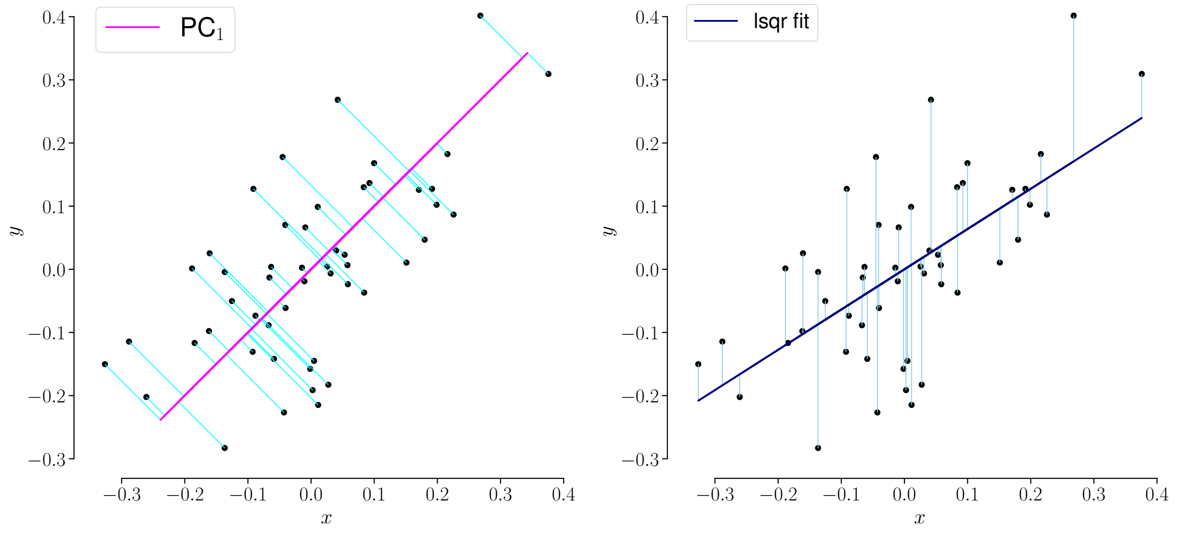
In the figures above we can visualise what PCA actually does, we can see that the minimised distance for all datapoints in our dataset it *perpendicular* to the line. On the other hand, in traditional least squares the minimisation objective changes and thus it results in a different line. However, note that we said that not centering the data before applying the SVD will not get us the same PC's if we do. To illustrate that, we now preprocess the dataset to be normalised,



(a) First PC direction, no preprocessing (b) Ordinary least squares fit, no preprocessing

Fig. 2.2 Comparison of the line dictated by the first PC after applying PCA against traditional Least Squares over a dataset of 50 datapoints in $\mathbb{R}^2$ without centering or being zero-mean.

centered, and have zero-mean. After performing that, we reapply the aforementioned methods once more, the results of which are shown in fig. 2.3.



(a) First PC direction (b) Ordinary least squares fit

Fig. 2.3 Comparison of the line dictated by the first PC after applying PCA against traditional Least Squares over a dataset of 50 datapoints in $\mathbb{R}^2$ after preprocessing ensuring it is normalised, centered, and has zero-mean.

We can see that PCA produces a *different* line as its slope changes.

In order to quantify how much information is "explained" or "captured" by each principal component we can compute the fraction of $\sigma_i/(n-1)$ where n is the number of samples. This property is a direct consequence of theorem 2.3.1 as the basis formed by the r singular vectors within $\mathbf{U}$ is able to explain the most variance out of any other set of vectors. Formally, we can define the explained variance as follows,

Definition (Explained variance of a matrix)**.** *The explained variance of a matrix* $\mathbf{A}$ *is given by,*

$$\mathrm{var}(\mathbf{A}) = \frac{\|\mathbf{A}\|_F^2}{n-1} = \frac{\sum_{i=1}^{n} \|\mathbf{a}_i\|^2}{n-1}$$

Or equivalently, we can reformulate this using the trace operator,

$$\mathrm{Tr}(\mathrm{cov}(\mathbf{A})) = \mathrm{Tr}\left(\frac{\mathbf{A}\mathbf{A}^T}{n-1}\right) = \frac{\mathrm{Tr}(\mathbf{A}\mathbf{A}^T)}{n-1} = \frac{\sum_{i=1}^{r} \sigma_i^2}{n-1}$$

Essentially, what PCA does is provide a principled mechanism of interpreting and understanding structure of a given matrix $\mathbf{A}$ with n datapoints within that reside in $\mathbb{R}^d$. It provides stronger guarantees than least squares, in the sense that normally regression tasks operate under the assumption that predictor variables are measured exactly and only the response variance has an error component. On the other hand, PCA makes no such assumption and thus can generalise better, meaning that requires no natural distinction between predictor and response variables [100, 45].

2.5 Differential privacy

Differential privacy (DP) is a formal framework first put forth by Dwork et al. [50, 51] and provides the mathematical constructions required for quantifying the privacy loss associated with *any* query result drawn from a statistical database. By using the term statistical database, we are referring to a set of data that are collected under the pledge of confidentiality to produce aggregate cohort statistics, that by their production, the privacy of individuals who provided the said data is not comprised. Intuitively, differential privacy can be motivated by the impossibility event. That is, for any given statistical database to achieve the privacy target of preventing disclosure of information about any individual with the said database against potential adversaries with arbitrary auxiliary data while still providing useful results. The curator of such a database for providing the queries is assumed to be a trusted entity whereas queries to it can be made by untrusted parties that make statistical queries to the said database and receive responses via randomised mechanisms or algorithms. Provided the above context, we can proceed to define differential privacy as follows,

Definition (Differential Privacy [51])**.** *A privacy mechanism* $\mathcal{A}$ *guarantees* ε*-differential privacy if and only if for any database* $D \in \mathcal{D}$ *and* $D' \in \mathcal{D}$ *differing on at most a single record, and for* *any possible result* $R \in \mathbf{Range}(\mathcal{A})$

$$e^{-\varepsilon}\mathbb{P}[\mathcal{A}(\mathcal{D}') = R] \leq \mathbb{P}[\mathcal{A}(\mathcal{D}) = R] \leq e^{\varepsilon}\mathbb{P}[\mathcal{A}(\mathcal{D}') = R]$$

where the probability is taken over the randomness of the privacy mechanism $\mathcal{A}$*.*

The above definition essentially guarantees us that an adversary, provided with the results of $\mathcal{A}$, can draw almost the same conclusions about any given individual within the database regardless if that individual was included in the input of $\mathcal{A}$ or not. More succinctly, ε-differential privacy requires that provided there is change of a single datapoint within the statistical database, then the output of the algorithm should not differ significantly, with the privacy risk of that occurring being controlled by the turning parameter ε. Following the definition of differential privacy, we can now derive the privacy loss as follows,

Definition (Privacy Loss [51])**.** *Let* $\mathcal{A}$ *be a privacy mechanism which provides a result* $R \subseteq \mathbf{Range}(\mathcal{A})$ *to a dataset* $D \in \mathcal{D}$*. Then we can define the privacy loss of* $\mathcal{A}$ *between datasets* $D, D' \in \mathcal{D}$ *at results* $R \in \mathbf{Range}(\mathcal{A})$ *as a random variable* $\mathcal{P}$ *as follows,*

$$\mathcal{P}(\mathcal{A}, D, D', R) = log\left(\frac{\mathbb{P}[\mathcal{A}(D) = R]}{\mathbb{P}[\mathcal{A}(D') = R]}\right)$$

where the probability is taken over the randomness of the privacy mechanism $\mathcal{A}$*.*

This formulation of differential privacy while robust suffers from two major limitations that hinder its practicality. The first is its intractability with respect to resources required for its computation in the context of large datasets, while the second being its sample complexity required to offer the aforementioned privacy guarantees [52]. To that end, a weaker definition of differential privacy exists that relaxes that guarantee of section 2.5 and allows it to be violated with probability $\delta > 0$.

Definition ((ε, δ)-Differential Privacy [51])**.** *A privacy mechanism* $\mathcal{A}$ *guarantees* (ε, δ)*-differential privacy if and only if for any database* $D \in \mathcal{D}$ *and* $D' \in \mathcal{D}$ *differing on at most a single record, and for any possible result* $S \subseteq \mathbf{Range}(\mathcal{A})$*,*

$$\mathbb{P}[\mathcal{A}(\mathcal{D}) \in S] \leq e^{\varepsilon}\mathbb{P}[\mathcal{A}(\mathcal{D}') \in S] + \delta$$

This can equivalently formulated as follows using our privacy definition from section 2.5,

$$\underset{R \sim \mathcal{A}(D)}{\mathbb{P}}[\mathcal{P}(\mathcal{A}, D, D', R) > \varepsilon] \leq \delta.$$

This corresponds to the definition of (ε, δ)-differential privacy, which enables tools for tighter privacy analysis over repeated queries to the data. The relaxation made to the original definition allows us to provide higher utility guarantees but at the cost of weaker privacy guarantees.

Namely, what this definition allows us to claim is that (ε, δ)-differential privacy ensures that for all adjacent databases $\mathcal{D}$ and $\mathcal{D}'$ the absolute value of the privacy loss will be bounded by ε with a probability of *at least* $1 - \delta$, where the probability space is over the domain of privacy mechanism $\mathcal{A}$. Another way to view this is that the (ε, δ) definition of differential privacy guarantees that every input to $\mathcal{A}$ is almost equally likely, up to ε, on datasets differing at most in a single datapoint except with probability at most equal to δ. Note, that δ preferably should be smaller than $1/|D|$ where $|D|$ is the cardinality of D meaning the number of individual datapoints within the given statistical database. Additionally, another important yet often missed remark, is if we *fix* δ to be equal to zero at all times in our provided definition of (ε, δ)-differential privacy then it yields our original definition of ε-differential privacy. Thus we can observe that ε (or pure) differential privacy is a special case of (ε, δ)-differential privacy and can be often seen as $(\varepsilon, 0)$-differential privacy.

However, a more quantitative explanation of the aforementioned definitions is in order, as there is a lot to unpack. In practice, what differential privacy does is to describe a *promise*, made by the data owner (e.g. a database manager), to a data subject (e.g. a participant in a dataset). This promise essentially affirms that the subject will not be affected, adversely or otherwise, by allowing their data to be used in any study or analysis no matter what other studies, datasets, or information sources are available [51]. What this promise unlocks is the ability to make confidential data widely available for accurate data analysis without resorting to any kind of special procedures such as data agreements, data protection plans, or similar schemes.

Differential privacy techniques exploit the paradox of learning *nothing* about an individual while learning *useful* information about the overall population. For example, a medical database might tell us that high blood pressure increases the risk of heart attack, in turn affecting the long-term insurance premiums of those suffering by it. Has an individual with high blood pressure been harmed? Maybe - as the insurance premiums might be higher as a result if the condition is disclosed. However, this disclosure might end up helping the individual as a suitable treatment plan to manage the underlying condition could be provided. Now, has the privacy of the individual been compromised? Obviously, more is known about the individual after this disclosure, but the question we need to answer is if the information was actually “leaked”? The stance of differential privacy is that in fact it was not, with the rationale being that the impact of suffering with high blood pressure is the same and independent of whether or not the individual disclosed that information. In a sense, differential privacy guarantees that the same conclusions, for example that high blood pressure increases the risk of heart attacks, will be reached, *independent* of whether any individual opts into or opts out of the dataset. More precisely for any sequence of outputs (e.g. responses to queries) ensures that “essentially” have an equal probability of occurring, independent of the presence or absence of any individual within the dataset. In that context, the probabilities of the event to occur are taken over random choices made by the privacy mechanism and the firmness of that guarantee is captured by the (tunable) parameter ε. A *smaller* ε will yield better privacy at the cost of the resulting query accuracy, which is also evident in the formulations presented previously. However, one important detail is that differential privacy is a *definition* rather than an actual specification for an algorithm. This means that for any task and a given set of differential privacy parameters there will be many algorithms for completing that task in a differentially private way.

Concluding, we note that due to practicality implications, our constructions put forth in Chapter 4 use the (ε, δ) definition of differential privacy to provide such guarantees for our Federated Principal Component framework.

2.6 Federated computation

With the advent of powerful mobile phones, tablets, and wearables a significant amount of users have switched to one as their primary computing device instead of a traditional personal computer. Such devices are highly portable with far better battery life than most laptops and are rarely separated from their owners. Further, most of these devices are equipped with a myriad of different sensors offering rich user interactions and activity monitoring, that in turn can provide unprecedented amount of data, much of it private in nature. Naturally, insights gathered from such volume of data hold the promise of unlocking features and experiences that would be infeasible otherwise. However, their sensitive nature means that there are risks and implications if gathered in a centralised location. Additionally, in most cases due to the sheer volume of the data or the communication overhead incurred centralisation might be impossible.

To exploit these datasets this while also providing data ownership for each client participating in the computation a new computing paradigm was put forth dubbed as federated learning [108, 128]. Concretely, it can be seen as a relaxation of the distributed computation model, whereas each client solves its own problem but also retains its data. The model is computed using local and incremental updates only propagating these updates sending them to clients holding the centralised model for aggregation. For each client, these updates are targeted improvements to its own model and thus are ephemeral. Meaning that under this model there is no need to store the training data resulting in considerable space savings. In other words, this framework allows *decentralised* training of the model without the massive communication costs incurred during centralisation procedures that traditional methods employ. Moreover, traditional training assumes that the data are evenly distributed across each client and the paradigm was primarily exploited to effectively parallelise the workload across the available computation nodes. Additionally, under more traditional models it was also assumed that the computing nodes would be relatively equal in terms of available resources. In contrast, federated learning aims to train unevenly distributed datasets in clients that are assumed to have heterogeneous hardware configurations and limited connectivity. For example, it is able to unlock training using different generations of mobile phones which are known to have large discrepancies in their capabilities and available resources. Practically speaking, one of the principal benefits of this approach is that it allows the decoupling of the model training from requiring constant and direct access to the entire dataset. However, this computation paradigm has been used for mostly for training of neural networks and federated adaptations of classical data analysis algorithms are still largely missing [162, 88, 65]. We exploit this method and refine it to be used as a building block that enables us to put forth a novel algorithm for federated computation of PCA, which is presented in Chapter 4.

One important consequence of using federated learning for model training is that it ensures data ownership within each client, which can guarantee a notion of privacy or more precisely, data-secrecy. Indeed, since no client data, apart from aggregates such as model weights or parameters are transmitted verbatim for processing this in turn makes it harder for a malicious actor to perform successful attacks on a federated model. This can be further strengthened if standard practices are used, such as encrypting data during transmission and having a verification scheme for when new clients join the federation. Even if the data transmitted are summaries, sufficient trust to the nodes coordinating the training is still required. This is because, as we previously mentioned, federated data can contain sensitive information about individuals and even if only summaries of data are transmitted they still can contain personally identifiable information. Despite these provisions data can still be potentially leaked if a malicious actor is able to make multiple specific queries on the dataset. To this end, if additional privacy is required we can exploit differential privacy randomisation techniques as previously discussed. Concretely, our federated PCA algorithm we introduce in Chapter 4 can also release a differentially private model which extends the work put forth by [32, 33] to be applicable in a federated setting and to non-symmetric matrices. This protects both local models as well as their aggregations which in turn safeguards against malicious leaf nodes and/or aggregation servers.

2.7 Notation

This section collectively states the notational conventions used throughout this thesis for the convenience of the reader. We use lowercase letters y for scalars, bold lowercase letters $\mathbf{y}$ for vectors, bold capitals $\mathbf{Y}$ for matrices, and calligraphic capitals $\mathcal{U}$ for subspaces. The orthogonal complement of a subspace $\mathcal{U}$ is depicted as $\mathcal{U}^{\perp}$. The field of rational numbers is denoted as $\mathbb{R}$ and we write the dimensionality of the domain as its exponent; e.g: $\mathbb{R}^{d\times n}$ depicting a two dimensional matrix of d rows and n columns, with $n, d \geq 0$. The transpose and inverse of any matrix $\mathbf{Y}$ is denoted as $\mathbf{Y}^T$ and $\mathbf{Y}^{-1}$ respectively. For any positive definite (PD) and semidefinite (PSD) matrix $\mathbf{Y}$, we write it as $\mathbf{Y} \succ 0$ and $\mathbf{Y} \succeq 0$ respectively. We denote the span$(\cdot)$ operator to be the linear span of a set of vectors. The expectation of a random variable x is written as $\mathbb{E}[x]$, while the probability measure over x is written as $\mathbb{P}[x]$. If $\mathbf{Y} \in \mathbb{R}^{d\times n}$ and $S \subset \{1, \ldots, m\}$, then $\mathbf{Y}_S$ is the block composed of columns indexed by S. We reserve $\mathbf{0}_{m\times n}$ for the zero matrix in $\mathbb{R}^{m\times n}$ and $\mathbf{I}_n$ for the identity matrix in $\mathbb{R}^{n\times n}$. Additionally, we use $\|\cdot\|_{\mathrm{F}}$ to denote the Frobenius norm, $\|\cdot\|$ to denote the ℓ_2 norm, and the trace operator with $\mathrm{Tr}(\cdot)$. We denote that the Singular Value Decomposition to be abbreviated as SVD and use it as an operator with $\mathrm{SVD}(\mathbf{Y})$. The truncated SVD is defined by extending the SVD operator as $\mathrm{SVD}(\mathbf{Y}, r)$ by adding the truncation rank r as an argument to the operator. If $\mathbf{Y} \in \mathbb{R}^{d\times n}$ we let $\mathbf{Y} = \mathbf{U\Sigma V}^T$ be its full SVD formed from unitary $\mathbf{U} \in \mathbb{R}^{d\times d}$ and $\mathbf{V} \in \mathbb{R}^{n\times n}$ and diagonal $\mathbf{\Sigma} \in \mathbb{R}^{d\times n}$. The values $\mathbf{\Sigma}_{i,i}^{r,r} = \sigma_i(\mathbf{Y}) \geq 0$ are the singular values of $\mathbf{Y}$. If $1 \leq r \leq \min(d, n)$, we let $\widehat{\mathbf{Y}}_r = \mathbf{U}_r\mathbf{\Sigma}_r\mathbf{V}_r^T = \mathrm{SVD}(\mathbf{Y}, r)$ be the singular value decomposition of its *best rank-r approximation* formed by unitary $\mathbf{U} \in \mathbb{R}^{d\times r}$ and $\mathbf{V} \in \mathbb{R}^{r\times n}$ and diagonal $\mathbf{\Sigma} \in \mathbb{R}^{r\times r}$. That is, the solution of $\min\{\|\mathbf{Z} - \mathbf{Y}\|_F : \mathrm{rank}\,(\mathbf{Z}) \leq r\}$. Using this notation, we define $[\mathbf{U}_r, \mathbf{\Sigma}_r]$ be the rank-r *principal subspace* of $\mathbf{Y}$. We also define $\mathrm{SVDS}(\mathbf{Y}, r)$ as $[\mathbf{U}_r, \mathbf{\Sigma}_r, \mathbf{V}_r^T] = \mathrm{SVDS}(\mathbf{Y}, r)$ which instead of the computed rank-r matrix returns the individual components of the SVD as defined above. When there is no risk of confusion, we will abuse notation and use $\mathrm{SVDS}(\mathbf{Y}, r)$ to denote the rank-r left principal subspace with the r leading singular values $[\mathbf{U}_r, \mathbf{\Sigma}_r]$. In a similar note, we also let $[\mathbf{Q}, \mathbf{R}] = \mathrm{QR}(\mathbf{Y})$ be the rank QR factorisation of $\mathbf{Y}$. We also let $\lambda_1(\mathbf{Y}) \geq \cdots \geq \lambda_k(\mathbf{Y})$ be the eigenvalues of $\mathbf{Y}$ when $d = n$. Finally, we let $\vec{\mathbf{e}}_k \in \mathbb{R}^d$ be the k-th canonical vector in $\mathbb{R}^d$.

Chapter 3

Beyond Regular Singular Value Decomposition

In Chapter 2, we introduced SVD, which is an invaluable tool for factorising a matrix with stringent quality guarantees. However, this optimality comes at a cost. Traditional SVD requires significant resources both in terms of storage as well as computation. Given that the datasets have exploded in size and rarely remain as-is, performing the full SVD every time the dataset is appended with incoming data seems highly inefficient. In this chapter we introduce an algorithm for Memory-limited Online Subspace Estimation (MOSES) that attempts to optimally solve this problem. Further, it can be used for *both* estimating the principal components of data and reducing its dimension, requiring only *one pass* over the data. Concretely, consider a scenario where the data vectors are presented sequentially to a user who has limited storage and processing time available, for example in the context of sensor networks. In this scenario, our proposed algorithm maintains an estimate of leading principal components of the data that has arrived so far while also reducing its dimension with *one pass* over the data.

In terms of its origins, our solution is based on generalising the popular incremental Singular Vale Decomposition (SVD) to handle *thin* blocks of data. Furthermore, this generalisation, is in part what allows us to complement our method with a comprehensive statistical analysis that is not available for incremental SVD, despite its widespread empirical success. Notably, it also enables us to concretely interpret the proposed algorithm as an approximate solver for the underlying non-convex optimisation program. We also find that our proposed method shows state-of-the-art performance in our numerical experiments with both synthetic and real-world datasets. As we will see later in Chapter 4 this method is one of our primary building blocks that allows to perform a federated computation of PCA.

3.1 Introduction

Linear models are pervasive in data and computational sciences and Principal Component Analysis (PCA) in particular is an indispensable tool for detecting linear structure in collected data and has been extensively studied in scientific literature [175, 6, 110, 169, 181]. Principal Components (PC's) are the directions that preserve most of the "energy" of a dataset and can be used for linear dimensionality reduction, among other things. In turn, successful dimensionality reduction is at the heart of classification, regression, and other learning tasks that often suffer from the "curse of dimensionality", where having a small number of training samples in relation to the dimension of data typically leads to overfitting [87].

We are interested in both computing the principal components *and* reducing the dimension of data that is presented sequentially to a client. Due to potential hardware limitations, the client can only store small amounts of data, which in turn severely limits the available processing time for each incoming data vector. For example, consider a network of battery-powered and cheap sensors that must relay their measurements to a central node on a daily basis. Each sensor has a limited storage and does not have the power to relay all the raw data to the central node. One solution is then for each sensor to reduce the dimension of its data to make transmission to the central node possible. Even if each sensor had unlimited storage, the frequent daily updates scheduled by the central node would force each sensor to reduce the dimension of its data "on the go" before transmitting it to the central node. In the context of this work we focus on incremental or "streaming" algorithms for the computation of "subspace tracking" or "streaming SVD" denoting that such algorithms are able to update and track a dataset subspace using streaming observations and thus we use these terms interchangeably. However, we understand that these methods have different connotations and have arisen from different scientific domains. Concretely, "subspace tracking" originated within signal processing literature where often is required to update a subspace within a dynamic environment [37]. On the other hand, the more recent terminology of "streaming“ or "online“ PCA can be found in machine learning and data-science literature stemming from the need to replicate the behaviour of batch PCA using streaming or too large for memory data [133, 130] A number of similar problems are listed in [12, 11].

Motivated by such scenarios, we are interested in developing a *streaming* algorithm for linear dimensionality reduction, namely an algorithm with minimal storage and computational requirements while providing strong guarantees about its output quality. As more and more data vectors arrive, this algorithm would keep a running estimate of the principal components of the data *and* project the available data onto this estimate to reduce its dimension. As we will see shortly, what we need here is a streaming algorithm to compute truncated SVD.

More specifically, existing *incremental* SVD algorithms in the literature update its estimate of truncated SVD of the data matrix with *every* new incoming vector [29, 26, 25, 37, 116]. This poses computational issues, as performing the SVD for *every* new incoming vector can prove costly and thus inefficient. However, more crucially, to the best of our knowledge and despite its popularity, incremental SVD lacked comprehensive statistical guarantees. In fact, [10] only very recently provided stochastic analysis for two of the variants of incremental SVD in [109, 137]. Concretely, in [10] the authors studied how well the output of incremental SVD approximates (only) the leading principal component of data, in expectation. In particular, [10] does *not* offer any guarantees for dimensionality reduction, see Section 3.6 for a detailed discussion. **Summary of contributions**: In this chapter we put forth a streaming algorithm able to operate on memory-limited devices that incrementally computes the rank-r truncation of a given data matrix. Contrary to prior art, our algorithm provides several innovations. Firstly, our algorithm is able to be computed using *thin* blocks of data the size of which are bounded by the target rank-r rather than the ambient dimension which normally is significantly larger. This increases both the rate of estimate updates, as well as improve its computational efficiency. Secondly, our algorithm is able to provide quality guarantees on all of the SVD components as well as the projected data while only requiring a *single* pass over the data. In fact, we show both theoretically and empirically that this algorithm is almost equivalent to the offline truncated SVD of equal rank, which is assumed to have infinite resources for its computation. More importantly, we complement our algorithm with comprehensive statistical guarantees about its output quality, which to the best of our knowledge and despite its popularity, incremental SVD lacked.

3.2 Memory-limited Online Subspace Estimation

Consider a sequence of vectors $\{\mathbf{y}_1, ..., \mathbf{y}_\tau\} \subset \mathbb{R}^d$, presented to us sequentially, and let

$$\mathbf{Y}_\tau := \left[\begin{array}{cccc} \mathbf{y}_1 & \mathbf{y}_2 & \cdots & \mathbf{y}_\tau \end{array} \right] \in \mathbb{R}^{d\times\tau}, \tag{3.1}$$

for short. We conveniently assume throughout that $\mathbf{Y}_\tau$ is *centred*, namely the entries of each row of $\mathbf{Y}_\tau$ sum up to zero. It is then a consequence of the theorem 2.3.1 that leading r principal components of $\mathbf{Y}_\tau$ in fact coincide with leading r left singular vectors of $\mathbf{Y}_\tau$. More specifically, let for a given matrix $\mathbf{Y}_\tau \in \mathbb{R}^{d\times\tau}$

$$\mathrm{SVD}\left(\mathbf{Y}_\tau\right) = \mathbf{U}_\tau \mathbf{\Sigma}_\tau {\mathbf{V}_\tau}^T, \tag{3.2}$$

be the data matrix $\widehat{\mathbf{Y}}_\tau$ using the outputs of the SVD$(\mathbf{Y}_\tau)$, where $\mathbf{U}_\tau \in \mathbb{R}^{d\times d}$ and $\mathbf{V}_\tau \in \mathbb{R}^{\tau\times n}$ are orthonormal bases, and the diagonal matrix $\mathbf{\Sigma}_\tau \in \mathbb{R}^{d\times\tau}$ contains the singular values of $\mathbf{Y}$ in decreasing order.

Now, provided with the above constructions let us assume that rank$(\mathbf{Y}_\tau) \geq r$. Then the first r columns of $\mathbf{U}_\tau$, which we collect in $\mathbf{U}_{\tau,r} \in \mathbb{R}^{d\times r}$, are leading r principal components of $\mathbf{Y}_\tau$. We accordingly decompose $\mathbf{Y}_\tau$ into two components, namely

$$\begin{aligned}
\mathrm{SVD}\left(\mathbf{Y}_\tau\right) &= \mathbf{U}_\tau \boldsymbol{\Sigma}_\tau \mathbf{V}_\tau^T \\
&= \left[\begin{array}{cc} \mathbf{U}_{\tau,r} & \mathbf{U}_{\tau,r^+} \end{array}\right] \left[\begin{array}{cc} \boldsymbol{\Sigma}_{\tau,r} & \\ & \boldsymbol{\Sigma}_{\tau,r^+} \end{array}\right] \left[\begin{array}{cc} \mathbf{V}_{\tau,r}^T & \mathbf{V}_{\tau,r^+}^T \end{array}\right] \\
&= \mathbf{U}_{\tau,r}\boldsymbol{\Sigma}_{\tau,r}\mathbf{V}_{\tau,r}^T + \mathbf{U}_{\tau,r^+}\boldsymbol{\Sigma}_{\tau,r^+}\mathbf{V}_{\tau,r^+}^T \\
&=: \mathbf{Y}_{\tau,r} + \mathbf{Y}_{\tau,r^+}, \qquad (3.3)
\end{aligned}$$

where the empty blocks are zero. It is easy to see that the column and row spaces of $\mathbf{Y}_{\tau,r}$ are orthogonal to those of $\mathbf{Y}_{\tau,r^+}$, namely

$$\mathbf{Y}_{\tau,r}\mathbf{Y}_{\tau,r^+}^T = \mathbf{0}_{d\times d}, \qquad \mathbf{Y}_{\tau,r}^T\mathbf{Y}_{\tau,r^+} = \mathbf{0}_{\tau\times\tau}. \qquad (3.4)$$

Moreover, Theorem 2.3.1 implies that $\mathbf{Y}_{\tau,r} = \mathrm{SVD}_r(\mathbf{Y}_\tau)$ is the best rank-r truncation of $\mathbf{Y}_\tau$. That is, $\mathbf{Y}_{\tau,r}$ is a best rank-r approximation of $\mathbf{Y}_\tau$ with the corresponding residual,

$$\begin{aligned}
\|\mathbf{Y}_\tau - \mathbf{Y}_{\tau,r}\|_F^2 &= \min_{\mathrm{rank}(\mathbf{X})=r} \|\mathbf{Y}_\tau - \mathbf{X}\|_F^2 \\
&= \|\mathbf{Y}_{\tau,r^+}\|_F^2 \qquad \text{(see (3.4))} \\
&= \sum_{i\geq r+1} \sigma_i^2(\mathbf{Y}_\tau) \\
&=: \rho_r^2(\mathbf{Y}_\tau), \qquad (3.5)
\end{aligned}$$

where $\sigma_1(\mathbf{Y}_\tau) \geq \sigma_2(\mathbf{Y}_\tau) \geq \cdots \sigma_r(\mathbf{Y}_\tau)$ are the singular values of $\mathbf{Y}_\tau$. Given the leading r principal components of $\mathbf{Y}_\tau$, namely $\mathbf{U}_{\tau,r}$ in (3.3), we can reduce the dimension of data from d to r by *projecting* $\mathbf{Y}_\tau$ onto the span of $\mathbf{U}_{\tau,r}$, that is

$$\begin{aligned}
\mathbf{U}_{\tau,r}^T \cdot \mathbf{Y}_\tau &= \mathbf{S}_{\tau,r}^T(\mathbf{Y}_{\tau,r} + \mathbf{Y}_{\tau,r^+}) \qquad \text{(see (3.3))} \\
&= \mathbf{U}_{\tau,r}^T\mathbf{Y}_{\tau,r} \qquad \text{(see (3.4))} \\
&= \boldsymbol{\Sigma}_{\tau,r}\mathbf{V}_{\tau,r}^T \in \mathbb{R}^{r\times\tau}. \qquad \text{(see (3.4))} \qquad (3.6)
\end{aligned}$$

The projected data matrix $\mathbf{U}_{\tau,r}^T\mathbf{Y}_\tau \in \mathbb{R}^{r\times\tau}$ again has τ data vectors (namely, columns) but these vectors are embedded in, often much smaller, $\mathbb{R}^r$ rather than $\mathbb{R}^d$. Note also that,

$$\begin{aligned}
\mathbf{Y}_{\tau,r} &= \mathrm{SVD}(\mathbf{Y}_\tau, r) \\
&= \mathbf{U}_{\tau,r}\mathbf{U}_{\tau,r}^T \cdot \mathbf{Y}_\tau \\
&= \mathbf{U}_{\tau,r}\mathbf{U}_{\tau,r}^T (\mathbf{Y}_{\tau,r} + \mathbf{Y}_{\tau,r^+}) \qquad \text{(see (3.3))} \\
&= \mathbf{U}_{\tau,r}\mathbf{U}_{\tau,r}^T \mathbf{Y}_{\tau,r} \qquad \text{(see (3.4))} \\
&= \underbrace{\mathbf{U}_{\tau,r}}_{\text{PCs}} \cdot \underbrace{\mathbf{\Sigma}_{\tau,r}\mathbf{V}_{\tau,r}^T}_{\text{projected data}} . \qquad \text{(see (3.3))}
\end{aligned} \tag{3.7}$$

That is, rank-r truncation of $\mathbf{Y}_\tau$ encapsulates both leading r principal components of $\mathbf{Y}_\tau$, namely $\mathbf{U}_{\tau,r}$, and the projected data matrix $\mathbf{U}_{\tau,r}^T\mathbf{Y}_\tau = \mathbf{\Sigma}_{\tau,r}\mathbf{V}_{\tau,r}^T$. In other words, computing a rank-r truncation of the data matrix both yields its principal components and reduces the dimension of data at once. We are in this work interested in developing a *streaming* algorithm to compute $\mathbf{Y}_{\tau,r} = \mathrm{SVD}(\mathbf{Y}_\tau, r)$, which is a rank-$r$ truncation of the data matrix $\mathbf{Y}_\tau$. More specifically, to compute $\mathbf{Y}_{\tau,r}$, we restrict ourselves to perform only a *single pass* through the columns of $\mathbf{Y}_\tau$, as they arrive. We also operate under the realistic assumption that each device has limited amount of storage available, namely $\mathcal{O}(d)$ bits. Meaning that the amount of memory available is proportional to the ambient dimension of the the matrix $\mathbf{Y}_\tau$ which is $\mathbb{R}^d$. See also Figure 3.1.

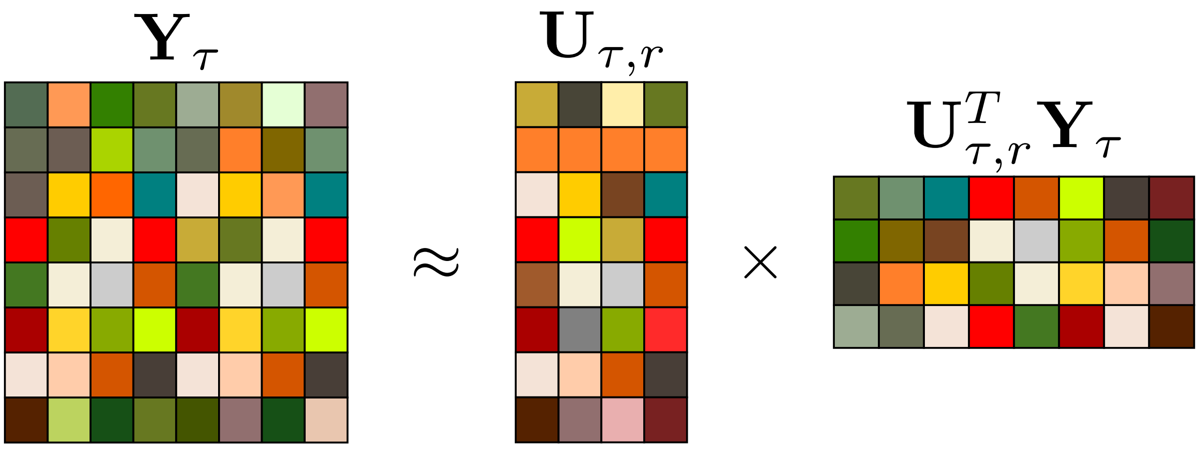


Fig. 3.1 Given a data matrix $\mathbf{Y}_\tau \in \mathbb{R}^{d\times\tau}$, truncated SVD finds the best low-dimensional linear model of rank r to represent the data. For a typically small integer r, we compute $\mathbf{Y}_{\tau,r} = \mathrm{SVD}(\mathbf{Y}_\tau, r) = \mathbf{U}_{\tau,r} \cdot \mathbf{U}_{\tau,r}^T\mathbf{Y}_\tau$, where $\mathbf{U}_{\tau,r} \in \mathbb{R}^{\tau\times r}$ contains leading r principal components of $\mathbf{Y}_\tau$. Further, $\mathbf{U}_{\tau,r}^T\mathbf{Y}_\tau \in \mathbb{R}^{r\times\tau}$ is the projected data matrix with reduced dimension r (instead of d).

For a block size $b \in \mathbb{N}$, our strategy is to iteratively group every b incoming vectors into an $d \times b$ block and then update a rank-r estimate of the data that has been received so far. We assume throughout that $r \le b \le \tau$ and in fact often take the block size as $b = \mathcal{O}(r)$. It is convenient to assume that the number of blocks is an integer and we can claim that $K = \{1, \dots, \lceil \tau/b \rceil\}$. Upon arrival of a new data block $\{\mathbf{y}_\tau\}_{\tau=(k-1)b+1}^{kb}$, we concatenate these vectors to form the matrix,

$$\mathbf{B}_k = \begin{bmatrix} \mathbf{y}_{(k-1)b+1} & \cdots & \mathbf{y}_{kb} \end{bmatrix} \in \mathbb{R}^{d\times b}.$$

For every $k \in [1:K] := \{1, \cdots, K\}$, we then set

$$[\mathbf{U}_{kb,r}, \mathbf{\Sigma}_{kb,r}, \mathbf{V}_{kb,r}^T] = \text{SVDS}\left(\begin{bmatrix} \widehat{\mathbf{Y}}_{(k-1)b,r} & \mathbf{B}_k \end{bmatrix}, r\right), \tag{3.8}$$

where $\mathbf{U}_{kb,r} \in \mathbb{R}^{d\times d}$, $\mathbf{\Sigma}_{kb,r} \in \mathbb{R}^{r\times r}$, $\mathbf{V}_{kb,r}^T \in \mathbb{R}^{r\times kb}$, and with the convention that $\widehat{\mathbf{Y}}_{0,r}$ is the empty matrix. Then to get the updated projected data matrix $\widehat{\mathbf{Y}}_{kb,r}$ at block kb, we can do so by simply multiplying the outputs of Equation 3.8. Concretely, we can do,

$$\widehat{\mathbf{Y}}_{kb,r} = \mathbf{U}_{kb,r}\mathbf{\Sigma}_{kb,r}\mathbf{V}_{kb,r}^T \in \mathbb{R}^{d\times kb}, \tag{3.9}$$

In practice this is a recursion using the SVD operator with its input parameters being previous output namely $\widehat{\mathbf{Y}}_{(k-1)b,r}$ concatenated with the newly gathered data block $\mathbf{B}_k$. We call this simple algorithm MOSES for Memory-limited Online Subspace Estimation. The output of MOSES after K iterations is,

$$\widehat{\mathbf{Y}}_{Kb,r} = \widehat{\mathbf{Y}}_{\tau,r},$$

which contains both an estimate of leading r principal components of $\mathbf{Y}_\tau$ and the projection of $\mathbf{Y}_\tau$ onto this estimate, as discussed below. For easy reference, MOSES is summarised in Algorithm 1, which is presented below.

Algorithm 1: MOSES: A streaming algorithm for linear dimensionality reduction

```
Data: Sequence of vectors $\{\mathbf{y}_t\}_{t\geq 1} \subset \mathbb{R}^d$, rank $r$, and block size $b \geq r$.
Result: Sequence $\{\widehat{\mathbf{Y}}_{kb,r}\}_k$, where $\widehat{\mathbf{Y}}_{kb,r} \in \mathbb{R}^{d\times kb}$ for every $k \geq 1$.
Function MOSES($\{\mathbf{y}_t\}_{t\geq 1} \subset \mathbb{R}^d$, $r$, $b$) is
    Set $\widehat{\mathbf{Y}}_{0,r} \leftarrow \{\}$.
    for $k \geq 1$ do
        Form $\mathbf{B}_k \in \mathbb{R}^{d\times b}$ by concatenating $\{y_\tau\}_{\tau=(k-1)b+1}^{kb}$.
        /* use SVD to get a rank-$r$ truncation of its argument. */
        Set $\widehat{\mathbf{Y}}_{kb,r} \leftarrow$ SVD($[\widehat{\mathbf{Y}}_{(k-1)b,r}\ \mathbf{B}_k]$, $r$)
    end
end
```

While this algorithm can get us the desired output, it is computationally inefficient. The most notable inefficiently is that in Algorithm 1 the full rank-r matrix seen thus far, namely $\widehat{\mathbf{Y}}_{(k-1)b,r}$, is required as an input to the recursion. Naturally, as this matrix grows by the size of each incoming block at every iteration, we can see that this algorithm quickly becomes intractable. To address this, we provide an efficient implementation of MOSES is given in Algorithm 2. This implementation, explicitly maintains both the estimates of principal components as well as the projected data *incrementally* and without requiring the expansion of $\widehat{\mathbf{Y}}_{(k-1)b,r}$ resulting in significant performance and storage gains. Additionally, as discussed below, we will see that the storage and computational requirements of Algorithm 2 are nearly optimal.

Discussion

MOSES maintains a rank-r estimate of the data that has been received so far, and updates its estimate in every iteration to account for the new incoming block of data. In other words, MOSES simultaneously keeps an estimate of principal components *and* the projection of the available data onto this estimate. More specifically, note that the final output of MOSES, namely $\widehat{\mathbf{Y}}_{\tau,r} \in \mathbb{R}^{d\times\tau}$, is at most rank-$r$, and let

$$\widehat{\mathbf{U}}_{\tau,r}\widehat{\mathbf{\Sigma}}_{\tau,r}\widehat{\mathbf{V}}_{\tau,r}^T = \text{SVD}(\widehat{\mathbf{Y}}_{\tau,r}), \tag{3.10}$$

be its SVD. We also know that by using the SVD components (though SVDS) that,

$$\widehat{\mathbf{Y}}_{\tau,r} = \widehat{\mathbf{U}}_{\tau,r}\widehat{\mathbf{\Sigma}}_{\tau,r}\widehat{\mathbf{V}}_{\tau,r} \in \mathbb{R}^{d\times\tau}$$

Then, $\widehat{\mathbf{U}}_{\tau,r} \in \mathbb{R}^{d\times r}$ is MOSES's estimate of principal components of the data matrix $\mathbf{Y}_\tau$, and,

$$\widehat{\mathbf{U}}_{\tau,r}^T\widehat{\mathbf{Y}}_{\tau,r} = \widehat{\mathbf{\Sigma}}_{\tau,r}\widehat{\mathbf{V}}_{\tau,r} \in \mathbb{R}^{r\times\tau}$$

is the projection of $\widehat{\mathbf{Y}}_{\tau,r}$ onto this estimate. That is, $\widehat{\mathbf{U}}_{\tau,r}^T\widehat{\mathbf{Y}}_{\tau,r}$ is the MOSES's estimate of the projected data matrix.

It is natural to ask how MOSES compares with the "offline" truncated SVD. To be concrete, recall that $\mathbf{Y}_{\tau,r} = \text{SVD}(\mathbf{Y}_\tau, r)$ is a rank-r truncation of the data matrix $\mathbf{Y}_\tau$ with the corresponding residual of $\rho_r^2(\mathbf{Y}_\tau)$, see (3.5). Because $\mathbf{Y}_{\tau,r}$ is a best rank-r approximation of $\mathbf{Y}_\tau$, the final output $\widehat{\mathbf{Y}}_{\tau,r}$ of MOSES cannot be a better approximation of $\mathbf{Y}_\tau$, that is,

$$\min_{\text{rank}(\mathbf{X})=r} \|\mathbf{Y}_\tau - \mathbf{X}\|_F^2 = \|\mathbf{Y}_\tau - \mathbf{Y}_{\tau,r}\|_F^2 = \rho_r^2(\mathbf{Y}_\tau) \leq \|\mathbf{Y}_\tau - \widehat{\mathbf{Y}}_{\tau,r}\|_F^2. \tag{3.11}$$

However, our main technical contribution in Theorem 3.4.2 below states that, under certain conditions, $\widehat{\mathbf{Y}}_{\tau,r}$ is not much worse than $\mathbf{Y}_{\tau,r}$, in the sense that

$$\rho_r^2(\mathbf{Y}_\tau) \leq \|\mathbf{Y}_\tau - \widehat{\mathbf{Y}}_{\tau,r}\|_F^2 \lesssim \text{poly}(\tau) \cdot \rho_r^2(\mathbf{Y}_\tau), \tag{3.12}$$

and the polynomial factor above is relatively small. That is, MOSES for streaming dimensionality reduction nearly matches the performance of its offline version that has access to unlimited storage and computing resources, see Section 3.4 for the details.

Origins

Incremental SVD is a streaming algorithm that updates its estimate of (truncated) SVD of the data matrix with every new incoming vector [29, 26, 25, 37, 116]. It is easy to verify that MOSES generalises traditional incremental SVD setting to update its estimate with every incoming block of data, rather than with every incoming data vector. As detailed later in Section 3.6, this small difference between incremental SVD and MOSES is in part what enables us to complement MOSES with a comprehensive statistical analysis in Theorem 3.4.2. To the best of our knowledge, such results are absent from the literature of incremental SVD, despite its popularity and empirical success. This refinement also allows us to concretely interpret MOSES as an approximate solver for the underlying non-convex program, as detailed in Section 3.3.

Storage and computational requirements

The efficient implementation of MOSES in Algorithm 2 is based on the ideas from incremental SVD and it is straightforward to verify that Algorithms 1 and 2 are indeed equivalent. Concretely, at iteration k, the relation between the output of Algorithm 1 ($\widehat{\mathbf{Y}}_{kb,r}$) and the output of Algorithm 2 ($\widehat{\mathbf{U}}_{kb,r}, \widehat{\mathbf{\Sigma}}_{kb,r}, \widehat{\mathbf{V}}_{kb,r}$) is,

$$\mathrm{SVD}(\widehat{\mathbf{Y}}_{kb,r}) = \widehat{\mathbf{U}}_{kb,r}\widehat{\mathbf{\Sigma}}_{kb,r}\widehat{\mathbf{V}}_{kb,r}^T,$$

where the right-hand side above is the SVD of $\widehat{\mathbf{Y}}_{kb,r}$. More specifically, $\widehat{\mathbf{U}}_{kb,r} \in \mathbb{R}^{d\times r}$ has orthonormal columns and is the MOSES's estimate of r leading principal components of $\mathbf{Y}_{kb} \in \mathbb{R}^{d\times kb}$, where we recall that $\mathbf{Y}_{kb}$ is the data received so far. Moreover,

$$\widehat{\mathbf{U}}_{kb,r}^T\widehat{\mathbf{Y}}_{kb,r} = \widehat{\mathbf{\Sigma}}_{kb,r}\widehat{\mathbf{V}}_{kb,r}^T \in \mathbb{R}^{r\times kb}$$

is the projection of $\widehat{\mathbf{Y}}_{kb,r}$ onto this estimate, namely $\widehat{\mathbf{U}}_{kb,r}^T\widehat{\mathbf{Y}}_{kb,r}$ is MOSES's estimate of the projected data matrix so far. In words, the efficient implementation of MOSES in Algorithm 2 explicitly maintains estimates of both principal components and the projected data, at every iteration.

Let us now evaluate the storage and computational requirements of the efficient MOSES algorithm. At the start of iteration k, Algorithm 2 stores the following matrices,

$$\widehat{\mathbf{U}}_{(k-1)b,r} \in \mathbb{R}^{d\times r}, \qquad \widehat{\mathbf{\Sigma}}_{(k-1)b,r} \in \mathbb{R}^{r\times r}, \qquad \widehat{\mathbf{V}}_{(k-1)b,r} \in \mathbb{R}^{(k-1)b\times r},$$

and after that also receives and stores the incoming block $\mathbf{B}_k \in \mathbb{R}^{d\times b}$. This requires $\mathcal{O}(r(d+(k-1)b+1)) + \mathcal{O}(bd)$ bits of memory, because $\widehat{\mathbf{\Sigma}}_{(k-1)b,r}$ is diagonal. Assuming that $\mathcal{O} = O(r)$, Algorithm 2 therefore requires $\mathcal{O}(r(d+kr))$ bits of memory at iteration k. Note that this is optimal, as it is impossible to store a rank-r matrix of size $d \times kb$ with fewer bits when $b = \mathcal{O}(r)$. It is also easy to verify that Algorithm 2 performs $\mathcal{O}(r^2(d+kb)) = \mathcal{O}(r^2(d+kr))$ flops in iteration k. The dependence of both storage and computational complexity on k is due to the fact that MOSES maintains both an estimate of principal components in $\widehat{\mathbf{U}}_{kb,r}$ and an estimate of the projected data in $\mathbf{\Sigma}_{kb,r}\mathbf{V}^T_{kb,r}$. To maximise the efficiency, one might optionally "flush out" the projected data after every $\lceil d/b \rceil$ iterations, as described in the last step in Algorithm 2.

Algorithm 2: An efficient implementation of MOSES

```
Data: Sequence of vectors $\{\mathbf{y}_t\}_{t\geq 1} \subset \mathbb{R}^d$, rank $r$, and block size $b \geq r$.
Result: Sequence $\{\widehat{\mathbf{S}}_{kb,r}, \widehat{\boldsymbol{\Sigma}}_{kb,r}, \widehat{\mathbf{V}}^T_{kb,r}\}_k$.
Function MOSES($\{\mathbf{y}_t\}_{t\geq 1} \subset \mathbb{R}^d, r, b,$ with_vt) is
    if $k$ is 1 then
        Form $\mathbf{B}_1 \in \mathbb{R}^{d\times b}$ by concatenating $\{\mathbf{y}_t\}_{t=1}^{b}$.
        /* set the first block */
        $[\widehat{\mathbf{U}}_{b,r}, \widehat{\boldsymbol{\Sigma}}_{b,r}, \widehat{\mathbf{V}}^T_{b,r}] \leftarrow \mathrm{SVDS}(\mathbf{B}_1,\ r),.$
    else
        Form $\mathbf{B}_k \in \mathbb{R}^{d\times b}$ by concatenating $\{\mathbf{y}_t\}_{t=(k-1)b+1}^{kb}$.
        /* project using previous estimate */
        $\dot{\mathbf{p}}_k = \widehat{\mathbf{U}}^T_{(k-1)b,r}\mathbf{B}_k \in \mathbb{R}^{r\times b}$.
        /* get the residual */
        $\widehat{\mathbf{z}}_k = \mathbf{U}_k - \widehat{\mathbf{U}}_{(k-1)b,r}\dot{\mathbf{p}}_k \in \mathbb{R}^{d\times b}$.
        /* use QR to get the partials, $\widehat{\mathbf{q}}_k \in \mathbb{R}^{d\times b}$ and $\mathbf{v}_k \in \mathbb{R}^{b\times b}$ */
        $[\widehat{\mathbf{q}}_k, \mathbf{v}_k] \leftarrow \mathrm{QR}(\widehat{\mathbf{z}}_k)$.
        /* get the $r$-SVDS to update, $\mathbf{u}_k, \widehat{\mathbf{v}}_k \in \mathbb{R}^{(r+b)\times r}$ and $\widehat{\boldsymbol{\Sigma}}_{kb,r} \in \mathbb{R}^{r\times r}$ */
        $\left[\mathbf{u}_k, \widehat{\boldsymbol{\Sigma}}_{kb,r}, \widehat{\mathbf{v}}_k^T\right] \leftarrow \mathrm{SVDS}\left(\begin{bmatrix} \widehat{\boldsymbol{\Sigma}}_{(k-1)b,r} & \dot{\mathbf{p}}_k \\ \mathbf{0}_{b\times r} & \mathbf{v}_k \end{bmatrix}, r\right)$.
        /* update the $\widehat{\mathbf{U}}_{kb,r} \in \mathbb{R}^{d\times r}$ estimate */
        $\widehat{\mathbf{U}}_{kb,r} = \left[\widehat{\mathbf{U}}_{(k-1)b,r} \quad \widehat{\mathbf{q}}_k\right]\mathbf{u}_k$.
        /* optionally, get the projected data, $\widehat{\mathbf{V}}^T_{kb,r} \in \mathbb{R}^{r\times kb}$ */
        if with_vt is true then
            $\widehat{\mathbf{V}}^T_{kb,r} = \begin{bmatrix} \widehat{\mathbf{V}}^T_{(k-1)b,r} & 0 \\ 0 & \mathbf{I}_b \end{bmatrix} \widehat{\mathbf{v}}_k^T$.
        else
            /* otherwise, flush, $\widehat{\mathbf{v}}_k \in \mathbb{R}^{r\times b}$ */
            $\widehat{\mathbf{V}}^T_{kb,r} = \widehat{\mathbf{v}}_k^T$.
        end
    end
end
```

It has to be noted that while Algorithm 2 is as discussed optimal, in practical implementations it is bounded by the quality of the auxiliary functions that are used internally. Concretely, in most implementations we believe that matrix operations are trivial and most packages have well defined routines. However, SVDS and QR implementations matter a lot and have high variability depending on the operating system, language, and packages used. To provide more information about potential implementations, we will now describe the computational complexity of the these functions as is described by the packages we used for our implementation, namely `MATLAB`. Note that these methods are employed one per block, thus these operate in $\mathbf{B} \in \mathbb{R}^{d\times b}$ as such we will report the complexity of these methods using the dimensions of each block. The `svds` function used a variant of the Lanczos bidiagonalization algorithm with partial reorthogonalization (BPRO) algorithm [111, 9], which has a storage complexity of $\mathcal{O}(r(d+b)) = \mathcal{O}(r(d+r))$ and requires $\mathcal{O}(r^2(d+b)) = \mathcal{O}(r^2(d+r))$ flops. As for QR, `MATLAB` does not provide any publicly available implementation details, however most certainly they use variants of the QR implementation used in `LAPACK`. In particular to provide a reference point one popular way of computing the QR decomposition is by employing Householder reflections. Its storage requirement is $\mathcal{O}(db) = \mathcal{O}(dr)$ and has a computational complexity of $\mathcal{O}(3db^2 - \frac{2}{3}db^3) = \mathcal{O}(3dr^2 - \frac{2}{3}dr^3)$ [70]. Ignoring the constants and only showing the asymptotically dominant factor, the final computational complexity for QR is $\mathcal{O}(dr^2)$. As we can see, in both instances the dominant factor is the ambient dimension d as asymptotically our assumption is that $d >> b \approx r$.

3.3 Optimisation Viewpoint

MOSES has a natural interpretation as an approximate solver for the non-convex optimisation program underlying PCA, which serves as its primary motivation. More specifically, recall that r leading principal components of $\mathbf{Y}_\tau$ are obtained by solving the non-convex program

$$\min_{\mathcal{U} \in \mathbb{G}(d,r)} \|\mathbf{Y}_\tau - \mathbf{P}_{\mathcal{U}} \mathbf{Y}_\tau\|_F^2 , \tag{3.13}$$

where the minimisation is over the Grassmannian $\mathbb{G}(d,r)$, the set of all r-dimensional subspaces in $\mathbb{R}^d$. Above, $\mathbf{P}_{\mathcal{U}} \in \mathbb{R}^{d\times d}$ is the orthogonal projection onto the subspace $\mathcal{U}$. By our problem formulation as per Section 3.2, note that,

$$\begin{aligned} \mathbf{Y}_\tau &= \begin{bmatrix} \mathbf{y}_1 & \mathbf{y}_2 & \cdots & \mathbf{y}_\tau \end{bmatrix} \qquad \text{(see eq. (3.1))} \\ &= \begin{bmatrix} \mathbf{B}_1 & \mathbf{B}_1 & \cdots & \mathbf{B}_K \end{bmatrix} \in \mathbb{R}^{d\times\tau}, \end{aligned} \tag{3.14}$$

where $\{\mathbf{B}_k\}_{k=1}^K$ are the incoming blocks of data. This allows us to rewrite Program (3.13) as

$$\begin{aligned} \min_{\mathcal{U} \in \mathbb{G}(d,r)} \|\mathbf{Y}_T - \mathbf{P}_{\mathcal{U}} \mathbf{Y}_\tau\|_F^2 &= \min_{\mathcal{U} \in \mathbb{G}(d,r)} \sum_{k=1}^{K} \|\mathbf{B}_k - \mathbf{P}_{\mathcal{U}} \mathbf{B}_k\|_F^2 \qquad \text{(see (3.14))} \\ &= \begin{cases} \min \sum_{k=1}^K \|\mathbf{B}_k - \mathbf{P}_{\mathcal{U}_K} \cdots \mathbf{P}_{\mathcal{U}_k} \mathbf{B}_k\|_F^2 \\ \mathcal{U}_1 = \mathcal{U}_2 = \cdots = \mathcal{U}_K, \end{cases} \end{aligned} \tag{3.15}$$

where the last minimisation above is over all identical subspaces $\{\mathcal{U}_k\}_{k=1}^K \subset \mathbb{G}(d,r)$. Our strategy is to make a sequence of approximations to the program in the last line above. In the first approximation, we only keep the first summand in the last line of (3.15). That is, our first approximation reads as

$$\begin{aligned} \begin{cases} \min \sum_{k=1}^K \|\mathbf{B}_k - \mathbf{P}_{\mathcal{U}_K} \cdots \mathbf{P}_{\mathcal{U}_k} \mathbf{B}_k\|_F^2 \\ \mathcal{U}_1 = \mathcal{U}_2 = \cdots = \mathcal{U}_K \end{cases} &\geq \begin{cases} \min \|\mathbf{B}_1 - \mathbf{P}_{\mathcal{U}_K} \cdots \mathbf{P}_{\mathcal{U}_1} \mathbf{B}_1\|_F^2 \\ \mathcal{U}_1 = \mathcal{U}_2 = \cdots = \mathcal{U}_K \end{cases} \\ &= \min_{\mathcal{U} \in \mathbb{G}(d,r)} \|\mathbf{B}_1 - \mathbf{P}_{\mathcal{U}} \mathbf{B}_1\|_F^2 , \end{aligned} \tag{3.16}$$

where the second line above follows by setting $\mathcal{U} = \mathcal{U}_1 = \cdots = \mathcal{U}_K$. Let the candidate subspace $\widehat{\mathcal{S}}_{b,r}$ be a minimiser of the program in the last line above. Note that $\widehat{\mathcal{S}}_{b,r}$ simply spans r leading principal components of the first block in the sequence $\mathbf{B}_1$, akin to Program (3.13). This indeed coincides with the output of MOSES in the first iteration. This because what both MOSES and SVD do, is practically identical for the first block as is shown below,

$$\begin{aligned}\widehat{\mathbf{Y}}_{b,r} &= \mathrm{SVD}(\mathbf{B}_1, r) \qquad \text{(see (3.8))}\\ &= \mathbf{P}_{\widehat{\mathcal{S}}_{b,r}} \mathbf{y}_1. \qquad \text{(similar to the second line of (3.7))}\end{aligned} \tag{3.17}$$

However, we must now consider the next approximation for all iterates. To do so, we keep two of the summands in the last line of (3.15), namely that the quantity M,

$$M = \begin{cases} \min \sum_{k=1}^{K} \|\mathbf{B}_k - \mathbf{P}_{\mathcal{U}_K} \cdots \mathbf{P}_{\mathcal{U}_k} \mathbf{B}_k\|_F^2 \\ \mathcal{U}_1 = \mathcal{U}_2 = \cdots = \mathcal{U}_K \end{cases}$$

is *greater or equal* than the following,

$$M \geq \begin{cases} \min \|\mathbf{B}_1 - \mathbf{P}_{\mathcal{U}_K} \cdots \mathbf{P}_{\mathcal{U}_1} \mathbf{B}_1\|_F^2 + \|\mathbf{B}_2 - \mathbf{P}_{\mathcal{U}_K} \cdots \mathbf{P}_{\mathcal{U}_2} \mathbf{B}_2\|_F^2 \\ \mathcal{U}_1 = \mathcal{U}_2 = \cdots = \mathcal{U}_K, \end{cases}$$

and then we substitute $\mathcal{U}_1 = \widehat{\mathcal{S}}_{b,r}$ above to arrive at the new program,

$$\begin{aligned}&\begin{cases} \min \ \|\mathbf{B}_1 - \mathbf{P}_{\mathcal{U}_K} \cdots \mathbf{P}_{\mathcal{U}_2} \mathbf{P}_{\widehat{\mathcal{S}}_{b,r}} \mathbf{B}_1\|_F^2 + \|\mathbf{B}_2 - \mathbf{P}_{\mathcal{U}_K} \cdots \mathbf{P}_{\mathcal{U}_2} \mathbf{B}_2\|_F^2 \\ \mathcal{U}_2 = \mathcal{U}_3 = \cdots = \mathcal{U}_K \end{cases}\\ &= \min_{\mathcal{U} \in \mathbb{G}(d,r)} \ \|\mathbf{B}_1 - \mathbf{P}_{\mathcal{U}} \mathbf{P}_{\widehat{\mathcal{S}}_{b,r}} \mathbf{B}_1\|_F^2 + \|\mathbf{B}_2 - \mathbf{P}_{\mathcal{U}} \mathbf{B}_2\|_F^2,\end{aligned} \tag{3.18}$$

where the second program above follows by setting $\mathcal{U} = \mathcal{U}_2 = \cdots = \mathcal{U}_K$. Provided this, then we can rewrite the above program as follows,

$$\begin{aligned}&\min_{\mathcal{U} \in \mathbb{G}(d,r)} \ \|\mathbf{B}_1 - \mathbf{P}_{\mathcal{U}} \mathbf{P}_{\widehat{\mathcal{S}}_{b,r}} \mathbf{B}_1\|_F^2 + \|\mathbf{B}_2 - \mathbf{P}_{\mathcal{U}} \mathbf{B}_2\|_F^2\\ &= \min_{\mathcal{U} \in \mathbb{G}(d,r)} \ \left\| \begin{bmatrix} \mathbf{B}_1 - \mathbf{P}_{\mathcal{U}} \mathbf{P}_{\widehat{\mathcal{S}}_{b,r}} \mathbf{B}_1 & \mathbf{B}_2 - \mathbf{P}_{\mathcal{U}} \mathbf{B}_2 \end{bmatrix} \right\|_F^2\\ &= \min_{\mathcal{U} \in \mathbb{G}(d,r)} \ \left\| \begin{bmatrix} \mathbf{P}_{\widehat{\mathcal{S}}_{b,r}^{\perp}} \mathbf{B}_1 & \mathbf{0}_{d \times b} \end{bmatrix} + \mathbf{P}_{\mathcal{U}^{\perp}} \begin{bmatrix} \mathbf{P}_{\widehat{\mathcal{S}}_{b,r}} \mathbf{B}_1 & \mathbf{B}_2 \end{bmatrix} \right\|_F^2\\ &= \|\mathbf{P}_{\widehat{\mathcal{S}}_{b,r}^{\perp}} \mathbf{B}_1\|_F^2 + \min_{\mathcal{U} \in \mathbb{G}(d,r)} \ \left\| \mathbf{P}_{\mathcal{U}^{\perp}} \begin{bmatrix} \mathbf{P}_{\widehat{\mathcal{S}}_{b,r}} \mathbf{B}_1 & \mathbf{B}_2 \end{bmatrix} \right\|_F^2 \qquad \text{(see the text below)}\\ &= \|\mathbf{P}_{\widehat{\mathcal{S}}_{b,r}^{\perp}} \mathbf{B}_1\|_F^2 + \min_{\mathcal{U} \in \mathbb{G}(d,r)} \ \left\| \mathbf{P}_{\mathcal{U}^{\perp}} \begin{bmatrix} \widehat{\mathbf{Y}}_{b,r} & \mathbf{B}_2 \end{bmatrix} \right\|_F^2, \qquad \text{(see (3.17))}\end{aligned} \tag{3.19}$$

and let $\widehat{\mathcal{S}}_{2b,r}$ be a minimiser of the last program above. Above, $\perp$ indicates the orthogonal complement of a subspace. The second to last line above follows because $\widehat{\mathcal{S}}_{2b,r}$ is always within the column span of $[\mathbf{P}_{\widehat{\mathcal{S}}_{b,r}}\mathbf{B}_1 \ \ \mathbf{B}_2]$. Note also that $\widehat{\mathcal{S}}_{2b,r}$ is the span of leading r principal components of the matrix $[\widehat{\mathbf{Y}}_{1,r} \ \ \mathbf{B}_2]$, similar to Program (3.13). This again coincides with the output of MOSES in the second iteration, because

$$\begin{aligned}\widehat{\mathbf{Y}}_{2b,r} &= \text{SVD}\left(\left[\begin{array}{cc}\widehat{\mathbf{Y}}_{b,r} & \mathbf{B}_2\end{array}\right], r\right) && \text{(see (3.8))}\\ &= \mathbf{P}_{\widehat{\mathcal{S}}_{2b,r}}\left[\begin{array}{cc}\widehat{\mathbf{Y}}_{b,r} & \mathbf{B}_2\end{array}\right]. && \text{(similar to the second line of (3.7))} \qquad (3.20)\end{aligned}$$

Continuing this procedure precisely produces the iterates of MOSES. Therefore we might interpret MOSES as an optimisation algorithm that can also be an approximate solver of Program (3.13) by making a sequence of approximations, namely one for each complete block $\mathbf{B}_k$ of input processed.

3.4 Performance of MOSES

In this section, we study the performance of MOSES in a stochastic setup. Consider the probability space $(\mathbb{R}^d, \mathcal{B}, \mu)$, where $\mathcal{B}$ is the Borel σ-algebra and μ is an *unknown* probability measure with zero mean, namely $\int_{\mathbb{R}^d} \mathbf{y}\, \mu(d\mathbf{y}) = 0$. We are interested in finding an r-dimensional subspace $\mathcal{U}$ that best approximates the probability measure μ. That is, with $\mathbf{y}$ drawn from this probability space, we are interested in finding an r-dimensional subspace $\mathcal{U}$ that minimises the *population risk*:

$$\begin{aligned}\min_{\mathcal{U}\in\mathbb{G}(d,r)} \mathbb{E}\,\|\mathbf{y}-\mathbf{P}_{\mathcal{U}}\mathbf{y}\|_2^2 &= \min_{\mathcal{U}\in\mathbb{G}(d,r)} \int_{\mathbb{R}^d} \|\mathbf{y}-\mathbf{P}_{\mathcal{U}}\mathbf{y}\|_F^2 \ \mu(d\mathbf{y})\\ &=: \rho_r^2(\mu). \qquad (3.21)\end{aligned}$$

Since μ is unknown, we cannot directly solve Program (3.21) but suppose that instead we have access to the *training samples* $\{\mathbf{y}_t\}_{t=1}^{\tau} \subset \mathbb{R}^d$ drawn independently from this probability space $(\mathbb{R}^d, \mathcal{B}, \mu)$. Let us form $\mathbf{Y}_\tau \in \mathbb{R}^{d\times\tau}$ by concatenating these vectors, see (3.1). In lieu of Program (3.21), we then replace the population risk above with the *empirical risk*:

$$\begin{aligned}\min_{\mathcal{U}\in\mathbb{G}(d,r)} \frac{1}{\tau}\sum_{t=1}^{\tau}\|\mathbf{y}_t-\mathbf{P}_{\mathcal{U}}\mathbf{y}_t\|_2^2 &= \min_{\mathcal{U}\in\mathbb{G}(d,r)} \frac{1}{\tau}\|\mathbf{Y}_\tau-\mathbf{P}_{\mathcal{U}}\mathbf{Y}_\tau\|_F^2 && \text{(see (3.1))}\\ &= \frac{1}{\tau}\|\mathbf{Y}_\tau-\mathbf{P}_{\mathcal{S}_{\tau,r}}\mathbf{Y}_\tau\|_F^2 && \text{(see the text below)}\\ &= \frac{1}{\tau}\|\mathbf{Y}_\tau-\mathbf{Y}_{\tau,r}\|_F^2 && (\mathbf{Y}_{\tau,r} = \text{SVD}_r(\mathbf{Y}_\tau))\\ &=: \frac{\rho_r^2(\mathbf{Y}_\tau)}{\tau}. && \text{(see (3.5))} \qquad (3.22)\end{aligned}$$

Here, $\mathcal{S}_{\tau,r} \in \mathbb{G}(d,r)$ is a minimiser of the above program with orthonormal basis $\mathbf{S}_{\tau,r} \in \mathbb{R}^{d\times r}$. Note that $\mathbf{S}_{\tau,r}$ consists of r leading principal components of $\mathbf{Y}_\tau$, namely it contains leading r *left* singular vectors of $\mathbf{Y}_\tau$ as per theorem 2.3.1. Given its principal components, we can then reduce the dimension of the data matrix $\mathbf{Y}_\tau \in \mathbb{R}^{d\times\tau}$ from d to r by computing $\mathbf{S}_{\tau,r}^T\mathbf{Y}_\tau \in \mathbb{R}^{r\times\tau}$. Note also that $\mathcal{S}_{\tau,r}$ is a possibly sub-optimal choice in Program (3.21), namely,

$$\mathbb{E}_{\mathbf{y}}\|\mathbf{y} - \mathbf{P}_{\mathcal{S}_{\tau,r}}\mathbf{y}\|_2^2 \geq \rho_r^2(\mu). \qquad \text{(see (3.21))} \tag{3.23}$$

But one would hope that $\mathcal{S}_\tau$ still nearly minimises Program (3.21), in the sense that

$$\mathbb{E}_{\mathbf{y}}\|\mathbf{y} - \mathbf{P}_{\mathcal{S}_{\tau,r}}\mathbf{y}\|_2^2 \approx \rho_r^2(\mu), \tag{3.24}$$

with high probability over the choice of training data $\{\mathbf{y}_t\}_{t=1}^{\tau}$. That is, one would hope that the *generalisation error* of Program (3.22) is small. Above, $\mathbb{E}_{\mathbf{y}}$ stands for expectation over $\mathbf{y}$, so that the left-hand side of (3.24) is still a random variable because of its dependence on $\mathcal{S}_\tau$.

If the training data $\{\mathbf{y}_t\}_{t=1}^{\tau}$ is presented to us sequentially and little storage is available, we cannot hope nor is practical, to directly solve Program (3.22). Moreover, even if we have enough storage, we might not want to wait for all the data to arrive before solving Program (3.22). In this streaming scenario, we may apply MOSES to obtain the (rank-r) output $\widehat{\mathbf{Y}}_{\tau,r}$. We then set

$$\widehat{\mathcal{S}}_{\tau,r} = \operatorname{span}(\widehat{\mathbf{Y}}_{\tau,r}), \tag{3.25}$$

with orthonormal basis $\widehat{\mathbf{S}}_\tau \in \mathbb{R}^{d\times r}$. Note that $\widehat{\mathbf{S}}_\tau$ is MOSES' estimate of leading r principal components of the data matrix $\mathbf{Y}_\tau$ and is possibly suboptimal in the sense that,

$$\|\mathbf{Y}_\tau - \widehat{\mathbf{Y}}_{\tau,r}\|_F \geq \rho_r(\mathbf{Y}_\tau). \qquad \text{(see (3.22))} \tag{3.26}$$

But we would still hope that the output $\widehat{\mathbf{Y}}_{\tau,r}$ of MOSES is a nearly optimal choice in Program (3.22), in the sense that,

$$\|\mathbf{Y}_\tau - \widehat{\mathbf{Y}}_{\tau,r}\|_F \approx \rho_r(\mathbf{Y}_\tau), \tag{3.27}$$

with high probability over the choice of $\{\mathbf{y}_t\}_{t=1}^{\tau}$. Moreover, as with (3.24), $\widehat{\mathcal{S}}_{\tau,r}$ is again a possibly sub-optimal choice for Program (3.21), and yet we would hope that,

$$\mathbb{E}_{\mathbf{y}}\|\mathbf{y} - \mathbf{P}_{\widehat{\mathcal{S}}_{\tau,r}}\mathbf{y}\|_2^2 \approx \rho_r^2(\mu), \tag{3.28}$$

with high probability over the choice of $\{\mathbf{y}_t\}_{t=1}^{\tau}$.

To summarise the discussion above, the key questions are whether (3.24, 3.27, 3.28) hold. Let us answer these questions for the important case where μ is a zero-mean Gaussian probability measure with covariance matrix $\mathbf{\Xi} \in \mathbb{R}^{d\times d}$. For this choice of μ in (3.21), it is not difficult to verify that

$$\rho_r^2(\mu) = \sum_{i=r+1}^{d} \lambda_i(\mathbf{\Xi}), \tag{3.29}$$

where $\lambda_1(\mathbf{\Xi}) \ge \lambda_2(\mathbf{\Xi}) \ge \cdots$ are the eigenvalues of the covariance matrix $\mathbf{\Xi}$. From now on, let us use the shorthand,

$$\rho_r = \rho_r(\mu), \qquad \lambda_i = \lambda_i(\mathbf{\Xi}), \qquad i \in [1:d].$$

For our choice of μ above as a Gaussian measure with covariance matrix $\mathbf{\Xi} \in \mathbb{R}^{d\times d}$, one can use standard tools from the covariance estimation literature to show that (3.24) holds when τ is sufficiently large, the proof of which is included in Appendix A of the supplementary material for completeness [57, 179].

Lemma 3.4.1. *Suppose that $\{\mathbf{y}_t\}_{t=1}^{\tau} \subset \mathbb{R}^d$ are drawn independently from a zero-mean Gaussian measure μ with covariance matrix $\mathbf{\Xi} \in \mathbb{R}^{d\times d}$ and form $\mathbf{Y}_\tau \in \mathbb{R}^{d\times\tau}$ by concatenating these vectors, see (3.1). Suppose also that $\mathcal{S}_{\tau,r} \in \mathbb{G}(d,r)$ is the span of leading r principal components of $\mathbf{Y}_\tau$. For $1 \le \alpha \le \sqrt{\tau/\log(\tau)}$, it then holds that*

$$\frac{\rho_r^2(\mathbf{Y}_\tau)}{\tau} \lesssim \alpha\rho_r^2, \tag{3.30}$$

$$\mathbb{E}_{\mathbf{y}}\|\mathbf{y} - \mathbf{P}_{\mathcal{S}_{\tau,r}}\mathbf{y}\|_2^2 \lesssim \alpha\rho_r^2 + \alpha(d-r)\lambda_1\sqrt{\frac{\log(\tau)}{\tau}}, \tag{3.31}$$

except with a probability of at most $\tau^{-C\alpha^2}$, where C is a universal constant. In presentation, we use $\lesssim$ for suppressing any universal constants for a more tidy presentation.

In words, (3.31) states that the generalisation error of Program (3.22) is sufficiently small, hence (3.24) holds. Indeed, as τ increases, the right-hand side of (3.31) approaches the residual squared of $\mathbf{Y}_\tau/\sqrt{\tau}$, which is bounded by $C\alpha\rho_r^2$. In particular, (3.24) holds when $\alpha = \mathcal{O}(1)$ and τ is sufficiently large. As the dimension r of the subspace that we fit to the data approaches the ambient dimension d, note that the right-hand side of (3.31) vanishes.

In contrast, MOSES operates in a streaming regime, where we are unable to fully store the data matrix $\mathbf{Y}_\tau$ and consequently unable to find its principal components directly. That is, we cannot directly solve Program (3.22) in the streaming regime. However, Theorem 3.4.2 below states that MOSES approximately solves Program (3.22). Put succinctly, MOSES approximately estimates the leading principal components of $\mathbf{Y}_\tau$ *and* reduces the dimension of data from d to r with only $\mathcal{O}(r(d+\tau))$ bits of memory. These can provide significant improvements, as solving Program (3.22) using "offline" truncated SVD would incur a storage cost in the order of $\mathcal{O}(d\tau)$ bits. Moreover, as we saw previously MOSES approximately solves Program (3.21). In other words, MOSES satisfies *both* (3.27, 3.28). These statements are made concrete below and proved in Appendix A.2 of the supplementary material.

Theorem 3.4.2. (Performance of MOSES) *Suppose that* $\{\mathbf{y}_t\}_{t=1}^{\tau} \subset \mathbb{R}^d$ *are drawn independently from a zero-mean Gaussian probability measure* μ *with covariance matrix* $\mathbf{\Xi} \in \mathbb{R}^{d\times d}$*. Moreover, let us define,*

$$\kappa_r^2 := \frac{\lambda_1}{\lambda_r}, \qquad \rho_r^2 = \sum_{i=r+1}^{d} \lambda_i, \qquad \eta_r := \kappa_r + \sqrt{\frac{2\alpha\rho_r^2}{p^{\frac{1}{3}}\lambda_r}}, \tag{3.32}$$

where $\lambda_1 \geq \lambda_2 \geq \cdots$ *are the eigenvalues of* $\mathbf{\Xi}$*. Let* $\widehat{\mathcal{S}}_{\tau,r} = \operatorname{span}(\widehat{\mathbf{Y}}_{\tau,r})$ *be the span of the output of* MOSES*, see (3.25). Then, for tuning parameters* $1 \leq \alpha \leq \sqrt{\tau/\log(\tau)}$ *and* $p > 1$*, it holds that*

$$\frac{\|\mathbf{Y}_\tau - \widehat{\mathbf{Y}}_{\tau,r}\|_F^2}{\tau} \lesssim \frac{\alpha p^{\frac{1}{3}} 4^{p\eta_r^2}}{(p^{\frac{1}{3}}-1)^2} \cdot \min\left(\kappa_r^2\rho_r^2, r\lambda_1 + \rho_r^2\right)\left(\frac{T}{p\eta_r^2 b}\right)^{p\eta_r^2-1}, \tag{3.33}$$

$$\mathbb{E}_{\mathbf{y}}\|\mathbf{y} - \mathbf{P}_{\widehat{\mathcal{S}}_{\tau,r}}\mathbf{y}\|_2^2 \lesssim \frac{\alpha p^{\frac{1}{3}} 4^{p\eta_r^2}}{(p^{\frac{1}{3}}-1)^2} \cdot \min\left(\kappa_r^2\rho_r^2, r\lambda_1 + \rho_r^2\right)\left(\frac{\tau}{\mathbf{p}\eta_r^2 b}\right)^{p\eta_r^2-1} + \alpha(d-r)\lambda_1\sqrt{\frac{\log(\tau)}{\tau}}, \tag{3.34}$$

except with a probability of at most $\tau^{-C\alpha^2} + e^{-C\alpha r}$ *and provided that*

$$b \geq \frac{\alpha p^{\frac{1}{3}} r}{(p^{\frac{1}{6}}-1)^2}, \qquad b \geq C\alpha r, \qquad \tau \geq p\eta_r^2 b. \tag{3.35}$$

The requirement $\tau \geq p\eta_r^2 b$ in the last line above is only for a compact bound for (3.33, 3.34) and is not necessary. A general expression for arbitrary τ is given in the proof, see (A.22). A few remarks about Theorem 3.4.2 are in order.

Discussion of Theorem 3.4.2

On the one hand, Theorem 3.4.2 and specifically (3.33) state that (3.27) holds under certain conditions. That is, MOSES approximately solves Program (3.13) or, in other words, MOSES successfully performs streaming (linear) dimensionality reduction. Indeed, (3.33) loosely speaking states that $\|\mathbf{Y}_\tau - \widehat{\mathbf{Y}}_{\tau,r}\|_F^2$ scales with $\rho_r^2\tau^{p\eta_r^2}/b^{p\eta_r^2-1}$, whereas the residual squared of $\mathbf{Y}_\tau$ scales with $\rho_r^2\tau$ by (3.30). That is,

$$\begin{aligned}\|\mathbf{Y}_\tau - \widehat{\mathbf{Y}}_{\tau,r}\|_F^2 &\propto \left(\frac{\tau}{b}\right)^{p\eta_r^2-1}\rho_r^2(\mathbf{Y}_\tau)\\ &= \left(\frac{\tau}{b}\right)^{p\eta_r^2-1}\|\mathbf{Y}_\tau - \mathbf{Y}_{\tau,r}\|_F^2, \qquad \text{(see (3.22))}\end{aligned} \tag{3.36}$$

after ignoring the less important terms. In words, applying offline truncated SVD to $\mathbf{Y}_\tau$, which is assumed to have *unlimited resources*, outperforms the streaming MOSES by only a polynomial factor in the order of τ/b.

- This polynomial factor can be negligible when the covariance matrix $\mathbf{\Xi}$ of the Gaussian data distribution μ is well-conditioned ($\kappa_r = \mathcal{O}(1)$) and has a small residual ($\rho_r^2 = \mathcal{O}(\lambda_r)$). In this case we will have $\eta_r = \mathcal{O}(1)$, see (3.32). With $p = \mathcal{O}(1)$, (3.36) then reads as

  $$\|\mathbf{Y}_\tau - \widehat{\mathbf{Y}}_{\tau,r}\|_F^2 \propto \left(\frac{\tau}{b}\right)^{\mathcal{O}(1)} \rho_r^2(\mathbf{Y}_\tau).$$

  In particular, in the case that the covariance matrix of the data distribution is rank-r, we have by (3.29) that $\rho_r = 0$. Consequently, (3.36) reads as $\widehat{\mathbf{Y}}_{\tau,r} = \mathbf{Y}_{\tau,r} = \mathbf{Y}_\tau$, namely the outputs of offline truncated SVD and MOSES coincide. In fact, MOSES correctly identifies the r-dimensional span of incoming data after processing the very first block.

- At the cost of a larger multiplicative factor on the right-hand side of (3.33), one might reduce the power of τ in the first term of (3.33) by choosing p closer to one.

- The dependence of our results on the condition number κ_r and residual ρ_r is very likely *not* an artefact of the proof techniques, see (3.32). Indeed, when $\kappa_r \gg 1$, certain directions are less often observed in the incoming data vectors $\{\mathbf{y}_t\}_{t=1}^T$, which tilts the estimate of MOSES towards the dominant principal components corresponding to the very large singular values. Moreover, if $\rho_r \gg 1$, there are too many significant principal components and MOSES can at most "remember" r of them from its previous iteration. In this scenario, approximating the incoming data with a rank-r subspace is not a good idea in the first place, in the sense that the residual $\rho_r(\mathbf{Y}_\tau)$ corresponding to the offline truncated SVD will be large too. An obvious solution to this would be perhaps to increase the dimension r of the subspace that we wish to fit to the incoming data $\{\mathbf{y}_t\}_{t=1}^T$.

- Note also that, as b increases, performance of MOSES naturally approaches that of the offline truncated SVD. In particular, when $b = \tau$, MOSES processes all of the data at once and practically reduces to offline truncated SVD. This trend is somewhat imperfectly reflected in (3.33).

On the other hand, Theorem 3.4.2 and specifically (3.34) state that (3.28) holds under certain conditions. Indeed, for sufficiently large τ, (3.34) loosely speaking reads as follows,

$$\begin{aligned}\mathbb{E}_{\mathbf{y}}\|\mathbf{y} - \mathbf{P}_{\widehat{\mathcal{S}}_{\tau,r}}\mathbf{y}\|_2^2 &\propto \left(\frac{\tau}{b}\right)^{p\eta_r^2-1} \rho_r^2 \\ &= \left(\frac{\tau}{b}\right)^{p\eta_r^2-1} \min_{\mathcal{U}\in\mathbb{G}(d,r)} \mathbb{E}\,\|\mathbf{y} - \mathbf{P}_{\mathcal{U}}\mathbf{y}\|_2^2. \qquad \text{(see Program (3.21))} \qquad (3.37)\end{aligned}$$

That is, the output of MOSES is sub-optimal for Program (3.21) by a only polynomial factor in τ. This factor can be negligible in the case that the covariance matrix Ξ of the data distribution μ is well-conditioned and has a small residual, see the discussion above.

Spiked covariance model

A popular model in the statistics literature is the spiked covariance model, where the data vectors $\{\mathbf{y}_t\}_{t=1}^{\tau}$ are drawn from a distribution with a covariance matrix Ξ. Under this model, Ξ is a low-rank perturbation of the identity matrix [99, 179], namely $\lambda_1(\Xi) = \cdots = \lambda_r(\Xi) = \lambda$ and $\lambda_{r+1}(\Xi) = \cdots = \lambda_d(\Xi) = 1$. Lemma 3.4.1 in this case reads as,

$$\mathbb{E}\|\mathbf{y} - \mathbf{P}_{\mathcal{S}_{\tau,r}}\mathbf{y}\|_2^2 \propto (d-r) + (d-r)\lambda\sqrt{\frac{\log(\tau)}{\tau}}, \tag{3.38}$$

where $\mathcal{S}_{\tau,r}$ spans leading r principal components of the data matrix $\mathbf{Y}_\tau$. In contrast, Theorem 3.4.2 roughly speaking states that,

$$\mathbb{E}\|\mathbf{y} - \mathbf{P}_{\widehat{\mathcal{S}}_{\tau,r}}\mathbf{y}\|_2^2 \propto (d-r)\left(\frac{\tau\lambda}{bd}\right)^{\frac{d}{\lambda}} + (d-r)\lambda\sqrt{\frac{\log(\tau)}{\tau}}, \tag{3.39}$$

where $\widehat{\mathcal{S}}_{\tau,r}$ spans the output of MOSES. When $\lambda \gtrsim d\log(\tau/b) = d\log(K)$ in particular, it follows that the error bounds in (3.38, 3.39) are of the same order. That is, under the spiked covariance model, MOSES for streaming truncated SVD matches the performance of "offline" truncated SVD, provided that the underlying distribution has a sufficiently large spectral gap. However, in practice, (3.39) is often a conservative bound.

Proof strategy

Starting with (3.34), the proof of Theorem 3.4.2 in Appendix A.2 of the supplementary material breaks down the error associated with MOSES into two primary components, namely

$$\mathbb{E}_{\mathbf{y}}\|\mathbf{y} - \mathbf{P}_{\widehat{\mathcal{S}}_{\tau,r}}\mathbf{y}\|_2 \le \frac{1}{\tau}\|\mathbf{Y}_\tau - \mathbf{P}_{\widehat{\mathcal{S}}_{\tau,r}}\mathbf{Y}_\tau\|_F^2 + \left|\frac{1}{\tau}\|\mathbf{Y}_\tau - \mathbf{P}_{\widehat{\mathcal{S}}_{\tau,r}}\mathbf{Y}_\tau\|_F^2 - \mathbb{E}_{\mathbf{y}}\|\mathbf{y} - \mathbf{P}_{\widehat{\mathcal{S}}_{\tau,r}}\mathbf{y}\|_2^2\right|. \tag{3.40}$$

That is, we bound the population risk with the empirical risk. We control the empirical risk in the first part of the proof by noting that,

$$\begin{aligned}\|\mathbf{Y}_\tau - \mathbf{P}_{\widehat{\mathcal{S}}_{\tau,r}}\mathbf{Y}_\tau\|_F &= \|\mathbf{P}_{\widehat{\mathcal{S}}_{\tau,r}^{\perp}}\mathbf{Y}_\tau\|_F \\ &= \|\mathbf{P}_{\widehat{\mathcal{S}}_{\tau,r}^{\perp}}(\mathbf{Y}_\tau - \widehat{\mathbf{Y}}_{\tau,r})\|_\tau \qquad \text{(see (3.25))} \\ &\le \|\mathbf{Y}_\tau - \widehat{\mathbf{Y}}_{\tau,r}\|_F,\end{aligned} \tag{3.41}$$

where the last line gauges how well the output of MOSES approximates the data matrix $\mathbf{Y}_\tau$, see (3.33). We then bound $\|\mathbf{Y}_\tau - \widehat{\mathbf{Y}}_{\tau,r}\|_F$ in two steps: As it is common in these types of arguments, the first step finds a deterministic upper bound for this norm, which is then evaluated for our particular stochastic setup.

- The deterministic bound appears in Lemma A.2.1 and gives an upper bound for $\|\mathbf{Y}_\tau - \widehat{\mathbf{Y}}_{\tau,r}\|_F$ in terms of the overall "innovation". Loosely speaking, the innovation $\|\mathbf{P}_{\mathcal{S}^\perp_{(k-1)b,r}}\mathbf{y}_k\|_F$ at iteration k is the part of the new data block $\mathbf{y}_k$ that cannot be described by the leading r principal components of data arrived so far, which span the subspace $\mathcal{S}_{(k-1)b,r}$.
- The stochastic bound is given in Lemma A.2.2 and uses a tight perturbation result.

Our argument so far yields an upper bound on the empirical loss $\|\mathbf{Y}_\tau - \mathbf{P}_{\widehat{\mathcal{S}}_{\tau,r}}\mathbf{Y}_\tau\|_F$ that holds with high probability. In light of (3.40), it remains to control the following quantity,

$$\left|\frac{1}{\tau}\|\mathbf{Y}_\tau - \mathbf{P}_{\widehat{\mathcal{S}}_{\tau,r}}\mathbf{Y}_T\|_F^2 - \mathbb{E}_{\mathbf{y}}\|\mathbf{y} - \mathbf{P}_{\widehat{\mathcal{S}}_{\tau,r}}\mathbf{y}\|_2^2\right| = \frac{1}{\tau}\left|\|\mathbf{Y}_\tau - \mathbf{P}_{\widehat{\mathcal{S}}_{\tau,r}}\mathbf{Y}_\tau\|_F^2 - \mathbb{E}\|\mathbf{Y}_\tau - \mathbf{P}_{\widehat{\mathcal{S}}_{\tau,r}}\mathbf{Y}_\tau\|_F^2\right|$$
$$= \frac{1}{\tau}\left|\|\mathbf{P}_{\widehat{\mathcal{S}}^\perp_{\tau,r}}\mathbf{Y}_\tau\|_F^2 - \mathbb{E}\|\mathbf{P}_{\widehat{\mathcal{S}}^\perp_{\tau,r}}\mathbf{Y}_\tau\|_F^2\right| \qquad (3.42)$$

with a standard large deviation bound.

Applicability on other stochastic models

While our current results were restricted to the Gaussian distribution, they extend easily and with minimal change to the larger class of subgaussian distributions. More concretely, to do so one can exploit the constructions we used for our proof from [180, 154] in order to prove this. Namely, they provide similar results for the more general class of subgaussian distributions which is only slightly different than the one we used for Gaussian distributions. Recall, that we arrived at (3.28) by not making any assumptions on the distribution of the probably measure μ. Thus by assuming that this probably measure μ is indeed subgaussian and following through the proof of Theorem 3.4.2 exploiting Theorem 5.58 in [180] under that assumption one can see that we arrive at a similar result, albeit with slightly different constraints as dictated by [180, 179] for subgaussian distributions. Hence, overall Theorem 3.4.2 would still hold even if the data-generating process changed from the original one with the constraint that the data-generating process would have to remain within the subgaussian class of distributions. Beyond subgaussian data models, Lemma A.2.1 is the key deterministic result, directly relating the MOSES error to the overall innovation. One might therefore control the overall innovation, namely the right-hand side of (A.19) in Lemma A.2.1, for any other stochastic model at hand.

3.5 Experimental Evaluation

In this section, we investigate the numerical performance of MOSES and compare it against competing algorithms, namely GROUSE [13], frequent directions (FD) [47, 120], and power method (PM) [130, 131], on both synthetic and real-world datasets. In all of our experiments, we reveal one by one the data vectors $\{\mathbf{y}_t\}_{t=1}^{\mathcal{T}} \subset \mathbb{R}^d$ and, for every t, wish to compute a rank-r truncated SVD of the data matrix arrived so far, namely $[\mathbf{y}_1, \cdots, \mathbf{y}_t]$. For the tests that use synthetic data, the vectors $\{\mathbf{y}_t\}_{t=1}^{\mathcal{T}}$ are drawn independently from a zero-mean Gaussian distribution with covariance matrix $\mathbf{\Xi} = \mathbf{S}\mathbf{\Lambda}\mathbf{S}^T$, where $\mathbf{S} \in \mathbb{R}^{d\times d}$ is a generic orthonormal basis obtained by orthogonalising a standard random Gaussian matrix. The entries of the diagonal matrix $\mathbf{\Lambda} \in \mathbb{R}^{d\times d}$, namely the eigenvalues of the covariance matrix $\mathbf{\Xi}$, are selected according to the power law: $\lambda_i = i^{-\alpha}$ for a given positive α. To be more succinct, where possible we use MATLAB's notation for specifying the value ranges in this section. Note that both MOSES and Power method updates its estimates after receiving each block of data. For the sake of an easier comparison with other algorithms (with different block sizes), we properly "interpolate" the outputs of all algorithms over time. All experiments were performed on a workstation using an AMD 3970X CPU with 32 cores, 256GB of 3600 MHz DDR4 RAM, and MATLAB R2021a (build 9.10.0.1602886). To foster reproducibility and dissemination of the contributions presented in this chapter we remark that the accompanying MATLAB code and the datasets used for the numerical evaluation are publicly available[1].

To assess the performance of MOSES, let $\mathbf{Y}_t = [\mathbf{y}_1, \cdots, \mathbf{y}_t] \in \mathbb{R}^{d\times t}$ be the data received by time t and let $\widehat{\mathbf{Y}}_{t,r}^m$ be the output of MOSES at time t. Then the error incurred by MOSES is as follows,

$$\frac{1}{t}\|\mathbf{Y}_t - \widehat{\mathbf{Y}}_{t,r}^m\|_F^2, \tag{3.43}$$

see Theorem 3.4.2. Recall from (3.5) that the above error is always worse (i.e. *larger*) than the residual of $\mathbf{Y}_t$, namely

$$\|\mathbf{Y}_t - \widehat{\mathbf{Y}}_{t,r}^m\|_F^2 \geq \|\mathbf{Y}_t - \mathbf{Y}_{t,r}\|_F^2 = \rho_r^2(\mathbf{Y}_t), \qquad \text{(see (3.5))} \tag{3.44}$$

where $\mathbf{Y}_{t,r} = \mathrm{SVD}_r(\mathbf{Y}_t)$ is a rank-r truncated SVD of $\mathbf{Y}_t$ and $\rho_r^2(\mathbf{Y}_t)$ is the corresponding residual.

Quantitative evaluation of MOSES.

To start, we would like to understand how MOSES behaves across different setups. Concretely, we would like to empirically quantify the error scaling when tweaking the ambient dimension, rank, and its block size. To so so, we set out to do various tests and report the results.

[1]https://github.com/andylamp/moses

Ambient dimension

On a synthetic dataset with $\alpha = 1$ and $\tau = 2000$, we first test MOSES by varying the ambient dimension as $d \in \{200 : 200 : 1200\}$, and setting the rank and block size to $r = 15$, $b = 2r = 30$. The average error over ten trials is reported in Figure 3.2a. Note that the error is increasing in d, see the discussion under spiked covariance model in Section 3.4.

Block size

On a synthetic dataset with $\alpha = 1$ and $\tau = 2000$, we test MOSES by setting the ambient dimension and rank to $d = 1200$, $r = 15$, and varying the block size as $b \in \{r : r : 15r\}$. The average error over ten trials is reported in Figure 3.2b. It is interesting to note that the MOSES is particularly robust against the choice of the block size and that, at the extreme case of $b = \tau$, error vanishes as MOSES reduces to be equal to the "offline" truncated SVD.

Rank

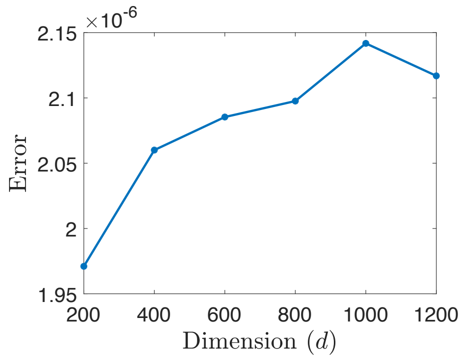
On a synthetic dataset with $\alpha = 1$ and $\tau = 2000$, we test MOSES by setting the ambient dimension and block size to $d = 1200$, $b = 2r$, and varying the rank as $r \in \{5 : 5 : 25\}$. The average error over ten trials is reported in Figure 3.2c. As expected, the error is decreasing in the dimension r of the subspace that we fit to the data and in fact, at the extreme case of $r = d$, there would be no error at all.

Comparisons against other methods

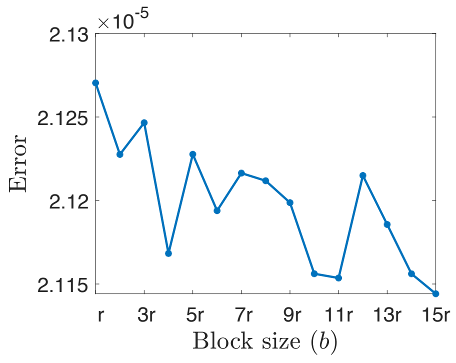
In this section, we compare MOSES against GROUSE [13], FD [47] and PM [130], as described in Section Section 3.6. These competing algorithms *only* estimate the principal components of the data, as opposed to MOSES which also projects the data onto these estimates. More specifically, let $\widehat{\mathcal{S}}^g_{t,r} \in \mathrm{G}(d,r)$ be the span of the output of GROUSE, $\widehat{\mathcal{S}}^f_{t,r} \in \mathrm{G}(d,r)$ be the span of the output of FD, and $\widehat{\mathcal{S}}^p_{t,r} \in \mathrm{G}(d,r)$ be the span of the output of PM. These algorithms then incur the errors,

$$\frac{1}{t}\|\mathbf{Y}_t - \mathbf{P}_{\widehat{\mathcal{S}}^g_{t,r}}\mathbf{Y}_t\|_F^2, \qquad \frac{1}{t}\|\mathbf{Y}_t - \mathbf{P}_{\widehat{\mathcal{S}}^f_{t,r}}\mathbf{Y}_t\|_F^2,$$
$$\frac{1}{t}\|\mathbf{Y}_t - \mathbf{P}_{\widehat{\mathcal{S}}^p_{t,r}}\mathbf{Y}_t\|_F^2, \tag{3.45}$$

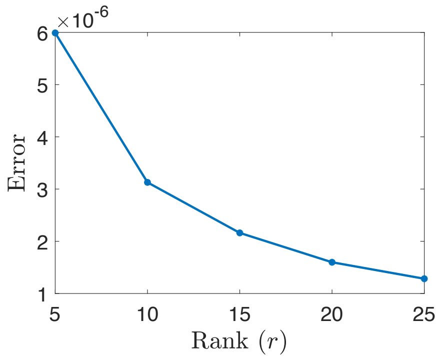
respectively. Above, $\mathbf{P}_{\mathcal{A}} \in \mathbb{R}^{d\times d}$ is the orthogonal projection onto the subspace $\mathcal{A}$. Even though robust FD [120] improves over FD in the quality of matrix sketching, since the subspaces produced by FD and robust FD coincide, there is no need here for computing a separate error for robust FD.



(a) varied d, for $r = 15$, $b = 2r$.



(b) varied b, for $n = 1.2$k, $r = 15$.



(c) varied r, for $n = 1.2$k, $b = 2r$.

Fig. 3.2 Performance of MOSES on synthetic datasets, see Section 3.5 for the details.

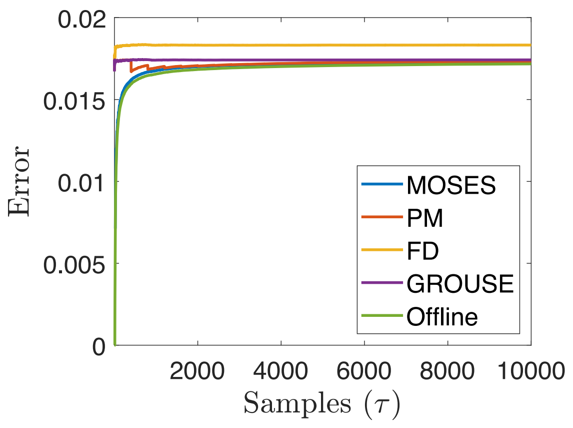
Comparisons on synthetic datasets.

On synthetic datasets with $\alpha \in \{0.01, 0.1, 0.5, 1\}$ and $\tau = 10000$, we compare MOSES against GROUSE, FD, and PM as well as the traditional offline SVD. More specifically, we set the ambient dimension to $d = 200$ and the rank to $r = 10$. For MOSES, the block size was set to $b = 2r$. For GROUSE and power method, we set the step size and block size to 2 and $2d = 400$, respectively, as these values seemed to produced the best results overall. Both GROUSE and power method were initialised randomly, as prescribed in [13, 131], whereas FD does not require any particular initialisation. For the implementation of the proposed algorithms, namely MOSES, FD, and PM we used MATLAB. Specifically, we based our FD implementation on the specific algorithm described in [66], a Python version of which can be found online[2]. PM was based on the algorithm described in [130] while GROUSE had already code publicly available[3], which we used. The average errors of all four algorithms over ten trials versus time is shown in Figure 3.3. Because of its large blocks size of $\mathcal{O}(d)$ [131]. Note that the power method updates its estimate of principal components much slower than MOSES, but the two algorithms converge to similar errors. The slow updates of power method will become

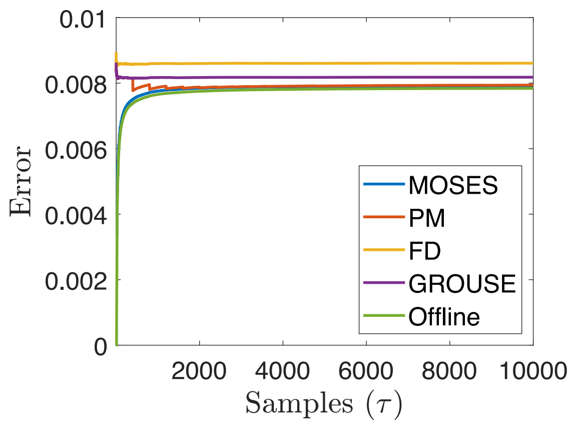
[2]At: https://github.com/edoliberty/frequent-directions/

[3]MATLAB code for GROUSE is publicly available at https://web.eecs.umich.edu/~girasole/grouse/.

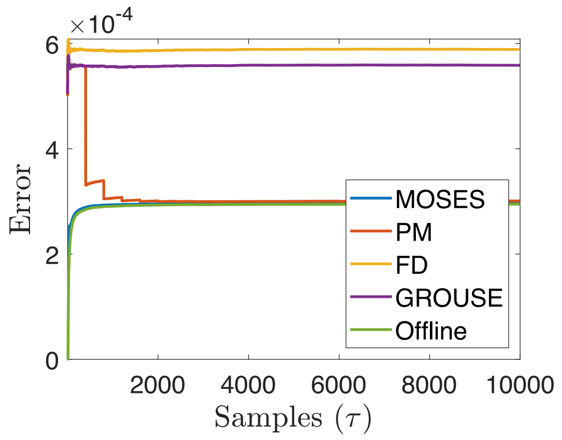
a problem when working with dynamic data, where the distribution of arriving data changes over time.



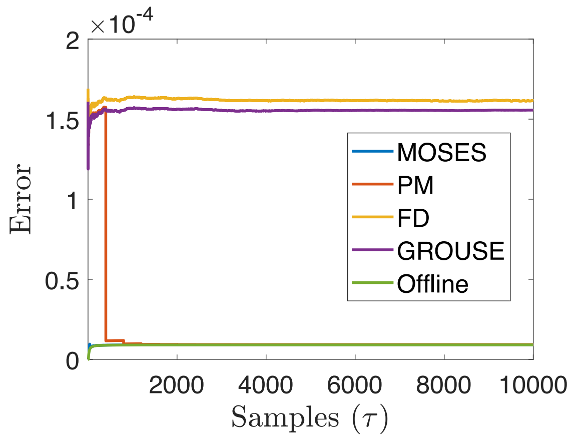
(a) $\alpha = 0.01$.



(b) $\alpha = 0.1$.



(c) $\alpha = 0.5$.

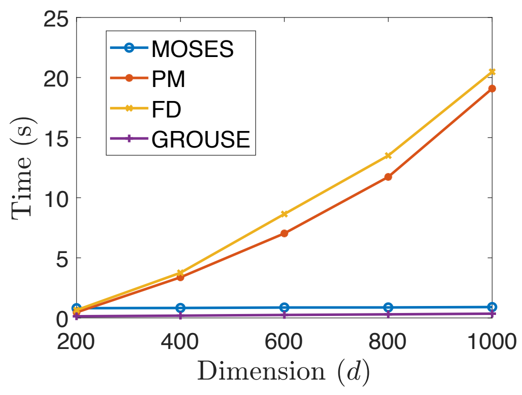


(d) $\alpha = 1$.

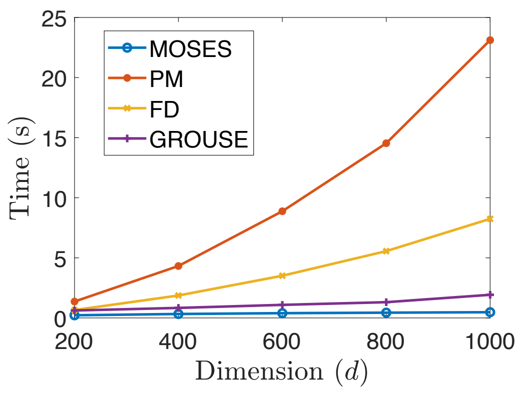
Fig. 3.3 Comparisons on synthetic datasets, see Section 3.5 for the details.

Computational complexity on synthetic datasets.

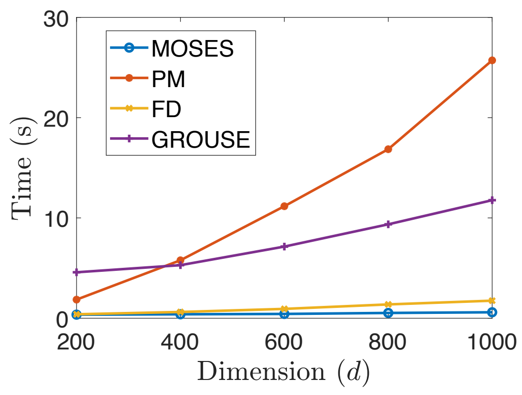
Let us now turn our attention to the computational performance of these three algorithms. On synthetic datasets with $\alpha = 1$ and $\tau = 10000$, we compare the run-time of MOSES to GROUSE and power method, where the block sizes of MOSES and power method, and the step size of GROUSE were set as described in the synthetic tests earlier. The average run-time of all three algorithms over five trials and for various choices of rank r is shown in Figure 3.4. We note that the computational cost of MOSES remains consistently small throughout these simulations, especially for large ambient dimensions and ranks where GROUSE and power method perform poorly, see Figure 3.4c. This appears to happen regardless of the desired recovery rank r used, as the impact on MOSES as the ambient dimension (d) increases is much less, see Figure 3.4c and 3.4d. Interestingly enough, FD performs poorly when attempting a relatively low rank recovery ($r \in \{1, 10\}$) and closely matches MOSES as r increases, which can be attributed to the buffering size of FD. However, overall MOSES seems to exhibit graceful performance scaling regardless of the ambient dimension of the data. In some instances some of the methods are able to compete with MOSES, such as GROUSE in the case when the target rank recovery is small as shown in Figures 3.4a and 3.4b, but fail to scale gracefully when it increases as shown in Figures 3.4c and 3.4d.



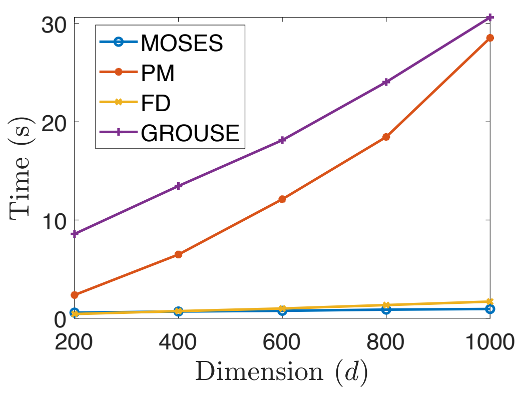
(a) Running time with $r = 1$.



(b) Running time with $r = 10$.



(c) Running time with $r = 50$.

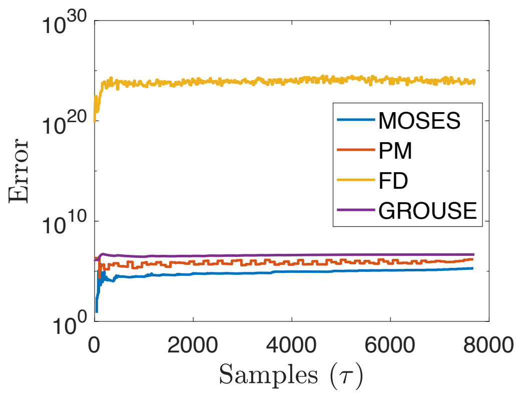


(d) Running time with $r = 100$.

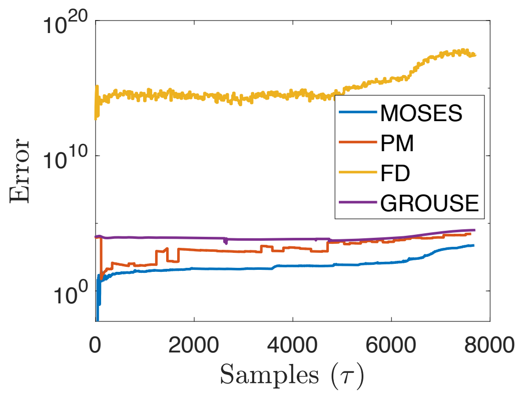
Fig. 3.4 Computational complexity of all algorithms on synthetic datasets, see Section 3.5 for the details.

Comparison using real datasets

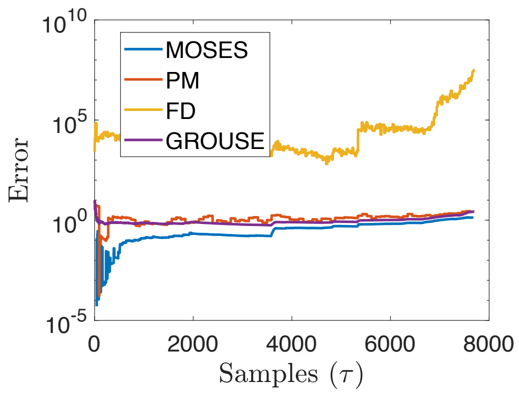
In order to better evaluate the practicality of our method we also evaluate all these three algorithms on actual, publicly available datasets. For our experiments, we use four different datasets that contain *mote* (sensor node) voltage, humidity, light, and temperature measurements over time [48]. These datasets were selected because they are publicly available and are representative of real-world applications due to their ambient dimension n being sufficiently large (> 45) to reflect practical deployments. These results are aggregated in Figure 3.5 and a qualitative analysis about the performance of the algorithms on each dataset follows. For improving presentation, please note that all errors are plotted in logarithmic scale due to GROUSE and FD exploding errors.



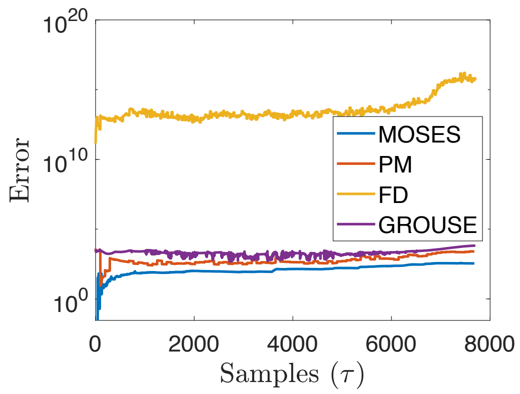
(a) Light dataset, $r = 20$.



(b) Temperature dataset, $r = 20$.



(c) Voltage dataset, $r = 20$.



(d) Humidity dataset, $r = 20$.

Fig. 3.5 Comparisons on voltage and humidity real-world datasets Section 3.5 for the details.

Comparison on the mote voltage dataset

The first dataset we evaluate has an ambient dimension of $n = 46$ and has $T = 7712$ columns. It is an inherently volatile dataset as it contains the rapid small voltage changes the motes exhibit during operation. With $r = 20$ and the rest of the parameters as described in the synthetic comparison above, the errors over time for all algorithms is shown in Figure 3.5c in logarithmic scale. MOSES here outperforms both GROUSE and power method.

Comparison on the mote humidity dataset

The second dataset evaluated has an ambient dimension of $n = 48$ and has $T = 7712$ columns. This dataset contains the humidity measurements of motes and is more periodic in nature with a larger range than the voltage dataset. With $r = 20$ and the rest of the parameters as described in the synthetic comparison above, the errors over time for all algorithms is shown in Figure 3.5d in logarithmic scale. MOSES again outperforms the three other algorithms.

Comparison on the mote light dataset

The third dataset has an ambient dimension $n = 48$ and has $T = 7712$ columns. This dataset contains the light measurements of the motes and contains much more frequent value changes while having the highest range of all four datasets studied in this section. With $r = 20$ and the rest of the parameters as described in the synthetic comparison above, the errors over time for all algorithms is shown in Figure 3.5a in logarithmic scale. As before, MOSES outperforms the three other algorithms.

Comparison on the mote temperature dataset

The last real dataset we consider in this instance has an ambient dimension of $n = 56$ and has $T = 7712$ columns. This dataset contains the temperature measurements of the sensor motes and has mostly periodic value changes and infrequent spikes. With $r = 20$ and the rest of the parameters as described in the synthetic comparison above, the errors over time for all algorithms is shown in Figure 3.5b in logarithmic scale. It is evident that MOSES outperforms the three other algorithms.

3.6 Discussion

In this chapter, we presented MOSES for streaming (linear) dimensionality reduction, an algorithm with minimal storage and computational requirements. One might think of MOSES as an online “subspace tracking” algorithm that identifies the linear structure of data as it arrives. Once the data has fully arrived, both principal components and the projected data are readily made available by MOSES and the user could immediately proceed with any additional learning and inference tasks. Note also that t in our notation need not correspond to time, but rather to a a sequence of vectors $\{\mathbf{y}_t\}_{t=1}^{\mathcal{T}} \subset \mathbb{R}^d$, presented to us sequentially; see eq. (3.1). For example, only a small portion of a large data matrix $\mathbf{Y}_\tau$ can be stored in the fast access memory of the processing unit, which could instead use MOSES to fetch and process the data in small chunks and iteratively update its estimate of r-leading principal components. Moreover, MOSES can be easily adapted to the *dynamic* case where the distribution of data changes over time. In dynamic subspace tracking, each data vector $\mathbf{y}_t$ is drawn from a subspace $\mathcal{S}(t) \in \mathbb{G}(d, r)$ that might vary with time.

A closely related line of work is the incremental SVD [29, 26, 25, 37, 116]. Incremental SVD is a streaming algorithm that, given the (truncated) SVD of $\mathbf{Y}_{t-1} \in \mathbb{R}^{d\times(t-1)}$, aims to compute the truncated) SVD of $\mathbf{Y}_t = [\mathbf{Y}_{t-1}\ \mathbf{y}_t] \in \mathbb{R}^{d\times t}$, where $\mathbf{y}_t \in \mathbb{R}^d$ is the newly arrived data vector and $\mathbf{Y}_{t-1}$ is the matrix formed by concatenating the previous data vectors, see eq. (3.1). It is easy to verify that MOSES generalises incremental SVD to handle data blocks, see Algorithm 1. This small difference between incremental SVD and MOSES is in part what enables us to complement MOSES with a comprehensive statistical analysis in Theorem 3.4.2 which is, to the best of our knowledge at the time of publication, not available for incremental SVD, despite its long history and popularity. Indeed, [10] only very recently provided stochastic analysis for two of the variants of incremental SVD in [109, 137]. The results in [10] hold in expectation and for the special case of $r = 1$, the first leading principal component. Crucially, these results measure the angle $\angle[\mathcal{S}_{\tau,r}, \widehat{\mathcal{S}}_{\tau,r}]$ between the true leading principal components of the data matrix and those estimated by incremental SVD. In this sense, these types of results are inconclusive because incremental SVD estimates both left and right leading singular vectors of the data matrix. More succinctly, incremental SVD both estimates the leading principal components of the data matrix $\widehat{\mathbf{S}}_{\tau,r}$ *and* reduces the dimension of data by computing $\widehat{\mathbf{S}}_{\tau,r}^T \widehat{\mathbf{Y}}_{\tau,r} \in \mathbb{R}^{r\times\tau}$, where $\widehat{\mathbf{Y}}_{\tau,r}$ is the final output of incremental SVD. In contrast to [10], Theorem 3.4.2 and specifically (3.33) assesses the quality of both of these tasks and establishes that, under certain conditions, MOSES performs nearly as well as offline SVD. GROUSE [13] is a closely related algorithm for streaming SVD (on data with possibly missing entries) that can be interpreted as projected stochastic gradient descent on the Grassmannian manifold. GROUSE is effectively identical to incremental SVD when the incoming data is low-rank [13]. In [192], the authors offer theoretical guarantees for GROUSE that again does not account for the projected data and are based on the proof techniques of [10]. Their results hold without any missing data, in expectation, and in a setup similar to the spiked covariance model. Recently, an alternative to GROUSE was introduced called SNIPE that has much stronger theoretical guarantees in case of missing data [58, 55]. In Section 3.5, we numerically compared MOSES with GROUSE.

One might also view MOSES as a stochastic algorithm for PCA. Indeed, note that Program (3.21) is equivalent to

$$\begin{cases} \max \ \mathbb{E}_{\mathbf{y}} \|\mathbf{U}\mathbf{U}^T\mathbf{y}\|_F^2 \\ \mathbf{U}^T\mathbf{U} = \mathbf{I}_r \end{cases} = \begin{cases} \max \ \mathbb{E}_{\mathbf{y}} \langle \mathbf{U}\mathbf{U}^T, \mathbf{y}\mathbf{y}^T \rangle \\ \mathbf{U}^T\mathbf{U} = \mathbf{I}_r \end{cases} = \begin{cases} \max \ \mathbb{E}_{\mathbf{y}} \langle \mathbf{U}\mathbf{U}^T, \mathbf{y}\mathbf{y}^T \rangle \\ \mathbf{U}^T\mathbf{U} \preccurlyeq \mathbf{I}_r, \end{cases} \tag{3.46}$$

where the maximisation is over matrix $\mathbf{U} \in \mathbb{R}^{d\times r}$. Above, $\mathbf{U}^T\mathbf{U} \preccurlyeq \mathbf{I}_r$ is the unit ball with respect to the spectral norm and if $\mathbf{A} \preccurlyeq \mathbf{B}$ then it follows that $\mathbf{B} - \mathbf{A}$ is a positive semi-definite matrix [166]. The last identity above holds because a convex function is always maximised on the boundary of the feasible set. Using the Schur's complement, we can equivalently write the last program above as,

$$\begin{cases} \max \ \mathbb{E}\langle \mathbf{U}\mathbf{U}^T, \mathbf{y}\mathbf{y}^T \rangle \\ \begin{bmatrix} \mathbf{I}_d & \mathbf{U} \\ \mathbf{U}^T & \mathbf{I}_r \end{bmatrix} \succcurlyeq \mathbf{0}_{(d+r)\times(d+r)}. \end{cases} = \begin{cases} \max \ \langle \mathbf{U}\mathbf{U}^T, \mathbf{\Xi} \rangle \\ \begin{bmatrix} \mathbf{I}_d & \mathbf{U} \\ \mathbf{U}^T & \mathbf{I}_r \end{bmatrix} \succcurlyeq \mathbf{0}_{(d+r)\times(d+r)}, \end{cases} \tag{3.47}$$

where $\mathbf{\Xi} = \mathbb{E}[\mathbf{y}\mathbf{y}^T] \in \mathbb{R}^{d\times d}$ is the covariance matrix of the data distribution μ. Note that Program (3.47) has a convex (in fact, quadratic) objective function that is *maximised* on a convex (conic) feasible set. We cannot hope to directly compute the gradient of the objective function above, namely $2\mathbf{\Xi}\mathbf{U}$, because the distribution of $\mathbf{y}$ and hence its covariance matrix $\mathbf{\Xi}$ are unknown. Given an iterate $\widehat{\mathbf{S}}_t$, one might instead draw a random vector $\mathbf{y}_{t+1}$ from the probability measure μ and move along the direction dictated by $2\mathbf{y}_{t+1}\mathbf{y}_{t+1}^T\widehat{\mathbf{S}}_t$. Our motivation to do so, stems from the observation that $\mathbb{E}[2\mathbf{y}_{t+1}\mathbf{y}_{t+1}^T\widehat{\mathbf{S}}_t] = 2\mathbf{\Xi}\widehat{\mathbf{S}}_t$. This is then followed by back projection onto the feasible set of Program (3.46). That is,

$$\widehat{\mathbf{S}}_{t+1} = \mathcal{P}\left(\mathbf{S}_t + 2\alpha_{t+1}\mathbf{y}_{t+1}\mathbf{y}_{t+1}^T\widehat{\mathbf{S}}_t\right), \tag{3.48}$$

for an appropriate step size α_{t+1}. Above, $\mathcal{P}(\mathbf{A})$ projects onto the unit spectral norm ball by setting to one all singular values of $\mathbf{A}$ that exceed one. The stochastic projected gradient ascent for PCA, described above, is itself closely related to the so-called *power method* and is at the heart of [130, 138, 158, 107, 7]. However, as mentioned previously, all of these methods lack a statistical analysis similar to Theorem 3.4.2. One notable exception is the power method in [130] which in a sense applies *mini-batch* stochastic projected gradient ascent to solve Program (3.47), with data blocks (namely, batches) of size $b = \Omega(n)$. There the authors offer statistical guarantees for the spiked covariance model, see Section 3.4. As before, these guarantees are solely for the quality of estimated principal components and silent about the quality of projected data, both of which are addressed in Theorem 3.4.2. Note also that, especially when the data dimension d is large, one disadvantage of this approach is its large block size; it takes a long time of $\Omega(d)$ for the algorithm to update its estimate of the principal components. In this setup, we may think of MOSES as a stochastic algorithm for PCA based on alternative minimisation rather than gradient ascent, see Section 3.3. Moreover, MOSES updates its estimate frequently, after receiving every $b = \mathcal{O}(r)$ data vectors, and also maintains the projected data. In Section 3.5, we numerically compared MOSES with the power method in [130]. A few closely related works are [84, 46, 97, 45].

In the context of online learning and *regret minimisation*, [188, 7] offer two algorithms the former of which is not memory optimal and the latter does not have guarantees similar to Theorem 3.4.2. See also [21]. A Bayesian approach to PCA is studied in [151, 168]. The *expectation maximisation* algorithm there could be implemented in an online fashion but without theoretical guarantees.

More generally, MOSES might be interpreted as a deterministic *matrix sketching* algorithm. Common sketching algorithms either randomly sparsify a matrix, randomly combine its rows (columns), or randomly subsample its rows (columns) according to its *leverage scores* [38, 49] Ideas from sketching and randomised linear algebra could be integrated into MOSES and other streaming dimensionality reduction algorithms [171, 34, 144, 68, 66, 67]. It is also perhaps worth pointing out that one might consider a streaming algorithm as a special case of distributed computing along the “cone” tree shown in Figure 3.6, which we later generalise in Chapter 4 to approximate PCA in the federated setting. When the data vectors have missing entries, a closely related problem is low-rank matrix completion [43, 59].

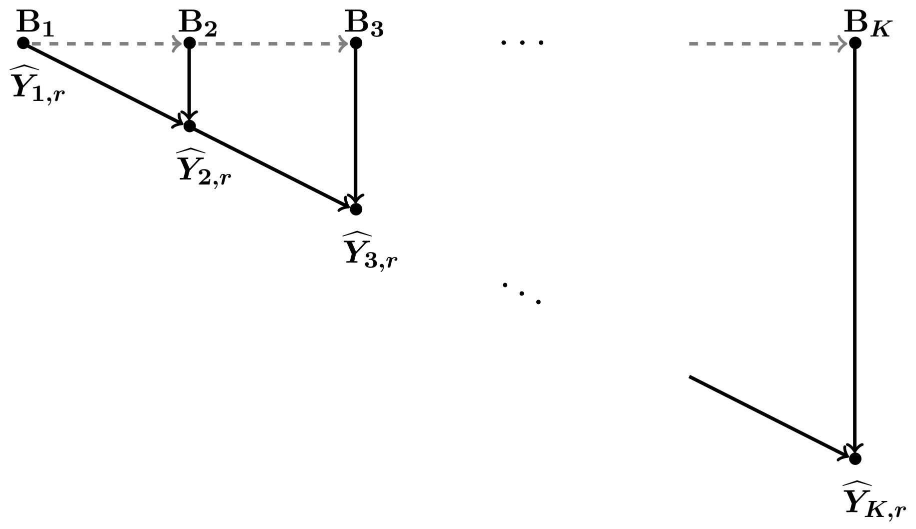

$\mathbf{B_1}$
$\mathbf{B_2}$
$\mathbf{B_3}$
$\mathbf{B}_{\boldsymbol{K}}$
$\widehat{\boldsymbol{Y}}_{\mathbf{1},\boldsymbol{r}}$
$\widehat{\boldsymbol{Y}}_{\mathbf{2},\boldsymbol{r}}$
$\widehat{\boldsymbol{Y}}_{\mathbf{3},\boldsymbol{r}}$
$\widehat{\boldsymbol{Y}}_{\boldsymbol{K},\boldsymbol{r}}$

Fig. 3.6 Streaming problems may be interpreted as a special case of distributed computing. Each data block $\mathbf{y}_k$ lives on a node of the chain graph and the nodes are combined, from left to right, following the structure of the “cone” tree.

However, even though MOSES has the benefit of online computation while exhibiting remarkable performance, even when compared to the offline SVD, fails to scale horizontally. Meaning that, as presented, MOSES cannot be used in multi-node environments. This is because, MOSES is currently formulated to expect the dataset input vectors to be streamed from a central location in order to successfully produce its iterates. In turn, this limitation, makes the algorithm unable to process distributed or federated datasets, thus limiting its usage in single-node scenarios. Moreover, MOSES requires a hyper-parameter to be provided for the intrinsic dimension of the dataset and cannot be adjusted during its execution. Naturally, in a streaming setup this cannot be possibly known exactly and thus can only be guessed. This can be problematic in cases of distribution shifts of rapid data changes in which the actual rank of the observed input changes over time. Further, it lacks the ability to guarantee differential privacy over its iterates, which is a highly desirable feature. In the next Chapter, we are going to build upon the ideas presented herein and introduce a novel algorithm for PCA that is applicable in massive, federated datasets, allows adaptive rank estimation, and is able to guarantee differential privacy.

Chapter 4

Federated Principal Component Analysis

In Chapter 3, we introduced a novel algorithm for performing SVD and by extension PCA in a streaming, memory-limited setup that exhibited remarkable overall performance. However, as it was initially formulated our proposed solution could not be used outside of the computational barriers of a single-node. In this chapter, we would like to scale out these ideas and make them suitable for decentralised computation that is able to harness the voracity of federated datasets. To do so, we put forth unified mathematical framework that makes PCA applicable to federated datasets. More specifically, we introduce an asynchronous, rank-adaptive, memory-limited, and (ε, δ)-differentially private algorithm for PCA in the federated setting. Our algorithm incrementally computes local model updates using a streaming procedure and adaptively estimates its r leading principal components when only $\mathcal{O}(dr)$ memory is available with d being the dimensionality of the data. We guarantee differential privacy via an input-perturbation scheme in which the covariance matrix of a dataset $\mathbf{X} \in \mathbb{R}^{d \times n}$ is perturbed with a non-symmetric random Gaussian matrix with variance in $\mathcal{O}\left(\left(\frac{d}{n}\right)^2 \log d\right)$, thus improving upon the state-of-the-art. Furthermore, contrary to previous distributed algorithms for PCA and in the absence of perturbation masks, our algorithm is also invariant to permutations in the incoming data which provides robustness against straggler or failed nodes. Numerical simulations show that, while using limited-memory, our algorithm exhibits performance that closely matches or outperforms traditional non-federated algorithms, and in the absence of communication latency, it exhibits attractive horizontal scalability.

4.1 Introduction

In recent years, the advent of edge computing in smartphones, IoT and cryptocurrencies has induced a paradigm shift in distributed model training and large-scale data analysis. Under this new paradigm, data is generated by commodity devices with hardware limitations and severe restrictions on data-sharing and communication, which makes the centralisation of the data extremely difficult. This has brought new computational challenges since algorithms do not only have to deal with the sheer volume of data generated by networks of devices, but also leverage the algorithm's voracity, accuracy, and complexity with constraints on hardware capacity, data access, and device-device communication. Moreover, concerns regarding data ownership and privacy have been growing in applications where sensitive datasets are crowd-sourced and then aggregated by *trusted* central parties to train machine learning models. In such situations, mathematical and computational frameworks to ensure data ownership and guarantee that trained models will not expose private client information are highly desirable. In light of this, the necessity of being able to analyse large-scale decentralised datasets and extract useful insights out of them is becoming more prevalent than ever before. A number of frameworks have been put forward to train machine-learning models while preserving data ownership and privacy like Federated Learning [128, 108]. In this work we pursue a federated learning framework to compute PCA in a decentralised way, using resource constrained devices, offering asynchronous computation, and the ability to guarantee differential privacy [50, 51]. Seminal work in federated learning has been made, but mainly in the context of deep neural networks, see [128, 108]. Specifically, in [108] a *federated* method for training of neural networks was proposed and was the first successful, publicly announced, application of federated learning in a production environment. In this setting one assumes that each of a large number of independent *clients* can contribute to the training of a centralised model by computing local updates with their own data and sending them to the client holding the centralised model for aggregation. Ever since the publication of this seminal work, interest in federated algorithms for training neural networks has surged, see [162, 88, 65, 115]. Despite of this, federated adaptations of classical data analysis techniques are still largely missing.

Out of the many techniques available, Principal Component Analysis (PCA) [143, 100] is arguably the most ubiquitous one for discovering linear structure or reducing dimensionality in data, so has become an essential component in inference, machine-learning, and data-science pipelines. In a nutshell, given a matrix $\mathbf{Y} \in \mathbb{R}^{d \times n}$ of n feature vectors of dimension d, PCA aims to build a low-dimensional subspace of $\mathbb{R}^d$ that captures the directions of maximum variance in the data contained in $\mathbf{Y}$. Apart from being a fundamental tool for data analysis, PCA is often used to reduce the dimensionality of the data in order to minimise the cost of computationally expensive operations. For instance, before applying t-SNE [121] or UMAP [127], as it has the ability to reduce the variance and only provide to these algorithms the most significant "parts of the data". The reason for doing so is that both of the aforementioned methods are much more expensive than even traditional PCA. Hence, a federated algorithm for PCA is not only desired when data-ownership is sought to be preserved, but also from a computational viewpoint.

Herein, we propose a federated algorithm for PCA (Algorithm 3). The computation of PCA is closely related to the Singular Value Decomposition (SVD) [54, 129] which can decompose any matrix into a linear combination of orthonormal rank-1 matrices weighted by positive scalars Section 2.4. In the context of high-dimensional data, the main limitation stems from the fact that, in the absence of structure, performing PCA on a matrix $\mathbf{Y} \in \mathbb{R}^{d \times n}$ requires $\mathcal{O}(d^2 n + d^3)$ computation time and $\mathcal{O}(d^2)$ memory. This cubic computational complexity and quadratic storage dependency on d makes the cost of PCA computation prohibitive for high-dimensional data, though it can often be circumvented when the data is sparse or admits another type of exploitable underlying structure. Moreover, in some decentralised applications, the computation has to be done in resource constrained, commodity devices with $\mathcal{O}(d)$ storage capabilities. Given these limitations a PCA algorithm with $\mathcal{O}(d)$ memory dependency is highly desirable. On this front, there have been numerous recent works in the streaming setting that try to tackle this problem, see [131, 130, 125, 7, 8, 21]. However, most of these methods do not naturally scale well nor can they be parallelised efficiently despite their widespread use, e.g. [23, 21]. Unfortunately, this limitation also holds for the MOSES algorithm which was introduced in our previous Chapter. To overcome these issues a reliable and federated scheme for large decentralised datasets is highly desirable. Distributed algorithms for PCA have been studied previously in [105, 117, 146]. Similar to this line of work in [134] proposed a federated subspace tracking algorithm in the presence of missing values. However, the focus in this line of work is in obtaining high-quality guarantees in communication complexity and approximation accuracy and do not the ability to guarantee differential privacy, if required.

Focusing on non-distributed but differentially private setups, a number of papers have attempted to address this and proposed algorithms for the computation of PCA in such settings. Broadly speaking, these can be roughly divided in two main groups: (i) those which are *model free* and provide guarantees for unstructured data matrices, (ii) those that are specifically tailored for instances where specific underlying structure of the input is assumed. In the model-free PCA we have (SuLQ) [18], (PPCA) and (MOD-SuLQ) [32], Analyse Gauss [53]. In the structured case, [85, 86, 84] studies approaches under the assumption of high-dimensional data [198], considers the case of achieving differential privacy by compressing the database with a random affine transformation, while [64] proposes a distributed privacy-preserving version for sparse PCA, but with a strong sparsity assumption in the underlying subspaces.

To the best of our knowledge, the federated, memory-limited setting for the computation of PCA while being able to guarantee differential privacy in the model free case has not been previously addressed in literature. This is not surprising as this case is especially difficult to address. In the one hand, distributed algorithms for computing principal directions are not generally *time-independent*. That is, the principal components are not invariant to permutations the data. On the other hand, guaranteeing (ε, δ)-differential privacy imposes an $\mathcal{O}(d^2)$ overhead in storage complexity, which might render the distributed procedure infeasible in limited-memory scenarios.

Summary of contributions: Our primary contribution is *Federated-PCA* (Algorithm 3) an asynchronous, rank-adaptive, memory-limited algorithm for PCA in the federated setting. It is also able to guarantee, (ε, δ)-differentially privacy. Our algorithm is comprised out of two independent innovations: (1) An algorithm for the incremental, private, and decentralised computation of local updates to PCA, (2) a low-complexity merging procedure to aggregate these incremental updates together. Further, contrary of prior-art in distributed PCA and in the absence of perturbation masks, is also invariant to permutations in the incoming data, which provides enhanced resilience against straggler or failed nodes. By design Federated-PCA is only allowed to do *one pass* through each column of the dataset $\mathbf{Y} \in \mathbb{R}^{d \times n}$ using an $\mathcal{O}(d)$-memory device which results in a $\mathcal{O}(dr)$ storage complexity. Federated-PCA achieves (ε, δ)-differential privacy by extending the symmetric input-perturbation scheme put forward in [32] to the non-symmetric case. In doing so, we improve the noise-variance complexity with respect to the state-of-the-art for non-symmetric matrices [18].

4.2 Federated PCA

We consider a decentralised dataset $\mathcal{D} = \{\mathbf{y}_1, \dots, \mathbf{y}_n\} \subset \mathbb{R}^d$ distributed across M clients. The dataset $\mathcal{D}$ can be stored in a matrix $\mathbf{Y} = \left[\mathbf{Y}^1 | \mathbf{Y}^2 | \cdots | \mathbf{Y}^M\right] \in \mathbb{R}^{d \times n}$ with $n \gg d$ and such that $\mathbf{Y}^i \in \mathbb{R}^{d \times n_i}$ is *owned* by client $i \in \{1, \dots, M\}$. We assume that each $\mathbf{Y}^i$ is generated in a streaming fashion and that due to resource limitations it cannot be stored in full. Our method resembles the *distributed agglomerative summary model* (DASM) [167] in which updates are aggregated in a “bottom-up” approach following a tree-structure. That is, by arranging the nodes in a tree-like hierarchy such that, for any sub-tree, the leaves compute and propagate intermediate results the their roots for merging or summarisation. Furthermore, under the DASM we assume that the M clients in the network can be arranged in a tree-like structure with $q > 1$ levels and approximately $\ell > 1$ leaves per node. Without loss of generality, in this paper we assume that $M = \ell^q$. An example of such tree-like structure is given in Figure 4.1. We note that such structure can be generated easily and efficiently using various schemes [190]. Our procedure is presented in Algorithm 3.

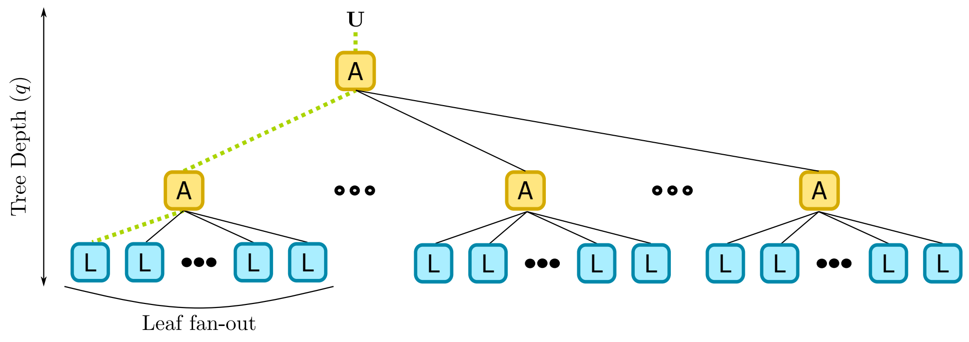


Fig. 4.1 Federated model: (1) Leaf nodes (**L**) independently compute local updates asynchronously, (2) The subspace updates are propagated upwards to aggregator nodes (**A**), (3) The process is repeated recursively until the root node is reached, (4) FPCA returns the global PCA estimate.

Algorithm 3: Federated PCA (FPCA)

```
Data: $\mathbf{Y} = \left[\mathbf{Y}^1 | \cdots | \mathbf{Y}^M\right] \in \mathbb{R}^{d \times n}$: Data for network with $M$ nodes // $(\varepsilon, \delta)$: DP
      parameters // $(\alpha, \beta)$: Bounds on energy, see (4.5) // $B$: Batch size for clients
      // $r$: Initial rank ;
Result: $[\mathbf{U}', \mathbf{\Sigma}'] \approx \text{SVDS}(\mathbf{Y}, r), \mathbf{U}' \in \mathbb{R}^{d \times r}, \ \mathbf{\Sigma}' \in \mathbb{R}^{r \times r}$
Function Federated-PCA($\mathbf{Y}, B, (\varepsilon, \delta), (\alpha, \beta), r$) is
    If using DP and to ensure it, compute $T_{\varepsilon,\delta,d,n}$ as minimum batch size,
      see Lemma 4.2.3
    /* 1. Initialise clients */
    Each client $i \in [M]$ :
        Initialises PC estimate to $(\mathbf{U}^i, \mathbf{\Sigma}^i) \leftarrow (0, 0)$, batch $\mathbf{B}^i \leftarrow [\,]$, and batch size
          $b^i \leftarrow T_{\varepsilon,\delta,d,n}$
    end
    /* 2. Computation of local updates */
    At time $t \in \{1, \ldots, n\}$ , each client $i \in \{1, \ldots, M\}$
        Observes data-point $\mathbf{y}_t^i \in \mathbb{R}^d$ and add it to batch $\mathbf{B}^i \leftarrow [\mathbf{B}^i, \mathbf{y}_t^i]$
        if $\mathbf{B}^i$ has $b^i$ columns then
            $[\mathbf{U}^i, \mathbf{\Sigma}^i] \leftarrow$ FPCA-Edge($\mathbf{B}^i, \mathbf{U}^i, \mathbf{\Sigma}^i, (\varepsilon, \delta), (\alpha, \beta), r$)
            Reset the batch $\mathbf{B}^i \leftarrow [\,]$, and set the batch size $b^i \leftarrow B$
        end
    end
    /* 3. Recursive subspace merge */
    Arrange clients' subspaces in a tree-like data structure an example of which is shown
      in Figure 4.1) and merge them recursively with Algorithm 6.
end
```

Note that Algorithm 3, invokes FPCA-Edge (Algorithm 10) to privately compute local updates to the centralised model and Algorithm 6 to recursively merge the local subspaces in the tree. To simplify the exposition we assume, without loss of generality, that every client $i \in [T]$ observes a vector $\mathbf{y}_t^i \in \mathbb{R}^d$ at time $t \in [T]$, but remark that this uniformity in data sampling need not hold in the general case. We also assume that clients accumulate observations in *batches* and that these are not merged until their size grows to b^i. However, we point out that in real-world device networks the batch size might vary from client to client due to heterogeneity in storage capacity and could indeed be merged earlier in the process. Finally, it is important to note that the network does not need to wait for all clients to compute a global estimation, so that subspace merging can be initiated a new local estimation has been computed without perturbing the global estimation. It operates under the assumption that the aggregation network is trusted and the resulting subspace and singular values are only released at the root, when the algorithm finishes. This in turn implies that that no adversary can eavesdrop on the estimation of any of the nodes during the computation. However, in the absence of differential privacy, we are able to guarantee *time independence* of the end result. In words, this property guarantees that the principal-component estimations after merging are invariant to permutations in the incoming data, see Lemma B.3.1 which is particularly useful in a federated setting. Merge and FPCA-Edge are described in Algorithms 6 and 10.

Subspace merging

Our algorithmic constructions are built upon the concept of *subspace merging* in which two subspaces $\mathcal{S}_1 = (\mathbf{U}_1, \mathbf{\Sigma}_1)$ with $\mathbf{U}_1 \in \mathbb{R}^{d\times r_1}$, $\mathbf{\Sigma}_1 \in \mathbb{R}^{r_1\times r_1}$ and $\mathcal{S}_2 = (\mathbf{U}_2, \mathbf{\Sigma_2})$ with $\mathbf{U}_2 \in \mathbb{R}^{d\times r_2}$ $\mathbf{\Sigma_2} \in \mathbb{R}^{r_2\times r_2}$ are *merged* together to produce a subspace $\mathcal{S} = (\mathbf{U}, \mathbf{\Sigma})$ with $\mathbf{U} \in \mathbb{R}^{d\times r}$, $\mathbf{\Sigma} \in \mathbb{R}^{r\times r}$ describing the combined r principal directions of $\mathcal{S}_1$ and $\mathcal{S}_2$ where $r = \max(r_1,\ r_2)$. Note we operate under the assumption that both subspaces originate from matrices $\mathbf{Y}_1^{d\times n_1}$ and $\mathbf{Y}_2^{d\times n_2}$ respectively, which might now be lost and not available. Our goal is to merge the two subspaces into one that spans the directions of both. Thankfully, one can perform the merging by computing a truncated SVD on their concatenation to get its components. Namely we can formulate the above problem as the solution to the following Program,

$$[\mathbf{U}, \mathbf{\Sigma}, \mathbf{V^T}] \leftarrow \text{SVDS}([\lambda \mathbf{U}_1\mathbf{\Sigma}_1, \mathbf{U}_2\mathbf{\Sigma}_2], r), \tag{4.1}$$

where $\lambda \in (0, 1]$ a *forgetting factor* that allocates less weight to the previous subspace $\mathbf{U}_1$. This is a direct consequence of what SVD achieves once applied. However, while Equation 4.1 works as expected it inefficient as $\mathbf{V}^T$ is not needed and we do not take into account that both bases $\mathbf{U}_1$ and $\mathbf{B}_2$ are already orthonormal. Recently, the utility provided by (4.1) was refined in [96] to be applicable to multiple subspaces when the computation is incremental, but not for streaming data. That is, when every subspace has to be computed in full in order to be processed, merged, and propagated synchronously, which is not ideal for use in a federated approach. We start by presenting partial SVD uniqueness, which expands the results of [96] to be applicable in the combined setting of federated computation using streaming matrices.

Lemma 4.2.1 (Federated SVD uniqueness). *Consider a network with M nodes where, at each timestep $t \in \mathbb{N}$, node $i \in \{1, \dots, M\}$ processes a dataset $\mathbf{D}_t^i \in \mathbb{R}^{d\times b}$. At time t, let $\mathbf{Y}_t^i = [\mathbf{D}_1^i \mid \cdots \mid \mathbf{D}_t^i] \in \mathbb{R}^{d\times tb}$ be the dataset observed by node i and $\mathbf{Y}_t = \left[\mathbf{Y}_t^1 | \mathbf{Y}_t^2 | \cdots | \mathbf{Y}_t^M\right] \in \mathbb{R}^{d\times tMb}$ be the dataset observed by the network. Moreover, let $\mathbf{Z}_t := [\mathbf{U}_t^1\mathbf{\Sigma}_t^1 \mid \cdots \mid \mathbf{U}_t^M\mathbf{\Sigma}_t^M]$ where $[\mathbf{U}_t^i, \mathbf{\Sigma}_t^i, (\mathbf{V}_t^i)^T] = \text{SVDS}(\mathbf{Y}_t^i)$. If $\left[\mathbf{U}_t, \mathbf{\Sigma}_t, \mathbf{V}_t^T\right] = \text{SVDS}(\mathbf{Y}_t)$ and $[\hat{\mathbf{U}}_t, \hat{\mathbf{\Sigma}}_t, (\hat{\mathbf{V}}_t)^T] = \text{SVDS}(\mathbf{Z}_t)$, then $\mathbf{\Sigma} = \hat{\mathbf{\Sigma}}_t$, and $\mathbf{U}_t = \hat{\mathbf{U}}_t\mathbf{B}_t$, where $\mathbf{B}_t \in \mathbb{R}^{r\times r}$ is a unitary block diagonal matrix with $r = \text{rank}(\mathbf{Y}_t)$ columns. If none of the nonzero singular values are repeated then $\mathbf{B}_t = \mathbf{I}_r$. A similar result holds if b differs for each worker as long as $b \geq \min \text{rank}(\mathbf{Y}_t^i)$ $\forall i \in [M]$.*

Lemma 4.2.1 is proved in the Appendix B along with a result on the time-independence property Lemma B.3.1. In order to expand the result of Lemmas 4.2.1 and B.3.1 we must first present a proper implementation of eq. (4.1) as an algorithm. This is presented in Algorithm 4.

Algorithm 4: BasicMerge algorithm

```
Data: $\mathbf{U}_1 \in \mathbb{R}^{d \times r_1}$, first subspace, $\boldsymbol{\Sigma}_1 \in \mathbb{R}^{r_1 \times r_1}$, first subspace singular values
$\mathbf{U}_2 \in \mathbb{R}^{d \times r_2}$, second subspace, $\boldsymbol{\Sigma}_2 \in \mathbb{R}^{r_2 \times r_2}$, second subspace singular values
$r \in [r]$, , the desired rank $r$, $\lambda_1 \in (0,1)$, forgetting factor, $\lambda_2 \geq 1$, enhancing factor
Result: $\mathbf{U}' \in \mathbb{R}^{d \times r}$, merged subspace, $\boldsymbol{\Sigma}' \in \mathbb{R}^{r \times r}$, merged singular values
Function BasicMerge($\mathbf{U}_1$, $\boldsymbol{\Sigma}_1$, $\mathbf{U}_2$, $\boldsymbol{\Sigma}_2$, $\lambda_1$, $\lambda_2$) is
    $[\mathbf{U}', \boldsymbol{\Sigma}', \tilde{}\,] \leftarrow \text{SVDS}([\lambda_1 \mathbf{U}_1 \boldsymbol{\Sigma}_1, \lambda_2 \mathbf{U}_2 \boldsymbol{\Sigma}_2], r)$
end
```

As per Lemma 4.2.1 we are able to use this algorithm in order to merge two subspaces with ease. However, as previously mentioned, there are a few things that we could improve in terms of both speed as well as storage requirements. Recall, that in our particular care we do not require $\mathbf{V}^T$, which is computed by default when using SVD; this incurs both computational and memory overheads. We now show how we can do better in this regard.

Our derivation begins by presenting an improved version for merging, shown Algorithm 5. Notably, this algorithm improves upon the basic merge (Algorithm 4) by exploiting the fact that the input subspaces are already *orthonormal*. In this case, we show how we can transform the Algorithm 4 to Algorithm 5. The key intuition comes from the fact that we can incrementally update $\mathbf{U}$ by using $\mathbf{U} \leftarrow \mathbf{Q}_p \mathbf{U}_R$. To do this we need to first create a subspace basis which spans $\mathbf{U_1}$ and $\mathbf{U_2}$, namely $\text{span}(\mathbf{Q}_p) = \text{span}([\mathbf{U_1}, \mathbf{U_2}])$. This is done by performing $[\mathbf{Q}_p, \mathbf{R}_p] = \text{QR}([\lambda_1 \mathbf{U}_1 \boldsymbol{\Sigma}_1, \lambda_2 \mathbf{U}_2 \boldsymbol{\Sigma}_2])$ and use $\mathbf{R}_p$ to perform an incremental update. Additionally, it is often the case that the subspaces spanned by $\mathbf{U_1}$ and $\mathbf{U_2}$ to intersect; in which case the rank of $\mathbf{Q}$ is less than the sum r_1 and r_2. Typically, practical implementations of QR will permute $\mathbf{R}$ pushing the diagonal zeros only after all non-zeros which preserves the intended diagonal shape in the upper left part of $\mathbf{R}$. However, this behaviour has no practical impact to our results; as in the event this occurs, $\mathbf{Q}$ is always permuted accordingly to reflect this [166]. Continuing, we know that $\mathbf{Q}_p$ is orthogonal but we are not finished yet since $\mathbf{R}_p$ is not diagonal, so an extra SVD needs to be applied on it which yields the singular values in question and the rotation that $\mathbf{Q}_p$ requires to represent the new subspace basis. Unfortunately, even with this improvement, this technique only yields a marginally better algorithm since the SVD has to now be performed at a much smaller matrix, namely, $\mathbf{R}_p$.
Provided with the above, we are now able to derive our final merge algorithm. More concretely, for each subsequent block, namely for $k > 1$, we can refactor Algorithm 2 to be performed in a recursive manner. Now we will present our final merge algorithm by showing how Algorithm 5 can be further improved when $\mathbf{V}^T$ is not needed and we have knowledge that $\mathbf{U}_1$ and $\mathbf{U}_2$ are already orthonormal. This is done by building a basis $\mathbf{U}'$ for $\text{span}((\mathbf{I} - \mathbf{U_1}\mathbf{U_1}^T)\mathbf{U_2})$ via the QR factorisation and then computing the SVD decomposition of a matrix $\mathbf{X}$ such that

$$[\mathbf{U_1}\boldsymbol{\Sigma_1}, \mathbf{U_2}\boldsymbol{\Sigma_2}] = [\mathbf{U_1}, \mathbf{U}']\mathbf{X}. \tag{4.2}$$

Algorithm 5: FasterMerge algorithm

```
Data: $\mathbf{U}_1 \in \mathbb{R}^{d\times r_1}$, first subspace, $\mathbf{\Sigma}_1 \in \mathbb{R}^{r_1\times r_1}$, first subspace singular values,
      $\mathbf{U}_2 \in \mathbb{R}^{d\times r_2}$, second subspace, $\mathbf{\Sigma}_2 \in \mathbb{R}^{r_2\times r_2}$, second subspace singular values,
      $r \in [r]$, the desired rank $r$, $\lambda_1 \in (0,1)$, scaling factor for $\mathbf{U}_1$, and $\lambda_2 \geq 1$, scaling
      factor for $\mathbf{U}_2$.
Result: $\mathbf{U}' \in \mathbb{R}^{d\times r}$, merged subspace, $\mathbf{\Sigma}' \in \mathbb{R}^{r\times r}$, merged singular values
Function FasterMerge($\mathbf{U}_1$, $\mathbf{\Sigma}_1$, $\mathbf{U}_2$, $\mathbf{\Sigma}_2$, $\lambda_1$, $\lambda_2$, $r$) is
    $[\mathbf{Q}_p, \mathbf{R}_p] \leftarrow \mathrm{QR}(\lambda_1\mathbf{U}_1\mathbf{\Sigma}_1 \mid \lambda_2\mathbf{U}_2\mathbf{\Sigma}_2)$
    $[\mathbf{U}_R, \mathbf{\Sigma}', \tilde{}\,] \leftarrow \mathrm{SVDS}(\mathbf{R_p}, r)$
    $\mathbf{U}' \leftarrow \mathbf{Q}_p\mathbf{U}_R$
end
```

It is shown in [148, Chapter 3] in an analytical derivation that this yields an $\mathbf{X}$ of the form

$$\mathbf{X} = \begin{bmatrix} \mathbf{U_1^T U_1 \Sigma_1} & \mathbf{U_1^T U_2 \Sigma_2} \\ \mathbf{U'}^T\mathbf{U_1} & \mathbf{U'^T U_2 \Sigma_2} \end{bmatrix} = \begin{bmatrix} \mathbf{\Sigma_1} & \mathbf{U_1^T U_2 \Sigma_2} \\ 0 & \mathbf{R_p \Sigma_2} \end{bmatrix}.$$

An improvement of Program (4.1) that incorporates the ideas presented above, which resulted in a refined subspace merging algorithm Algorithm 6, is presented below.

Algorithm 6: Merge, part of Algorithm 2 and [148]

```
Data: $(\mathbf{U}_1, \mathbf{\Sigma}_1) \in \mathbb{R}^{d\times r_1} \times \mathbb{R}^{r_1\times r_1}$: *First subspace* // $(\mathbf{U}_2, \mathbf{\Sigma}_2) \in \mathbb{R}^{d\times r_2} \times \mathbb{R}^{r_2\times r_2}$:
      *Second subspace* // with_vt: flag for projected data
Result: $(\mathbf{U}'', \mathbf{\Sigma}'', \mathbf{W}), \mathbf{U}'' \in \mathbb{R}^{d\times r}, \mathbf{\Sigma}'' \in \mathbb{R}^{r\times r}$, optionally $\mathbf{W} \times \mathbb{R}^{r\times(r_1+r_2)}$ merged
        subspace
Function Merge($\mathbf{U}_1, \mathbf{\Sigma}_1, \mathbf{U}_2, \mathbf{\Sigma}_2$, with_vt=False) is
    /* project */
    $\mathbf{Z} \leftarrow \mathbf{U}_1^T\mathbf{U}_2 \in \mathbb{R}^{r\times r}$, $r$ is max($r_1$, $r_2$)
    /* get the QR */
    $[\mathbf{Q}, \mathbf{R}] \leftarrow \mathrm{QR}(\mathbf{U}_2 - \mathbf{U}_1\mathbf{Z})$
    /* perform the SVDS to get the components */
    if with_vt=False then
        $[\mathbf{U}', \mathbf{\Sigma}'', \sim] \leftarrow \mathrm{SVDS}\left(\begin{bmatrix} \mathbf{\Sigma}_1 & \mathbf{Z}\mathbf{\Sigma}_2 \\ 0 & \mathbf{R}\mathbf{\Sigma}_2 \end{bmatrix}, r\right)$
        $\mathbf{W} \leftarrow \sim$
    else
        $[\mathbf{U}', \mathbf{\Sigma}'', \mathbf{W}] \leftarrow \mathrm{SVDS}\left(\begin{bmatrix} \mathbf{\Sigma}_1 & \mathbf{Z}\mathbf{\Sigma}_2 \\ 0 & \mathbf{R}\mathbf{\Sigma}_2 \end{bmatrix}, r\right)$
    end
    /* finally update the subspace */
    $\mathbf{U}'' \leftarrow [\mathbf{U}_1, \mathbf{Q}]\mathbf{U}'$
end
```

Note, that the flushing step is performed when the SVDS practically ignores the $\mathbf{W}$ that would be returned. This is reflected by putting a tilde ($\sim$) as its target output for the projected data upon calling the function, similar to what MATLAB does. In our instance and for presentation reasons it is omitted, however this invocation style is used throughout. Moreover, [148] presented a similar algorithm and was independently discovered but without the statistical guarantees provided by MOSES. In practice what Algorithm 6 does, can be roughly checked in Figure 4.2 shown below. In a nutshell, provided two subspaces as arguments and their associated singular values, then it is able to combine them into a resulting subspace and its associated values. These, in turn, reflect the directions of both, whereas the resulting rank of the subspace and singular value matrices is $\max(r_1, r_2)$. In Chapter B we also complement our algorithm with an empirical evaluation which attempts to show the performance gains by using it.

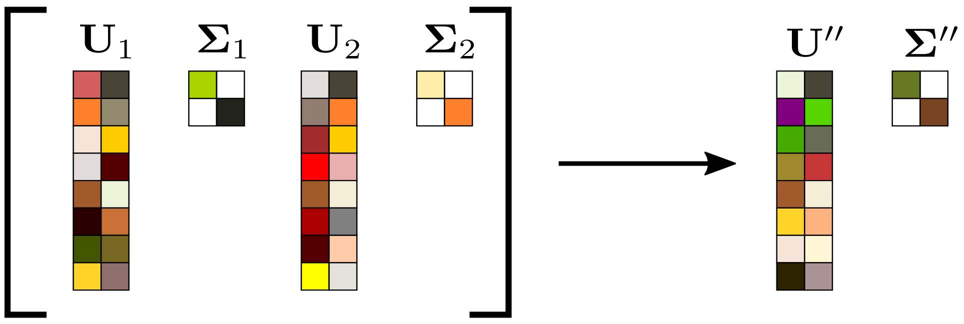


Fig. 4.2 The final Merge procedure as in Algorithm 6, which given as arguments two subspaces and their associated singular values, namely $\mathbf{U}_1 \in \mathbb{R}^{d\times r_1}$, $\mathbf{\Sigma}_1 \in \mathbb{R}^{r_1\times r_1}$ and $\mathbf{U}_2 \in \mathbb{R}^{d\times r_2}$, $\mathbf{\Sigma}_2 \in \mathbb{R}^{r_2\times r_2}$ is able to combine them. The resulting subspace and associated singular values reflect the directions of both, whereas the resulting rank of the subspace and singular value matrices is $\max(r_1, r_2)$.

Local update estimation: Subspace tracking

Similar to the setting we had in the previous chapter, we consider a sequence $\{\mathbf{y}_1, \ldots, \mathbf{y}_n\} \subset \mathbb{R}^d$ of feature vectors. A block of size $b \in \mathbb{N}$ is formed by taking b contiguous columns of $\{\mathbf{y}_1, \ldots, \mathbf{y}_n\}$. Assume $r \le b \le \tau \le n$. If $\widehat{\mathbf{Y}}_0$ is the empty matrix, the r principal components of $\mathbf{Y}_\tau := [\mathbf{y}_1, \cdots, \mathbf{y}_\tau]$ can be estimated by running the following iteration for $k = \{1, \ldots, \lceil \tau/b \rceil\}$ can be computed as,

$$[\widehat{\mathbf{U}}_r, \widehat{\mathbf{\Sigma}}_r, \widehat{\mathbf{V}}_r^T] = \text{SVDS}\left(\begin{bmatrix} \widehat{\mathbf{Y}}_{(k-1)b} & \mathbf{y}_{(k-1)b+1} & \cdots & \mathbf{y}_{kb} \end{bmatrix}, r\right) \tag{4.3}$$

or, more succinctly, we can represent the approximated r-rank estimation of $\mathbf{Y}_\tau$ at k-th block as follows,

$$\widehat{\mathbf{Y}}_{kb} = \widehat{\mathbf{U}}_r \widehat{\mathbf{\Sigma}}_r \widehat{\mathbf{V}}_r^T \in \mathbb{R}^{d\times kb}. \tag{4.4}$$

The above uses the same formulation as used in the previous chapter, namely eq. (3.8) and its output after $K = \lceil \tau / b \rceil$ iterations contains an estimate $\widehat{\mathbf{U}}$ of the r leading principal components of $\mathbf{Y}_\tau$ and the projection $\widehat{\mathbf{Y}}_\tau = \widehat{\mathbf{U}}\widehat{\mathbf{\Sigma}}\widehat{\mathbf{V}}^T$ of $\mathbf{Y}_\tau$ onto this estimate. The local subspace estimation of ((4.3), (4.4)) was presented in the previous chapter, specifically in Algorithm 1. However, while this algorithm is intuitive, it is not particularly computationally efficient. To this end, in last chapter we introduced an improved version, that requires considerably fewer resources to run and was materialised in Algorithm 2.

However, this algorithm was iterative and, as we indicated in Algorithm 6 could be refactored. Essentially, what this allows us to do is separate the incremental update of the subspace and the associated singular values as an isolated operation, while being able to append the projected data if required. Streaming procedures are inherently recursive, however the original formulation of both Algorithm 1, and 2 is not, as they are both based on iterative schemes. To alleviate this shortcoming, we can exploit these procedures and fuse them into an algorithm that is recursive and stateless, suitable for federated computation. This refined version based on the building blocks of MOSES (1) and Merge (6) is called Streaming SVD (S-SVDS) and is presented below.

Algorithm 7: Streaming SVDS (S-SVDS)

```
Data: $\mathbf{D} \in \mathbb{R}^{d \times b}$: block to process // $\mathbf{U} \in \mathbb{R}^{d \times r}$: previous subspace estimate //
      $\mathbf{\Sigma} \in \mathbb{R}^{r \times r}$: previous singular value estimate // (optionally) rank-$r$, default is
      $r = \dim(\Sigma)$ // (optionally) $\mathbf{V}^T \in \mathbb{R}^{r \times (k-1)b}$ projected data estimate after $k$
      blocks, default is $\mathbf{V}^T = \text{NIL}$
Result: $\mathbf{U}' \in \mathbb{R}^{d \times r}$, $\mathbf{\Sigma}' \in \mathbb{R}^{r \times r}$, (optionally) $\mathbf{V}'^T \in \mathbb{R}^{r \times b}$
Function S-SVDS($\mathbf{D}, \mathbf{U}, \mathbf{\Sigma}, r = \dim(\Sigma), \mathbf{V}^T = \text{NIL}$) is
    if $\mathbf{U\Sigma}$ is 0 then
        if $\mathbf{V}^T$ is NIL then
            $\mathbf{U}', \mathbf{\Sigma}', \sim \leftarrow$ SVDS($\mathbf{D}$)
        else
            $\mathbf{U}', \mathbf{\Sigma}', \mathbf{V}^T \leftarrow$ SVDS($\mathbf{D}$)
        end
    else
        /* otherwise, perform block update */
        if $\mathbf{V}^T$ is NIL then
            $\mathbf{U}', \mathbf{\Sigma}', \sim \leftarrow$ Merge($\mathbf{U}, \mathbf{\Sigma}, \mathbf{D}, \mathbf{I}$).
        else
            $\mathbf{U}', \mathbf{\Sigma}', \mathbf{W} \leftarrow$ Merge($\mathbf{U}, \mathbf{\Sigma}, \mathbf{D}, \mathbf{I}$).
            /* append projected data */
            $\mathbf{V}'^T \leftarrow \begin{bmatrix} \widehat{\mathbf{V}}^T & 0 \\ 0 & \mathbf{I}_b \end{bmatrix} \mathbf{W}$.
        end
    end
end
```

Adaptive rank estimation

One of the major limitations that many streaming schemes suffer from, is that they do not explicitly cater for distribution shifts. In our particular case, this can materialise by the estimation changing its effective rank as new data arrives, either upwards or downwards. Meaning that the data observed after a while, might not be adequately captured by the current rank r estimate and we need to increase it. Conversely, it might be the case that we notice increased redundancy and thus can decrease the effective rank resulting in optimal space and resource allocation for each client. This is especially important in a federated setting, since the dataset spans across many heterogeneous clients such events are not only expected, but bound to occur. To tackle this, we introduce a novel solution that tries to adaptively estimate the rank over time while providing concrete bounds on the resulting reconstruction quality. That is, by enforcing within each client,

$$\mathcal{E}_r(\mathbf{Y}_\tau) = \frac{\sigma_r(\mathbf{Y}_\tau)}{\sum_{i=1}^{r} \sigma_i(\mathbf{Y}_\tau)} \in [\alpha, \beta], \tag{4.5}$$

and increasing r whenever $\mathcal{E}_r(\mathbf{Y}_\tau) > \beta$ or decreasing it when $\mathcal{E}_r(\mathbf{Y}_\tau) < \alpha$. In our algorithm, this adjustment happens only once per block, though a number of variations to this strategy are possible. Notably, we can express the global bound in a different form which can give us a more descriptive overall bound, at least w.r.t. to the reconstruction quality. To this end we know that for each local worker its $\|\cdot\|_F$ accumulated error any given time is bounded by the ratio of the summation of its singular values.

Lemma 4.2.2. *Let $\|\cdot\|_F^M \in \{1, \ldots, M\}$ be the error accumulated for each of the M clients at block τ; then, after merging operations the global error will be $\sum_{i=1}^{M} \mathcal{E}_M^{\mathbf{Y}_\tau}$.*

By exploiting Lemma 4.2.2, it allows us to bound the reconstruction quality of $\mathbf{Y}_\tau$ by using its Frobenious norm operator ($\|\widehat{\mathbf{Y}}_\tau\|_F$). This method is a heuristic, but comes with a bound on the reconstruction quality that throughout our empirical evaluation proved to be an effective solution for this problem.

Naturally, there can be corner cases in which rounding errors regarding the fraction of the singular values over time could potentially prove problematic. Another likely source of numerical issues might be the case when there are rapid (consecutive) phase transitions in the input data. These changes pose abrupt shift to the directions of the captured PC's and might require more significant rank adjustments over consecutive blocks than currently allowed. However, dropping components quickly can be equally problematic; thus, we settled for allowing only a single change to the effective rank per block be it either upwards or downwards. We note, that such corner cases as the ones previously described, are not explicitly studied in this work. However, we conjecture in real-world scenarios that these are unlikely events or, when they do manifest, it is probable for these to be isolated to just few clients at any given moment. Thus we believe that their overall impact to the federation scheme would be minimal.

Now, provided with the results of Lemma 4.2.2 we are now able to present the adjust rank algorithm which is directly based on the ideas discussed previously. The implementation of which is presented in Algorithm 8 that follows.

Algorithm 8: AdjustRank

```
Data: $\mathbf{U} \in \mathbb{R}^{d\times r}, \mathbf{\Sigma} \in \mathbb{R}^{r\times r}$, $r = \dim(\Sigma)$, $\alpha$, $\beta$
Result: $\mathbf{U}', \mathbf{\Sigma}' \in \mathbb{R}^{d\times r'}, \mathbf{\Sigma} \in \mathbb{R}^{r'\times r'}$
Function AdjustRank($\mathbf{U}, \mathbf{\Sigma}, r, \alpha, \beta$) is
    if $\mathcal{E}_r(\mathbf{\Sigma}) > \beta$ then
        /* Increase the rank */
        $\mathbf{U}', \mathbf{\Sigma}' \leftarrow [\mathbf{U}, \vec{\mathbf{e}}_{r+1}], \mathbf{\Sigma}_{[r+1]}$
    else if $\mathcal{E}_r(\mathbf{\Sigma}) < \alpha$ then
        /* Decrease the rank */
        $\mathbf{U}', \mathbf{\Sigma}' \leftarrow \mathbf{U}_{[r-1]}, \mathbf{\Sigma}_{[r-1]}$
    else
        /* $\mathcal{E}_r(\mathbf{\Sigma}) \in [\alpha, \beta]$, no need to do anything */
        $\mathbf{U}', \mathbf{\Sigma}' \leftarrow \mathbf{U}, \mathbf{\Sigma}$
    end
end
```

In words, what the adjust rank algorithm does, is that based on the singular values provided as its input it either increases or decreases the effective rank r as described previously. More specifically, in the case of increasing the rank we append the $r+1$-th canonical vector ($\vec{\mathbf{e}}_{r+1}$) to the subspace and its singular value (σ_{r+1}) to a sufficiently small scalar. Typical values for that scalar would be $\sigma_{r+1} = 0.0001$. Another strategy that could work equally well would be to keep all previously captured singular values in a d-dimensional vector and fetch their previous values in case of adjustment. Note that, even in the case of storing all of the previous singular values, asymptotically this does not change the memory requirements. This because this scheme at most requires $\mathcal{O}(d)$ memory since Σ is a diagonal matrix and has at most d entries, which is linear w.r.t. to the ambient dimension (d). On the other hand, if need to decrease the rank we just truncate both the target subspace and singular values to their $r-1$ elements, which are then returned. In the case we are within bounds the function does not do anything to the provided input and returns it as-is. However, we note that one might still want to cap the range the rank can operate for various reasons. For example in nodes that are particularly resource restricted then perhaps it might worth the trade-off to cap the *maximum* rank that these nodes can reach. Conversely, one might want to cap the minimum rank because at certain nodes we want *at least* as many components to be tracked due. Practically speaking what this means is that the rank bounds are provided with respect to min and max rank-r observed within the clients that are participating in the federation. As per our initial conjecture we assume that the input matrices have a low-dimensional intrinsic rank and thus that enable us to keep the dimensions of the respective matrices tractable.

Federated-PCA **without DP**

In the previous sections we discussed on how to incrementally merge blocks and provide the required iterates in a recursive manner. Further we described how we can adjust the rank of the estimates based on the distribution data seen thus far while providing a bound on the reconstruction quality. With these building blocks we are now able to introduce the first version of our FPCA-Edge algorithm. The implementation of which is presented in Algorithm 9 shown below.

Algorithm 9: Federated PCA Edge (FPCA-Edge) (No DP)

```
Data: $\mathbf{B} \in \mathbb{R}^{d\times b}$: Batch $\mathbf{Y}_{\{(k-1)b+1,\dots,kb\}}$ // $(\widehat{\mathbf{U}}_{k-1}, \widehat{\mathbf{\Sigma}}_{k-1})$: SVD estimate for
      $\mathbf{Y}_{\{1,\dots,(k-1)b\}}$ // $r$: Initial rank estimate // $(\alpha, \beta)$: Bounds on energy, see (4.5)
      // $r$: Initial rank estimate
Result: $(\widehat{\mathbf{U}}, \widehat{\mathbf{\Sigma}})$, principal $r$-subspace of $\mathbf{Y}_{\{1,\dots,kb\}}$.
Function FPCA-Edge($\mathbf{B}, \widehat{\mathbf{U}}_{k-1}, \widehat{\mathbf{\Sigma}}_{k-1}, r, \alpha, \beta$) is
    /* Subspace tracking */
    $(\widehat{\mathbf{U}}', \widehat{\mathbf{\Sigma}}', \sim) \leftarrow$ S-SVDS($\mathbf{B}, \widehat{\mathbf{U}}_{k-1}, \widehat{\mathbf{\Sigma}}_{k-1}, r$)
    /* Adjust rank */
    $(\widehat{\mathbf{U}}, \widehat{\mathbf{\Sigma}}) \leftarrow$ AdjustRank($\widehat{\mathbf{U}}', \widehat{\mathbf{\Sigma}}', r, \alpha, \beta$)
end
```

Note, that this function is slated to be executed within each client *independently* and the global dataset is *never* materialised or propagated upwards but only the estimates. Further, the rank adjustment also is able to happen in isolation within each client without any causing any issues during propagation. This is the case, because as we saw previously the Merge algorithm is able to compose subspaces that have different ranks. Further, note that the storage and computational requirements of the *Subspace tracking* procedure of Algorithm 9 are nearly optimal for the given objective since, at iteration k, only requires $\mathcal{O}(r(d+kr))$ bits of memory and $\mathcal{O}(r^2(d+kr))$ flops. However, until now none of the algorithms discussed previously are able to guarantee differential privacy.

Differential Privacy: Streaming MOD-SuLQ

So far, we have introduced a version of FPCA-Edge algorithm but without providing any guarantees for differential privacy, which is an increasingly desired property - especially in a federated setting. However, one might ask - why do we even need privacy in that particular setting given the associated costs with its computation and maintenance. To illustrate a practical attack which could reveal private information when performing dimensionality reduction can serve as an indicator why privacy, in some settings, is indeed quite important and a valid concern. The kind of attackers we consider in this setting are *external adversaries* that know the final output of our algorithm as well as additional prior information about individuals. For example, one such adversary could be one that had gathered partial input from a node while also having at their disposal the final output. Suppose we have a matrix $\mathbf{W} \in \mathbb{R}^{d\times n}$, where d the number of features and n the number of samples, we know that its covariance is given by,

$$\mathbf{C} = \frac{1}{n}\mathbf{W}\mathbf{W}^T.$$

Notably, if we perform the SVD on matrix $\mathbf{W}$ we have,

$$\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T = \mathrm{SVD}(\mathbf{W}). \tag{4.6}$$

As we mentioned previously this decomposes $\mathbf{W}$ into the unitary matrices $\mathbf{U} \in \mathbb{R}^{d\times d}$ and $\mathbf{V}^T \in \mathbb{R}^{n\times n}$ and the diagonal matrix $\mathbf{\Sigma} \in \mathbb{R}^{d\times n}$. The matrices $\mathbf{U}$ and $\mathbf{V}^T$ contain the left and right singular vectors respectively, while the singular values are contained within $\mathbf{\Sigma}$. However, since the covariance matrix $\mathbf{C} \in \mathbb{R}^{d\times d}$ is Hermitian then we know that its eigendecomposition yields eigenvectors given by the unitary matrix $\mathbf{U} \in \mathbb{R}^{d\times d}$ and with associated eigenvalues λ placed within the diagonal matrix $\mathbf{\Lambda} \in \mathbb{R}^{d\times d}$ [165]; more formally we have,

$$\mathbf{C}\mathbf{U} = \mathbf{U}\mathbf{\Lambda}.$$

Since $\mathbf{C}$ is symmetric, we can factorise it as follows,

$$\mathbf{C} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T. \tag{4.7}$$

Which uses its eigenvectors and eigenvalues. One key observation is that we can also perform this by rearranging (4.6) to produce (4.7), thus we can compute the eigenvectors of $\mathbf{W}$ by applying the SVD on its covariance matrix. In order to fully prove this relationship, we can do the following expansions,

$$
\begin{aligned}
\mathbf{C} &= \frac{1}{n}\mathbf{W}\mathbf{W}^T && (4.8)\\
&= \frac{1}{n}(\mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T)(\mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T)^T && (4.9)\\
&= \frac{1}{n}(\mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T\mathbf{V}\boldsymbol{\Sigma}\mathbf{U}^T) && (4.10)\\
&= \frac{1}{n}(\mathbf{U}\boldsymbol{\Sigma}^2\mathbf{U}^T) && (4.11)\\
&\approx \frac{1}{n}(\mathbf{U}_r\boldsymbol{\Sigma}_r^2\mathbf{U}_r^T). && (4.12)
\end{aligned}
$$

It is worth mentioning that we can perform the last step since $\mathbf{V}$ is a unitary matrix and thus $\mathbf{V}^T\mathbf{V} = \mathbf{I}$. The covariance contains important information about the data which in many instances we want to safeguard. Given the tools above and assuming an adversary gathered partial input from a node while also having at their disposal the final output of Federated-PCA then it could easily reconstruct the covariance matrix. In addition, it might be safe to assume that the adversary might also know some of the entries of the target covariance matrix, let the number of the known entries be m. It is worth pointing, that knowing the final output of Algorithm 3 is given, as we assume that anyone has access to it. Thus, the unknown parameters of (4.12) are comprised out of the unknown entries in $\mathbf{C} \in \mathbb{R}^{d\times d}$ that the adversary wants to find to fully reconstruct $\mathbf{C}$ which is $d^2 - m$. However, that there are d^2 linear (approximated) equations in (4.12) and the unknown entries of $\mathbf{C}$ can be recovered if $d^2 \geq d^2 - m$, namely $m \geq 0$. Which is always the case, as Federated-PCA returns both the $\mathbf{U} \in \mathbb{R}^{d\times r}$ and its associated singular values $\boldsymbol{\Sigma} \in \mathbb{R}^{r\times r}$. More broadly speaking this and similar attacks fall into the "reconstruction attacks" category, which is one of the attack types that differential privacy aims to protect against. Now that we have described a viable attack surface, we can proceed on how to apply differential privacy techniques in order to protect against similar attacks.

Our primary innovation lies in refining the seminal work presented by [32] to be applicable to non-symmetric matrices. This is the case, because in the streaming setting not only the dataset cannot be fully materialised but is not known beforehand thus making traditional MOD-SuLQ inapplicable. We refine this algorithmic construction and extend it to be applicable to non-symmetric matrices which correspond to the blocks presented within each client. We start to formulate the provided by having a data matrix $\mathbf{X} \in \mathbb{R}^{d\times n}$ and differential privacy parameters (ε, δ), the MOD-SuLQ algorithm [32] privately computes the k-leading principal components of

$$
\mathbf{A} = \frac{1}{n}\mathbf{X}\mathbf{X}^T + \mathbf{N}_{\varepsilon,\delta,d,n} \in \mathbb{R}^{d\times d}, \qquad (4.13)
$$

the covariance matrix of $\mathbf{X}$ perturbed with a *symmetric* random Gaussian matrix $\mathbf{N}_{\varepsilon,\delta,d,n} \in \mathbb{R}^{d\times d}$. This *symmetric* perturbation mask is such that $(\mathbf{N}_{\varepsilon,\delta,d,n})_{i,j} \sim \mathcal{N}(0,\omega^2)$ for $i \geq j$ where

$$\omega := \omega(\varepsilon,\delta,d,n) = \frac{d+1}{n\varepsilon}\sqrt{2\log\left(\frac{d^2+d}{2\delta\sqrt{2\pi}}\right)} + \frac{1}{n\sqrt{\varepsilon}}. \tag{4.14}$$

Materialising (4.13) requires $\mathcal{O}(d^2)$ memory which is prohibitive in given our complexity budgets. We can reduce the memory requirements to $\mathcal{O}(cdn)$ by computing $\mathbf{X}\mathbf{X}^T$ incrementally in batches of size $c \leq d$. That is, by drawing $\mathbf{N}^{d\times c}_{\varepsilon,\delta,d,n} \in \mathbb{R}^{d\times c}$ and merging the *non-symmetric* updates

$$\mathbf{A}_{k,c} = \frac{1}{b}\mathbf{X}\left[(\mathbf{X}^T)_{(k-1)c+1} \quad \cdots \quad (\mathbf{X}^T)_{ck}\right] + \mathbf{N}^{d\times c}_{\varepsilon,\delta,d,n} \tag{4.15}$$

by using Algorithm 6.

In Lemma 4.2.3 we extend the results in [32] to guarantee (ε,δ)-differential privacy in (4.15). While the SuLQ algorithm [18], guarantees (ε,δ)-differential privacy with non-symmetric noise matrices, it requires a variance rate of $\omega^2 = \frac{8d^2\log^2(d/\delta)}{n^2\varepsilon^2}$, which is sub-optimal with respect to the $\mathcal{O}(\frac{d^2\log(d/\delta)}{n^2\varepsilon^2})$ guaranteed by Lemma 4.2.3. The full proof for Lemma 4.2.3 is in Appendix B (see B).

Lemma 4.2.3 (Streaming Differential Privacy)**.** *Let* $\mathbf{X} = [\mathbf{x}_1 \cdots \mathbf{x}_n] \in \mathbb{R}^{d\times n}$ *be a dataset with* $\|\mathbf{x}_i\| \leq 1$, $\mathbf{N}_{\varepsilon,\delta,d,n} \in \mathbb{R}^{d\times d}$ *and* $\mathbf{A} = \frac{1}{n}\mathbf{X}\mathbf{X}^T + \mathbf{N}_{\varepsilon,\delta,d,n}$. *Let* $\{\mathbf{v}_1,\ldots,\mathbf{v}_d\}$ *be the eigenvectors of* $\frac{1}{n}\mathbf{X}\mathbf{X}^T$ *and* $\{\hat{\mathbf{v}}_1,\ldots,\hat{\mathbf{v}}_d\}$ *be the eigenvectors of* $\mathbf{A}$. *Further, let*

$$\omega(\varepsilon,\delta,d,n) = \frac{4d}{\varepsilon n}\sqrt{2\log\left(\frac{d^2}{\delta\sqrt{2\pi}}\right)} + \frac{\sqrt{2}}{\sqrt{\varepsilon n}}. \tag{4.16}$$

1. *If* $(\mathbf{N}_{\varepsilon,\delta,d,n})_{i,j} \sim \mathcal{N}(0,\omega^2)$ *drawn independently, then* (4.15) *is* (ε,δ)*-differentially private.*
2. *If* $n \geq T_{\varepsilon,\delta,d,n} := \omega_0^{-1}\left[4d\varepsilon^{-1}\sqrt{2\log\left(d^2\delta^{-1}(2\pi)^{-1/2}\right)} + \sqrt{2\varepsilon^{-1}}\right]^{-1}$, *then* (4.15) *is* (ε,δ)*-differentially private for a noise mask with variance* ω_0^2.
3. *Iteration* (4.15) *inherits* MOD-SuLQ*'s sample complexity guarantees, and asymptotic utility bounds on* $\mathbb{E}\left[|\langle\mathbf{v}_1,\hat{\mathbf{v}}_1\rangle|\right]$ *and* $\mathbb{E}\left[\|\mathbf{v}_1-\hat{\mathbf{v}}_1\|\right]$.

Having provided the differential privacy foundations by extending MOD-SuLQ to be applicable to non-symmetric matrices, we are now able to introduce our FPCA-Edge algorithm that takes advantage of that. One additional detail that we need to adhere to is that after the sample complexity bounds are satisfied the output of each FPCA-Edge client is, and remains, differentially private for each consecutive estimate. This is especially convenient in a federated setting, as after the initial estimate lag (in order to guarantee DP) we can then provide an estimate for each consecutive block without losing the provided DP guarantees.

This practically means that we can process *all* blocks however, the estimate is only guaranteed to be differentially private after the sample complexity bound is satisfied. Furthermore, our asymptotic analysis indicates that the more blocks we process the stronger the differential privacy guarantees become. Naturally, this is also affected by the choice of the block size. Moreover, our conclusions seem to be further validated by recent developments which put forth the conjecture that online learning is, by definition, private [71].

By using the previous building blocks, namely S-SVDS, Merge, AdjustRank, and the results of Lemma 4.2.3 we are able to present the (ε, δ)-differentially private Federated-PCA Edge algorithm below.

Algorithm 10: Federated PCA Edge (FPCA-Edge)

```
Data: $\mathbf{B} \in \mathbb{R}^{d \times b}$: Batch $\mathbf{Y}_{\{(k-1)b+1,\dots,kb\}}$ // $(\widehat{\mathbf{U}}_{k-1}, \widehat{\mathbf{\Sigma}}_{k-1})$: SVD estimate for
        $\mathbf{Y}_{\{1,\dots,(k-1)b\}}$ // $r$: Initial rank estimate // $(\alpha, \beta)$: Bounds on energy, see (4.5)
        // $(\varepsilon, \delta)$: DP parameters // $r$: Initial rank estimate
Result: $(\widehat{\mathbf{U}}, \widehat{\mathbf{\Sigma}})$, principal $r$-subspace of $\mathbf{Y}_{\{1,\dots,kb\}}$.
Function FPCA-Edge($\mathbf{B}, \widehat{\mathbf{U}}_{k-1}, \widehat{\mathbf{\Sigma}}_{k-1}, (\varepsilon, \delta), (\alpha, \beta), r$) is
    if $(\varepsilon, \delta)$ is NIL then
        /* No DP, just do subspace tracking */
        $(\widehat{\mathbf{U}}', \widehat{\mathbf{\Sigma}}', \sim) \leftarrow$ S-SVDS($\mathbf{B}, \widehat{\mathbf{U}}_{k-1}, \widehat{\mathbf{\Sigma}}_{k-1}, r$)
    else
        /* Streaming MOD-SuLQ */
        $(\mathbf{U}, \mathbf{\Sigma}) \leftarrow (0, 0)$
        for $\ell \in \{1, \dots, d/c\}$ do
            $\mathbf{B}_s \leftarrow \frac{1}{b}\mathbf{B}(\mathbf{B}_{\{(\ell-1)c+1,\dots,\ell c\}})^T + \mathbf{N}^{d \times c}_{\varepsilon,\delta,d,b}$ such that $\left(\mathbf{N}^{d \times c}_{\varepsilon,\delta,d,b}\right)_{i,j} \sim \mathcal{N}(0, \omega^2)$ and
             $\omega$ as in (4.16)
            $(\mathbf{U}, \mathbf{\Sigma}, \sim) \leftarrow$ S-SVDS($\mathbf{B}_s, \mathbf{U}, \mathbf{\Sigma}, r$)
        end
        /* Subspace merge */
        $(\widehat{\mathbf{U}}', \widehat{\mathbf{\Sigma}}', \sim) \leftarrow$ Merge($\mathbf{U}, \mathbf{\Sigma}, \widehat{\mathbf{U}}_{k-1}, \widehat{\mathbf{\Sigma}}_{k-1}$)
    end
    /* Adjust rank */
    $(\widehat{\mathbf{U}}, \widehat{\mathbf{\Sigma}}) \leftarrow$ AdjustRank($\widehat{\mathbf{U}}', \widehat{\mathbf{\Sigma}}', r, \alpha, \beta$)
end
```

The above algorithm in order to guarantee DP uses the result of Lemma 4.2.3 for $\mathbf{X} = \mathbf{B} \in \mathbb{R}^{d \times b}$ and computes an input-perturbation in a streaming way in batches of size c. If c is taken as a fixed small constant the memory complexity of this procedure reduces to $\mathcal{O}(db)$, which is linear w.r.t. to the ambient dimension (d). However, note that in the presence of perturbation masks, while we are able to reduce the memory complexity the computational complexity remains $\mathcal{O}(d^2)$ due to the incremental covariance expansion per block, see Section 4.2. A value for ε can be obtained from Apple's differential privacy guidelines [5]. However, in our experiments, we

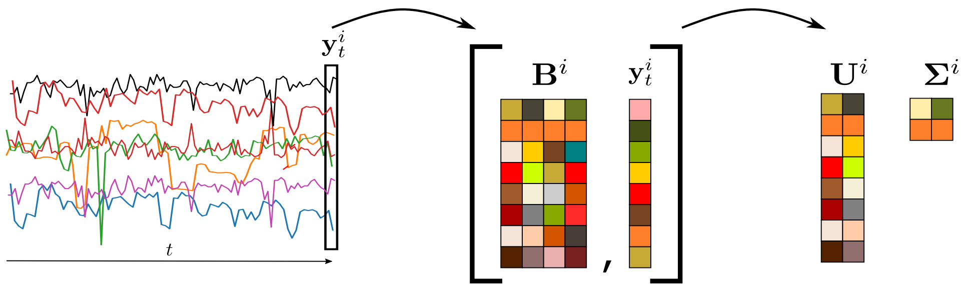


Fig. 4.3 Federated-PCA edge at client i at time t, takes as an argument sequences of vectors $\mathbf{y}_i^t \in \mathbb{R}^d$ which are concatenated into blocks within each client. These blocks are then processed, incrementally, in order to produce the new estimates for $\mathbf{U}^i \in \mathbb{R}^{d\times r}$ and $\Sigma^i \in \mathbb{R}^{r\times r}$ for the i-th client. These results are then propagated upwards to be merged using Algorithm 6.

benchmark across a wider spectrum of values. A rough outline of the FPCA-Edge procedure can be seen in Figure 4.3, which indicates that at every time t each client observes a vector in $\mathbb{R}^d$ which forms a block that is processed accordingly. The differentiation happens depending if there is a requirement to guarantee differential-privacy or not.

4.3 Experimental Evaluation

To validate our proposed scheme, we divide our evaluation into two separate segments. The first focuses on the differentially private aspects of our scheme while the second one presents the results in the absence of perturbation masks - *i.e.* traditional PCA. We try to evaluate our scheme using both established real-world (e.g. MNIST, Wine) as well as synthetic datasets in order to provide a better overall view of its performance. All of our experiments for this section were performed on a workstation using an AMD 1950X CPU with 16 cores at 4.0GHz having 128 GB of 3200 MHz DDR4 RAM, and MATLAB R2020a (build 9.8.0.1380330). To foster reproducibility and accelerate dissemination of the contributions presented both code and datasets used for our numerical evaluation are made publicly available[1].

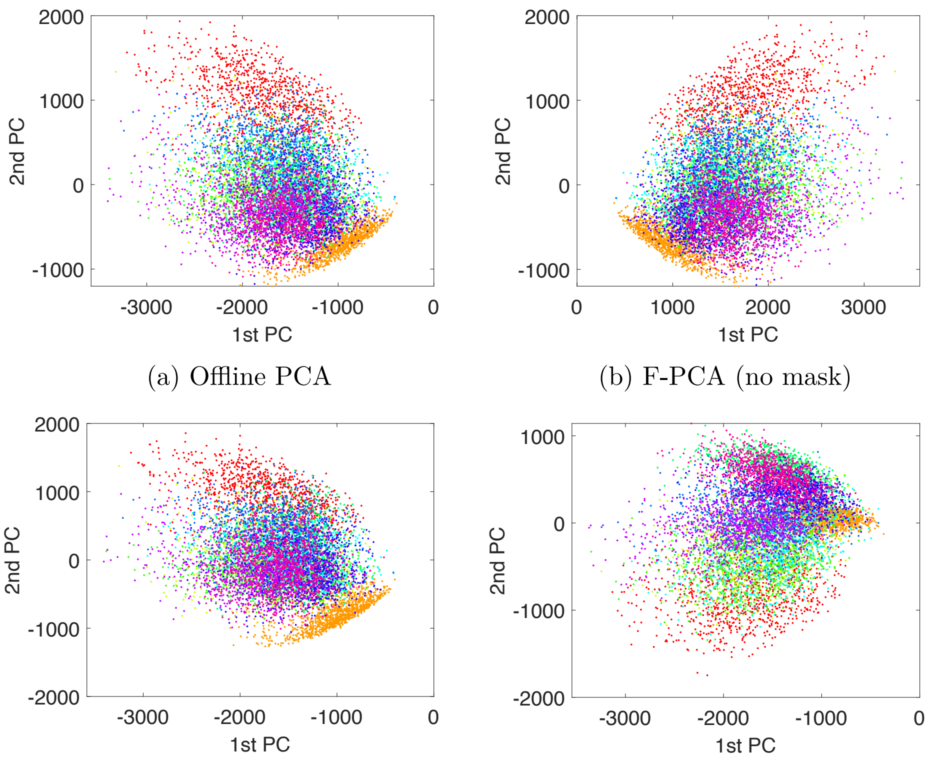
Differential Privacy empirical evaluation

In this section we focus on on comparing Federated-PCA against an offline differentially private algorithm, MOD-SuLQ [32]. To quantify the loss with the application of differential private that our scheme has we compare the quality of the projections using the MNIST [113] and Wine [41] datasets which contain, respectively, 10000 labelled images of handwritten digits and physicochemical data for 6498 variants of red and white wine. Further, we empirically test the utility loss across a variety of spectrums and show that Federated-PCA exhibits attractive performance when compared to the non-streaming variants.

[1] At: https://www.github.com/andylamp/federated_pca

MNIST and Wine evaluation

To retrieve our baseline we performed the full-rank PCA on the MNIST and (red) Wine datasets and retrieved the first and second principal components, see Figs. 4.4a and 4.5a. Then, on the same datasets, we applied FPCA with rank estimate $r = 6$, block size $b = 25$, and DP budget $(\varepsilon, \delta) = (0.1, 0.1)$. The projections for Offline PCA, FPCA with no DP mask, FPCA with DP mask, and vanilla MOD-SuLQ for the MNIST and (red) Wine datasets are shown in Fig. 4.4. We note that to keep the comparison fair with MOD-SuLQ, the rank estimation was disabled in this first round of experiments.

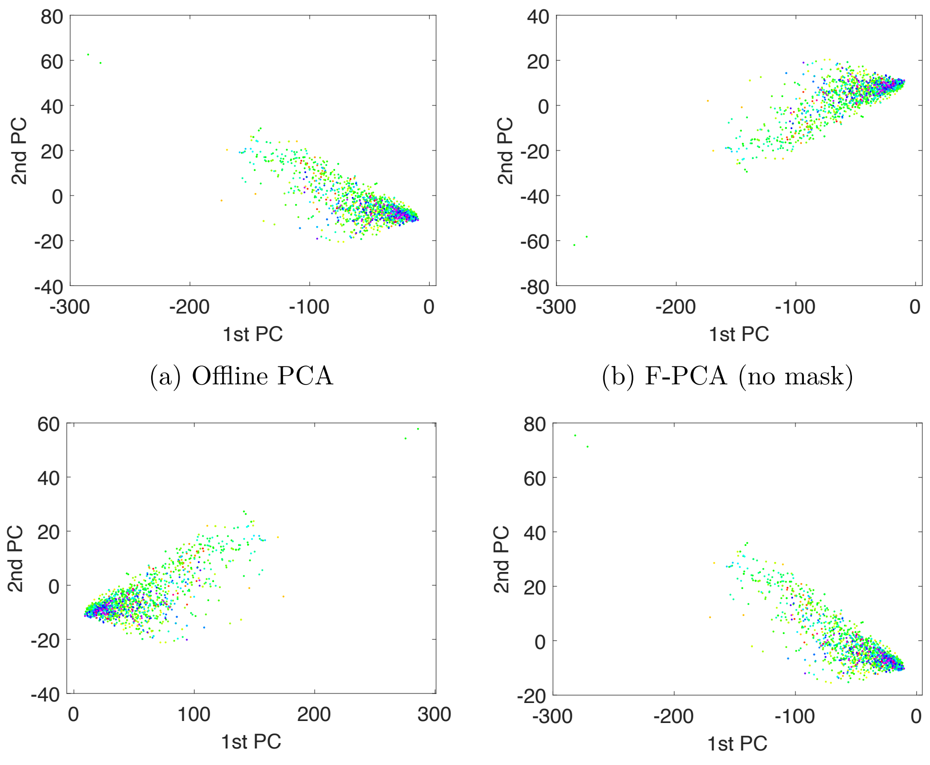


(a) Offline PCA (b) F-PCA (no mask)

(c) F-PCA (with mask) (d) MOD-SuLQ

Fig. 4.4 MNIST projections, for (a) Offline PCA, (b) F-PCA without DP mask, (c) F-PCA with DP mask, (d) (symmetric) MOD-SuLQ. Computed with DP budget of $(\varepsilon, \delta) = (0.1, 0.1)$.

We start by discussing the MNIST dataset results. It can be seen from Figure 4.4 that in all cases FPCA correctly learnt the principal subspace of Offline PCA (up to a rotation) and managed to preserve the underlying structure of the data. In fact, in this case FPCA outperformed traditional MOD-SuLQ. We believe that this is because the MNIST is a large enough dataset with a relatively low true intrinsic dimension of $r = 10$. We also conjecture that the large volume of samples contained within MNIST when compared to the dataset rank aided the preservation of the underlying structure.



(a) Offline PCA (b) F-PCA (no mask) (c) F-PCA (with mask) (d) MOD-SuLQ

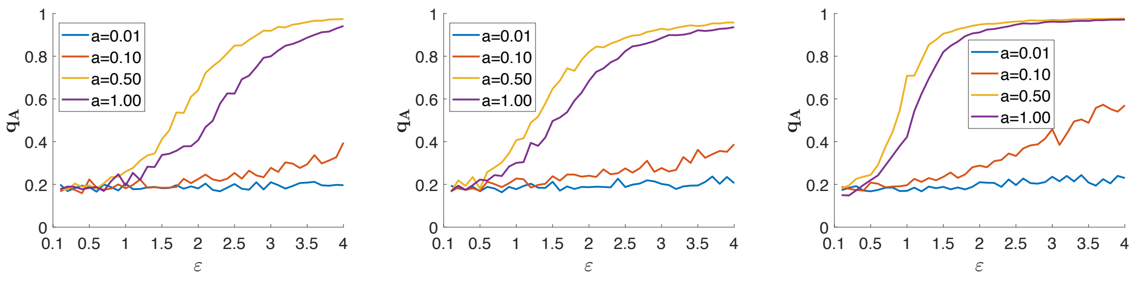
Fig. 4.5 Wine projections, for (a) Offline PCA, (b) F-PCA without DP mask, (c) F-PCA with DP mask, (d) (symmetric) MOD-SuLQ. Computed with DP budget of $(\varepsilon, \delta) = (0.1, 0.1)$.

Next we proceed to discuss the Wine dataset results. In this instance it seems that although FPCA performed well, traditional MOD-SuLQ had the edge. We conjecture that this is because the sample complexity of this dataset relative to its rank is considerably lower than the MNIST and thus there ware not enough samples for a streaming algorithm to completely capture the full structure of the data.

Overall, as we saw from Figure 4.4 and 4.5, we note that in all cases FPCA correctly learnt the principal subspace of Offline PCA (up to a rotation) and managed to preserve the underlying structure of the data. This was particularly evident in traditional dimensionality reduction workloads, meaning when the target recovery rank (r) was *less* than the ambient dimension of the data (d). Moreover, we observed that in datasets that the samples far outnumber the intrinsic dimensionality FPCA performed remarkably well, even beating offline methods, which have no restrictions on the availability of computing resources. We note that the presence of arbitrary rotations is expected as the guarantees for our algorithm hold up to a unitary transform, for more details see Appendix C. Further, when we used our adaptive rank estimation in separate runs, we note that the final estimated rank for MNIST when using differential privacy was $r = 8$ and expressed most of the dataset's variance. Notably, this was very close to the true rank of the MNIST dataset which is $r = 10$ and equal to the number of different handwritten digit digit classes, namely 0 through 9.

Utility loss evaluation

To evaluate the utility loss with respect to the privacy-accuracy trade-off we fix $\delta = 0.01$ and plot $q_A = \langle \mathbf{v}_1, \hat{\mathbf{v}}_1 \rangle$ for $\varepsilon \in \{0.1k : k \in \{1, \ldots, 40\}\}$ where $\mathbf{v}_1$ and $\hat{\mathbf{v}}_1$ are defined as in Lemma 4.2.3. Synthetic data was generated from a power-law spectrum using the Synth function[2] as $\mathbf{Y}_\alpha \sim \text{Synth}(\alpha)^{d \times n} \subset \mathbb{R}^{d \times n}$ and using $\alpha \in \{0.01, 0.1, .5, 1\}$. The results are shown in Figure 4.6 where we see that a larger ε increases the utility, but at the cost of lower DP. Quantitatively speaking, our experiments suggest that the more uniform the spectrum is, the harder it is to guarantee DP and preserve the utility.



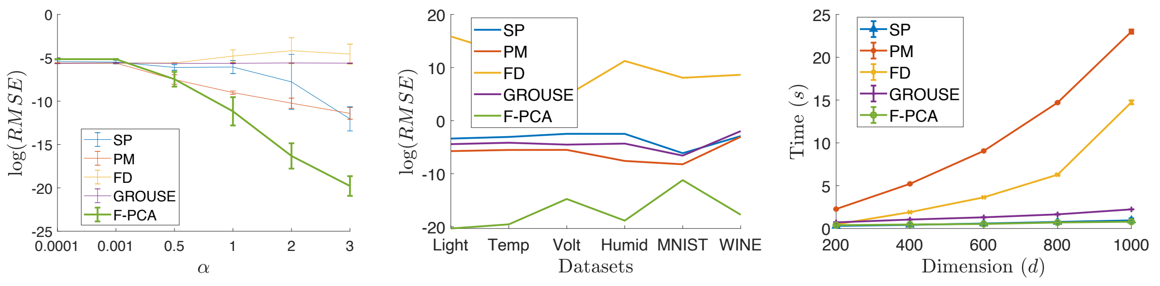
(a) F-PCA (with mask). (b) MOD-SuLQ (non-sym.). (c) MOD-SuLQ (symmetric).

Fig. 4.6 Utility loss of q_A for (a) F-PCA, (b) non-symmetric MOD-SuLQ, and (c) symmetric MOD-SuLQ using $\delta = 0.05$, $N = 5\text{k}$, and $d = 20$ across different ε and $\mathbf{Y}_\alpha \sim \text{Synth}(\alpha)^{d \times n}$.

Federated-PCA **results in the absence of perturbation masks.**

To further evaluate our scheme and its efficacy in more traditional setups we compare its accuracy without perturbation masks on both synthetic, real datasets. The algorithms considered in this instance are: FPCA-Edge (on a single node network), GROUSE [13], Frequent Directions (FD) [47, 120], the Power Method (PM) [130], and a variant of Projection Approximation Subspace Tracking (PAST) [194], named SPIRIT (SP) [142]. In the spirit of a fair comparison, we run FPCA-Edge without its DP features; we elected to do this, as no other streaming algorithm out of the ones we compare against is able to guarantee DP. Figures 4.7b, 4.7a, and 4.7c evaluate the performance of FPCA-Edge against the competing streaming algorithms.

[2]If $\mathbf{Y} \sim \text{Synth}(\alpha)^{d \times n}$ iff $\mathbf{Y} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$ with $[\mathbf{U}, \sim] = \text{QR}(\mathbf{N}^{d \times d})$, $[\mathbf{V}, \sim] = \text{QR}(\mathbf{N}^{d \times n})$, and $\mathbf{\Sigma}_{i,i} = i^{-\alpha}$, and $\mathbf{N}^{m \times n}$ is an $m \times n$ matrix with i.i.d. entries drawn from $\mathcal{N}(0, 1)$.



(a) Errors on synthetic datasets. (b) Errors on real datasets. (c) Average execution time.

Fig. 4.7 Approximation (a-b) and execution (c) benchmarks against other streaming algorithms for a single-node network and without DP masks. Note that the RMSE errors for Figs. 4.7b and 4.7a are in logarithmic scale. The execution time shown in Fig. 4.7c is in seconds and indicates the average time per trial for each of the algorithms across the different ambient dimensions tested (d).

Concretely, in Figure 4.7a we evaluate the resulting subspace using synthetic datasets; these were generated according to the power law distribution and in a similar fashion as in the utility evaluation by using the Synth function[3] as $\mathbf{Y}_\alpha \sim \text{Synth}(\alpha)^{d\times n} \subset \mathbb{R}^{d\times n}$ albeit with a wider gamut for α, namely $\alpha \in \{0.0001, 0.001, 0.5, 1, 2, 3\}$. Moving on, in the second figure, namely Figure 4.7b we examine the performance of the algorithms against the sensor readings from the `Berkeley Mote` dataset which contains measurements of *humidity*, *voltage*, *temperature*, and *light* [48]. We also compare against the traditional MNIST and Wine datasets as in the previous section. In both Figures 4.7b and 4.7a we report log(RMSE) errors with respect to the offline full-rank PCA and show that FPCA exhibits state-of-the-art performance across all datasets. In particular, we note that the performance of FPCA in real datasets was significantly better when compared with the other methods tested.

The third figure attempts to quantify the computational performance of FPCA against the algorithms tested. In this comparison, we measured the time it took for each method to complete across different ambient dimensions (d) over 3 trials. The total number of columns for each ambient dimension was $n = 10^4$ while the recovery rank for all methods was set to $r = 10$. The results of these measurements are depicted in 4.7c in which we observe that both FPCA and SP scale gracefully and exhibit state of the art performance while the other methods suffer from degraded performance as the ambient dimension grows. Overall, we believe that FPCA in most scenarios outperforms the state-of-the-art while also having the desirable ability to scale gracefully as the ambient dimension grows.

[3]If $\mathbf{Y} \sim \text{Synth}(\alpha)^{d\times n}$ iff $\mathbf{Y} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$ with $[\mathbf{U}, \sim] = \text{QR}(\mathbf{N}^{d\times d})$, $[\mathbf{V}, \sim] = \text{QR}(\mathbf{N}^{d\times n})$, and $\mathbf{\Sigma}_{i,i} = i^{-\alpha}$, and $\mathbf{N}^{m\times n}$ is an $m \times n$ matrix with i.i.d. entries drawn from $\mathcal{N}(0, 1)$.

Memory Evaluation

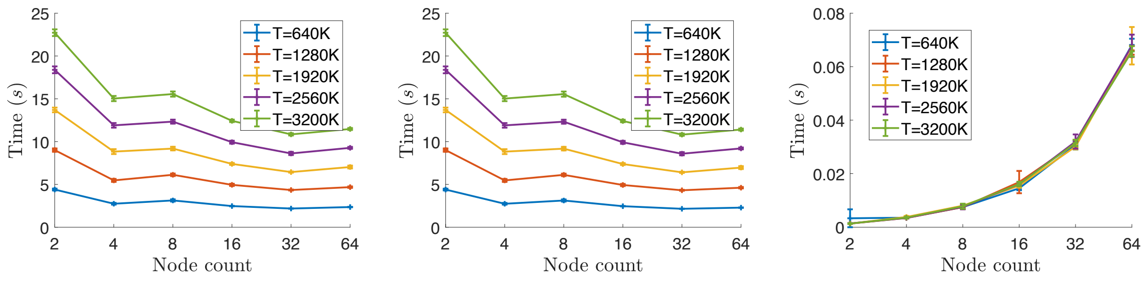
We benchmarked each of the methods used against its competitors and found that our Federated-PCA performed favourably. With respect to the experiments, in order to ensure accurate measurements, we started measuring after clearing the previous profiler contents. The tool used in all profiling instances was MATLAB's built-in memory profiler which provides a rough estimate about the memory consumption; however, it has been reported that can cause issues in some instances. These empirical results support the theoretical claims about the storage optimality of FPCA. In terms of average and median memory allocations, FPCA is most of the times better than the competitors. Naturally, since by design, PM requires the materialisation of larger block sizes it requires more memory than both FPCA as well as FD. Moreover, GROUSE, in its reference implementation requires the instantiation of the whole matrix again; this is because the reference version of GROUSE is expected to run on a subset of a sparse matrix which is copied locally to the function - since in this instance we require the entirety of the matrix to be allocated and thus results in a large memory overhead. An improved, more efficient implementation of GROUSE would likely solve this particular issue. Concluding, we note that although Federated-PCA when using perturbation masks consumes slightly more memory, this is due to the inherent added for supporting differential privacy; however, this cost appears to be in line with our $\mathcal{O}(db)$ memory bound and not quadratic with respect to d, as with competing algorithms.

Table 4.1 Average / median memory allocations (Kb) for a set of real-world datasets.

	Humidity	Light	Voltage	Temperature
FPCA (with mask)	166.57 / 81.23 Kb	172.00 / 99.17 Kb	289.02 / 143.79 Kb	257.00 / 195.30 Kb
FPCA (no mask)	**138.11** / **58.99** Kb	**104.00** / **76.03** Kb	204.58 / **23.47** Kb	**187.74** / **113.28** Kb
PM	905.45 / 666.11 Kb	685.48 / 685.44 Kb	649.12 / 644.35 Kb	657.57 / 668.27 Kb
GROUSE	2896.61 / 2896.62 Kb	2896.84 / 2896.62 Kb	2772.86 / 2772.62 Kb	3379.62 / 3376.62 Kb
FD	162.70 / 117.92 Kb	170.48 / 127.91 Kb	**114.46** / 112.66 Kb	196.11 / 118.59 Kb
SP	476.68 / 405.01 Kb	1009.03 / 508.11 Kb	348.84 / 351.98 Kb	541.56 / 437.61 Kb

Federated performance evaluation

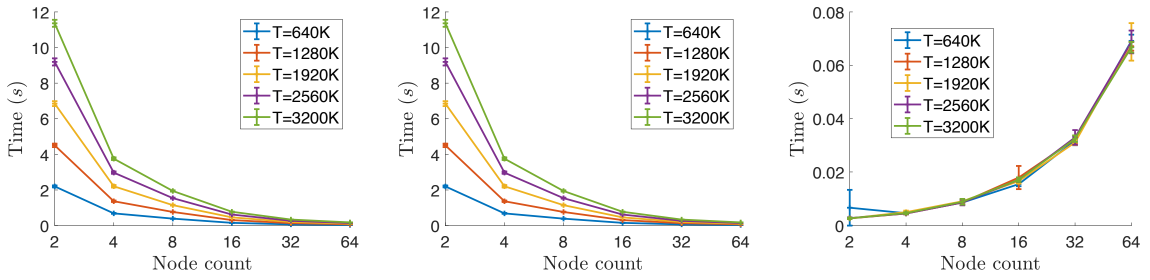
In this part of the evaluation we aim to simulate a federated environment and how the performance of our scheme would look like. Specifically, we create a tree like network structure following a binary tree pattern with different depths and number of nodes. The depth of the tree was provided by $\ell = \log_2(\text{Node count})$ where the node count was picked from $2^i \in \{1:6\}$. This was primarily done for practical reasons rather than a limitation of the algorithm itself, as it can work in the absence of structure. However, MATLAB did not allow through its parallel toolbox easy sharing of non-aligned memory blocks without a huge penalty impact thus everything had to be properly aligned, which the binary tree topology ensured. In particular, we measured the total time it took to execute each trial for which we reported the average. We used an ambient dimension of $d = 10^3$ for all trials while we scaled the column of the dataset accordingly; the number of columns we tested for was $n \in \{640\text{k}, 1.28\text{M}, 1.92\text{M}, 2.56\text{M}, 3.2\text{M}\}$. To provide deeper insights into where the CPU cycles were spent, we separated the global computation time into its components, namely the time to compute Merge and PCA. We report our findings in for the average of total CPU time spend for the total trial duration, Merge, and PCA in Figures 4.8a, 4.8b, and 4.8c respectively.



(a) FPCA: Total time. (b) FPCA: PCA time. (c) FPCA: Merging time.

Fig. 4.8 Computational scaling of FPCA on multi-node networks with binary-trees of depth $\ell = \log_2(\text{Node count})$. We report the average execution time over 5 trials for total execution time (fig. 4.8a), PCA (fig. 4.8b), and Merge (fig. 4.8c) operations respectively. The dataset had $d = 10^3$ features, varying columns n with $n \in \{640\text{k}, 1.28\text{M}, 1.92\text{M}, 2.56\text{M}, 3.2\text{M}\}$ using node count equal to 2^i, $i \in \{1 : 6\}$.

The results indicate that there is a regression after exceeding the number of physical cores available within our machine, which is an expected behaviour. This behaviour is expected, as due to the lack of processing nodes, not all of the sub-problems can be executed concurrently. One way to estimate how this would be extrapolated into networks that would have enough nodes for processing each subproblem independently is to amortise the computation. Notably, we see that the majority of the time is spent for PCA computation as is expected, however we remark Merge exhibits favourable scalability. This empirical discovery further validates its applicability in a federated setting, where it is desirable that the aggregation procedure is as lightweight as possible. In order to provide additional information with respect to the evaluation we also report the amortised execution times per number of workers, as if the workers exceed the number of available compute nodes in our workstation then computation cannot be completed in parallel thus hindering the potential speedup. In Figure 4.9 we show the amortised total (fig. 4.9a), PCA (fig. 4.9b), and Merge (fig. 4.9c) times respectively.
These results indicate, that in the presence of enough resources, Federated-PCA exhibits an extremely favourable scalability curve emphasising the practical potential of the method if used in conjunction with thin clients (*i.e.* mobile phones).



(a) Amortised execution time. (b) Amortised PCA time. (c) Amortised merging time.

Fig. 4.9 Amortised computational scaling of FPCA on multi-node networks with binary-trees of depth $\ell = \log_2(\text{Node count})$. We report the amortised average execution times over 5 trials for total execution time (fig. 4.9a), PCA (fig. 4.9b), and Merge (fig. 4.9c) operations respectively. The dataset had $d = 10^3$ features, varying columns n with $n \in \{640\text{k}, 1.28\text{M}, 1.92\text{M}, 2.56\text{M}, 3.2\text{M}\}$ using node count equal to 2^i, $i \in \{1 : 6\}$.

4.4 Discussion

In this chapter, we have put forth Federated-PCA, which is a unified mathematical framework that makes traditional PCA applicable in the federated setting, consisting of several innovations. More specifically, our framework offers several attractive properties, such as the ability of streaming computation, adaptive rank estimation, and the ability to guarantee differential privacy. We note, in the streaming setup the notion of time used is the same as in Chapter 3; namely that it corresponds to sequence of vector arrivals which are presented to each node rather than actual time units. Its origins are based on the ideas previously established in Chapter 3; but as previously discussed in Section 3.6, even with its considerable merits, MOSES had several drawbacks. Notably, one its major limitations was its lack of horizontal scalability due to its design that restricted its applicability to single node use-cases. In fact, this limitation is shared across most established methods [130, 13, 8] as they all lack the ability to operate in a distributed or decentralised setting. Recently however, there has been a surge of methods trying to tackle PCA computation while offering horizontal scalability [21, 105]. Unfortunately, none of aforementioned methods are able to estimate the rank of the input, rely on the incoming data to admit special structure, or require explicit synchronisation. In our case, we were able to address these limitations by reformulating the problem presented in Chapter 3 such that the actual merging was decoupled from the iterate computation. In doing so, allowed us to introduce Federated-PCA Edge (Algorithm 10) algorithm that offered horizontal scalability by construction. In turn, this separation, is what provided the required insights that helped us to devise our intrinsic rank estimation scheme and addresses another significant limitation that many algorithms share [13, 8, 131], including MOSES. We complement this result with bounds on the reconstruction error which is provided in Lemma 4.2.2. Notably, there might be instances in which one might still want to cap the range of the rank for various reasons. Specifically, one such case would be in the event that some of the nodes are particularly resource restricted; then perhaps it might worth the trade-off to cap the *maximum* rank that these nodes can reach in order to conserve resources. We note that, if desired, this can be trivially supported from our scheme. Moreover, our Merge (Algorithm 6 procedure is lightweight and as empirically observed throughout our evaluation exhibited remarkable scalability. This is a desirable trait, as we expect to perform these merges frequently in a federated setting. Another attribute of our proposed scheme that can be particularly useful, is time-independence of the result, which for Federated-PCA holds in the absence of perturbation masks (*i.e.* when we do not require DP). This property was formalised in Lemma B.3.1 and enables the input samples to arrive in any order while guaranteeing that the final output of the algorithm will remain the same. Such explicit guarantees for the end result and input perturbation invariance were not present in prior art. However, we note, that in works such as [66] or [130] the result was provided in expectation and only for one of the principal subspaces rather than both; in our case, we provide a deterministic result for both principal subspaces as well as the singular values.

Interestingly, one particular question that might arise is why not use Stochastic Gradient Descent (SGD) for solving this problem. In general, SGD based approaches work reasonably well in practice. However, they suffer from a few drawbacks that are worth considering when picking a suitable algorithm in a particular domain. Firstly, they are stochastic - meaning that they results are not *deterministic*, which might hinder their explainability [155]. This property can be a necessity in sensitive decision making applications. In addition, normally SGD based approaches require multiple iterations to converge, which in turn implies multiple passes over the data (and their availability). On the other hand, our approach follows a line of work that tries to provide *deterministic* guarantees on the output while only requiring *one pass* over the data. In particular, our approach requires only a single block of data to be stored at any given time within each node and is able to iteratively update both the estimates as well as the projected data. Normally, SGD based approaches are only able to produce the subspace, rather than also provide access to the projected data - if that is desired. More importantly however, SGD based approaches can perform, at best, as well as the SVD. This is a consequence of the Eckart-Young-Mirsky Theorem [54, 129], as the SVD can produce the *best possible* rank-r approximation of any given matrix. However, in the past methods similar to ours were computationally expensive or had probabilistic bounds. Our line of work tries to alleviate that by firstly introducing MOSES in the previous chapter and Federated-PCA herein, attempting to make such methods scalable as well as practical while providing their attractive guarantees.

Unfortunately, none of the aforementioned algorithms provided the ability to guarantee differential privacy, which is an increasingly desirable property and, in some instances, a requirement. Indeed recently, there have been many seminal works in the model-free PCA setting, namely (SuLQ) [18] which was improved in [32] that yielded (PPCA) and (MOD-SuLQ) as well as Analyse Gauss [53]. On the other hand, if we assume that the input data admits specific underlying structure [85, 86, 84] studies approaches under the assumption of high-dimensional data whereas in [198], considers the case of guaranteeing differential privacy by compressing the database by using a random affine transformation. More recently and closely related to our work [64] proposed a distributed privacy-preserving version for sparse PCA, but requires synchronisation and a strong sparsity assumption in the underlying subspaces. In our work, to guarantee differential privacy we exploited the celebrated MOD-SuLQ algorithm [32, 33] and extended its results to be applicable in the streaming and non-symmetric setting. We note, that we are able to do so while preserving the same nearly-optimal asymptotic guarantees provided by MOD-SuLQ and our provided bound of $\mathcal{O}\left(\left(\frac{d}{n}\right)^2 \log d\right)$ improves the bound previously provided by [18]. In addition, we complement our analysis with bounds for both utility (Lemma B.2.5) as well as a sample complexity (Lemma B.2.6). More importantly, having visibility on the sample complexity is an extremely useful insight as it allows to have a precise minimum sample horizon on when the result is able to be released while guaranteeing differential privacy. On the other hand, one limitation is that when our scheme guarantees differential privacy then this voids its time-independence property. This is because as currently formulated, Lemma B.3.1 preserves the absolute *weights* of the end result. Naturally, this is something that cannot possibly hold in the case of DP due to the noise added in order to guarantee it. Note, that in the context of perturbation masks, we have assumed that the network is trusted and that no adversary can eavesdrop on the estimation of any of the nodes. A number of interesting DP settings occur when this assumption is relaxed. For instance, such cases are when the network is not trusted or when the values of $(\mathbf{U}, \mathbf{\Sigma})$ are not initialised to the zero matrix in Algorithm 10. In these cases, it is important to quantify the privacy loss of observing several queries to the dataset which can be addressed via composition theorems [51, 102]. Concretely, such an example is [64] in which they assume that the central server is an untrusted entity and thus during each iteration of the algorithm a privacy cost has to be paid. For the purposes of this work these are not cases we consider and leave them as future work. Another avenue for future work, is to devise a formulation of Lemma B.3.1 that measures the time independence with respect to the *utility* rather than the absolute weights. In doing so, such a formulation could yield the same time-independent guarantees, but using a metric that is appropriate in the setting of DP. The final avenue for potential future work that we will address is to extend Federated-PCA in the setting of missing values while preserving differential privacy. Indeed, one of the current limitation of our approach is its inability to operate with datasets that have missing values and obtaining a formulation that can handle this would greatly enhance its applicability.

Concluding, in this chapter we introduced an algorithm to compute PCA that is able to scale to federated datasets, can estimate their intrinsic dimension, and also guarantee (ε, δ)-differential privacy. At the edge, our proposed method is able to process the individual sub-problems using limited memory and resources. It is also streaming, meaning that is able to produce its iterates incrementally while using small blocks of data. This allows faster updates of its iterates, resulting into more frequent and accurate iterate estimations. Further, the lightweight merging procedure enables us to guarantee that aggregation will be swift and accurate with the ability to process aggregates from multiple nodes with minimal latency. These properties along with their provided theoretical guarantees make this method an ideal candidate to be used as building block for a federated task scheduler in decentralised setups. In the next chapter, we are going to introduce a novel task scheduler that is able to operate on traditional but is also to federated data centres of the future. We are also going to exploit the produced subspace iterates within each node that employ the learnt embeddings to make job acceptance decisions while maximising the overall system responsiveness.

Chapter 5

Federated Task Scheduling

In this chapter, using the previous constructions we present a federated, asynchronous, memory-limited algorithm for online task scheduling. Our scheme is able to handle large scale job allocations across networks comprising with hundreds of workers. This is achieved by using a variant of Federated-PCA which was presented in Chapter 4 and exploiting its ability to incrementally compute local model updates. This local model is then used along with incoming data to generate a rejection signal which reflects the overall node responsiveness and if it is able to accept an incoming task without resulting in degraded performance. Through this innovation, we allow each node to execute scheduling decisions on whether to accept an incoming job independently based on the workload seen thus far. Further, using the aggregate of the iterates a global view of the system can be constructed, as needed, and could be used to produce a holistic perspective of the system. We complement our findings, by an empirical evaluation on a large-scale real-world dataset of traces from a virtualised production data centre that shows, while using limited-memory, that our algorithm exhibits state-of-the-art performance. Concretely, it is able to predict changes in the system responsiveness ahead of time based on the industry standard `CPU Ready` metric and, in turn, can lead to better scheduling decisions and overall utilisation of the available resources. Finally, in the absence of communication latency, it exhibits attractive horizontal scalability.

5.1 Introduction

Data centre resource allocation or scheduling is the fundamental task of allocating resources (e.g., CPU, memory, network bandwidth, and disk space) to workloads such that their performance objectives are satisfied and the overall data centre utilisation is kept high. Even small deviations from the desired objectives can have substantial detrimental effects with millions of dollars in revenue potentially lost [14].

There exist many different data centre scheduling approaches, e.g., [20, 106, 178, 69, 123, 139]. Such approaches rely on estimates of nodes' future resource availability to schedule workloads in ways to avoid saturation and to efficiently utilise resources across data centre nodes. For predicting resource availability, schedulers either probe available nodes on an on-demand basis [139, 178], or collectively analyse time-series of performance data generated by the underlying physical and virtual infrastructure (e.g., servers and virtual machines (VMs)) across data centre nodes [41, 123]. For example, Microsoft's Recourse Central gathers VM utilisation data on a centralised cluster and uses offline machine learning prediction to tackle servers' over-subscriptions [41]. Although performance data (usually referred to as *telemetry* data) can provide a very detailed view of resource consumption over time, it is a challenge how to effectively analyse this to accurately predict in *near real-time* resource availability across nodes in a scalable manner.

Related work has shown that when schedulers have access to performance data from all data center nodes, they can generate improved holistic models for efficient provisioning [178, 69, 20, 123, 41]. However, these works operate on a near offline fashion and consume network bandwidth to transfer data from servers to centralised locations for processing. As such, they lack the ability to react to performance problems in real-time. Furthermore, to tackle data centre scalability the vast majority of schedulers are distributed or hierarchical and continuously probe subsets of servers about their resource consumption, such as CPU and memory utilisation, to make scheduling decisions based on nodes' availability after probing [139]. Although they can identify resource availability relatively fast they operate on a partial view of the data centre and so they lack global efficiency. It remains a challenge how to collectively analyse performance data in near real-time for efficient scheduling across data centre nodes.

To tackle the problem of large-scale performance data analysis with minimal latency we exploit recent advancements in edge computing. Under this new paradigm, data is generated by commodity devices with potential hardware limitations and important restrictions on data-sharing and communication, which makes centralised processing of data extremely challenging. The dominant, scalable training model that has shown to be able to tackle such challenges is Federated Learning (FL) [108, 128]. Since the publication of these seminal works, we observe an increased interest in federated algorithms for training neural networks [162, 88]. In this setting, there is a large number of independent *clients* running at the edges that contribute to the training of a centralised model by computing local updates with their own data and sending them to a designated client holding the centralised model for aggregation [195]. Over the year, FL has attracted significant interest and has eventually expanded into its own field with most of the literature to focus on deep neural networks, see [115]. The first notable example of a truly decentralised and federated cluster was put forth by Google to collaboratively train the Gboard[1] Android keyboard with great success. Specifically, their approach used the technique from [108] which was the first publicly announced *federated* method for training of neural networks.

[1]https://ai.googleblog.com/2017/04/federated-learning-collaborative.html

Despite the wide adoption of FL in many areas, to the best of our knowledge, this paradigm has not been yet applied into solving the large-scale, data analysis and data centre task allocation problems. Notably, traditional data centres are not federated, as all nodes operate under the same administration. Here, we advocate that the need for real-time large-scale analysis of performance data makes FL the ideal solution for the problem in-hand.

We also believe that based on the trends observed [115, 19] that federated schemes as the one used for training Gboard could expand in-scope and soon pave the way for the formation of *federated data centres*. In this case, traditional schedulers which assume full access to all data centre nodes will not be applicable. The scope of this paper is on exploiting FL for scheduling on today's data centres which could be further relevant to potentially future federated data centres.

In this paper, we present PRONTO, a federated algorithm that uses real-time performance data from virtualised data centre nodes for online task scheduling. Pronto, to the best of our knowledge, is the first to approach the data centre scheduling problem as a federated processing one. In particular, our approach focuses on predicting the spikes in values of the `CPU Ready` performance metric generated by the VMware vSphere leading virtualisation platform as these are generated by each data centre node. The `CPU Ready` VMware metric captures the percentage of time a VM is ready to run but is not scheduled in one of the available CPUs. As a rule of thumb the `CPU Ready` values should be kept low and below a predefined threshold for the system to operate well and without performance problems [44]. Higher `CPU Ready` values than the predefined threshold typically indicate that the VM's performance is degrading as the VM in question does not run despite being ready [184]. The `CPU Ready` is used extensively as an industry standard indicator of performance problems [183]. Despite being an instrumental performance indicator with years of application in the industry, we are not aware of any data centre schedulers based on this metric.

In PRONTO, we aim to predict the performance degradation of a node by detecting `CPU Ready` spikes. At its core, PRONTO predicts an incoming `CPU Ready` spike based on our empirical observation from a real-world data centre trace that a spike in the weighted summation of the top-r tracked projections within a node is highly indicative of an incoming spike. To this end, PRONTO tracks in real-time the top-r projections within each node and by exploiting spikes detected over a sliding window it decides whether to accept an incoming job or not. Concretely, at each timestep and for every node, PRONTO generates a Boolean flag based on the weighted summation of the number of projection spikes detected over a sliding window. The flag is raised (*i.e.*, `true`) if the node at time t *cannot* accept a job, and lowered (*i.e.*, `false`) otherwise. This Boolean flag over-time can be thought as a binary signal, which which we deb as the "*Rejection signal*". In other words, we treat a system to be in a degraded state, if at any given time t its `CPU Ready` value exceeds a predefined threshold.

PRONTO is designed to be *federated, streaming*, and *unsupervised.* It is federated as it executes scheduling plans in a decentralised fashion without knowledge of the global performance dataset. This is one of the key benefits of our approach, as nodes are able to immediately take decisions about incoming workloads without the need for global synchronisation and so reducing communication overhead and scheduling latency. To achieve this, we combine recent advances of federated PCA to accurately compute the d-dimensional, r-rank embedding space $\mathbf{U} \in \mathbb{R}^{d \times r}$ of data incrementally [79]. This approach also provides an efficient mechanism to adaptively estimate the embedding rank [79] with high accuracy which is likely to happen as workload trends evolve. Additionally, it is *streaming* and only requires memory linear to the number of features considered, namely the required memory is proportional to $\mathcal{O}(d)$. Furthermore, it ensures *data ownership*, as each node keeps its own incremental and evolving estimate while catering for distributional shifts; which can be crucial if some parts of the data centre process orthogonal or sensitive workloads. Finally, experimental evaluation shows that it is *fast* as it can incrementally track thousands of features per second while having no “offline” components. This means that the computation of PRONTO is *streaming* requiring only a single pass over the incoming data without having to store any historical data in order to update its estimates. This method can provide tangible benefits to data centres as it enables, in an online fashion, the allocation of incoming jobs across thousands of nodes efficiently and effectively while improving overall allocation. To the best of our knowledge this is the first work to tackle large-scale online workload scheduling using a FL-based algorithm.

Summary of contributions: In this chapter, we introduce a novel federated task scheduler that is able to operate in the streaming and memory-limited setting. It allows each of the computing nodes to independently take decisions for upcoming task assignments without the need for global synchronisation. Further, each node only accepts an incoming job if its responsiveness will not be affected; a node’s deterioration is captured by spikes of the `CPU Ready` metric. To the best of our knowledge, although `CPU Ready` is widely used, it has not been used as a task scheduling predictor before. We also provide a thorough discussion about the importance of `CPU Ready` for performance predictions and we present exploratory results on the use of traditional offline methods to predict `CPU Ready` values. Finally, we evaluate our proposed scheme using traces gathered from the virtualised data centre of an international bank organisation (hereafter referred to as the Company) to validate our claims.

5.2 Importance of `CPU Ready`

The `CPU Ready` is an important VM performance metric and it is widely used by system administrators to identify CPU saturation at-run-time. It reports the % of time a VM is ready to run but was not scheduled in a CPU. The higher the `CPU Ready` values are, the longer a VM is waiting for a virtual CPU (vCPU) to run and so its hosted workloads are not executing and suffer from performance degradation. When the `CPU Ready` values of a VM are higher than a threshold then this VM is saturated and it needs more resources to run efficiently.

Although the `CPU Ready` metric is a key performance indicator, most schedulers focus on using the CPU utilisation metric to allocate workloads to nodes. Typically schedulers assume known CPU workload demands which they schedule on node(s) where the predicted CPU availability matches the workload demands. To accommodate for mis-predictions on workload demands and nodes' availability and time-varying workload CPU utilisation, workloads are typically allocated with a higher than demanding share of nodes' CPU resources. Despite CPU over-subscription, workloads can still saturate nodes and the `CPU Ready` metric is able to capture at real-time when workloads are inadequately provisioned with CPU resources.

The empirical rule-of-thumb is to keep the `CPU Ready` values below a predefined threshold number, however there does not exist an exact number as this depends on setup particular, expected workloads, and hardware specifications. Normally, this can be successfully after trial and error during initial cluster deployment setup procedures. This is shown by many online sources, e.g., [36, 182, 94, 185] and this is inline with private discussions we had with the Company's system administrators regarding the available to us dataset. In a nutshell, one of the main innovations of this scheduler is the ability to *accurately* predict the upcoming `CPU Ready` *spikes* of a VM. We define a spike when the `CPU Ready` values exceed a certain threshold. The `CPU Ready` spikes are caused by CPU saturation and so they can be used to identify CPU performance bottlenecks. However, forecasting `CPU Ready` values is challenging [74], because: a) the `CPU Ready` values are highly variable and do not follow regular patterns, b) spikes of different values occur abruptly, and c) the values of the spikes vary significantly.

Our ultimate goal is to accurately predict `CPU Ready` spikes given past `CPU Ready` values and so to identify future CPU contentions. In past literature, the attempts to predict the occurrence of `CPU Ready` spikes followed two main approaches. The first approach revolved around forecasting `CPU Ready` values and then uses the predicted values to find spikes above a certain threshold. On the other hand, the second approach focused more on predicting the `CPU Ready` spikes directly using past spikes only. For both approaches a more systematic review and why traditional methods failed was conducted in [74].

5.3 A Federated Approach to Real-time Resource Monitoring

An important element in task scheduling is knowing the resource availability of all nodes across the data centre for globally informed allocation decisions. Maintaining such a global resource view is challenging and often centralised schedulers operate on cached data and distributed approaches work with different subsets of nodes to tackle large-scale scalability. To this end, we explore a different avenue for tackling this problem by exploiting recent advances on federated edge computing to *accurately* track both the individual node status ("specialised view") while also having the ability to get a holistic view of the system by intelligently synthesising data from different nodes. This approach allows each node to independently accept tasks, without the need for frequent global synchronisation.

Formally, we consider a data centre to be comprised out of M nodes each having finite computing resources and each node has to accept the maximum number of jobs without impacting their own overall responsiveness. Further, nodes produce a wealth of performance telemetry data used for scheduling, crash recovery, health monitoring, etc. However, the number of such metrics that each node outputs can be highly dimensional, thus making their understanding and exploitation for scheduling in a streaming fashion very difficult.

We exploit unsupervised learning techniques as they are able to discover hidden correlations within unstructured data, with minimal assumptions about the input underlying structure. Such methods, could eventually lead to improved allocation decisions with minimal user effort. Out of the many techniques available, Principal Component Analysis (PCA) [143, 100] is arguably the most ubiquitous one for discovering linear structure or reducing dimensionality in data, and so it has become an essential component in inference, machine-learning, and data-science pipelines. In a nutshell, given a matrix $\mathbf{Y} \in \mathbb{R}^{d \times n}$ of n feature vectors of dimension d, PCA aims to build a low-dimensional subspace of $\mathbb{R}^d$ that captures the directions of maximum variance in the data contained in $\mathbf{Y}$. Each of these captured directions are d-dimensional vectors called Principal Components (PC's) which contain linear combinations of the features exhibiting their contribution to the total magnitude of each PC. Conveniently, the PCs resulting from PCA are ordered in descending significance which is provided by the associated singular values. However, even if the iterates can capture most of the information within the data they still lie in d-dimensional space. This high dimensionality makes their interpretation difficult and limits their usefulness. To this end, and to effectively reduce its dimensionality, we exploit the resulting subspace estimate along with the incoming data in order to reveal the hidden patterns within the data. These patterns can then be leveraged to improve our scheduling decisions.

The general intuition behind our scheme stems from the observation that each PC can be *projected* to the incoming data, which yields a scalar value for each PC that can be tracked over time. This value is a key indicator of the PC magnitude evolution over time and can reveal hidden trends and capture fluctuations of the overall direction pattern without knowing precisely which feature contributed to it at any given time. Such trends and fluctuations could then be easily captured using traditional algorithms like averaging and Kalman filters. This is because for each PC its projection is a single scalar that is tracked over time.

More formally, we assume that each of the compute nodes produces at each time-step a feature vector in $\mathbb{R}^d$ that contains all the metrics gathered. Each node can be thought as part of a decentralised dataset $\mathcal{D} = \{\mathbf{y}_1, \ldots, \mathbf{y}_n\} \subset \mathbb{R}^d$ distributed across these M clients. The dataset $\mathcal{D}$ can be stored in a matrix $\mathbf{Y} = \left[\mathbf{Y}^1 | \mathbf{Y}^2 | \cdots | \mathbf{Y}^M\right] \in \mathbb{R}^{d \times n}$ with $n \gg d$ and such that $\mathbf{Y}^i \in \mathbb{R}^{d \times n_i}$ is *wholly owned* by each compute client $i \in [M]$. We assume that each $\mathbf{Y}^i$ is generated in a streaming fashion and that due to resource limitations it cannot be stored in full. Furthermore, under the DASM and as defined previously in Section 4.2, we assume that the M clients in the federation can be arranged in a computation graph with q levels and computation nodes only as its "leaves". The computation graph contains computation nodes and aggregators which can be joined to create an independent federation group. An example of such a computation graph is given in Figure 5.1. We note that such structure can be generated easily and efficiently using various schemes [190]. Moreover, we fully expect the computation graph to be shallow yet exhibit a very large fan-out which is typical for federated applications.

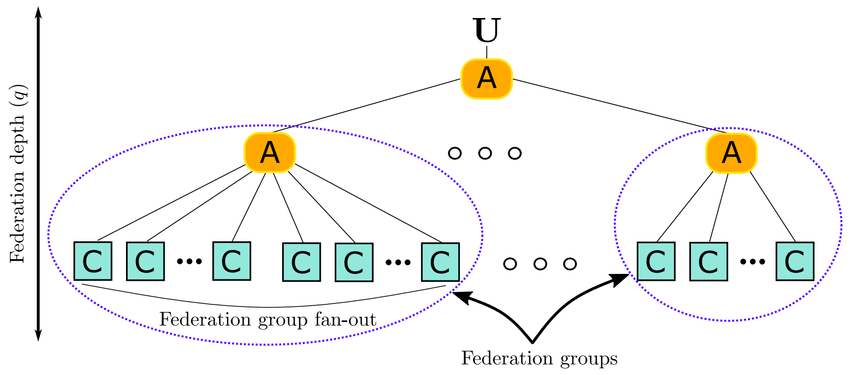


Fig. 5.1 An indicative federation network structure in which compute nodes (**C**) independently compute and make decisions based on the data they have seen so far. The updated subspaces are propagated upwards to aggregator nodes (**A**) upon completion of every block and the difference between the previous subspace estimate is below ϵ. Each subspace is propagated upwards until the root is reached. At this point we update the global estimate for the cluster workload seen thus far.

Local Iterates

We now introduce our local update scheme which is responsible for producing the iterates we use within our scheduler. The local iterates are produced by exploiting FPCA-Edge which was introduced in Section 4.2 which is also rank-adaptive. This is particularly convenient as it allows each of the processing nodes to adjust, independently of each other, their rank estimate based on the workload seen so far. Its iterates are then used by our scheduler and play an important role for deciding at any given time if accepting a new job at the node has the potential to impact its performance. That is because these iterates contain a “specialized” view tailored for each node specifically, which reflects its overall responsiveness at any given time. If the new job will severely impact the nodes’ performance according to our metrics discussed below, the scheduler rejects the job, otherwise the job is accepted.

To elaborate on the local update algorithm internals, let us consider a sequence $\{\mathbf{y}_1, \ldots, \mathbf{y}_n\} \subset \mathbb{R}^d$ of feature vectors and let their concatenation at time $\tau \le n$ be

$$\mathbf{Y}_{[\tau]} = \left[\begin{array}{cccc} \mathbf{y}_1 & \mathbf{y}_2 & \cdots & \mathbf{y}_\tau \end{array} \right] \in \mathbb{R}^{d\times\tau}. \tag{5.1}$$

A block of size $b \in \mathbb{N}$ is formed by taking b contiguous columns of $\mathbf{Y}_{[\tau]}$. Hence, a matrix $\mathbf{Y}_{[\tau]}$ with $r \le b \le \tau$ induces $K = \lceil \tau/b \rceil$ blocks. For convenience, we assume $K \in \mathbb{N}$, so that $\tau = Kb \in \mathbb{N}$. In this case, block $k \in [K]$ corresponds to the sub-matrix containing columns

$S_k = \{(k-1)b+1, \ldots, kb\}$. It is assumed that all blocks S_k are owned and observed exclusively by client $i \in [M]$, but that due to resource or time constraints can only store a small subset of them. Hence, once client $i \in [M]$ has observed $\mathbf{Y}_{S_k} \in \mathbb{R}^{d\times b}$ it uses it to update its estimate of the r principal components of $[\mathbf{Y}_{[(k-1)b]}, \mathbf{Y}_{S_k}]$. If $\widehat{\mathbf{Y}}_{0,r}$ is the empty matrix, the r principal components of $\mathbf{Y}_{[\tau]}$ can be estimated by computing the following iteration,

$$[\mathbf{U}_{k,r}, \mathbf{\Sigma}_{k,r}, \mathbf{V}_{k,r}^T] = \mathrm{SVDS}\left(\left[\begin{array}{cc} \widehat{\mathbf{Y}}_{[(k-1)b],r} & \mathbf{Y}_{S_k} \end{array}\right], r\right), \quad \widehat{\mathbf{Y}}_{[kb],r} = \mathbf{U}_{k,r}\mathbf{\Sigma}_{k,r}\mathbf{V}_{k,r}^T \in \mathbb{R}^{d\times kb}. \tag{5.2}$$

Its output after K iterations contains an estimate $\mathbf{U}_{K,r}$ of the leading r principal components of $\mathbf{Y}_{[\tau]}$ and the projection $\widehat{\mathbf{Y}}_{[\tau],r} = \mathbf{U}_{K,r}\mathbf{\Sigma}_{K,r}\mathbf{V}_{K,r}^T$ of $\mathbf{Y}_{[\tau]}$ onto this estimate. Naturally, the closer $\widehat{\mathbf{Y}}_{[\tau],r}$ is to $\mathbf{Y}_{[\tau],r}$ the better our embedding. Further, for every processed block each client keeps r PC projections such that $\mathbf{Y}_{b,r}\mathbf{U}_{k,r} \in \mathbb{R}^{r\times b}$ which allow us to track over the block duration if any anomalies (e.g. peaks) were found in any of the tracked PC's.

Global Updates

In this section we describe how the global federation of the embedding happens in our scheme. Notably, the only data structure that is propagated upwards is the actual embedding and no nodes can perform predictions other than the leaf (computation) nodes themselves which happens in real-time and allows the scheduling to be both timely and independent. To efficiently transfer knowledge of the workload embeddings seen so far, each node periodically requests a copy of the global embedding which can be merged against its local. Further, this strategy can also be employed for new or transient nodes as they join into the computation pool. Our algorithmic constructions are built upon the concept of *subspace merging* in which two subspaces $\mathcal{S}_1 = (\mathbf{U}_1, \mathbf{\Sigma}_1)$ with $\mathbf{U}_1 \in \mathbb{R}^{d\times r_1}$, $\mathbf{\Sigma}_1 \in \mathbb{R}^{r_1\times r_1}$ and $\mathcal{S}_2 = (\mathbf{U}_2, \mathbf{\Sigma_2})$ with $\mathbf{U}_2 \in \mathbb{R}^{d\times r_2}$ $\mathbf{\Sigma_2} \in \mathbb{R}^{r_2\times r_2}$ are *merged* together to produce a subspace $\mathcal{S} = (\mathbf{U}, \mathbf{\Sigma})$ with $\mathbf{U} \in \mathbb{R}^{d\times r}$, $\mathbf{\Sigma} \in \mathbb{R}^{r\times r}$ describing the combined r principal directions of $\mathcal{S}_1$ and $\mathcal{S}_2$ where $r = \max(r_1,\ r_2)$. One can merge two sub-spaces by computing a truncated SVD on a concatenation of their bases. Namely,

$$\mathbf{U\Sigma V^T} = \mathrm{SVD}([\lambda\mathbf{U}_1\mathbf{\Sigma}_1, \mathbf{U}_2\mathbf{\Sigma}_2], r), \tag{5.3}$$

where $\lambda \in (0, 1]$ a *forgetting factor* that allocates less weight to the previous subspace $\mathbf{U}_1$. In the previous chapter, through Merge (Algorithm 6) it is shown how eq. (5.3) can be further optimised, resulting in additional resource savings.

5.4 Pronto Scheduler

Herein, we propose a new task scheduler called PRONTO that is designed to accept an incoming job only if by doing so the performance of the existing running job(s) in the proposed node will not deteriorate significantly. This deterioration is measured by spikes in `CPU Ready`, as previously discussed. Formally, at any given time t, PRONTO decides, if by accepting a new job at time t will result in a `CPU Ready` spike in the next few intervals after t. In this section, we describe how by exploiting and extending the algorithmic constructions introduced in the previous chapter, namely Section 4.2, we can derive a federated and reliable task cluster scheduler by exploiting the `CPU Ready` metric.

The intuition behind PRONTO originates from our initial exploratory analysis and empirical observations on the available to us dataset that a spike in the top-r tracked projections is indicative of incoming `CPU Ready` spikes. Our proposed scheme exploits this observation by using it to compute a binary rejection signal based on the currently tracked embedding projections within each node. The rejection signal is raised if the weighted summation of the current nodes' tracked projections at time t exceeds a predefined threshold. This event, indicates there enough projections spikes which implies rapid change in the magnitude of the tracked principal components. Such rapid change is highly correlated with imminent `CPU Ready` spikes, which as previously discussed, are highly indicative of degraded node performance. An overview of the system architecture for the scheduling process within each node is shown in Figure 5.2.

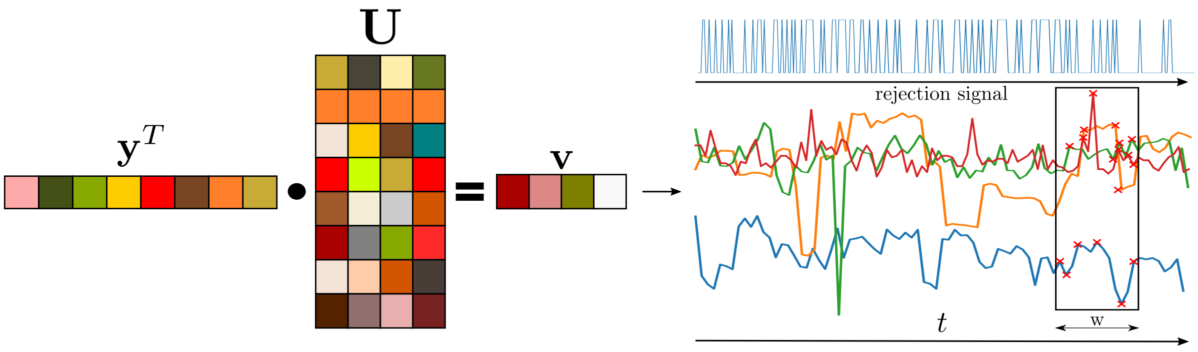
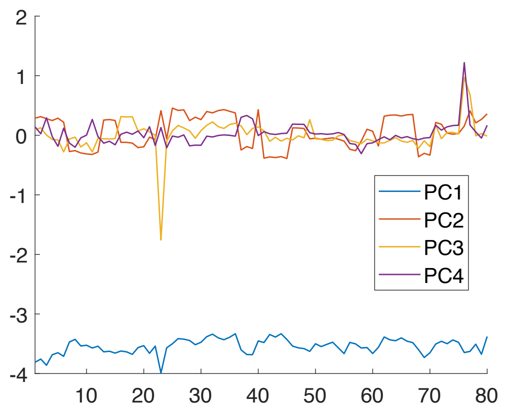
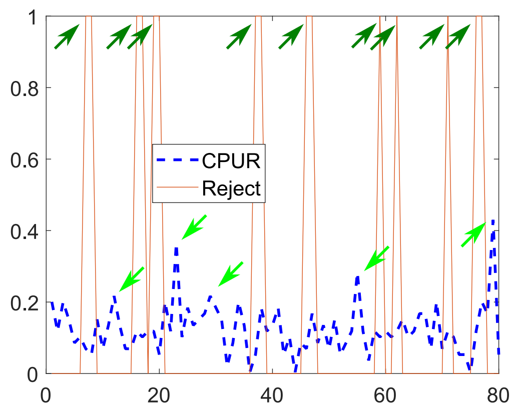


Fig. 5.2 Projection of incoming $\mathbf{y} \in \mathbb{R}^d$ onto embedding $\mathbf{U} \in \mathbb{R}^{d \times r}$ producing r projections in $\mathbf{v} \in \mathbb{R}^{1 \times r}$. These projections are then tracked over time for spikes which form the basis of our rejection signal. The sliding window for spike detection for each projection is of size $\mathbf{w}$ as is shown in the figure.

To better illustrate the way the projections, the rejection signal, and the `CPU Ready` signal work together consider Figure 5.3. At its left, we can see the tracked projections over time for a particular node, for which we see various rapid changes to them over time. On the right, we observe the rejection signal over the same period of time along with the detected spikes of `CPU Ready`. In this instance the spike threshold for `CPU Ready` is set to be below .2 and the rejection signal is produced by observing the spikes in an online fashion of the projections shown in Figure 5.3a. Provided with this, we can observe that each of the detected `CPU Ready` spikes is *preceded* by at least one raise of the rejection signal within few timesteps of its occurrence. Note, that consecutive `CPU Ready` spikes might indicate that for the next few intervals the node will experience deteriorated performance and thus we cannot possibly accept a job - which is precisely what PRONTO exploits.



(a) PC Projections



(b) Rejection and CPU-Ready Signals

Fig. 5.3 Left (fig. 5.3a): An example of the projections, Right (fig. 5.3b): An example of the rejection signal based on the projections seen in the left drawn concurrently with the `CPU Ready` signal serving as the ground-truth. Further, *green* and *teal* arrows depict spikes in the rejection and `CPU Ready` signals respectively. Our goal is to show that spikes in the rejection signal *precede* spikes in `CPU Ready` over time.

Elaborating, each node at a given time t uses its own subspace iterate $\mathbf{U} \in \mathbb{R}^{d \times r}$ as produced by FPCA-Edge and each incoming vector of features in $\mathbb{R}^d$ is projected into it to produce the projections of that node at time t. As mentioned previously for each of the tracked projections, we monitor their abrupt changes (*i.e.*, the signal spikes) in a streaming fashion over a sliding window of size w. Our spike detection algorithm is based on a z-score based scheme put forth in [24] which we implement within our rejection signal computation. The results of the detection are stored as r binary variables setting each to 1 if a positive and to -1 if a negative spike is detected for that projection at time t while being 0 otherwise. Moreover, at each time t we compute the weighted sum of these r binary variables multiplied by their associated singular value as, $R_s = \sum_{i=0}^{r} r_{i,t}\sigma_{r_i,t}$ where $r_{i,t}$ and $\sigma_{i,t}$ the i-th binary variable and i-th singular value at time t respectively. Then, the rejection signal value at time t is set to 1 if the weighted sum of R_s is above a certain threshold and 0 otherwise; throughout our experiments we set the threshold value to be equal to 1. We present the implementation of Reject-Job in Algorithm 11.

This principled approach in predicting `CPU Ready` spikes is enabled due to the properties inherited by the incremental embedding computation as described in Section 4.2 and specifically Algorithm 10. Exploiting the algorithm introduced previously, not only provides us with concrete guarantees with respect to the embedding quality but is also systematic and deterministic with the only variable left to tune being the threshold for raising the rejection signal. Note, that in this instance we do not require the differential privacy aspects provided by FPCA-Edge and thus the perturbation masks are disabled throughout.

Algorithm 11: Reject-Job

```
Data: $\mathbf{U} \in \mathbb{R}^{d \times r}$: embedding estimate at time $t$, $\mathbf{\Sigma} \in \mathbb{R}^{r}$: embedding singular values at
      time $t$, $\mathbf{y} \in \mathbb{R}^{d}$: observed data-point at time $t$, $\mathbf{w_p} \in \mathbb{R}^{r \times \text{lag}}$: the dampened
      signal, $\mathbf{w}_{\text{avg}} \in \mathbb{R}^{r \times \text{lag}}$: the average filter, $\mathbf{w}_{\text{std}} \in \mathbb{R}^{r \times \text{lag}}$: the std filter
Result: True if a job should be rejected at $t$, false otherwise.
Function Reject-Job($\mathbf{U}, \mathbf{\Sigma}, \mathbf{y}, \mathbf{w_p}, \mathbf{w}_{\text{avg}}, \mathbf{w}_{\text{std}}$) is
    /* Init.  */
    $\text{lag} \leftarrow 10$, $\alpha \leftarrow 3.5$, $\beta \leftarrow 0.5$, $\mathbf{b} \leftarrow 0_{1 \times r}$, $\text{tr} \leftarrow 1$
    /* Compute projections */
    $\mathbf{p} \leftarrow \mathbf{y}^T \mathbf{U} \in \mathbb{R}^{1 \times r}$
    /* Has lag buffer filled?  */
    if $\text{len}(\mathbf{w}_{\text{avg}} < \text{lag})$ then
        return false
    end
    /* Find peaks for each projection */
    for $i = 0; i < \text{len}(\mathbf{p}); i{+}{+}$ do
        if $\text{abs}(\mathbf{p}[i] - \mathbf{w}_{\text{avg}}[i][\text{lag} - 1]) > \alpha \mathbf{w}_{\text{std}}[i][\text{lag} - 1]$ then
            /* Peak detected, check sign */
            if $\mathbf{p}[i] > \mathbf{w}_{\text{avg}}[i][\text{lag} - 1])$ then
                $\mathbf{b}[i] \leftarrow 1$
            else
                $\mathbf{b}[i] \leftarrow -1$
                /* Reduce $\mathbf{w_p}$ influence */
                $\mathbf{w_p}[i][\text{lag}] \leftarrow \beta \mathbf{w_p}[i][\text{lag}] + (1 - \beta)\mathbf{w_p}[i][\text{lag} - 1]$
            end
        else
            /* No peaks */
            $\mathbf{b}[i] \leftarrow 0$
            $\mathbf{w_p}[i][\text{lag}] \leftarrow \mathbf{p}[i]$
        end
        /* Adjust the buffers */
        $\mathbf{w}_{\text{avg}}[i,:] \leftarrow \text{mean}([\mathbf{w}_{\text{avg}}[i, 2:], \mathbf{p}[i]])$
        $\mathbf{w}_{\text{std}}[i,:] \leftarrow \text{std}([\mathbf{w}_{\text{std}}[i, 2:], \mathbf{p}[i]])$
    end
    /* Do we reject a job?  */
    if $\sum_i (\mathbf{b}[i]\mathbf{\Sigma}[i])$ greater or equal than tr threshold then
        return true
    else
        return false
    end
end
```

In order to see how the spikes are detected in practice, we can observe what happens to a node upon initialisation. This is depicted in Figure 5.4 which shows three indicative snapshots at different timesteps that illustrate how spike detection works in practice. As we previously said we cannot start making predictions until at least **w** observations have been processed. At that point, we set out a reference point through which we set our prediction horizon. This is shown in the first row of the figure. In the second row of the same figure we can see that a full window has been observed and thus we can start to reliably detect potential spikes. This relative point is essentially what PRONTO considers its current time. This means that the spikes happen to the left of the reference point are considered to be incoming predictions while the ones on the right side are considered to have already happened in the past. This segmentation is shown in the third row of the aforementioned figure.

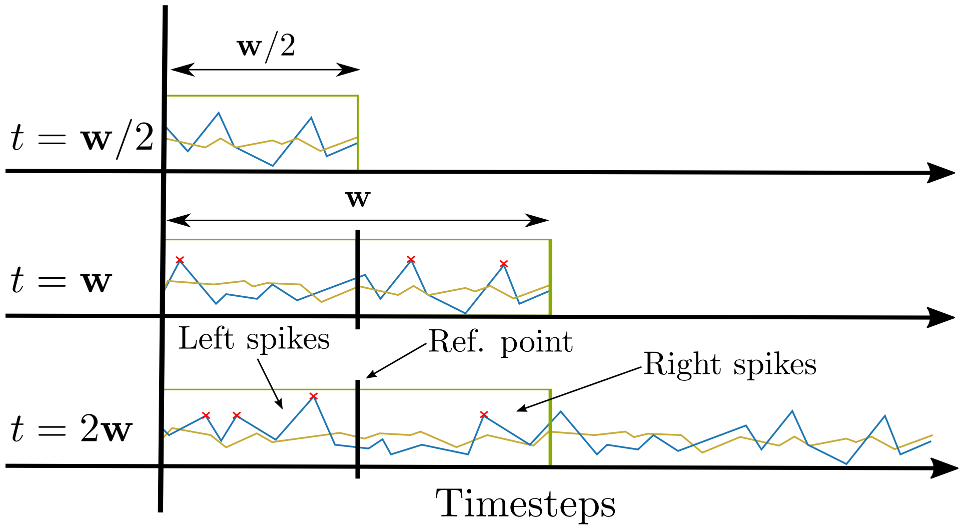


Fig. 5.4 Illustration using two projection signals over three indicative timesteps, namely $t = \{\mathbf{w}/2, \mathbf{w}, 2\mathbf{w}\}$. It shows how the sliding window operates and indicates that the minimum observations required to start predicting is equal to the size of our window **w**. In the final row we show where left and right sided spikes reside with respect to the reference point, which also acts as the current time for the scheduler. These spikes for each projection - after **w** observations - are then used to compute the rejection signal as described previously.

Having fully described how the rejection signal is computed, we now describe our federated scheduling scheme. To do so, we will exploit the technique we previously introduced in Chapter 4, namely Federated-PCA. Practically, we aim to establish a principled way that the iterates can be propagated upwards in order to be able to generate a holistic view of the system exactly like the way Federated-PCA is designed to do. Provided these details, we are now ready to present our PRONTO scheduler implementation which is depicted in Algorithm 12.
Note, that in our current setting and in order to save bandwidth we can elect to propagate only if the estimate has changed above a certain threshold. To achieve that, we employ a heuristic which checks if the absolute weights of the subspace iterate surpassed the set threshold.

Algorithm 12: Pronto scheduler

```
Data: $\mathbf{Y} = \left[\mathbf{Y}^1 | \cdots | \mathbf{Y}^M\right] \in \mathbb{R}^{d \times n}$: Data vectors
$(\alpha, \beta)$: Upper and lower bounds on $\mathcal{E}(\mathbf{Y})$
$b$: Batch size at the client level
Function Pronto($\mathbf{Y}, b \mid \alpha, \beta$) is
    /* Initialise clients */
    Each client $i \in [M]$ :
        Initialises PC estimate to $(\mathbf{U}^i, \mathbf{\Sigma}^i) \leftarrow (0, 0)$
        Initialises batch to $\mathbf{B}^i \leftarrow [\,]$
    end
    Arrange clients' merge paths in a computation graph such as in Figure 4.1
    /* Computation of local updates */
    At time $t \in [T]$ , each client $i \in [M]$ with job set $[J]$
        Observes data-point $\mathbf{y}_t^i \in \mathbb{R}^d$
        Adds $\mathbf{y}_t^i$ to batch $\mathbf{B}^i \leftarrow [\mathbf{B}^i, \mathbf{y}_t^i]$
        Accepts job $j' \in [J]$ based on Reject-Job
        if $\mathbf{B}^i$ has $b$ columns then
            $(\mathbf{U}^i, \mathbf{\Sigma}^i) \leftarrow$ FPCA-Edge
            $\mathbf{B}^i \leftarrow [\,]$
        end
    end
    /* Merge only if needed */
    if absdiff$(U^{t,i}, U^{t-1,i}) > \epsilon$ then
        Use Algorithm 6 to merge subspaces recursively within the computation graph
         (Fig. 5.1)
    end
end
```

5.5 Experimental Evaluation

This section is focused on the empirical evaluation of PRONTO against real-world traces. In a nutshell, the primary goal of this evaluation is to quantify the efficiency of our scheme to predict the `CPU Ready` spikes based only on the Company's unstructured trace observed from each of the compute nodes. To do so, we use a real-world dataset of performance data traces gathered from the data centres of a global bank organisation (hereafter referred to as the Company) from 2012. The dataset includes time-series VMware performance data from the Company's virtualised data centres. The Company collected such data across their virtualised infrastructure and stored it on NFS servers but derived little further value from it. The available to us dataset contains performance data from 100 virtualised clusters in a data centre over four weeks. Each cluster has about 14 VMware ESX hosts, supporting 250–350 VMs. The performance data consists of metrics output by the VMware ESX hypervisor every 20 seconds related to the CPU, memory, disk and network resource consumption of physical hosts and

VMs. There are 134 different resource metrics for a typical ESX host in our dataset, and 52 metrics for a VM. The total size of the dataset is 1TB

Formally, we use PRONTO to predict at any given time t that a raise in the rejection signal occurs shortly before or coincides with an observed `CPU Ready` spike contained in a sliding window of size w. This enables us to accurately predict incoming `CPU Ready` spikes which we can use to prevent further accepting additional jobs on a node with performance shortages. We use the Company's dataset trace to evaluate our approach. The rejection signal is then compared with the baseline (i.e., the actual `CPU Ready` values) and we check if a spike is indeed detected in `CPU Ready` and if that's reflected in the rejection signal. If the rejection signal was *raised* a few timesteps before or coincides with the actual `CPU Ready` spike we classify it as a successful prediction. As noted in the previous section, the window w of timesteps can be easily adjusted. However, throughout our evaluation we observe that values close to ten timesteps give us good performance so we use this value for all experiments.

Practically speaking, to generate the rejection signal we only need the actual embedding ($\mathbf{U}$) and its singular values ($\mathbf{\Sigma}$). The actual subspace embedding $\mathbf{U}$ can be generated using a plethora of methods which we also evaluate. We evaluate against, in addition to Federated-PCA Edge, with SPIRIT [142] (SP), frequent directions [118] (FD), and finally power method [130] (PM). These algorithms represent the state-of-art in the area, are well-established, and each uses a different mathematical approach into tackling this problem. Unfortunately, none of these algorithms are neither inherently distributed and, apart from SP, rank adaptive. We note however, that the singular values are needed for PRONTO to work in a truly federated setting which can only be reliably produced by Federated-PCA Edge and partially by [142]. Concretely, SP is able to produce singular values but without any guarantees about their quality, while both FD and PM lack the ability to produce any. For the methods that were not able to generate their own singular values we used predefined values for them that were generated using an exponential decay spectrum, namely $\sigma_r = 1/r$. While not exact, this approximation enables us to test against all of the competing methods. We now set forth to present our simulation results.

Simulation Results

As we discussed before, our goal is to evaluate how efficiently incoming `CPU Ready` spikes can be predicted and so we use the actual Company's trace as our baseline. PRONTO is designed to operate using a sliding window of size w for which we use the Reject-Job algorithm to decide if at time t the node can accept an incoming job or not. In reality, this sliding window imposes a slight "lag" between the observed values and actual prediction time when the scheduler has to decide. We elect to at least see one window in order to make a prediction which equates in our case to w timesteps throughout our experiments. We believe this delay will, in practice, be insignificant as the rate of incoming trace observation is much higher than the number of incoming jobs. Note also that typical window sizes in practical applications should range between $10 - 50$ timesteps[2]. Further, throughout our experiments we use a rank r equal to 4. Evaluation on higher values provided little to no benefit in terms of prediction quality with the added downside of increasing computation cost.

Before presenting the results we need to describe how we define a successful prediction. Hence, in this context we classify a successful prediction if a `CPU Ready` spike is preceded by *at least* one rejection signal raise within the current window. Moreover, we use the reference point within the window which equals half of the window size, namely $\mathbf{w}/2$. Thus, at any given time we can classify the detected spikes into *left* and *right* based on their location in the current window relative to our reference point.

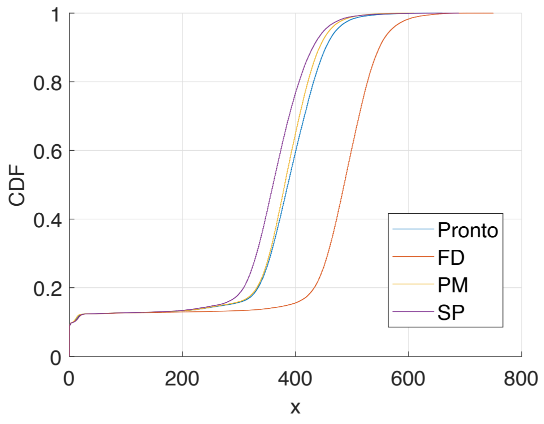
Formally, we measure at each timestep t how many spikes are contained within the current sliding window as well as their "side" with respect to the reference point t. We also measure the overall downtime, which is the amount of time the rejection signal is raised during our evaluation. This reflects the overall job acceptance availability at each node. Ideally, we would like to have the rejection signal raised before or coincide with a `CPU Ready` spike, but also minimise the downtime so that the node can accept more jobs.

We start by quantifying the spike behaviour and we evaluate the types of spikes observed throughout the trace and the empirical Cumulative Distribution Functions (CDF) are presented in Figure 5.5a and Figure 5.5b for *left-* and *right-sided* spikes respectively. We note, that *left-sided* spikes are the most important ones, as these indicate that a `CPU Ready` spike is imminent in the next few timesteps. *Right-sided* spikes can be an indication of consecutive `CPU Ready` spikes or delayed detection; this is because `CPU Ready` spikes might occur consecutively or very close to each other thus indicating a significantly deteriorated node.

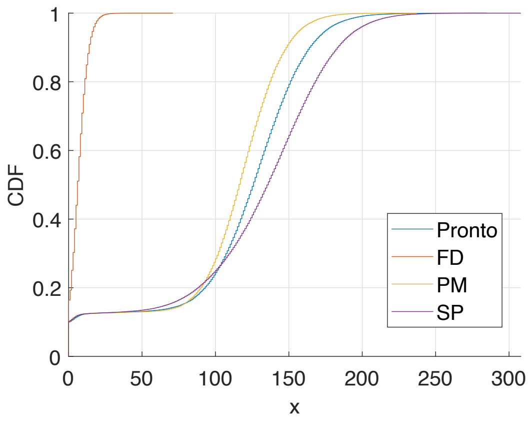
The above Figures show that the *left-sided* spikes are considerably more frequent than the *right-sided* ones. This means that we are able to detect incoming `CPU Ready` spikes with high accuracy. Notably PRONTO and FD have the highest number of *left-sided* spikes, followed by PM and SP. Although our goal is to predict `CPU Ready` spikes as accurately as possible we also want to have the highest availability (i.e., nodes *can* accept jobs) for a higher overall utilisation. To quantify this we need to show the CDFs of the overall downtime and the contained spike percentages across the traces; this is shown in Figure 5.6a and Figure 5.6b for the downtime and contained spike percentages respectively.

These figures indicate that PRONTO, SPIRIT, and PM have very little downtime compared to the actual spikes detected and thus are able to utilise each compute node more efficiently than FD. Notably, we observe that FD performs poorly as its downtime is greater than half of the total time - which means it could be similar to using a random scheduler. The contained spike percentage CDF shows the amount of total spikes that are contained by all methods considered in this evaluation. In this context, values greater than 100% show the proportion of

[2]This applies to PRONTO itself, SPIRIT, and FD. PM needs to have a block size at least equal to the dimensionality of the data, which necessitates a larger window than the other methods.



(a) *Left-sided* peaks CDF



(b) *Right-sided* peaks CDF

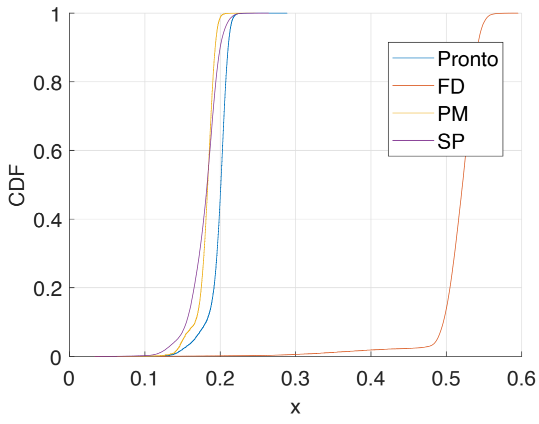
Fig. 5.5 Left: The empirical CDF of the number of *left-sided* spikes. Right: The empirical CDF of the *right-sided* spikes. Both are measured against our reference point at time t.

the spikes detected over the actual `CPU Ready` signal spikes. This could either indicate that these methods detect more congestion points or overestimate and raise the rejection signal without having to do so. Most methods are always near or over 100% and the skewing present in the graph is mostly attributed to FD. However, this is to be expected as it is also reflected by the downtime percentage CDF, that indicates FD has almost constantly the rejection signal raised - thus accepting minimal jobs.

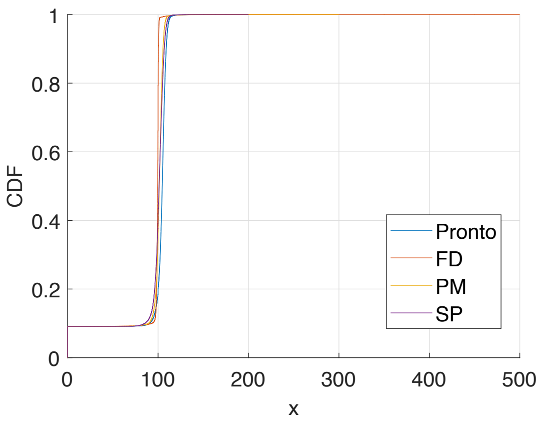
Performance

Another important consideration is the performance of each method. Our prototype implementation is developed using `Python` employing standard libraries and established packages (e.g. `scipy`, `numpy`). To this end we measure the performance to update the predictions for each incoming data vector. Note, that, even if the embedding is updated *per block* the actual predictions job acceptance happens *per data vector*. We amortise the cost of block methods by averaging the running time over each block to extrapolate the cost to perform the rejection signal computation per incoming data vector. For memory we opt to use the maximum amount per block or per vector required. However due to the built-in methods use of "slack" space all costs end up being fairly similar. Further, to avoid confusion we report rounded numbers. The results for all methods considered are shown in Table 5.1.

All of the considered algorithms do not consume a lot of resources and predictions happen in near real-time as they are updated every second (or around 100ms). However, we note that even with very small matrices when using built-in functions `splinalg.svds` and `linalg.svd` of `scipy` package the required memory was about $\approx$ 100 MB at its peak without increasing much even for larger ones. That is, we observed that the memory difference for performing SVD for 40×7 and 300×7, was $\approx$ 0.45MB as measured using the `memory_profiler` module.



(a) CDF of the percentage of time that the rejection signal was raised



(b) Contained peak percentage CDF

Fig. 5.6 Left: The empirical CDF of the percentage of the contained spikes, as we can see all methods perform desirably containing almost all spikes. We note that, values above 100% indicate that methods detected more spikes than the ones detected by just using CPU-Ready. Right: Shows the empirical CDF of the *downtime* of the rejection signal per method, indicating the percentage of the total time a compute job is able to accept an incoming job.

Table 5.1 Extrapolated average execution time (ms) per timestep (*i.e.*: per vector that the rejection signal is computed) and approximate mean memory allocation to process each block/vector, including any overheads (MB).

	Execution time	Memory allocation (rounded)
PRONTO	**15** ms	≈ **148** MB
PM	**22** ms	≈ **155** MB
FD	**25** ms	≈ **151** MB
SP	**9** ms	≈ **123** MB

We conjecture that this is due to the use of generous slack space and debug symbols used by `Python`. However during our experiments memory consumption never reached over ≈ 150 MB. Practically speaking, the most expensive operation throughout the whole pipeline is the computation of SVD. Though, given the fact that since all methods are able to exploit the truncated SVD instead of the normal one we are able to keep resource runtime requirements reasonable and able to update the estimates using either per vector or per small blocks. Finally, we note that the measurements are taken with debug symbols enabled and no particular optimisations; thus potential optimisations could help bring the cost further down.

5.6 Discussion

Today's data centres (especially virtualised) generate a plethora of real-time telemetry performance data consisting of fine-grained time-series measurements such as CPU utilisation, memory consumption, and I/O transactions. This data uniquely captures real-time behaviour of the data centre and can be used for resource predictions. Numerous related works highlight the importance to accurately analyse this data for resource prediction and modelling. We further discuss the most relevant works below.

VM resource management and prediction. In the past, there had been a lot of interest in the area of autonomic VM resource management and prediction using *feedback control*, mostly targeting small-scale cluster deployments.

Early works [187, 199, 141] use non-linear controllers to regulate the relative CPU utilisation across VMs under contention. The problem of resource contention in shared virtualised clusters is also studied by [119]; their approach allocates CPU resources based on a response time ratio between workloads on saturated VMs. For instance, [140] use a second-order ARMA model to capture the relationship between resource utilisation and application performance. [17] address the problem of boot storms when multiple VM workloads start to execute at the same time. They propose a feedback approach to control the concurrency level, improving end-to-end latency. [103, 104] propose linear feedback controllers that are based on the Kalman filtering technique and a widely applicable model of CPU utilisation. These controllers use online measurements to predict future demands and configure their parameters. The use of the min-max $\mathcal{H}_\infty$ filters to minimise the maximum error during under-provisioning was explored in [31]. Although the above approaches use resource utilisation data to derive performance models in a real-time manner they however target small-scale virtualised clusters.

The previous feedback control-based approaches ignore regular long-term patterns, which may exist in the time-series data and could potentially assist with performance management issues. For example, [193] present a predictive controller that regulates the relative utilisation of a single-tier virtualised server based on three time-series prediction algorithms (AR autoregressive model, the ANOVA decomposition and the MP multi-pulse model). Their results show that, when utilization exhibits regular patterns, their predictive controller outperforms the relative-utilisation feedback controller proposed by [187].

Additionally, a wavelet-based approach was proposed by [136] for online demand prediction of resource utilisation. The advantage of such an approach is the decomposition of the original signal into multiple detailed signals that capture different patterns and finally are synthesised into an approximation signal for predictions. Their approach uses a history of the past several minutes to generate a new model. To make short-term predictions in the absence of long-term patterns, PRESS by [72] combines state or signature-based pattern predictors for resource allocation using previous utilisation measurements. CloudScale by [161] targets the problem of fast reaction to hot-spots in deployments. It combines online adaptive padding based on burst detection with additional allocation correction using feedback from SLO violations and relative utilisation. While this works efficiently for patterns reported in training data, it is unclear if CloudScale can handle scaling of multiple metrics and application with multiple components with correlated utilisation.

The above predictive approaches show that resource utilisation predictions are possible when considering long-term history of a measured metric and that this could assist towards resource management problems. However, these approaches have either focused on short-term prediction with a small history or long-term patterns as found in a single or small group of monitored resource utilisation traces. More recent works recognise the need to analyse telemetry data in the large-scale. Most notably, Microsoft's Recourse Central gathers all VM telemetry data on a centralised cluster and it focuses on offline analysis for predictions to tackle servers' over-subscriptions [40]. When compared to these works, PRONTO is the first approach to target real-time predictions of the `CPU Ready` spikes using data from large-scale deployments.

Further, there exists different centralised and distributed approaches in the area of large-scale resource management. In the remainder of this section, we will outline the key ideas behind each and describe a number of seminar works for each. We note that our approach does not strictly belong to any of the two categories but shares similarities with distributed approaches, as it operates on a more relaxed model.

Centralised schedulers: Centralised schedulers gather performance data from nodes periodically to take globally informed scheduling decisions. In doing so, they have a global view of the data centre availability and so ultimately they could take well informed, even optimal, allocation decisions under specific constraints and performance goals, e.g., Decima [123], Firmament [69], Quincy [95], Resource Central [40], Google's Omega [160], and Tetrisched [173]. For example, Google's Borg centralised scheduler provisions for thousands of machines per cell by employing a similar to Omega's cached state [160] and a number of heuristics such as score caching and relaxed randomisation [178]. More recently, Microsoft's Resource Central uses performance prediction based on workload characteristics and machine learning to increase resource utilisation but requires gathering of resource utilisation data in centralised locations [40]. Although powerful, centralised schedulers do not scale easily and often employ heuristics that decrease allocations' efficiency. Their approach increases the overall end-to-end processing and adds traffic to the data centre network. In fact, related papers do not adequately discuss their approach to maintaining a consistent central view of the data centre, e.g., [69, 40, 123]. Furthermore, their decisions are based on old, cached data as by the time solutions are found and new allocations are in place, resource utilisation at the server nodes have changed risking to make scheduling decisions obsolete, e.g., Borg [178].

Distributed approaches: To handle scalability, fault-tolerance, and speedup scheduling time, different decentralised schedulers have been proposed. Popular systems such as Kubernetes [30], Mesos [89], Autopilot [157], Yarn [177], Sparrow [139], Medea [63], and Apollo [20] provide scalable and fault-tolerant managers for scheduling but they do not work on a global view of the data centre. Their focus is on engineering robust, practical and scalable frameworks tailored for specific workloads with very good performance. However, we would like to generalise and thus PRONTO is the first scheme to use unsupervised learning techniques in the context of scheduling and shows that is able to predict saturation indicated by the `CPU Ready` metric and uses several innovations to do so. Firstly, it relies on method that was presented in Chapter 4 to accurately update the local estimates that reside within each node. These iterates are then exploited by projecting incoming traces onto them, which results in the generation of the projections reflecting the overall trend of each of the Principal Components contained in that iterate. Finally, these insights are used to generate a binary rejection signal which unlocks the ability for each node to perform scheduling decisions independently. To the best of our knowledge, this is the first scheme that exploits projection tracking in the scheduling domain and is able to accurately predict `CPU Ready` spikes in the unsupervised setting within minimal assumptions over the input data.

Chapter 6

Reflections and outlook

Understanding the intricate underlying structure within a given dataset has been long been sought after and is thought to be one of the most fundamental research questions posed over the years. Answering this question has been the focus of many scientific disciplines attempting to tackle this problem from various perspectives with rich literature spanning decades. This is because such discoveries can unlock previously unknown patterns or provide novel insights paving the way for important scientific advancements. Such prescience holds the potential to have numerous economical, societal, and scientific implications.

More importantly, in recent years these implications have been further amplified by the rise of data-driven decisions that many governments, companies, and policy makers have shifted towards. Naturally, these have been primarily guided by insights, patterns, and sagacity extracted out of datasets of interest. However, practical analysis of the aforementioned data sources has proven challenging in numerous occasions, primarily due to the inherent scalability issues with traditional analytical tools. Further, another important factor is the that conventional algorithms for data analysis were designed in an era when privacy was an afterthought, rather than an essential feature. This is further motivated, as a large percentage of datasets involved for critical decision making stems from privacy sensitive-domains, such as health, socioeconomic status records, and movement tracking data just to name a few.

In this dissertation introduced several innovations that aimed in advancing the state of the art within one of the most cardinal research domains in large-scale data analytics, which is dimensionality reduction. In the context of this work, the focus of our contributions was in arguably the most ubiquitous method at our disposal to perform that task, namely, Principal Component Analysis (PCA). Concretely, through the contributions presented herein, we were able to scale PCA and make it applicable to both traditional, but more importantly, to federated datasets. This allowed us to create scalable reductions of massive datasets under practical constraints, such as the ability to offer efficient computation, adaptability, and guarantee privacy which are commonly encountered in real-world data analysis tasks.

6.1 Summary of contributions

In this section we will summarise and reflect on the contributions presented within the context of this dissertation. We begin by discussing the contributions put forth in Chapter 3, which essentially started as an exploration on how to leverage techniques to accelerate matrix factorisation techniques in a streaming setting. Unfortunately, most techniques around that topic explicitly require the input data to admit specific underlying structure or exhibit sparsity. On the other hand, methods such as Singular Value Decomposition (SVD) work under minimal assumptions and have been popularised due to their optimality guarantees as well as their utility. However, this optimality comes at a cost. Traditional SVD requires significant resources both in terms of storage as well as computation. This necessitated the introduction of approximate techniques for its calculation resulting in dramatic computational gains. Nevertheless, there was still a gap in the literature as none of the previous approaches was able to operate using *thin* blocks of data nor provide concrete guarantees about the produced iterates. One of the first contributions of this dissertation was to provide such a solution in the form of a Memory-limited Online Subspace Estimation (MOSES) algorithm that attempts provide these highly desirable traits. In a nutshell, our solution exploited the idea of using *thin* blocks of data and adapted incremental r-truncated SVD to be able to operate using such blocks instead. We demonstrated both theoretically and empirically that our proposed algorithm was almost the same in terms of performance to the offline r-truncated SVD, which is assumed to have infinite resources for its computation.

However, even though our proposed scheme had the benefit of online computation while exhibiting state of the art performance, failed to scale horizontally. Meaning that, as initially formulated, MOSES could not be used in distributed environments or leverage multiple computation nodes. This inherent limitation stemmed from the fact that MOSES was designed to expect the dataset input vectors to be streamed from a central location in order to successfully produce its iterates. Therefore, this significant impediment, rendered the proposed solution inapplicable for distributed computation or federated datasets, thus constraining its use-cases to single-node scenarios. Another drawback of our initial approach, was that MOSES required a hyper-parameter to be provided for the "estimated" rank of the dataset and could not be adjusted during its execution. This could result into sub-optimal choices for the target rank as, in a streaming setting, one could not possibly know its value exactly upon initialisation and thus could only be estimated. Naturally, this could prove problematic in the event of distribution drifts or phase transitions within the incoming data that could potentially necessitate on-the-fly rank adjustments of the estimates. Moreover, our initial formulation does not offer the ability to guarantee differential privacy, which is a highly desirable feature.

We attempted to address all of the aforementioned limitations in Chapter 4, by introducing a novel algorithm for Federated-PCA that required several innovations. We started by using the aforementioned primitives presented in Chapter 3 and reformulated the problem in a recursive manner. In doing so, it forced us to rethink the problem in a different way and introduced two novel algorithms; one that is able to merge two subspaces together and one that allowed incremental update of the iterate estimates. In practice, this was what facilitated the federated computation of PCA as decoupling the iterate updates from the actual merging resulted into the reliable and federated aggregation of the produced iterates. Another property we introduced, that is particularly useful in a federated setting, is time-independence of the end result in the absence of perturbation masks. Essentially, we provided the first proof that the merging of the iterates will result in the same global iterate regardless of the order of merging thus enabling the asynchronous computation of each subproblem. However, until now we have not addressed the limitation of the hyper-parameter required by MOSES for the rank estimate. To address this, we exploited a heuristic and proposed a novel solution that attempts to adaptively estimate the rank over time by monitoring the ratio of the last singular value against their summation and ensuring it remains within a certain range. This provided us with the ability to detect potential distribution shifts or rapid changes and adjust the rank estimate of the iterates as required. To ensure robustness of our approach, we also provided concrete bounds about the reconstruction quality for both local as well as global estimates.

So far, we have discussed how we addressed most of the limitations regarding the method presented in Chapter 3. Their resolution, is what enabled the computation of PCA in a federated setting, albeit without any differential privacy guarantees. To provide (ε, δ)-differential privacy we used an input-perturbation scheme in which the covariance matrix of a dataset is perturbed using a non-symmetric random Gaussian matrix. However, to enable this we had to extend the state of the art and provide three distinct contributions, which we will now summarise. Firstly, we refined the popular MOD-SuLQ algorithm to be applicable when using non-symmetric matrices but also able to work in a streaming setting, which was an essential properly in the context of federated computation. Secondly, we improved the current state-of-the-art lower bound of the variance required in order to guarantee differential privacy when using non-symmetric matrices, while also offering similar asymptotic guarantees. Thirdly, we provided sample complexity bound for guaranteeing differential privacy and thus delivered an exact sample horizon onto when we would be able to allow the release of differentially-private iterates. We noted that the sample complexity bound yielded was dependent only with respect to the ambient dimension (d) and the privacy budget parameters (*i.e.* ε and δ). This is particularly useful in a streaming setting, where the exact samples contained in a dataset are unknown upon initialisation. By alleviating all of the limitations discussed so far, we were able to offer the first complete mathematical framework for the computation of PCA in the combined federated, model free, and differential private setting.

In the final chapter, we presented a novel practical application of the methods previously introduced. Concretely, in Chapter 5 instituted a federated, asynchronous, memory-limited algorithm for online task scheduling that is applicable to data centres. The basis of our approach exploited Federated-PCA as described in Chapter 4 and used its ability to incrementally compute the iterates of each subproblem. Through the proposed scheme, we unlocked the ability for every node to execute scheduling decisions independently on whether to accept an incoming job based on the workload seen thus far. The workload seen thus far was reflected within the local iterates produced by each node. However, even if the iterates could indeed capture most of the information within the data they still resided in d-dimensional space. To this end, and to effectively reduce its dimensionality, we tried to exploit the resulting subspace estimate along with the incoming data in order to reveal the hidden patterns within the data. These patterns could then be leveraged to improve each nodes' scheduling decisions. Further, we can utilised the federated properties in order to generate a "global" view of the system by aggregating the iterates, as needed. In turn, this aggregated iterate could yield a holistic perspective of the system and potentially indicate its overall responsiveness. Through our empirical evaluation on a large-scale real-world dataset of traces gathered from a production data-centre we validated the practicality of our approach. More specifically, it was able to predict changes in the system responsiveness ahead of time based on industry standard metrics and, in turn, could lead to better scheduling decisions and overall utilisation of the available resources. Notably, we noted that our proposed task scheduling algorithm was explicitly designed to be applicable to both traditional as well as federated data centres of the future.

6.2 Future research directions

The contributions presented in this dissertation unlocked the ability to perform the federated computation of PCA in a tractable and reliable fashion while also indicating a potential practical application in the form a federated task scheduling algorithm. Yet they allude to a large set of unexplored research directions, some of which we outline in the remainder of this section.

One of the primal questions that can be posed, is how to extend the proposed algorithmic frameworks in the setting of missing values. Keeping track of the embeddings accurately and effectively is a line of work that resonates closely with the contributions in this thesis. Indeed, lately there has been a resurgence of studying how to compute traditional methods while being able to tolerate missing values. This is because in many practical scenarios, at each timestep, only a small subset of the data features may be observed, be it due to hardware limitations, resource constraints, privacy concerns, or just simply lack of enough observations. Therefore, traditional algorithms which are not designed to handle for missing data may be outright inapplicable or yield highly sub-optimal performance [11]. Investigating, how the current framework presented can be adapted in the case of missing values while also retaining all of the proposed properties remains a challenge. Notably, in the absence of differential privacy potentially [58] could be extended and be applicable in our setting and such an extension that exploits the contributions contained in this thesis is already in the works. However, since our proposed scheme for guaranteeing differential privacy requires each block in full renders this scheme inapplicable as-is. Broadly speaking, the solution to this problem lies in two main directions. Namely, either attempt to *fill* the missing data within each incoming block and apply our proposed DP scheme or redesign how we are able to guarantee DP. Operating under the assumption of recovering the missing values within each block, is indeed a very interesting setting as it can be explored through the lens of *imputation* or *inference*, both of which have rich literature surrounding them [153, 124]. However, in some instances a complete redesign of the algorithms to inherently handle missing data, might indeed be required. Which avenue is best to pursue, remains unanswered, but we conjecture it is unlikely that a universal solution exists [11]. This is because the best scheme in each use-case will depend on the expected missing data of the input, along with its ambient dimension. It is likely that the *less* information contained within each block (*i.e.* the more missing entries it has), the harder it will be to reconstruct the incoming data.

More recently, while federated learning can guarantee by design better privacy and efficiency, most frameworks operate under the common assumption that collected data are received in an i.i.d manner. This, depending on circumstances, could result to data distribution shift issues between nodes in practical scenarios [132]. One promising avenue to achieve that would be to pursue models learnt using causal features as they can generalise better to unseen data. In particular, it has been shown on data from different distributions than the train distribution [61]. Hence, it would be interesting to explore how our proposed framework could be formulated to operate in a casual inference setting.

Another interesting direction to explore is to provide a time-invariance property in the case of differential privacy. Currently, our time-invariance guarantee is able to hold with the formulation presented herein, only in the absence of perturbation masks *i.e.* without differential privacy. That is because, in the current setting we aim to preserve the absolute values in the end result. In the case of DP, we try to preserve the utility rather than the absolute value of the resulting iterate. Thus by reformulating the time-invariance lemma to describe the preservation of *utility* rather than the absolute iterate *weights* we can then, perhaps, extend this property in the DP setting. This is a direction we are currently pursuing for an upcoming journal submission that pertains an extended version of Federated-PCA we previously presented in Chapter 4.

So far, we have addressed only potential improvements to the theoretical contributions contained in this dissertation. However, a very exciting avenue for future work is addressing the problem of how the federated computation graph is constructed and maintained from a systems point of view. In this dissertation, we assumed that the network topology was static for the context of our experiments. Naturally, this assumption cannot hold always in practice and our algorithmic constructions are able to cater for abstract network topologies. In its more general case, our scheme only requires each node to have, or be able to discover, a path to the "root" of the federation node. Defining how this process happens remains an open questions and there are a number of potential strategies to do so. Moreover, peer discovery is something that is very important in federated topologies, which is also a direction to consider. Both of these topics can benefit of prior art in peer-to-peer and mesh networks. Concrete examples would be to potentially exploit distributed hash table (DHT) based solutions such as Chord [163] or mesh routing [147] for topology maintenance and peer discovery. Very recently, solutions such as Braintorrent [152] have been put forth that indeed exploit DHT's for federated learning with promising results. Such solutions, could potentially be used in the case of self-organising federated data centres as well, meaning that nodes would be able to use peer discovery schemes to join or leave the data centre at will. This could pave the way for a form of "work-sharing" scheme for edge devices in which they would participate in the federated computation only when idle in a user transparent way.

On the practical side, there have been a number of discussions on how the contributions presented in this thesis could be applied to different real-world problems. Firstly, there was a proposal to exploit the Federated-PCA and its differential privacy capabilities to help plot individual COVID-19 cases in London boroughs, which could help facilitate better policies for introducing localised measures, quarantine, and isolation. Another potential application that was discussed, is to exploit the computational gains within the context of COVID-19 Sounds project phone applications. The contributions presented in this thesis could help to perform on-device processing of the audio samples instead of uploading them to our servers. Naturally, this would overall enhance user privacy as sensitive voice related data would ever leave the device and is being explored as a possibility in a future version of the applications.

6.3 Outlook

The introduction of Federated Learning (FL) as a computation paradigm has paved the way for a new generation of services, applications, and methodologies. Coupled with the focus on privacy and data-ownership these constraints make FL the de-facto computation method of choice for datasets that are scattered in variable sized pieces within edge nodes across the globe, each with their own set of limitations *i.e.* they form *federated datasets*. Such drastic shifts in how computation is actually performed has rendered many of the traditional methods inapplicable and hence necessitated their redesign in order to be usable in a federated context. We conjecture, it will take some time until federated datasets are fully digested by academics, institutions, as well as the industry and even more time to tap into the full potential within these datasets. Naturally, when this eventuality occurs and as history has shown, more data and more questions will emerge.

In this dissertation we have attempted to make one oldest of the most commonly used methods in dimensionality reduction, namely PCA, applicable to federated datasets while offering attractive properties and concrete guarantees. Specifically, through the contributions presented herein, we are able scale PCA to "federated-scale" datasets and aspire that our methods serve as a powerful analytical tool that will help practitioners discover interesting underlying structures within them. Finally, we hope that apart from the obvious quantitative applications of our work, through this dissertation we can inspire researchers to improve upon our findings but, perhaps more importantly, empower them to embrace the FL paradigm by rethinking how other classical algorithms could be made applicable in a federated setting.

Appendix A

Supplementary Material for Chapter 3

This comes as supplementary material to the work presented in Chapter 3. Most of the material in this section is to aid the reader in understanding the intricate details about MOSES, if desired.

Let us start by recalling some of the necessary spectral properties of a standard random Gaussian matrix, namely a matrix populated with independent random Gaussian variables with zero-mean and unit variance. For a standard Gaussian matrix $\mathbf{G} \in \mathbb{R}^{a \times b}$ with $a \geq b$ and for fixed $\alpha \geq 1$, Corollary 5.35 in [180] dictates that,

$$\sqrt{a} - \alpha\sqrt{b} \leq \sigma_b(\mathbf{G}) \leq \sigma_1(\mathbf{G}) \leq \sqrt{a} + \alpha\sqrt{b}, \tag{A.1}$$

except with a probability of at most $e^{-C\alpha^2 b}$. Note that the t in Corollary 5.35 mentioned previously is translated as $t = (\alpha - 1)\sqrt{b}$, hence the requirement for $\alpha \geq 1$. For the next part, we will exploit the Hanson-Wright inequality, which for brevity presented below,

Theorem A.0.1 (Hanson-Wright inequality [154])**.** *Let $\mathbf{x} = (x_1, ..., x_d) \in \mathbb{R}^d$ be a random vector with independent components x_i which satisfy $\mathbb{E}[x_i] = 0$ and $\|x_i\|_{\psi_2} \leq K$. Let $\mathbf{A}$ be a matrix in $\mathbb{R}^{d\times d}$. Then, for every $t \geq 0$,*

$$\mathbb{P}\Big[|\mathbf{x}^T\mathbf{A}\mathbf{x} - \mathbb{E}[\mathbf{x}^T A\mathbf{x}]| > t\}\Big] \leq 2exp\Big[-c\min\left(\frac{t^2}{K^4\|\mathbf{A}\|_{HS}^2}, \frac{t}{K^2\|\mathbf{A}\|}\right)\Big].$$

A random variable ξ is called subgaussian if its distribution is dominated by that of a normal random variable. This can be expressed by requiring that $\mathbb{E}[\exp(\xi^2/K^2] \leq 2$ for some $K > 0$. The infimum of such K is traditionally called the subgaussian or ψ_2 norm of ξ. This turns the set of subgaussian random variables into the Orlicz space with the Orlicz function $\psi_2(t) = \exp(t^2) - 1$. For more details and an expanded reasoning about this, see [154].

Moreover, for a matrix $\mathbf{\Gamma} \in \mathbb{R}^{a' \times a}$ and $\alpha \geq 1$, an application of the Hanson-Wright inequality (as in Theorem A.0.1) yields that,

$$\left| \|\mathbf{\Gamma G}\|_F^2 - \mathbb{E}\|\mathbf{\Gamma G}\|_F^2 \right| \leq \beta, \tag{A.2}$$

for $\beta \geq 0$ and except with a probability of at most,

$$\exp\left(- \min\left(\frac{\beta^2}{b\|\mathbf{\Gamma}\|^2\|\mathbf{\Gamma}\|_F^2}, \frac{\beta}{\|\mathbf{\Gamma}\|^2} \right) \right),$$

where $\|\cdot\|$ stands for the spectral norm. In particular, with the choice $\beta = \alpha^2\|\mathbf{\Gamma}\|_F^2 b$ above and $\alpha \geq 1$, we find that

$$\|\mathbf{\Gamma G}\|_F^2 \leq (1+\alpha^2)\|\mathbf{\Gamma}\|_F^2 b \leq 2\alpha^2\|\mathbf{\Gamma}\|_F^2 b, \tag{A.3}$$

except with a probability of at most,

$$\exp\left(-C\alpha^2 b \frac{\|\mathbf{\Gamma}\|_F^2}{\|\mathbf{\Gamma}\|^2} \right) \leq \exp(-C\alpha^2 b).$$

In a different regime, with the choice of $\beta = \alpha^2\|\mathbf{\Gamma}\|_F^2\sqrt{b}$ in (A.2) and $\alpha^2 \leq \sqrt{b}$, we arrive at,

$$\left| \|\mathbf{\Gamma G}\|_F^2 - \mathbb{E}\|\mathbf{\Gamma G}\|_F^2 \right| = \left| \|\mathbf{\Gamma G}\|_F^2 - b\|\mathbf{\Gamma}\|_F^2 \right| \leq \alpha^2\|\mathbf{\Gamma}\|_F^2\sqrt{b}, \tag{A.4}$$

except with a probability of at most,

$$\exp\left(-C\alpha^4 \frac{\|\mathbf{\Gamma}\|_F^2}{\|\mathbf{\Gamma}\|^2} \right) \leq \exp(-C\alpha^4).$$

We are now ready to proceed with the proof Lemma 3.4.1.

A.1 Proof of Lemma 3.4.1

We start by letting,

$$\mathbf{\Xi} = \mathbf{S\Lambda S}^T = \mathbf{S\Sigma}^2\mathbf{S}^T \in \mathbb{R}^{d\times d} \tag{A.5}$$

be the eigendecomposition of the covariance matrix $\mathbf{\Xi}$, where $\mathbf{S} \in \mathbb{R}^{d\times d}$ is an orthonormal matrix and the diagonal matrix $\mathbf{\Lambda} = \mathbf{\Sigma}^2 \in \mathbb{R}^{d\times d}$ contains the eigenvalues of $\mathbf{\Xi}$ in non-increasing order, namely

$$\mathbf{\Lambda} = \mathbf{\Sigma}^2 = \begin{bmatrix} \sigma_1^2 & & & \\ & \sigma_2^2 & & \\ & & \ddots & \\ & & & \sigma_d^2 \end{bmatrix} \in \mathbb{R}^{d\times d}, \qquad \sigma_1^2 \geq \sigma_2^2 \geq \cdots \geq \sigma_d^2. \tag{A.6}$$

Throughout, we also make use of the condition number and residual, namely

$$\kappa_r = \frac{\sigma_1}{\sigma_r}, \qquad \rho_r^2 = \rho_r^2(\mathbf{\Xi}) = \sum_{i=r+1}^{d} \sigma_i^2. \qquad \text{(see (3.29))} \tag{A.7}$$

Recall that $\{\mathbf{y}_t\}_{t=1}^{\tau} \subset \mathbb{R}^d$ are the data vectors drawn from the Gaussian measure μ with zero mean and covariance matrix $\mathbf{\Xi}$, and that $\mathbf{Y}_\tau \in \mathbb{R}^{d\times\tau}$ is obtained by concatenating $\{\mathbf{y}_t\}_{t=1}^{\tau}$. It follows that,

$$\begin{aligned} \mathbf{y}_t &= \mathbf{S}\mathbf{\Sigma}\mathbf{g}_t, \qquad t \in [1:\tau], \\ \mathbf{Y}_\tau &= \mathbf{S}\mathbf{\Sigma}\mathbf{G}_\tau, \end{aligned} \tag{A.8}$$

where $\mathbf{g}_t \in \mathbb{R}^d$ and $\mathbf{G}_\tau \in \mathbb{R}^{d\times\tau}$ are standard random Gaussian vector and matrix, respectively. That is, $\mathbf{g}_t$ and $\mathbf{G}_\tau$ are populated with independent Gaussian random variables with zero mean and unit variance. With these preparations, we are now ready to prove Proposition 3.4.1. For $\mathbf{y}$ drawn from the Gaussian measure μ, note that,

$$\begin{aligned} \mathbb{E}_{\mathbf{y}}\|\mathbf{y} - \mathbf{P}_{\mathcal{S}_{\tau,r}}\mathbf{y}\|_2^2 &= \mathbb{E}_{\mathbf{y}}\|\mathbf{P}_{\mathcal{S}_{\tau,r}^\perp}\mathbf{y}\|_2^2 \\ &= \mathbb{E}_y\langle \mathbf{P}_{\mathcal{S}_{\tau,r}^\perp}, \mathbf{y}\mathbf{y}^T\rangle \\ &= \langle \mathbf{P}_{\mathcal{S}_{\tau,r}^\perp}, \mathbf{\Xi}\rangle \\ &= \left\langle \mathbf{P}_{\mathcal{S}_{\tau,r}^\perp}, \mathbf{\Xi} - \frac{\mathbf{Y}_\tau\mathbf{Y}_\tau^T}{\tau}\right\rangle + \frac{1}{\tau}\langle \mathbf{P}_{\mathcal{S}_{\tau,r}^\perp}, \mathbf{Y}_\tau\mathbf{Y}_\tau^T\rangle \\ &= \left\langle \mathbf{P}_{\mathcal{S}_{\tau,r}^\perp}, \mathbf{\Xi} - \frac{\mathbf{Y}_\tau\mathbf{Y}_\tau^T}{\tau}\right\rangle + \frac{1}{\tau}\|\mathbf{P}_{\mathcal{S}_{\tau,r}^\perp}\mathbf{Y}_\tau\|_F^2 \\ &= \left\langle \mathbf{P}_{\mathcal{S}_{\tau,r}^\perp}, \mathbf{\Xi} - \frac{\mathbf{Y}_\tau\mathbf{Y}_\tau^T}{\tau}\right\rangle + \frac{\rho_r^2(\mathbf{Y}_\tau)}{\tau} \qquad \text{(see Program (3.22))} \\ &= \frac{1}{\tau}\left(\mathbb{E}\|\mathbf{P}_{\mathcal{S}_{\tau,r}^\perp}\mathbf{Y}_\tau\|_F^2 - \|\mathbf{P}_{\mathcal{S}_{\tau,r}^\perp}\mathbf{Y}_\tau\|_F^2\right) + \frac{\rho_r^2(\mathbf{Y}_\tau)}{\tau}. \qquad \text{(see (A.8))} \end{aligned} \tag{A.9}$$

Next let us now, attempt to control the two components in the last line above. The first component above involves the deviation of random variable $\|\mathbf{P}_{\mathcal{S}_{\tau,r}^\perp}\mathbf{Y}_\tau\|_F^2$ from its expectation. By invoking the Hanson-Wright inequality (as in Theorem A.0.1) and for $\widetilde{\alpha}^2 \le \sqrt{\tau}$, we write that

$$\begin{aligned} \mathbb{E}\|\mathbf{P}_{\mathcal{S}_{\tau,r}^\perp}\mathbf{Y}_\tau\|_F^2 - \|\mathbf{P}_{\mathcal{S}_{\tau,r}^\perp}\mathbf{Y}_\tau\|_F^2 &= \mathbb{E}\|\mathbf{P}_{\mathcal{S}_{\tau,r}^\perp}\mathbf{S}\mathbf{\Sigma}\cdot\mathbf{G}_\tau\|_F^2 - \|\mathbf{P}_{\mathcal{S}_{\tau,r}^\perp}\mathbf{S}\mathbf{\Sigma}\cdot\mathbf{G}_\tau\|_F^2 \qquad \text{(see (A.8))} \\ &\le \widetilde{\alpha}^2\|\mathbf{P}_{\mathcal{S}_{\tau,r}^\perp}\mathbf{S}\mathbf{\Sigma}\|_F^2\sqrt{\tau} \qquad \text{(see (A.4))} \\ &\le \widetilde{\alpha}^2\|\mathbf{P}_{\mathcal{S}_{\tau,r}^\perp}\mathbf{S}\|_F^2\,\|\mathbf{\Sigma}\|^2\sqrt{\tau} \\ &\le \widetilde{\alpha}^2(d-r)\sigma_1^2\sqrt{\tau}, \qquad \text{(see (A.6, A.7))} \end{aligned} \tag{A.10}$$

except with a probability of at most $e^{-C\widetilde{\alpha}^4}$. In particular, for the choice of $\widetilde{\alpha}^2 = \alpha^2\sqrt{\log\tau}$ with $\alpha^2 \le \sqrt{\frac{\tau}{\log\tau}}$, we find that

$$\mathbb{E}\|\mathbf{P}_{\mathcal{S}_{\tau,r}^{\perp}}\mathbf{Y}_\tau\|_F^2 - \|\mathbf{P}_{\mathcal{S}_{\tau,r}^{\perp}}\mathbf{Y}_\tau\|_F^2 \le \alpha^2(d-r)\sigma_1^2\sqrt{\frac{\tau}{\log\tau}}, \tag{A.11}$$

except with a probability of $\tau^{-C\alpha^4}$. We next bound the second term in the last line of (A.9), namely the residual of $\mathbf{Y}_\tau$. To do so, let us start by noting that,

$$\begin{aligned}\rho_r^2(\mathbf{Y}_\tau) &= \rho_r^2(\mathbf{S}\boldsymbol{\Sigma}\mathbf{G}_\tau) \qquad \text{(see (A.8))}\\ &= \rho_r^2(\boldsymbol{\Sigma}\mathbf{G}_\tau) \qquad \left(\mathbf{S}^T\mathbf{S} = \mathbf{I}_d\right)\\ &= \min_{\text{rank}(\mathbf{X})=r} \|\boldsymbol{\Sigma}\mathbf{G}_\tau - \mathbf{X}\|_F^2. \qquad \text{(see (A.7))}\end{aligned} \tag{A.12}$$

By substituting above the suboptimal choice of,

$$\mathbf{X}_o = \left[\begin{array}{c} \boldsymbol{\Sigma}[1:r,1:r]\cdot\mathbf{G}_\tau[1:r,:] \\ \mathbf{0}_{(d-r)\times\tau} \end{array}\right], \tag{A.13}$$

we can deduce that,

$$\begin{aligned}\rho_r^2(\mathbf{Y}_\tau) &= \min_{\text{rank}(\mathbf{X})=r} \|\boldsymbol{\Sigma}\mathbf{G}_\tau - \mathbf{X}\|_F^2 \qquad \text{(see (A.12))}\\ &\le \|\boldsymbol{\Sigma}\mathbf{G}_\tau - \mathbf{X}_o\|_F^2\\ &= \|\boldsymbol{\Sigma}[r+1:d,r+1:d]\cdot\mathbf{G}_\tau[r+1:d,:]\|_F. \qquad \text{(see (A.13))}\end{aligned} \tag{A.14}$$

Note that $\mathbf{G}_\tau[r+1:d,:] \in \mathbb{R}^{(d-r)\times\tau}$ is a standard Gaussian matrix. For $\alpha \ge 1$, an application of the Hanson-Wright inequality (as in Theorem A.0.1) therefore implies that,

$$\begin{aligned}\rho_r^2(\mathbf{Y}_\tau) &\le \|\boldsymbol{\Sigma}[r+1:d,r+1:d]\cdot\mathbf{G}_\tau[r+1:d,:]\|_F^2 \qquad \text{(see (A.14))}\\ &\le 2\alpha^2\|\boldsymbol{\Sigma}[r+1:d,r+1:d]\|_F^2\tau \qquad \text{(see (A.3))}\\ &= 2\alpha^2\rho_r^2\tau, \qquad \text{(see (A.7))}\end{aligned} \tag{A.15}$$

except with a probability of at most $e^{-C\alpha^2\tau}$. Let us now substitute the bounds in (A.11) and (A.15) back into (A.9) then we are arrive to arrive that the following,

$$\mathbb{E}\|\mathbf{y} - \mathbf{P}_{\mathcal{S}_{\tau,r}}\mathbf{y}\|_2^2 \le \alpha^2(d-r)\sigma_1^2\sqrt{\frac{\tau}{\log\tau}} + 2\alpha^2\rho_r^2, \tag{A.16}$$

when $\alpha^2 \le \sqrt{\frac{\tau}{\log\tau}}$ and except with a probability of at most equal to,

$$\tau^{-C\alpha^4} + e^{-C\alpha^2\tau} \le \tau^{-C\alpha^4}, \qquad \left(\alpha^2 \le \sqrt{\frac{\tau}{\log\tau}}\right)$$

where we abuse the notation in which C is a universal constant with the property is allowed to change in every appearance. This completes the proof of Proposition 3.4.1.

A.2 Proof of Theorem 3.4.2

In the rest of this paper, we slightly unburden the notation by using $\mathbf{Y}_k \in \mathbb{R}^{d\times kb}$ to denote $\mathbf{Y}_{kb}$. For example, we will use $\mathbf{Y}_K \in \mathbb{R}^{d\times\tau}$ instead of $\mathbf{Y}_\tau$ because $\tau = Kb$. We also write $\widehat{\mathcal{S}}_{k,r}$ instead of $\widehat{\mathcal{S}}_{kb,r}$. As with the proof of Proposition 3.4.1, we argue that,

$$\begin{aligned}
\mathbb{E}_{\mathbf{y}}\|\mathbf{y} - \mathbf{P}_{\widehat{\mathcal{S}}_{K,r}}\mathbf{y}\|_2^2 &\le \frac{1}{\tau}\left(\mathbb{E}\|\mathbf{P}_{\widehat{\mathcal{S}}_{K,r}^\perp}\mathbf{Y}_\tau\|_F^2 - \|\mathbf{P}_{\widehat{\mathcal{S}}_{K,r}^\perp}\mathbf{Y}_\tau\|_F^2\right) + \frac{1}{\tau}\|\mathbf{P}_{\widehat{\mathcal{S}}_{K,r}^\perp}\mathbf{Y}_K\|_F^2 \qquad \text{(similar to (A.9))}\\
&\le \alpha^2(d-r)\sigma_1^2\sqrt{\frac{\log\tau}{\tau}} + \frac{1}{\tau}\|\mathbf{P}_{\widehat{\mathcal{S}}_{K,r}^\perp}\mathbf{Y}_K\|_F^2 \qquad \text{(see (A.11))}\\
&= \alpha^2(d-r)\sigma_1^2\sqrt{\frac{\log\tau}{\tau}} + \frac{1}{\tau}\|\mathbf{P}_{\widehat{\mathcal{S}}_{K,r}^\perp}(\mathbf{Y}_K - \widehat{\mathbf{Y}}_{K,r})\|_F^2 \qquad \text{(see (3.25))}\\
&\le \alpha^2(d-r)\sigma_1^2\sqrt{\frac{\log\tau}{\tau}} + \frac{1}{\tau}\|\mathbf{Y}_K - \widehat{\mathbf{Y}}_{K,r}\|_F^2,
\end{aligned} \tag{A.17}$$

except with a probability of at most $\tau^{-C\alpha^4}$ and provided that $\alpha^2 \le \sqrt{\frac{\tau}{\log\tau}}$. It therefore remains to control the norm in the last line above. Let us recall that the output of MOSES, namely $\widehat{\mathbf{Y}}_{K,r}$, is intended to approximate a rank-r truncation of $\mathbf{Y}_K$. We will therefore compare the error $\|\mathbf{Y}_K - \widehat{\mathbf{Y}}_{K,r}\|_F$ in (A.17) with the true residual $\rho_r(\mathbf{Y}_K)$. To that end, our analysis consists of a deterministic bound and a stochastic evaluation of this bound. The deterministic bound is as follows, see Appendix A.3 for the proof.

Lemma A.2.1. *For every $k \in [1:K]$, let $\mathbf{Y}_{k,r} = \mathrm{SVD}(\mathbf{Y}_k, r) \in \mathbb{R}^{d\times kb}$ be a rank-r truncation of $\mathbf{Y}_k$ and set $\mathcal{S}_{k,r} = \mathrm{span}(\mathbf{Y}_{k,r}) \in \mathbb{G}(d,r)$. For $\mathbf{p} > 1$, we also set,*

$$\theta_k := 1 + \frac{\mathbf{p}^{\frac{1}{3}}\|\mathbf{B}_k\|^2}{\sigma_r(\mathbf{Y}_{k-1})^2}. \tag{A.18}$$

Where $\mathbf{B}_k$ is the concatenated vectors $\mathbf{y} \in \mathbb{R}^d$ contained in the k-th block. Then the output of MOSES*, namely $\widehat{\mathbf{Y}}_{K,r}$, satisfies the following inequality,*

$$\|\mathbf{Y}_K - \widehat{\mathbf{Y}}_{K,r}\|_F^2 \le \frac{\mathbf{p}^{\frac{1}{3}}}{\mathbf{p}^{\frac{1}{3}} - 1}\sum_{k=2}^{K}\left(\prod_{l=k+1}^{K}\theta_l\right)\|\mathbf{P}_{\mathcal{S}_{k-1,r}^\perp}\mathbf{B}_k\|_F^2, \tag{A.19}$$

where $\mathbf{P}_{\mathcal{S}_{k-1,r}^\perp} \in \mathbb{R}^{d\times d}$ *is the orthogonal projection onto the orthogonal complement of* $\mathcal{S}_{k-1,r}$*. Above, we use the convention that* $\prod_{l=K+1}^{K} \theta_l = 1$*.*

In words, (A.19) gives a deterministic bound on the performance of MOSES. The term $\|\mathbf{P}_{\mathcal{S}_{k-1,r}^\perp}\mathbf{B}_k\|_F$ in (A.19) is in a sense the "innovation" at iteration k, namely the part of the new data block $\mathbf{B}_k$ that cannot be described by the current estimate $\mathcal{S}_{k-1,r}$. We note, that the overall innovation in (A.19) clearly controls the performance of MOSES. In particular, if the data blocks are drawn from the same distribution, this innovation gradually reduces as k increases. For example, if $\{\mathbf{B}_k\}_{k=1}^K$ are drawn from a distribution with a rank-r covariance matrix, then the innovation term vanishes almost surely after finitely many iterations. In contrast, when the underlying covariance matrix is high-rank, the innovation term decays more slowly and never completely disappears even as $k\to\infty$. We will next evaluate the right-hand side of (A.19) in a stochastic setup, see Appendix A for the proof.

Lemma A.2.2. *Suppose that* $\{\mathbf{y}_t\}_{t=1}^\tau$ *are drawn from a zero-mean Gaussian probability measure with the covariance matrix* $\boldsymbol{\Xi}\in\mathbb{R}^{d\times d}$*. Further, let* $\sigma_1^2\ge\sigma_2^2\ge\cdots\ge\sigma_d$ *be the eigenvalues of* $\boldsymbol{\Xi}$ *and recall the notation in (A.7). For* $\mathbf{p}>1$*, also we let,*

$$\eta_r := \kappa_r + \frac{\sqrt{2}\alpha\rho_r}{\mathbf{p}^{\frac{1}{6}}\sigma_r}.$$

For $\alpha\ge 1$*, it then holds that,*

$$\|\mathbf{Y}_K-\widehat{\mathbf{Y}}_{K,r}\|_F^2 \le \frac{50\mathbf{p}^{\frac{4}{3}}\alpha^2}{(\mathbf{p}^{\frac{1}{3}}-1)^2}\cdot\min\left(\kappa_r^2\rho_r^2, r\sigma_1^2+\rho_r^2\right)\eta_r^2 b\left(\frac{2K}{\mathbf{p}\eta_r^2}+2\right)^{\mathbf{p}\eta_r^2}, \tag{A.20}$$

except with a probability of at most $e^{-C\alpha^2 r}$ *and provided that,*

$$b\ge\frac{\mathbf{p}^{\frac{1}{3}}\alpha^2 r}{(\mathbf{p}^{\frac{1}{6}}-1)^2},\qquad b\ge C\alpha^2 r.$$

Substituting the right-hand side of (A.20) back into (A.17) yields that,

$$\begin{aligned}\mathbb{E}_{\mathbf{y}}\|\mathbf{y}-\mathbf{P}_{\widehat{\mathcal{S}}_{K,r}}\mathbf{y}\|_2^2 &\le \alpha^2(d-r)\sigma_1^2\sqrt{\frac{\log\tau}{\tau}}+\frac{1}{\tau}\|\mathbf{Y}_K-\widehat{\mathbf{Y}}_{K,r}\|_F^2, \qquad \text{(see (A.17))}\\ &\le \alpha^2(d-r)\sigma_1^2\sqrt{\frac{\log\tau}{\tau}}+\frac{50\mathbf{p}^{\frac{4}{3}}\alpha^2}{(\mathbf{p}^{\frac{1}{3}}-1)^2}\cdot\min\left(\kappa_r^2\rho_r^2, r\sigma_1^2+\rho_r^2\right)\frac{\eta_r^2}{K}\left(\frac{2K}{\mathbf{p}\eta_r^2}+2\right)^{\mathbf{p}\eta_r^2}.\end{aligned} \tag{A.21}$$

In particular, if $K \ge p\eta_r^2$, we may simplify the above bound to read as follows,

$$\mathbb{E}_{\mathbf{y}}\|\mathbf{y}-\mathbf{P}_{\widehat{\mathcal{S}}_{K,r}}\mathbf{y}\|_2^2 \le \alpha^2(d-r)\sigma_1^2\sqrt{\frac{\log\tau}{\tau}} + \frac{50\mathbf{p}^{\frac{1}{3}}\alpha^2 4^{p\eta_r^2}}{(\mathbf{p}^{\frac{1}{3}}-1)^2}\cdot\min\left(\kappa_r^2\rho_r^2, r\sigma_1^2+\rho_r^2\right)\left(\frac{K}{\mathbf{p}\eta_r^2}\right)^{p\eta_r^2-1}, \tag{A.22}$$

which completes our proof of Theorem 3.4.2.

A.3 Proof of Lemma A.2.1

Recall that the output of MOSES is the sequence of rank-r matrices $\{\widehat{\mathbf{Y}}_k\}_{k=1}^K$. For every $k<K$, it is more convenient in the proof of Lemma A.2.1 to pad both $\mathbf{Y}_k, \widehat{\mathbf{Y}}_{k,r} \in \mathbb{R}^{d\times kb}$ with zeros to form the $d\times Kb$ matrices which can be formalised as,

$$\begin{bmatrix} \mathbf{Y}_k & \mathbf{0}_{d\times(K-k)b} \end{bmatrix}, \qquad \begin{bmatrix} \widehat{\mathbf{Y}}_{k,r} & \mathbf{0}_{d\times(K-k)b} \end{bmatrix}. \tag{A.23}$$

We overload the notation $\mathbf{Y}_k, \widehat{\mathbf{Y}}_{k,r}$ to show the new $d\times Kb$ matrices in (A.23). Let,

$$\widehat{\mathcal{S}}_{k,r} = \text{span}(\widehat{\mathbf{Y}}_{k,r}) \in \mathbb{G}(d,r),$$

$$\widehat{\mathcal{Q}}_{k,r} = \text{span}(\widehat{\mathbf{Y}}_{k,r}^T) \in \mathbb{G}(Kb,r) \tag{A.24}$$

denote the (r-dimensional) column and row spaces of the rank-r matrix $\widehat{\mathbf{Y}}_{k,r} \in \mathbb{R}^{d\times Kb}$, respectively. Let also $\widehat{\mathbf{S}}_{k,r} \in \mathbb{R}^{d\times r}$ and $\widehat{\mathbf{Q}}_{k,r} \in \mathbb{R}^{Kb\times r}$ be orthonormal bases for these subspaces. We also let $\mathcal{I}_k \subset \mathbb{R}^{Kb}$ denote the b-dimensional subspace spanned by the coordinates $[(k-1)b+1 : bk]$, namely

$$\mathcal{I}_k = \text{span}\left(\begin{bmatrix} \mathbf{0}_{(k-1)b\times b} \\ \mathbf{I}_b \\ \mathbf{0}_{(K-k)b\times b} \end{bmatrix}\right) \in \mathbb{G}(Kb,b), \tag{A.25}$$

and we use the notation

$$\mathcal{J}_k := \mathcal{I}_1 \oplus \mathcal{I}_2 \cdots \oplus \mathcal{I}_k \in \mathbb{G}(Kb,kb), \qquad k\in[1:K], \tag{A.26}$$

to denote the kb-dimensional subspace that spans the first kb coordinates in $\mathbb{R}^{Kb}$. The following technical lemma, proved in Chapter A, gives another way of expressing the output of MOSES, namely $\{\widehat{\mathbf{Y}}_{k,r}\}_{k=1}^K$.

Lemma A.3.1. *For every $k\in[1:K]$, it holds that,*

$$\widehat{\mathbf{Y}}_{k,r} = \mathbf{Y}_K\mathbf{P}_{\widehat{\mathcal{Q}}_{k,r}}, \tag{A.27}$$

or equivalently

$$\widehat{\mathbf{Y}}_{k-1,r} + \mathbf{Y}_k \mathbf{P}_{\mathcal{I}_k} = \mathbf{Y}_K \mathbf{P}_{\widetilde{\mathcal{Q}}_k}, \tag{A.28}$$

where

$$\widetilde{\mathcal{Q}}_k := \widehat{\mathcal{Q}}_{k-1,r} \oplus \mathcal{I}_k \subset \mathbb{R}^{Kb} \tag{A.29}$$

is the direct sum of the two subspaces $\widehat{\mathcal{Q}}_{k-1,r}$ and $\mathcal{I}_k$. In particular, the update rule (3.8) can be written as follows,

$$\mathbf{Y}_K \mathbf{P}_{\widehat{\mathcal{Q}}_{k,r}} = \text{SVD}\left(\mathbf{Y}_K \mathbf{P}_{\widetilde{\mathcal{Q}}_k}, r\right), \qquad k \in [2:K]. \tag{A.30}$$

Lastly we have the inclusion,

$$\widehat{\mathcal{Q}}_{k,r} \subset \widetilde{\mathcal{Q}}_k \subset \mathcal{J}_k \in \mathbb{G}(Kb, kb). \tag{A.31}$$

In particular, (A.27) and (A.31) together imply that,

$$\widehat{\mathbf{Y}}_{k,r} = \mathbf{Y}_K \mathbf{P}_{\mathcal{J}_k} \mathbf{P}_{\widehat{\mathcal{Q}}_{k,r}} = \mathbf{Y}_k \mathbf{P}_{\widehat{\mathcal{Q}}_{k,r}},$$

that is, only $\mathbf{Y}_k$ (containing the first kb data vectors) contributes to the formation of $\widehat{\mathbf{Y}}_{k,r}$, the output of algorithm at iteration k, which was to be expected of course. Now, recall that $\widehat{\mathbf{Y}}_{k,r}$ is intended to approximate $\mathbf{Y}_{k,r} = \text{SVD}(\mathbf{Y}_k, r)$. In light of Lemma A.3.1, let us now derive a simple recursive expression for the residual $\mathbf{Y}_k - \widehat{\mathbf{Y}}_{k,r}$. For every $k \in [2:K]$, it holds that,

$$\begin{aligned}
\mathbf{Y}_k - \widehat{\mathbf{Y}}_{k,r} &= \mathbf{Y}_K \mathbf{P}_{\mathcal{J}_k} - \mathbf{Y}_K \mathbf{P}_{\widehat{\mathcal{Q}}_{k,r}} \qquad \text{(see (A.26) and (A.27))} \\
&= \mathbf{Y}_K \mathbf{P}_{\mathcal{J}_{k-1}} + \mathbf{Y}_K \mathbf{P}_{\mathcal{I}_k} - \mathbf{Y}_K \mathbf{P}_{\widehat{\mathcal{Q}}_{k,r}} \qquad \text{(see (A.26))} \\
&= \mathbf{Y}_{k-1} + \mathbf{Y}_K \mathbf{P}_{\mathcal{I}_k} - \mathbf{Y}_K \mathbf{P}_{\widehat{\mathcal{Q}}_{k,r}} \qquad \text{(see (A.26))} \\
&= \mathbf{Y}_{k-1} - \widehat{\mathbf{Y}}_{k-1,r} + \mathbf{Y}_K \mathbf{P}_{\widehat{\mathcal{Q}}_{k-1,r}} + \mathbf{Y}_K \mathbf{P}_{\mathcal{I}_k} - \mathbf{Y}_K \mathbf{P}_{\widehat{\mathcal{Q}}_{k,r}} \qquad \text{(see (A.27))} \\
&= \left(\mathbf{Y}_{k-1} - \widehat{\mathbf{Y}}_{k-1,r}\right) + \mathbf{Y}_K \left(\mathbf{P}_{\widehat{\mathcal{Q}}_{k-1,r}} + \mathbf{P}_{\mathcal{I}_k}\right) - \mathbf{Y}_K \mathbf{P}_{\widehat{\mathcal{Q}}_{k,r}} \\
&= \left(\mathbf{Y}_{k-1} - \widehat{\mathbf{Y}}_{k-1,r}\right) + \mathbf{Y}_K \left(\mathbf{P}_{\widetilde{\mathcal{Q}}_k} - \mathbf{P}_{\widehat{\mathcal{Q}}_{k,r}}\right). \qquad \text{(see (A.29))}
\end{aligned} \tag{A.32}$$

Interestingly, the two terms in the last line of (A.32) are orthogonal, as proved by induction in Chapter A.

Lemma A.3.2. *For every $k \in [2:K]$, it holds that,*

$$\left\langle \mathbf{Y}_{k-1} - \widehat{\mathbf{Y}}_{k-1,r}, \mathbf{Y}_K \left(\mathbf{P}_{\widetilde{\mathcal{Q}}_k} - \mathbf{P}_{\widehat{\mathcal{Q}}_{k,r}}\right)\right\rangle = 0. \tag{A.33}$$

For fixed $k \in [2:K]$,then Lemma A.3.2 immediately implies that,

$$\begin{aligned}
\|\mathbf{Y}_k - \widehat{\mathbf{Y}}_{k,r}\|_F^2 &= \left\| \left(\mathbf{Y}_{k-1} - \widehat{\mathbf{Y}}_{k-1,r}\right) + \mathbf{Y}_K \left(\mathbf{P}_{\widetilde{\mathbf{Q}}_k} - \mathbf{P}_{\widehat{\mathbf{Q}}_{k,r}}\right)\right\|_F^2 && \text{(see (A.32))} \\
&= \|\mathbf{Y}_{k-1} - \widehat{\mathbf{Y}}_{k-1,r}\|_F^2 + \|\mathbf{Y}_K(\mathbf{P}_{\widetilde{\mathcal{Q}}_k} - \mathbf{P}_{\widehat{\mathcal{Q}}_{k,r}})\|_F^2 && \text{(see Lemma A.3.2)} \\
&= \|\mathbf{Y}_{k-1} - \widehat{\mathbf{Y}}_{k-1,r}\|_F^2 + \rho_r\left(\widehat{\mathbf{Y}}_{k-1,r} + \mathbf{Y}_k \mathbf{P}_{\mathcal{I}_k}\right). && \text{(see (A.30) and (A.28))}
\end{aligned} \tag{A.34}$$

Recalling from (A.24) that $\widehat{\mathcal{S}}_{k-1,r} = \text{span}(\widehat{\mathbf{Y}}_{k-1,r})$, and thus we are able to bound the above expression as follows,

$$\begin{aligned}
\|\mathbf{Y}_k - \widehat{\mathbf{Y}}_{k,r}\|_F^2 &= \|\mathbf{Y}_{k-1} - \widehat{\mathbf{Y}}_{k-1,r}\|_F^2 + \rho_r\left(\widehat{\mathbf{Y}}_{k-1,r} + \mathbf{Y}_k \mathbf{P}_{\mathcal{I}_k}\right) \\
&\leq \|\mathbf{Y}_{k-1} - \widehat{\mathbf{Y}}_{k-1,r}\|_F^2 + \left\|\mathbf{P}_{\widehat{\mathcal{S}}_{k-1,r}^{\perp}}\left(\widehat{\mathbf{Y}}_{k-1,r} + \mathbf{Y}_k \mathbf{P}_{\mathcal{I}_k}\right)\right\|_F^2 \\
&= \|\mathbf{Y}_{k-1} - \widehat{\mathbf{Y}}_{k-1,r}\|_F^2 + \|\mathbf{P}_{\widehat{\mathcal{S}}_{k-1,r}^{\perp}}\mathbf{B}_k\|_F^2, \qquad \text{(see (A.24))}
\end{aligned} \tag{A.35}$$

where the second line follows from the sub-optimality of the choice of subspace $\widehat{\mathcal{S}}_{k-1,r}$. Let us focus on the last norm above. For every k, let $\mathbf{Y}_{k,r} = \text{SVD}(\mathbf{Y}_k, r)$ be a rank-r truncation of $\mathbf{Y}_k$ with the column span $\mathcal{S}_{k,r} = \text{span}(\mathbf{Y}_{k,r})$. We now write that,

$$\begin{aligned}
\|\mathbf{P}_{\widehat{\mathcal{S}}_{k-1,r}^{\perp}}\mathbf{B}_k\|_F &\leq \|\mathbf{P}_{\widehat{\mathcal{S}}_{k-1,r}^{\perp}}\mathbf{P}_{\mathcal{S}_{k-1,r}}\mathbf{B}_k\|_F + \|\mathbf{P}_{\widehat{\mathcal{S}}_{k-1,r}^{\perp}}\mathbf{P}_{\mathcal{S}_{k-1,r}^{\perp}}\mathbf{B}_k\|_F && \text{(triangle inequality)} \\
&\leq \|\mathbf{P}_{\widehat{\mathcal{S}}_{k-1,r}^{\perp}}\mathbf{P}_{\mathcal{S}_{k-1,r}}\|_F \cdot \|\mathbf{B}_k\| + \|\mathbf{P}_{\mathcal{S}_{k-1,r}^{\perp}}\mathbf{B}_k\|_F.
\end{aligned} \tag{A.36}$$

The first norm in the last line above gauges the principal angles between the two r-dimensional subspaces $\widehat{\mathcal{S}}_{k-1,r}$ and $\mathcal{S}_{k-1,r}$. It turns out that we can bound this norm with a standard perturbation result, for example see [58, Lemma 6] or [189]. More specifically, we may imagine that $\mathbf{Y}_{k-1}$ is a perturbed copy of $\mathbf{Y}_{k-1,r}$. Then the angle between $\mathcal{S}_{k-1,r} = \text{span}(\mathbf{Y}_{k-1,r})$ and $\widehat{\mathcal{S}}_{k-1,r} = \text{span}(\widehat{\mathbf{Y}}_{k-1,r})$ is controlled by the amount of perturbation. Namely given matrices $\mathbf{A}$ and $\mathbf{W}$ with $\mathbf{W}_r$ the rank-r SVD truncation of $\mathbf{W}$ then the perturbation is controlled as $\mathbf{A} = \widehat{\mathbf{Y}}_{k-1,r}, \mathbf{W} = \mathbf{Y}_{k-1}, \mathbf{W}_r = \mathbf{Y}_{k-1,r}$ proved in [58, Lemma 6]. Provided the above result we can then find that,

$$\|\mathbf{P}_{\widehat{\mathcal{S}}_{k-1,r}^{\perp}}\mathbf{P}_{\mathcal{S}_{k-1,r}}\|_F \leq \frac{\|\mathbf{Y}_{k-1} - \widehat{\mathbf{Y}}_{k-1,r}\|_F}{\sigma_r\left(\mathbf{Y}_{k-1}\right)}. \tag{A.37}$$

By plugging (A.37) back into (A.36), we find that the following holds,

$$\|\mathbf{P}_{\widehat{\mathcal{S}}_{k-1,r}^{\perp}}\mathbf{B}_k\| \leq \frac{\|\mathbf{B}_k\|}{\sigma_r\left(\mathbf{Y}_{k-1}\right)} \cdot \|\mathbf{Y}_{k-1} - \widehat{\mathbf{Y}}_{k-1,r}\|_F + \|\mathbf{P}_{\mathcal{S}_{k-1,r}^{\perp}}\mathbf{Y}_k\|_F. \tag{A.38}$$

In turn, for $\mathbf{p} > 1$, substituting the above inequality into (A.35) yields that,

$$\begin{aligned}
\|\mathbf{Y}_k - \widehat{\mathbf{Y}}_{k,r}\|_F^2 &\le \|\mathbf{Y}_{k-1} - \widehat{\mathbf{Y}}_{k-1,r}\|_F^2 + \|\mathbf{P}_{\widehat{\mathcal{S}}_{k-1,r}^{\perp}}\mathbf{B}_k\|_F^2 \qquad \text{(see (A.35))} \\
&\le \left(1 + \frac{\mathbf{p}^{\frac{1}{3}}\|\mathbf{B}_k\|^2}{\sigma_r\left(\mathbf{Y}_{k-1}\right)^2}\right)\|\mathbf{Y}_{k-1} - \widehat{\mathbf{Y}}_{k-1,r}\|_F^2 + \frac{\mathbf{p}^{\frac{1}{3}}}{\mathbf{p}^{\frac{1}{3}}-1}\|\mathbf{P}_{\mathcal{S}_{k-1,r}^{\perp}}\mathbf{B}_k\|_F^2 \qquad \text{(see (A.38))} \\
&=: \theta_k\|\mathbf{Y}_{k-1} - \widehat{\mathbf{Y}}_{k-1,r}\|_F^2 + \frac{\mathbf{p}^{\frac{1}{3}}}{\mathbf{p}^{\frac{1}{3}}-1}\|\mathbf{P}_{\mathcal{S}_{k-1,r}^{\perp}}\mathbf{B}_k\|_F^2. \qquad \text{(A.39)}
\end{aligned}$$

where we used the inequality $(a_1 + a_2)^2 \le qa_1^2 + \frac{qa_2^2}{q-1}$ for scalars a_1, a_2 and $q > 1$, with the choice of $q = p^{\frac{1}{3}}$. By unfolding the recursion in (A.39), we arrive at

$$\|\mathbf{Y}_K - \widehat{\mathbf{Y}}_{K,r}\|_F^2 \le \frac{\mathbf{p}^{\frac{1}{3}}}{\mathbf{p}^{\frac{1}{3}}-1}\sum_{k=2}^{K}\left(\prod_{l=k+1}^{K}\theta_l\right)\|\mathbf{P}_{\mathcal{S}_{k-1,r}^{\perp}}\mathbf{B}_k\|_F^2, \qquad \text{(A.40)}$$

which completes the proof of Lemma A.2.1.

A.4 Proof of Lemma A.3.1

The proof is by induction. For $k = 1$, it holds that,

$$\begin{aligned}
\widehat{\mathbf{Y}}_{1,r} &= \mathrm{SVD}(\mathbf{Y}_1, r) \qquad \text{(see Algorithm 1)} \\
&= \mathbf{Y}_1 P_{\widehat{\mathcal{Q}}_{1,r}} \qquad \text{(see (A.24))} \\
&= \mathbf{Y}_K \mathbf{P}_{\mathcal{I}_1}\mathbf{P}_{\widehat{\mathcal{Q}}_{1,r}} \\
&= \mathbf{Y}_K \mathbf{P}_{\widehat{\mathcal{Q}}_{1,r}}, \qquad \left(\widehat{\mathcal{Q}}_{1,r} \subseteq \mathcal{I}_1\right) \qquad \text{(A.41)}
\end{aligned}$$

which proves the base case of the induction. Let us now suppose that (A.27-A.31) hold for $[2:k]$ with $k < K$. We now show that (A.27-A.31) hold also for $k+1$. We can then write that,

$$\begin{aligned}
\widehat{\mathbf{Y}}_{k+1,r} &= \mathrm{SVD}\left(\widehat{\mathbf{Y}}_{k,r} + \begin{bmatrix} \mathbf{0}_{d\times kb} & \mathbf{B}_{k+1} & \mathbf{0}_{d\times(K-k-1)b} \end{bmatrix}, r\right) \qquad \text{(see 1)} \\
&= \mathrm{SVD}\left(\mathbf{Y}_K\mathbf{P}_{\widehat{\mathcal{Q}}_{k,r}} + \mathbf{Y}_K\mathbf{P}_{\mathcal{I}_{k+1}}, r\right) \qquad \text{(assumption of induction)} \\
&= \mathrm{SVD}\left(\mathbf{Y}_K\mathbf{P}_{\widetilde{\mathcal{Q}}_{k+1}}, r\right), \qquad \text{(see (A.29))} \qquad \text{(A.42)}
\end{aligned}$$

which completes the proof of Lemma A.3.1.

A.5 Proof of Lemma A.3.2

In this proof only, it is convenient to use the notation rowspan($\mathbf{A}$) to denote the row span of a matrix $\mathbf{A}$, namely rowspan($\mathbf{A}$) = span($\mathbf{A}^T$). For every $k \in [1:K]$, recall from (A.30) that

$\mathbf{Y}_K(\mathbf{P}_{\widetilde{\mathcal{Q}}_k} - \mathbf{P}_{\widehat{\mathcal{Q}}_{k,r}})$ is the residual of rank-r truncation of $\mathbf{Y}_K\mathbf{P}_{\widetilde{\mathcal{Q}}_k}$. Consequently,

$$\mathbf{Y}_K(\mathbf{P}_{\widetilde{\mathcal{Q}}_k} - \mathbf{P}_{\widehat{\mathcal{Q}}_{k,r}}) = \mathbf{Y}_K\mathbf{P}_{\widehat{\mathcal{Q}}^C_{k,r}}, \qquad k \in [1:K], \tag{A.43}$$

where $\widehat{\mathcal{Q}}^C_{k,r}$ is the orthogonal complement of $\widehat{\mathcal{Q}}_{k,r}$ with respect to $\widetilde{\mathcal{Q}}_k$, namely

$$\widetilde{\mathcal{Q}}_k = \widehat{\mathcal{Q}}_{k,r} \oplus \widehat{\mathcal{Q}}^C_{k,r}, \qquad \widehat{\mathcal{Q}}_{k,r} \perp \widehat{\mathcal{Q}}^C_{k,r} \qquad k \in [1:K], \tag{A.44}$$

in which we conveniently set $\widetilde{\mathcal{Q}}_1 = \mathcal{I}_1$, see (A.25). Now by exploiting (A.43), we can rewrite (A.32) as

$$\begin{aligned}\mathbf{Y}_k - \widehat{\mathbf{Y}}_{k,r} &= (\mathbf{Y}_{k-1} - \widehat{\mathbf{Y}}_{k-1,r}) + \mathbf{Y}_k(\mathbf{P}_{\widetilde{\mathcal{Q}}_k} - \mathbf{P}_{\widehat{\mathcal{Q}}_{k,r}}) \qquad \text{(see (A.32))} \\ &= (\mathbf{Y}_{k-1} - \widehat{\mathbf{Y}}_{k-1,r}) + \mathbf{Y}_K\mathbf{P}_{\widehat{\mathcal{Q}}^C_{k,r}}, \qquad k \in [2:K].\end{aligned} \tag{A.45}$$

With the preliminaries out of the way, let us rewrite the claim of Lemma A.3.2 as follows,

$$\left\langle \mathbf{Y}_{k-1} - \widehat{\mathbf{Y}}_{k-1,r}, \mathbf{Y}_K\mathbf{P}_{\widehat{\mathcal{Q}}^C_{k,r}} \right\rangle = 0, \qquad k \in [2:K], \tag{A.46}$$

see (A.33) and (A.43). Because $\widehat{\mathcal{Q}}^C_{k,r} \subset \widetilde{\mathcal{Q}}_k$ by consequence of (A.44), it suffices to instead prove the stronger claim that,

$$\text{rowspan}(\mathbf{Y}_{k-1} - \widehat{\mathbf{Y}}_{k-1,r}) \perp \widetilde{\mathcal{Q}}_k, \qquad k \in [2:K]. \tag{A.47}$$

We next prove (A.47) by induction. The base case of the induction, namely $k = 2$, is trivial. Suppose now that (A.47) holds for $[2:k]$ with $k < K$. We next show that (A.47) holds for $k+1$ as well. Note that,

$$\begin{aligned}\text{rowspan}(\mathbf{Y}_k - \widehat{\mathbf{Y}}_{k,r}) &= \text{rowspan}\left((\mathbf{Y}_{k-1} - \widehat{\mathbf{Y}}_{k-1,r}) + \mathbf{Y}_K\mathbf{P}_{\widehat{\mathcal{Q}}^C_{k,r}}\right) \qquad \text{(see (A.45))} \\ &\subseteq \text{rowspan}(\mathbf{Y}_{k-1} - \widehat{\mathbf{Y}}_{k-1,r}) \oplus \widehat{\mathcal{Q}}^C_{k,r}.\end{aligned} \tag{A.48}$$

As we next show, both subspaces in the last line above are orthogonal to $\widetilde{\mathcal{Q}}_{k+1}$. Indeed, on the one hand we have the following,

$$\begin{aligned}&\begin{cases}\text{rowspan}(\mathbf{Y}_{k-1} - \widehat{\mathbf{Y}}_{k-1,r}) \perp \widetilde{\mathcal{Q}}_k \supseteq \widehat{\mathcal{Q}}_{k,r}, & \text{(induction hypothesis and (A.31))} \\ \text{rowspan}(\mathbf{Y}_{k-1} - \widehat{\mathbf{Y}}_{k-1,r}) \subset \mathcal{J}_{k-1} \perp \mathcal{I}_{k+1}, & \text{(see (A.31) and (A.26))}\end{cases} \\ &\Longrightarrow \text{rowspan}(\mathbf{Y}_{k-1} - \widehat{\mathbf{Y}}_{k-1,r}) \perp (\widehat{\mathcal{Q}}_{k,r} \oplus \mathcal{I}_{k+1}) = \widetilde{\mathcal{Q}}_{k+1}. \qquad \text{(see (A.29))}\end{aligned} \tag{A.49}$$

While on the other hand,

$$\begin{cases} \widehat{\mathcal{Q}}_{k,r}^C \perp \widehat{\mathcal{Q}}_{k,r}, \\ \widehat{\mathcal{Q}}_{k,r}^C \subset \widetilde{\mathcal{Q}}_k \subset \mathcal{J}_k \perp \mathcal{I}_{k+1}, \quad \text{(see (A.31) and (A.26))} \end{cases}$$
$$\Longrightarrow \widehat{\mathcal{Q}}_{k,r}^C \perp (\widehat{\mathcal{Q}}_{k,r} \oplus \mathcal{I}_{k+1}) = \widetilde{\mathcal{Q}}_{k+1}. \qquad \text{(see (A.29))} \tag{A.50}$$

By combining (A.49) and (A.50), we can deduce that,

$$\begin{aligned} \operatorname{rowspan}(\mathbf{Y}_k - \widehat{\mathbf{Y}}_{k,r}) &\subseteq \operatorname{rowspan}(\mathbf{Y}_{k-1} - \widehat{\mathbf{Y}}_{k-1,r}) \oplus \widehat{\mathcal{Q}}_{k,r}^C \qquad \text{(see (A.48))} \\ &\perp \widetilde{\mathcal{Q}}_{k+1}. \qquad \text{(see (A.49, A.50))} \end{aligned} \tag{A.51}$$

Therefore, (A.47) holds for every $k \in [2:K]$ by induction. Thus, by extension, this proves Lemma A.3.2 and concludes our proof.

A.6 Proof of Lemma A.2.2

Recall that $\mathbf{B}_k \in \mathbb{R}^{d\times b}$, $\mathbf{Y}_k \in \mathbb{R}^{d\times kb}$ denote the k-th block and the concatenation of the first k blocks of data, respectively. Since the data vectors are independently drawn from a zero-mean Gaussian probability measure with covariance matrix $\mathbf{\Xi}$, it follows from (A.5, A.6) that,

$$\mathbf{B}_k = \mathbf{S}\mathbf{\Sigma}\mathbf{W}_k,$$

$$\mathbf{Y}_k = \mathbf{S}\mathbf{\Sigma}\mathbf{G}_k, \tag{A.52}$$

and that holds for every $k \in [1:K]$, where $\mathbf{W}_k \in \mathbb{R}^{d\times b}$ and $\mathbf{G}_k \in \mathbb{R}^{d\times kb}$ are standard random Gaussian matrices. For fixed $k \in [2:K]$, let us now study each of the random quantities on the right-hand side of (A.19). The following results are proved in Appendices A.7 and A.8, respectively.

Lemma A.6.1. (Bound on $\|\mathbf{B}_k\|$) *For $\alpha \geq 1$, $\mathbf{p} > 1$, and fixed $k \in [1:K]$, it holds that*

$$\|\mathbf{B}_k\| \leq \mathbf{p}^{\frac{1}{6}}(\sigma_1 + \sqrt{2}\alpha\mathbf{p}^{-\frac{1}{6}}\rho_r)\sqrt{b}, \tag{A.53}$$

except with a probability of at most $e^{-C\alpha^2 b}$ and provided that,

$$b \geq \frac{\alpha^2 r}{(p^{\frac{1}{6}}-1)^2}. \tag{A.54}$$

Lemma A.6.2. (Bound on $\sigma_r(\mathbf{Y}_k)$) *For $\alpha \geq 1$, $\mathbf{p} > 1$, and fixed $k \in [1:K]$, it holds that,*

$$\sigma_r(\mathbf{Y}_k) \geq \mathbf{p}^{-\frac{1}{6}}\sigma_r\sqrt{kb}, \tag{A.55}$$

except with a probability of at most $e^{-C\alpha^2 r}$ and provided that,

$$b \geq \frac{\alpha^2 r}{(1-p^{\frac{-1}{6}})^2}. \tag{A.56}$$

By combining Lemma A.6.1 and A.6.2, we can then find for fixed $k \in [2:K]$ that,

$$\begin{aligned}
\theta_k &= 1 + \frac{\mathbf{p}^{\frac{1}{3}}\|\mathbf{B}_k\|^2}{\sigma_r(\mathbf{Y}_{k-1})^2} \qquad \text{(see (A.18))} \\
&\leq 1 + \frac{\mathbf{p}(\sigma_1 + \sqrt{2}\alpha\mathbf{p}^{-\frac{1}{6}}\rho_r)^2 b}{\sigma_r^2(k-1)b} \qquad \text{(see Lemma A.6.1 and A.6.2)} \\
&=: 1 + \frac{\mathbf{p}\eta_r^2}{k-1},
\end{aligned} \tag{A.57}$$

except with a probability of at most $e^{-C\alpha^2 r}$ and provided that (A.56) holds. In particular, it follows that,

$$\begin{aligned}
\prod_{l=k+1}^{K} \theta_l &\leq \prod_{l=k+1}^{K}\left(1 + \frac{\mathbf{p}\eta_r^2}{l-1}\right) \qquad \text{(see (A.57))} \\
&\leq \frac{(K-1+p\eta_r^2)^{K-1+p\eta_r^2}}{(K-1)^{K-1}} \cdot \frac{(k-1)^{k-1}}{(k-1+p\eta_r^2)^{k-1+p\eta_r^2}} \qquad \text{(see below)} \\
&= \left(1 + \frac{\mathbf{p}\eta_r^2}{K-1}\right)^{K-1}\left(1 + \frac{\mathbf{p}\eta_r^2}{k-1}\right)^{-k+1}\left(\frac{K-1+\mathbf{p}\eta_r^2}{k-1+\mathbf{p}\eta_r^2}\right)^{\mathbf{p}\eta_r^2},
\end{aligned} \tag{A.58}$$

holds for every $k \in [2:K]$ and except with a probability of at most $Ke^{-C\alpha r}$, where the failure probability follows from an application of the union bound. The second line above is obtained by bounding the logarithm of the product in that line with the corresponding integral. More specifically, it holds that,

$$\begin{aligned}
&\log\left(\prod_{l=k+1}^{K}\left(1 + \frac{\mathbf{p}\eta_r^2}{l-1}\right)\right) \\
&= \sum_{l=k}^{K-1} \log\left(1 + \frac{\mathbf{p}\eta_r^2}{l}\right) \\
&\leq \int_{k-1}^{K-1} \log\left(1 + \frac{\mathbf{p}\eta_r^2}{x}\right) dx \\
&= (K-1+\mathbf{p}\eta_r^2)\log(K-1+\mathbf{p}\eta_r^2) - (K-1)\log(K-1) \\
&\quad - (k-1+\mathbf{p}\eta_r^2)\log(k-1+\mathbf{p}\eta_r^2) + (k-1)\log(k-1),
\end{aligned} \tag{A.59}$$

where the third line above follows because the integrand is decreasing in x. Let us further simplify (A.58). Note that $K \geq k \geq 2$ and that $\mathbf{p}\eta_r^2 \geq 1$ by its definition in (A.57). Consequently, by using the relation $2 \leq (1+1/x)^x \leq e$ for $x \geq 1$, we can then write that,

$$2 \leq \left(1+\frac{\mathbf{p}\eta_r^2}{k-1}\right)^{\frac{k-1}{\mathbf{p}\eta_r^2}} \leq e, \qquad 2 \leq \left(1+\frac{\mathbf{p}\eta_r^2}{K-1}\right)^{\frac{K-1}{\mathbf{p}\eta_r^2}} \leq e. \tag{A.60}$$

In turn, by exploiting (A.60) is what allows us to simplify (A.58) as follows:

$$\begin{aligned}\prod_{l=k+1}^{K}\theta_l &\leq \left(1+\frac{\mathbf{p}\eta_r^2}{K-1}\right)^{K-1}\left(1+\frac{\mathbf{p}\eta_r^2}{k-1}\right)^{-k+1}\left(\frac{K-1+\mathbf{p}\eta_r^2\eta_r}{k-1+\mathbf{p}\eta_r^2}\right)^{\mathbf{p}\eta_r^2} && \text{(see (A.58))}\\ &\leq \left(\frac{e}{2}\right)^{\mathbf{p}\eta_r^2}\left(\frac{K-1+\mathbf{p}\eta_r^2}{k-1+\mathbf{p}\eta_r^2}\right)^{\mathbf{p}\eta_r^2}. && \text{(see (A.60))}\end{aligned} \tag{A.61}$$

Next we control the random variable $\|\mathbf{P}_{\mathcal{S}_{k-1}^\perp}\mathbf{B}_k\|_F$ in (A.19) with the following result, proved in Section A.9.

Lemma A.6.3. (Bound on the Innovation) *For $\alpha \geq 1$ and fixed $k \in [2:K]$, it holds that,*

$$\|\mathbf{P}_{\mathcal{S}_{k-1,r}^\perp}\mathbf{B}_k\|_F \leq 5\alpha \min\left(\kappa_r\rho_r, \sqrt{r}\sigma_1+\rho_r\right)\sqrt{b}, \tag{A.62}$$

except with a probability of at most $e^{-C\alpha^2 r}$ and provided that $b \geq C\alpha^2 r$.

By combining Lemma A.6.3 and (A.61), we are able to finally deduce a stochastic bound for the right-hand side of (A.19). More specifically, it holds that,

$$\begin{aligned}&\|\mathbf{Y}_K-\widehat{\mathbf{Y}}_{K,r}\|_F^2\\ &\leq \frac{\mathbf{p}^{\frac{1}{3}}}{\mathbf{p}^{\frac{1}{3}}-1}\sum_{k=2}^{K}\left(\prod_{l=k+1}^{K}\theta_l\right)\|P_{\mathcal{S}_{k-1,r}^\perp}\mathbf{B}_k\|_F^2 \qquad \text{(see (A.19))}\end{aligned}$$

by using (A.61) and Lemma A.6.3 we have that,

$$\begin{aligned}&\leq \frac{50\mathbf{p}^{\frac{1}{3}}\alpha^2}{\mathbf{p}^{\frac{1}{3}}-1}\min\left(\kappa_r^2\rho_r^2, r\sigma_1^2+\rho_r^2\right)b\cdot\left(\frac{e}{2}\right)^{\mathbf{p}\eta_r^2}\left(K-1+\mathbf{p}\eta_r^2\right)^{\mathbf{p}\eta_r^2}\sum_{k=2}^{K}\left(k-1+\mathbf{p}\eta_r^2\right)^{-\mathbf{p}\eta_r^2}\\ &\leq \frac{50\mathbf{p}^{\frac{1}{3}}\alpha^2}{\mathbf{p}^{\frac{1}{3}}-1}\min\left(\kappa_r^2\rho_r^2, r\sigma_1^2+\rho_r^2\right)b\cdot\left(\frac{e}{2}\right)^{\mathbf{p}\eta_r^2}\left(K-1+\mathbf{p}\eta_r^2\right)^{\mathbf{p}\eta_r^2}\int_{\mathbf{p}\eta_r^2}^{\infty}x^{-\mathbf{p}\eta_r^2}\,dx\\ &= \frac{50\mathbf{p}^{\frac{1}{3}}\alpha^2}{\mathbf{p}^{\frac{1}{3}}-1}\min\left(\kappa_r^2\rho_r^2, r\sigma_1^2+\rho_r^2\right)b\cdot\left(\frac{e}{2}\right)^{\mathbf{p}\eta_r^2}\left(K-1+\mathbf{p}\eta_r^2\right)^{\mathbf{p}\eta_r^2}\cdot\frac{(\mathbf{p}\eta_r^2)^{-\mathbf{p}\eta_r^2+1}}{\mathbf{p}\eta_r^2-1}\\ &\leq \frac{50\mathbf{p}^{\frac{1}{3}}\alpha^2}{\mathbf{p}^{\frac{1}{3}}-1}\min\left(\kappa_r^2\rho_r^2, r\sigma_1^2+\rho_r^2\right)b\left(\frac{2K}{\mathbf{p}\eta_r^2}+2\right)^{\mathbf{p}\eta_r^2}\frac{\mathbf{p}\eta_r^2}{\mathbf{p}\eta_r^2-1}\\ &\leq \frac{50\mathbf{p}^{\frac{4}{3}}\alpha^2}{(\mathbf{p}^{\frac{1}{3}}-1)^2}\cdot\min\left(\kappa_r^2\rho_r^2, r\sigma_1^2+\rho_r^2\right)\eta_r^2 b\left(\frac{2K}{\mathbf{p}\eta_r^2}+2\right)^{\mathbf{p}\eta_r^2}, \qquad (p,\eta_r \geq 1)\end{aligned} \tag{A.63}$$

except with a probability of at most $e^{-C\alpha^2 r}$ and provided that,

$$b \geq \frac{\mathbf{p}^{\frac{1}{3}}\alpha^2 r}{(\mathbf{p}^{\frac{1}{6}}-1)^2}, \qquad b \geq C\alpha^2 r.$$

This completes the proof of Lemma A.2.2.

A.7 Proof of Lemma A.6.1

Let us start our proof by noting that,

$$\begin{aligned}
\|\mathbf{B}_k\| &= \|\mathbf{S}\mathbf{\Sigma}\mathbf{W}_k\| \qquad \text{(see (A.52))} \\
&= \|\mathbf{\Sigma}\mathbf{W}_k\| \qquad \left(\mathbf{S}^T\mathbf{S} = \mathbf{I}_d\right) \\
&\leq \|\mathbf{\Sigma}[1:r,1:r]\cdot\mathbf{W}_k[1:r,:]\| + \|\mathbf{\Sigma}[r+1:d,r+1:d]\cdot\mathbf{W}_k[r+1:d,:]\| \qquad \text{(triangle inequality)} \\
&\leq \sigma_1\cdot\|\mathbf{W}_k[1:r,:]\| + \|\mathbf{\Sigma}[r+1:d,r+1:d]\cdot\mathbf{W}_k[r+1:d,:]\| \\
&\leq \sigma_1\cdot\|\mathbf{W}_k[1:r,:]\| + \|\mathbf{\Sigma}[r+1:d,r+1:d]\cdot\mathbf{W}_k[r+1:d,:]\|_F, \qquad \text{(A.64)}
\end{aligned}$$

where we used MATLAB's matrix notation as usual. Note that both $\mathbf{W}_k[1:r,:]\in\mathbb{R}^{r\times b}$ and $\mathbf{W}_k[r+1:d,:]\in\mathbb{R}^{(d-r)\times b}$ in (A.64) are standard Gaussian random matrices. For $\alpha\geq 1$ and $\mathbf{p}>1$, invoking the results about the spectrum of Gaussian random matrices as presented in the beginning of Chapter A yields that,

$$\begin{aligned}
\|\mathbf{B}_k\| &\leq \sigma_1\cdot\|\mathbf{W}_k[1:r,:]\| + \|\mathbf{\Sigma}[r+1:d,r+1:d]\cdot\mathbf{W}_k[r+1:d,:]\|_F \qquad \text{(see (A.64))} \\
&\leq \sigma_1(\sqrt{b}+\alpha\sqrt{r}) + \sqrt{2}\alpha\|\mathbf{\Sigma}[r+1:d,r+1:d]\|_F\sqrt{b} \qquad \text{(see (A.1,A.3) and } b\geq r) \\
&= \sigma_1(\sqrt{b}+\alpha\sqrt{r}) + \alpha\rho_r\sqrt{2b} \qquad \text{(see (A.6,A.7))} \\
&\leq \mathbf{p}^{\frac{1}{6}}\sigma_1\sqrt{b} + \alpha\rho_r\sqrt{2b}, \qquad \left(\text{if } b\geq\frac{\alpha^2 r}{(p^{\frac{1}{6}}-1)^2}\right) \qquad \text{(A.65)}
\end{aligned}$$

except with a probability of at most $e^{-C\alpha^2 r}+e^{-C\alpha^2 b}\leq e^{-C\alpha^2 r}$, where this final inequality follows from the assumption that $b\geq r$. This completes the proof of Lemma A.6.1. We remark that a slightly stronger bound could be obtained by using Slepian's inequality for comparing Gaussian processes, see [180, Section 5.3.1] and [114, Section 3.1].

A.8 Proof of Lemma A.6.2

For a matrix $\mathbf{A}\in\mathbb{R}^{d\times kb}$, it follows from the Fisher-Courant representation of the singular values that,

$$\sigma_r(\mathbf{A}) \geq \sigma_r(\mathbf{A}[1:r,:]). \qquad \text{(A.66)}$$

Alternatively, (A.66) might be verified using Cauchy's interlacing theorem applied to $\mathbf{A}\mathbf{A}^T$. For a vector $\gamma \in \mathbb{R}^{r\times r}$ and matrix $\mathbf{A} \in \mathbb{R}^{r\times r}$, we also have the useful inequality,

$$\sigma_r(\text{diag}(\gamma)\mathbf{A}) \geq \min_{i\in[r]} |\gamma[i]| \cdot \sigma_r(\mathbf{A}), \tag{A.67}$$

where $\text{diag}(\gamma) \in \mathbb{R}^{r\times r}$ is the diagonal matrix formed from the entries of γ. Using the above inequalities, we may write that,

$$\begin{aligned}
\sigma_r(\mathbf{Y}_k) &= \sigma_r(\mathbf{S}\boldsymbol{\Sigma}\mathbf{G}_k) \qquad \text{(see (A.52))}\\
&= \sigma_r(\boldsymbol{\Sigma}\mathbf{G}_k) \qquad \left(\mathbf{S}^T\mathbf{S} = \mathbf{I}_d\right)\\
&\geq \sigma_r\left(\boldsymbol{\Sigma}[1:r,1:r]\cdot\mathbf{G}_k[1:r,:]\right) \qquad \text{(see (A.66))}\\
&\geq \sigma_r \cdot \sigma_r\left(\mathbf{G}_k[1:r,:]\right). \qquad \text{(see (A.67, A.6))}
\end{aligned} \tag{A.68}$$

Note also that $\mathbf{G}_k[1:r,:] \in \mathbb{R}^{r\times kb}$ above is a standard Gaussian random matrix. Now by using the spectral properties listed in Chapter A, we can therefore write the following,

$$\begin{aligned}
\sigma_r(\mathbf{Y}_k) &\geq \sigma_r \cdot \sigma_r\left(\mathbf{G}_k[1:r,:]\right) \qquad \text{(see (A.68))}\\
&\geq \sigma_r \cdot (\sqrt{kb} - \alpha\sqrt{r}) \qquad \text{(see (A.1) and } b \geq r)\\
&\geq \sigma_r \cdot p^{-\frac{1}{6}}\sqrt{kb}, \qquad \left(\text{if } b \geq \frac{\alpha^2 r}{(1-p^{-\frac{1}{6}})^2}\right)
\end{aligned} \tag{A.69}$$

except with a probability of at most $e^{-C\alpha^2 r}$. This completes our proof for Lemma A.6.2.

A.9 Proof of Lemma A.6.3

Without loss of generality, we set $\mathbf{S} = \mathbf{I}_d$ in (A.5) to simplify the presentation, as this renders the contribution of the bottom rows of $\mathbf{B}_k$ to the innovation typically small. We first separate this term via the following inequality,

$$\begin{aligned}
\|\mathbf{P}_{\mathcal{S}^\perp_{k-1,r}}\mathbf{B}_k\|_F &= \left\|\mathbf{P}_{\mathcal{S}^\perp_{k-1,r}}\begin{bmatrix}\mathbf{B}_k[1:r,:]\\ \mathbf{B}_k[r+1:d,:]\end{bmatrix}\right\|_F\\
&\leq \left\|\mathbf{P}_{\mathcal{S}^\perp_{k-1,r}}\begin{bmatrix}\mathbf{B}_k[1:r,:]\\ \mathbf{0}_{(d-r)\times b}\end{bmatrix}\right\|_F + \|\mathbf{B}_k[r+1:d,:]\|_F. \qquad \text{(triangle inequality)}
\end{aligned} \tag{A.70}$$

Now, in order to control the last norm above, we can simply write the following to do so,

$$\begin{aligned}\|\mathbf{B}_k[r+1:d,:]\|_F &= \|\mathbf{\Sigma}[r+1:d,r+1:d]\cdot\mathbf{W}_k[r+1:d,:]\|_F \qquad \text{(see (A.52))}\\ &\le \alpha\|\mathbf{\Sigma}[r+1:d,r+1:d]\|_F\sqrt{2b} \qquad \text{(see (A.3))}\\ &= \alpha\rho_r\sqrt{2b}, \qquad \text{(see (A.7))}\end{aligned} \tag{A.71}$$

except with a probability of at most $e^{-C\alpha^2 b}$. In the second line above, we used the fact that $\mathbf{W}_k$ is a standard Gaussian random matrix. It therefore remains to control the first norm in the last line of (A.70). Note that,

$$\begin{aligned}\left\|\mathbf{P}_{\mathcal{S}^\perp_{k-1,r}}\begin{bmatrix}\mathbf{B}_k[1:r,:]\\ \mathbf{0}_{(d-r)\times b}\end{bmatrix}\right\|_F &= \left\|\mathbf{P}_{\mathcal{S}^\perp_{k-1,r}}\begin{bmatrix}\mathbf{I}_r & \\ & \mathbf{0}_{d-r}\end{bmatrix}\cdot\begin{bmatrix}\mathbf{B}_k[1:r,:]\\ \mathbf{0}_{(d-r)\times b}\end{bmatrix}\right\|_F\\ &=: \left\|\mathbf{P}_{\mathcal{S}^\perp_{k-1,r}}\mathbf{J}_r\cdot\begin{bmatrix}\mathbf{B}_k[1:r,:]\\ \mathbf{0}_{(d-r)\times b}\end{bmatrix}\right\|_F\\ &\le \|\mathbf{P}_{\mathcal{S}^\perp_{k-1,r}}\mathbf{J}_r\|_F\cdot\|\mathbf{B}_k[1:r,:]\|\\ &\le \|\mathbf{P}_{\mathcal{S}^\perp_{k-1,r}}\mathbf{J}_r\|_F\cdot\|\mathbf{\Sigma}[1:r,1:r]\|\cdot\|\mathbf{W}_k[1:r,:]\| \qquad \text{(see (A.52))}\\ &\le \|\mathbf{P}_{\mathcal{S}^\perp_{k-1,r}}\mathbf{J}_r\|_F\cdot\sigma_1\cdot(\sqrt{b}+\alpha\sqrt{r}) \qquad \text{(see (A.6, A.1))}\\ &\le \|\mathbf{P}_{\mathcal{S}^\perp_{k-1,r}}\mathbf{J}_r\|_F\cdot\sigma_1\sqrt{2b}, \qquad \left(\text{if } b\ge C\alpha^2 r\right)\end{aligned} \tag{A.72}$$

except with a probability of at most $e^{-C\alpha^2 r}$ and provided that $b \ge C\alpha^2 r$. The fifth line above again exploits the fact that $\mathbf{W}_k$ is a standard Gaussian random matrix. Let us now estimate the norm in the last line above. To do so, let us recall that $\mathbf{P}_{\mathcal{S}_{k-1,r}} \in \mathbb{R}^{d\times d}$ projects onto the span of $\mathbf{Y}_{k-1,r} = \mathrm{SVD}(\mathbf{Y}_{k-1}, r)$. That is, $\mathbf{P}_{\mathcal{S}_{k-1,r}}$ projects onto the span of leading r left singular vectors of $\mathbf{Y}_{k-1} = \mathbf{\Sigma}\mathbf{G}_{k-1}$, see (A.52). Further, because the diagonal entries of $\mathbf{\Sigma} \in \mathbb{R}^{d\times d}$ are in non-increasing order, it is natural to expect that $\mathbf{P}_{\mathcal{S}_{k-1,r}} \approx \mathbf{J}_r$. We now formalise this notion using standard results from perturbation theory. Note that one might think of $\mathbf{Y}_{k-1,r} = \mathrm{SVD}(\mathbf{Y}_{k-1}, r)$ as a perturbed copy of $\mathbf{Y}_{k-1}$. Further, note also that $\mathbf{J}_r$ is the orthogonal projection onto the subspace

$$\mathrm{span}\left(\begin{bmatrix}\mathbf{Y}_{k-1}[1:r,:]\\ \mathbf{0}_{(d-r)\times(k-1)b}\end{bmatrix}\right),$$

because $\mathbf{Y}_{k-1}[1:r,:]$ is almost surely full-rank. An application of Lemma 6 in [58] with the first matrix $\mathbf{A}$ as specified inside the parenthesis above and for the second matrix we can use

$\mathbf{Y}_{k-1}$ then it follows that,

$$\begin{aligned}
\|\mathbf{P}_{\mathcal{S}_{k-1,r}^{\perp}} \mathbf{J}_r\|_F &\le \frac{\left\| \mathbf{Y}_{k-1} - \left[\begin{array}{c} \mathbf{Y}_{k-1}[1:r,:] \\ \mathbf{0}_{(d-r)\times(k-1)b} \end{array} \right] \right\|_F}{\sigma_r(\mathbf{Y}_{k-1})} \\
&= \frac{\|\mathbf{Y}_{k-1}[r+1:d,:]\|_F}{\sigma_r(\mathbf{Y}_{k-1})} \\
&= \frac{\|\mathbf{\Sigma}[r+1:d,r+1:d]\cdot \mathbf{G}_{k-1}[r+1:d,:]\|_F}{\sigma_r(\mathbf{Y}_{k-1})} \qquad \text{(see (A.52))} \\
&\le \frac{\alpha \|\mathbf{\Sigma}[r+1:d,r+1:d]\|_F \sqrt{2(k-1)b}}{\sigma_r \sqrt{(k-1)b/2}} \qquad \text{(see (A.3) and Lemma A.6.2 with } p=8) \\
&= \frac{2\alpha\rho_r}{\sigma_r}, \qquad \text{(see (A.7))}
\end{aligned} \tag{A.73}$$

provided that $b \ge C\alpha^2 r$ and except with a probability of at most $e^{-C\alpha^2 b} + e^{-C\alpha^2 r} \le e^{-C\alpha^2 r}$, where this last inequality follows from the assumption that $b \ge r$. It also trivially holds that,

$$\|\mathbf{P}_{\mathcal{S}_{k-1,r}^{\perp}} \mathbf{J}_r\|_F \le \|\mathbf{P}_{\mathcal{S}_{k-1,r}^{\perp}}\| \cdot \|\mathbf{J}_r\|_F \le \|\mathbf{J}_r\|_F = \|\mathbf{I}_r\|_F = \sqrt{r},$$

where we used above the definition of $\mathbf{J}_r$ in (A.72). Therefore, overall we find that,

$$\|\mathbf{P}_{\mathcal{S}_{k-1,r}^{\perp}} \mathbf{J}_r\|_F \le \min\left(\frac{2\alpha\rho_r}{\sigma_r}, \sqrt{r}\right). \tag{A.74}$$

Substituting the above bound back into (A.72) yields that,

$$\begin{aligned}
\left\| \mathbf{P}_{\mathcal{S}_{k-1,r}^{\perp}} \left[\begin{array}{c} \mathbf{B}_k[1:r,:] \\ \mathbf{0}_{(d-r)\times b} \end{array} \right] \right\|_F &\le \|\mathbf{P}_{\mathcal{S}_{k-1,r}^{\perp}} \mathbf{J}_r\|_F \cdot \sigma_1 \sqrt{2b} \qquad \text{(see (A.72))} \\
&\le \min\left(\alpha\kappa_r\rho_r, \sigma_1\sqrt{r}\right)\sqrt{8b}, \qquad \text{(see (A.74, A.7))}
\end{aligned} \tag{A.75}$$

except with a probability of at most $e^{-C\alpha^2 r}$. Combining (A.71) and (A.75) finally controls the innovation as follows,

$$\begin{aligned}
\|\mathbf{P}_{\mathcal{S}_{k-1,r}^{\perp}} \mathbf{B}_k\|_F &\le \left\| \mathbf{P}_{\mathcal{S}_{k-1,r}^{\perp}} \left[\begin{array}{c} \mathbf{B}_k[1:r,:] \\ \mathbf{0}_{(n-r)\times b} \end{array} \right] \right\|_F + \|\mathbf{B}_k[r+1:d,:]\|_F \qquad \text{(see (A.70))} \\
&\le \min\left(\alpha\kappa_r\rho_r, \sigma_1\sqrt{r}\right)\sqrt{8b} + \alpha\rho_r\sqrt{2b} \qquad \text{(see (A.75, A.71))} \\
&\le 5\alpha \min\left(\kappa_r\rho_r, \sigma_1\sqrt{r} + \rho_r\right)\sqrt{b}, \qquad (\alpha, \kappa_r \ge 1)
\end{aligned} \tag{A.76}$$

except with a probability of at most $e^{-C\alpha^2 r}$ and provided that $b \ge C\alpha^2 r$. Finally, this step completes the proof of Lemma A.6.3.

Appendix B

Supplementary Material for Chapter 4

This comes as supplementary material for Chapter 5. The appendix is structured as follows:

1. Federated-PCA's local update guarantees,
2. Federated-PCA's differential privacy properties,
3. In-depth analysis of algorithm's federation,
4. Additional evaluation and discussion.

Furthermore, we complement our theoretical analysis with additional empirical evaluation on synthetic and real datasets which include details on memory consumption.

B.1 Local Update Guarantees

We note that the local updating procedure in Algorithm 10 inherits the theoretical guarantees from the incremental SVD algorithm, namely MOSES, we proposed in Chapter 3. This is used as a primary building block for Federated-PCA and is not discussed again, as is the case in the original paper which was done for brevity.

B.1.1 Adaptive Rank Estimation

Our algorithm provides a scheme to *adaptively* adjust the rank of each individual estimation based on the distribution seen so far. This can be helpful when there are distribution shifts and/or changes in the data over time. The scheme uses a thresholding procedure that consists in bounding the minimum and maximum contributions of $\sigma_r(\mathbf{Y}_\tau)$ to the variance $\sum_{i=1}^{r} \sigma_i(\mathbf{Y}_\tau)$ of the dataset. That is, by enforcing

$$\mathcal{E}_r^{\mathbf{Y}_\tau} = \frac{\sigma_r(\mathbf{Y}_\tau)}{\sum_{i=1}^{r} \sigma_i(\mathbf{Y}_\tau)} \in [\alpha, \beta], \tag{B.1}$$

for some $\alpha, \beta > 0$ and increasing r whenever $\mathcal{E}_r(\mathbf{Y}_\tau) > \beta$ or decreasing it when $\mathcal{E}_r(\mathbf{Y}_\tau) < \alpha$. As a guideline, from our experiments a typical ratio of α/β should be less or equal to 0.2 which could be used as an reference point when picking their values. This ensure that each client will have a bounded Frobenius norm at any given point in time. With this procedure, we are able to bound the global error as

$$\rho_{r_{\max(\alpha,\beta)}}(\mathbf{Y}_{kb}) \leq \mathbf{Y}_{\text{err}} \leq \rho_{r_{\min(\alpha,\beta)}}(\mathbf{Y}_{kb}). \tag{B.2}$$

Proof. At iteration $k \in \{1, \dots, K\}$, each node computes $\hat{\mathbf{Y}}^{\text{local}}_{kb}$, the best rank-$r$ approximation of $\mathbf{Y}_{kb}$ using iteration (4.3). Hence, for each $k \in \{1, \dots, K\}$, the error of the approximation is given by $\|\mathbf{Y}_{kb} - \hat{\mathbf{Y}}^{\text{local}}_{kb}\|_F = \rho_r(\mathbf{Y}_{kb})$. Let $r_{\min} = r_{\min}(\alpha, \beta)$ and $r_{\max} = r_{\max}(\alpha, \beta) > 0$ be the minimum and maximum rank estimates in when running FPCA. The result follows from

$$\rho_{r_{\max(\alpha,\beta)}}(\mathbf{Y}_{kb}) \leq \mathbf{Y}_{\text{err}} \leq \rho_{r_{\min(\alpha,\beta)}}(\mathbf{Y}_{kb}).$$

Where $\mathbf{Y}_{\text{err}} = \|\mathbf{Y}_{kb} - \hat{\mathbf{Y}}^{\text{local}}_{kb}\|_F$ □

Furthermore, we can express the global bound in a different form which can give us a more descriptive overall bound. To this end we know that for each local worker its $\|\cdot\|_F$ accumulated error any given time is bounded by the ratio of the summation of its singular values.

Lemma B.1.1. *Let $\|\cdot\|_F^M \in \{1, \dots, M\}$ be the error accumulated for each of the M clients at block τ; then, after merging operations the global error will be $\sum_{i=1}^M \mathcal{E}_M^{\mathbf{Y}_\tau}$.*

Proof. By Equation (B.1) we know that the error is deterministically bounded for each of the M clients at any given block τ. Further, we also know that the merging as in (Algorithm 6) is able to merge the target subspaces with minimal error and thus at any given block τ we can claim that $\sum_{i=1}^M \mathcal{E}_M^{\mathbf{Y}_\tau} + c_m$ where c_m is a small constant depicting the error accumulated during the merging procedure of the subspaces, thus when asymptotically eliminating the constant factors the final error is $\sum_{i=1}^M \mathcal{E}_M^{\mathbf{Y}_\tau}$. □

B.2 Privacy Preserving Properties of Federated PCA

In this section we prove Lemma 4.2.3, which summarises the differential privacy properties of our method. The arguments are based on the proofs given by [32]. Lemma B.2.1 proves the first part of Lemma 4.2.3 by extending MOD-SuLQ to the case of non-symmetric noise matrices. The second part of Lemma 4.2.3 is a direct corollary of Lemma B.2.1. The third part follows directly from Lemmas B.2.5 and B.2.6.

Lemma B.2.1 (Differential privacy)**.** *Let $\mathbf{X} \in \mathbb{R}^{d\times n}$ be a dataset with orthonormal columns and $\mathbf{A} = \frac{1}{n}\mathbf{X}\mathbf{X}^T$. Let*

$$\omega(\varepsilon, \delta, d, n) = \frac{4d}{\varepsilon n}\sqrt{2\log\left(\frac{d^2}{\delta\sqrt{2\pi}}\right)} + \frac{\sqrt{2}}{\sqrt{\varepsilon n}}, \tag{B.3}$$

and $\mathbf{N}_{\varepsilon,\delta,d,n} \in \mathbb{R}^{d\times d}$ be a non-symmetric random Gaussian matrix with i.i.d. entries drawn from $\mathcal{N}(0,\omega^2)$. Then, the principal components of $\frac{1}{n}\mathbf{X}\mathbf{X}^T + \mathbf{N}_{\varepsilon,\delta,d,n}$ are (ε,δ)-differentially private.

Proof. Let $\mathbf{N}, \hat{\mathbf{N}} \in \mathbb{R}^{d\times d}$ be two random matrices such that $\mathbf{N}_{i,j}$ and $\hat{\mathbf{N}}_{i,j}$ are i.i.d. random variables drawn from $\mathcal{N}(0,\omega^2)$. Let $\mathcal{D} = \{\mathbf{x}_i : i \in [n]\} \subset \mathbb{R}^d$ be a dataset and let $\hat{\mathcal{D}} = \mathcal{D} \cup \{\hat{\mathbf{x}}_n\} \setminus \{\mathbf{x}_n\}$. Form the matrices

$$\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_{n-1}, \mathbf{x}_n] \tag{B.4}$$

$$\hat{\mathbf{X}} = [\mathbf{x}_1, \dots, \mathbf{x}_{n-1}, \hat{\mathbf{x}}_n]. \tag{B.5}$$

Let $\mathbf{Y} = [\mathbf{x}_1, \dots \mathbf{x}_{n-1}]$. Then, the covariance matrices for these datasets are

$$\mathbf{A} = \frac{1}{n}[\mathbf{Y}\mathbf{Y}^T + \mathbf{x}_n\mathbf{x}_n^T] \tag{B.6}$$

$$\hat{\mathbf{A}} = \frac{1}{n}[\mathbf{Y}\mathbf{Y}^T + \hat{\mathbf{x}}_n\hat{\mathbf{x}}_n^T]. \tag{B.7}$$

Now, let $\mathbf{G} = \mathbf{A} + \mathbf{B}$ and $\hat{\mathbf{G}} = \hat{\mathbf{A}} + \hat{\mathbf{B}}$ and consider the log-ratio of their densities at point $\mathbf{H} \in \mathbb{R}^{d\times d}$.

$$\begin{aligned}\log\frac{f_{\mathbf{G}}(\mathbf{H})}{f_{\hat{\mathbf{G}}}(\mathbf{H})} &= \frac{1}{2\omega^2}\sum_{i,j=1}^{d}\left(-(\mathbf{H}_{i,j} - \mathbf{A}_{i,j})^2 + (\mathbf{H}_{i,j} - \hat{\mathbf{A}}_{i,j})^2\right) \\ &= \frac{1}{2\omega^2}\sum_{i,j=1}^{d}\left(\frac{2}{n}(\mathbf{A}_{i,j} - \mathbf{H}_{i,j})(\hat{\mathbf{x}}_n\hat{\mathbf{x}}_n^T - \mathbf{x}_n\mathbf{x}_n^T)_{i,j} + \frac{1}{n^2}(\hat{\mathbf{x}}_n\hat{\mathbf{x}}_n^T - \mathbf{x}_n\mathbf{x}_n^T)_{i,j}^2\right) \\ &= \frac{1}{2\omega^2}\sum_{i,j=1}^{d}\left(\frac{2}{n}(\mathbf{A}_{i,j} - \mathbf{H}_{i,j})(\hat{\mathbf{x}}_{n,i}\hat{\mathbf{x}}_{n,j} - \mathbf{x}_{n,i}\mathbf{x}_{n,j}) + \frac{1}{n^2}(\hat{\mathbf{x}}_{n,i}\hat{\mathbf{x}}_{n,j} - \mathbf{x}_{n,i}\mathbf{x}_{n,j})^2\right).\end{aligned} \tag{B.8}$$

Note that if $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$ are such that $\|\mathbf{x}\| = \|\mathbf{y}\| = 1$ are unit vectors, then

$$\sum_{i,j=1}^{d}(\mathbf{x}_i\mathbf{x}_j - \mathbf{y}_i\mathbf{y}_j)^2 \leq 4. \tag{B.9}$$

Moreover,

$$
\begin{aligned}
\sum_{i,j=1}^{d} (\hat{\mathbf{x}}_{n,i}\hat{\mathbf{x}}_{n,j} - \mathbf{x}_{n,i}\mathbf{x}_{n,j}) &\leq \sum_{i,j=1}^{d} |\hat{\mathbf{x}}_{n,i}\hat{\mathbf{x}}_{n,j}| + \sum_{i,j=1}^{d} |\mathbf{x}_{n,i}\mathbf{x}_{n,j}| && \text{(B.10)}\\
&\leq 2 \max_{\mathbf{z}:\|\mathbf{z}\|\leq 1} \sum_{i,j=1}^{d} \mathbf{z}_i\mathbf{z}_j && \text{(B.11)}\\
&\leq 2 \max_{\mathbf{z}:\|\mathbf{z}\|\leq 1} \|\mathbf{z}\|_1^2 && \text{(B.12)}\\
&\leq 2 \max_{\mathbf{z}:\|\mathbf{z}\|\leq 1} (\sqrt{d}\|\mathbf{z}\|_2)^2 && \text{(B.13)}\\
&\leq 2d. && \text{(B.14)}
\end{aligned}
$$

Using these observations to bound (B.8), and using the fact that for any $\gamma \in \mathbb{R}$ the events $\{\forall\, i,j : \mathbf{N}_{i,j} \leq \gamma\}$ and $\{\exists\, i,j : \mathbf{N}_{i,j} > \gamma\}$ are complementary, we obtain that for any measurable set $\mathcal{S}$ of matrices,

$$
\mathbb{P}(\mathbf{G} \in \mathcal{S}) \leq \exp\left(\frac{1}{2\omega^2}\left(\frac{4}{n}d\gamma + \frac{4}{n^2}\right)\right) + \mathbb{P}(\exists\, i,j : \mathbf{N}_{i,j} > \gamma). \tag{B.15}
$$

Moreover, if $\gamma > \omega$, we can use the union bound with a Gaussian tail bound to obtain

$$
\begin{aligned}
\delta := \mathbb{P}(\exists\, i,j : \mathbf{N}_{i,j} > \gamma) &= \mathbb{P}\left(\bigcup_{i,j=1}^{d} \{\mathbf{N}_{i,j} > \gamma\}\right)\\
&\leq \sum_{i,j=1}^{d} \mathbb{P}\left(\mathbf{N}_{i,j} > \gamma\right)\\
&\leq \sum_{i,j=1}^{d} \left(\frac{1}{\sqrt{2\pi}} e^{-\frac{\gamma^2}{2\omega^2}}\right)\\
&= \frac{d^2}{\sqrt{2\pi}} e^{-\frac{\gamma^2}{2\omega^2}} && \text{(B.16)}
\end{aligned}
$$

Now, solving for γ in (B.16) we obtain,

$$
\gamma = \omega\sqrt{2\log\left(\frac{d^2}{\delta\sqrt{2\pi}}\right)} \tag{B.17}
$$

Substituting (B.17) in (B.15) we can give an expression for (ε, δ)-differential privacy by letting

$$
\varepsilon = \frac{1}{2\omega^2}\left(\frac{4}{n}d\left(\omega\sqrt{2\log\left(\frac{d^2}{\delta\sqrt{2\pi}}\right)}\right) + \frac{4}{n^2}\right). \tag{B.18}
$$

This yields a quadratic equation on ω, which we can rewrite as

$$2\varepsilon\omega^2 - \frac{4}{n}d\left(\omega\sqrt{2\log\left(\frac{d^2}{\delta\sqrt{2\pi}}\right)}\right)\omega - \frac{4}{n^2} = 0. \tag{B.19}$$

Using the quadratic formula to solve for ω in (B.19) yields,

$$\begin{aligned}\omega &= \frac{2d}{\varepsilon n}\sqrt{2\log\left(\frac{d^2}{\delta\sqrt{2\pi}}\right)} \pm \frac{2}{\varepsilon n}\sqrt{2d^2\log\left(\frac{d^2}{\delta\sqrt{2\pi}}\right)+\frac{\varepsilon}{2}} \\ &\leq \frac{2d}{\varepsilon n}\sqrt{2\log\left(\frac{d^2}{\delta\sqrt{2\pi}}\right)} + \frac{2}{\varepsilon n}\left(\sqrt{2d^2\log\left(\frac{d^2}{\delta\sqrt{2\pi}}\right)}+\sqrt{\frac{\varepsilon}{2}}\right) \\ &= \frac{4d}{\varepsilon n}\sqrt{2\log\left(\frac{d^2}{\delta\sqrt{2\pi}}\right)} + \frac{\sqrt{2}}{\sqrt{\varepsilon}n}.\end{aligned}$$

□

To prove the utility bound in Lemma B.2.5 of Streaming MOD-SuLQ, we will exploit Lemmas B.2.2, B.2.3, and B.2.4.

Lemma B.2.2 (Packing result [32])**.** *For $\phi \in [(2\pi d)^{-1/2}, 1)$, there exists a set $\mathcal{C} \subset \mathbb{S}^{d-1}$ with*

$$|\mathcal{C}| = \frac{1}{8}\exp\left((d-1)\log\frac{1}{\sqrt{1-\phi^2}}\right) \tag{B.20}$$

and such that $|\langle\boldsymbol{\mu}, \mathbf{v}\rangle| \leq \phi$ for all $\boldsymbol{\mu}, \mathbf{v} \in \mathcal{C}$.

Lemma B.2.3 (Kullback-Leibler for Gaussian random variables)**.** *Let $\boldsymbol{\Sigma}$ be a positive definite matrix and let f and g denote, respectively, the densities $\mathcal{N}(\mathbf{a}, \boldsymbol{\Sigma})$ and $\mathcal{N}(\mathbf{b}, \boldsymbol{\Sigma})$. Then,*

$$\mathbf{KL}(f \mid\mid g) = \frac{1}{2}(\mathbf{a}-\mathbf{b})^T\boldsymbol{\Sigma}(\mathbf{a}-\mathbf{b}). \tag{B.21}$$

Proof. The proof follows directly by using the definition of the Kullback-Leibler (KL) divergence and simplifying. □

Lemma B.2.4 (Fano's inequality [197])**.** *Let $\mathcal{R}$ be a set and Θ be a parameter space with a pseudo-metric $d(\cdot)$. Let $\mathcal{F}$ be a set of r densities $\{f_1, \ldots, f_r\}$ on $\mathcal{R}$ corresponding to parameter values $\{\theta_1, \ldots, \theta_r\}$ in Θ. Let X have a distribution $f \in \mathcal{F}$ with corresponding parameter θ and let $\hat{\theta}(X)$ be an estimate of θ. If for all i, j, $d(\theta_i, \theta_j) \geq \tau$ and $\mathbf{KL}(f_i \mid\mid f_j) \geq \gamma$, then*

$$\max_j \mathbb{E}_j\left[d(\hat{\theta}, \theta_j)\right] \geq \frac{\tau}{2}\left(1 - \frac{\gamma + \log 2}{\log r}\right). \tag{B.22}$$

We are now ready to give a bound on the utility for Streaming MOD-SuLQ. We note that the proof for Lemma B.2.5 is identical as the one given in [32] except for a few equations where the dimension of the object considered changes from $\frac{d(d+1)}{2}$ to d^2. We also note that while the utility bound has the same functional form, it is not identical to the one given in [32] since it depends on the value of $\omega = \omega(\varepsilon, \delta, d, n)$ given in Lemma 4.2.3.

Lemma B.2.5 (Utility bounds)**.** *Let $d, n \in \mathbb{N}$ and $\varepsilon > 0$ be given and let ω be given as in Lemma 4.2.3, so that the output of Streaming* MOD-SuLQ *is (ε, δ) differentially private for all datasets $\mathbf{X} \in \mathbb{R}^{d\times n}$. Then, there exists a dataset with n elements such that if $\hat{\mathbf{v}}_1$ denotes the output of the Streaming* MOD-SuLQ *and $\mathbf{v}_1$ is the top eigenvector of the empirical covariance matrix of the dataset, the expected correlation $\langle \mathbf{v}_1, \hat{\mathbf{v}}_1 \rangle$ is upper bounded,*

$$\mathbb{E}\left[|\langle \mathbf{v}_1, \hat{\mathbf{v}}_1 \rangle|\right] \leq \min_{\phi\in\Phi} \left(1 - \frac{1-\phi}{4}\left(1 - \frac{1/\omega^2 + \log 2}{(d-1)\log\frac{1}{\sqrt{1-\phi^2}} - \log 8}\right)^2\right) \tag{B.23}$$

where

$$\Phi \in \left[\max\left\{\frac{1}{\sqrt{2\pi d}}, \sqrt{1-\exp\left(-\frac{2\log(8d)}{d-1}\right)}, \sqrt{1-\exp\left(-\frac{2/\omega^2+\log 256}{d-1}\right)}\right\}\right]. \tag{B.24}$$

Proof. Let $\mathcal{C}$ be an orthonormal basis in $\mathbb{R}^d$. Then, $|\mathcal{C}| = d$, so solving for ϕ in (B.20) yields

$$\phi = \sqrt{1-\exp\left(-\frac{2\log(8d)}{d-1}\right)}. \tag{B.25}$$

For any unit vector $\boldsymbol{\mu}$ let $\mathbf{A}(\boldsymbol{\mu}) = \boldsymbol{\mu}\boldsymbol{\mu}^T + \mathbf{N}$ where $\mathbf{N}$ is a symmetric random matrix such that $\{\mathbf{N}_{i,j} : i \leq i \leq j \leq d\}$ are i.i.d. $\mathcal{N}(0, \omega^2)$ and ω^2 is the noise variance used in the Streaming MOD-SuLQ algorithm. The matrix $\mathbf{A}(\boldsymbol{\mu})$ can be thought of as a jointly Gaussian random vector on d^2 variables. The mean and covariance of this vector is

$$\mathbb{E}[\boldsymbol{\mu}] = (\boldsymbol{\mu}_1^2, \ldots, \boldsymbol{\mu}_d^2, \boldsymbol{\mu}_1\boldsymbol{\mu}_2, \ldots, \boldsymbol{\mu}_{d-1}\boldsymbol{\mu}_d, \boldsymbol{\mu}_2\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_d\boldsymbol{\mu}_{d-1}) \in \mathbb{R}^{d^2}, \tag{B.26}$$
$$\mathrm{Cov}[\boldsymbol{\mu}] = \omega^2 \mathbf{I}_{d^2\times d^2} \in \mathbb{R}^{d^2\times d^2}. \tag{B.27}$$

For $\boldsymbol{\mu}, \boldsymbol{\nu} \in \mathcal{C}$, the divergence can be calculated using Lemma B.2.3 yielding

$$\mathbf{KL}(f_{\boldsymbol{\mu}} \mid\mid f_{\boldsymbol{\nu}}) \leq \frac{1}{\omega^2}. \tag{B.28}$$

For any two vectors $\boldsymbol{\mu}, \boldsymbol{\nu} \in \mathcal{C}$, we have that $|\langle \boldsymbol{\mu}, \boldsymbol{\nu} \rangle| \leq \phi$, so that $-\phi \leq -\langle \boldsymbol{\mu}, \boldsymbol{\nu} \rangle$. Therefore,

$$\begin{aligned} \|\boldsymbol{\mu} - \boldsymbol{\nu}\|^2 &= \langle \boldsymbol{\mu} - \boldsymbol{\nu}, \boldsymbol{\mu} - \boldsymbol{\nu} \rangle && \text{(B.29)} \\ &= \|\boldsymbol{\mu}\|^2 + \|\boldsymbol{\nu}\|^2 - 2\langle \boldsymbol{\mu}, \boldsymbol{\nu} \rangle && \text{(B.30)} \\ &= 2(1 - \langle \boldsymbol{\mu}, \boldsymbol{\nu} \rangle) && \text{(B.31)} \\ &\geq 2(1 - \phi). && \text{(B.32)} \end{aligned}$$

From (B.28) and (B.32), the set $\mathcal{C}$ satisfies the conditions of Lemma B.2.4 with $\mathcal{F} = \{f_{\boldsymbol{\mu}} : \boldsymbol{\mu} \in \mathcal{C}\}$, $r = K$ and $\tau = \sqrt{2(1-\phi)}$, and $\gamma = 1/\omega^2$. Hence, this shows that for Streaming MOD-SuLQ,

$$\max_{\boldsymbol{\mu} \in \mathcal{C}} \mathbb{E}_{f_{\boldsymbol{\mu}}} \left[\|\hat{\boldsymbol{v}} - \boldsymbol{\mu}\| \right] \geq \frac{\sqrt{2(1-\phi)}}{2} \left(1 - \frac{1/\omega^2 + \log 2}{\log K} \right) \tag{B.33}$$

As mentioned in [32] this bound is vacuous when the term inside the parentheses is negative which imposes further conditions on ϕ. Setting $K = 1/\omega^2 + \log 2$, we can solve to find another lower bound on ϕ:

$$\phi \geq \sqrt{1 - \exp\left(-\frac{2/\omega^2 + \log 256}{d-1} \right)} \tag{B.34}$$

Using Jensen's inequality on the left hand side of (B.33) yields

$$\max_{\boldsymbol{\mu} \in \mathcal{C}} \mathbb{E}_{f_{\boldsymbol{\mu}}} \left[2(1 - |\langle \hat{\mathbf{v}}, \boldsymbol{\mu} \rangle|) \right] \geq \frac{(1-\phi)}{2} \left(1 - \frac{1/\omega^2 + \log 2}{\log K} \right)^2 \tag{B.35}$$

so there is a $\boldsymbol{\mu}$ such that

$$\mathbb{E}_{f_{\boldsymbol{\mu}}} \left[|\langle \hat{\mathbf{v}}, \boldsymbol{\mu} \rangle| \right] \leq 1 - \frac{(1-\phi)}{4} \left(1 - \frac{1/\omega^2 + \log 2}{\log K} \right)^2 . \tag{B.36}$$

Now, consider the dataset $\mathbf{D} = [\boldsymbol{\mu} \cdots \boldsymbol{\mu}] \in \mathbb{R}^{d^2 \times n}$. This dataset has covariance matrix equal to $\boldsymbol{\mu}\boldsymbol{\mu}^T$ and has top eigenvector equal to $\mathbf{v}_1 = \boldsymbol{\mu}$. The output of the algorithm Streaming MOD-SuLQ applied to $\mathbf{D}$ approximates $\boldsymbol{\mu}$, so satisfies (B.36). Minimising this equation over ϕ yields the required result. □

Lemma B.2.6 (Sample complexity)**.** *For (ϵ, δ) and $d \in \mathbb{N}$, there are constants $C_1 > 0$ and $C_2 > 0$ such that with*

$$n \geq C_1 \frac{d^{3/2} \sqrt{\log(d/\delta)}}{\varepsilon} \left(1 - C_2 \left(1 - \mathbb{E}_{f_{\boldsymbol{\mu}}} \left[|\langle \hat{\mathbf{v}}, \boldsymbol{\mu} \rangle| \right] \right) \right), \tag{B.37}$$

where $\boldsymbol{\mu}$ is the first principal component of the dataset $\mathbf{X} \in \mathbb{R}^{d \times n}$ and $\hat{\mathbf{v}}$ is the first principal component estimated by Streaming MOD-SULQ.

Proof. Using (B.36), and letting $\mathbb{E}_{f_\mu}\left[|\langle \hat{\mathbf{v}}, \boldsymbol{\mu}\rangle|\right] = \rho$, we obtain,

$$2\sqrt{1-\rho} \geq \min_{\phi\in\Phi} \sqrt{1-\phi}\left(1 - \frac{1/\omega^2 + \log 2}{(d-1)\log\frac{1}{\sqrt{1-\phi^2}} - \log 8}\right) \tag{B.38}$$

Picking ϕ so that the fraction in the right-hand side becomes 0.5, we obtain,

$$4\sqrt{1-\rho} \geq \sqrt{1-\phi}. \tag{B.39}$$

Moreover, as $d, n \to \infty$, this value of ϕ guarantee implies an asymptotic of the form

$$\log\frac{1}{\sqrt{1-\phi^2}} \sim \frac{2}{\omega^2 d} + o(1). \tag{B.40}$$

This implies that $\phi = \Theta(\omega^{-1}d^{-1/2})$, and by (4.16) that $\omega \gtrsim d^2(\varepsilon n)^{-2}\log(d/\delta)$. Therefore, there exists $C > 0$ such that $\omega^2 > Cd^2(n\varepsilon)^{-2}\log(d/\delta)$. Since $\phi = \Theta(\omega^{-1}d^{-1/2})$ we have that for some $D > 0$

$$\phi^2 \leq D\frac{n^2\varepsilon^2}{d^3\log(d/\delta)}. \tag{B.41}$$

By (B.39) we get

$$(1 - 16(1-\rho)) \leq D\frac{n^2\varepsilon^2}{d^3\log(d/\delta)} \tag{B.42}$$

Solving for n in (B.42) yields

$$n \geq C_1\frac{d^{3/2}\sqrt{\log(d/\delta)}}{\varepsilon}(1 - C_2(1-\rho)), \tag{B.43}$$

for some constants C_1 and C_2. □

B.3 Federated PCA Analysis

In this section we will present a detailed analysis of Federated-PCA in which we will describe the merging process in detail as well as provide a detailed error analysis in the *streaming* and *federated* setting that is based is based on the mathematical tools introduced in [96].

B.3.1 Asynchronous Independent Block based SVD

We begin our proof by proving Lemma 4.2.1 (Streaming partial SVD uniqueness) which applies in the absence of perturbation masks and is the cornerstone of our federated scheme.
Proof. Let the reduced SVD representation of each of the M nodes at time t be,

$$\mathbf{Y}_t^i = \sum_{j=1}^{r} \mathbf{u}_j^i \boldsymbol{\sigma}_j^i (\mathbf{v}_j^i)^T = \hat{\mathbf{U}}_t^i \hat{\boldsymbol{\Sigma}}_t^i (\hat{\mathbf{V}}_t^i)^T, \quad i = 1, 2, \ldots, M. \tag{B.44}$$

We also know that each of the blocks $\mathbf{Y}_t^i \in [M]$ can be at most of rank d. Note that in this instance, the definition applies for only *fully* materialised matrices. However, substituting each block of $\mathbf{Y}_i^t$ with our local updates procedure as in Algorithm 10 then will generate an estimation of the reduced SVD$_r$ of that particular $\mathbf{Y}_i^t$ block with an error at most as presented in Theorem 3.4.2 subject to each update chunk being in $\mathbb{R}^{d\times b}$ with $b \geq \min \operatorname{rank}(\mathbf{Y}_t^i)$ $\forall i \in [M]$. Now, let the singular values of $\mathbf{Y}_t$ be the positive square root of the eigenvalues of $\mathbf{Y}_t\mathbf{Y}_t^T$, where as defined previously $\mathbf{Y}_t$ is the data seen so far from the M nodes. Then, by using the previously defined streaming block decomposition of a matrix $\mathbf{Y}_t$ we have the following,

$$\mathbf{Y}_t\mathbf{Y}_t^T = \sum_{i=1}^{M} \mathbf{Y}_t^i (\mathbf{Y}_t^i)^T = \sum_{i=1}^{M} \hat{\mathbf{U}}_t^i \hat{\boldsymbol{\Sigma}}_t^i (\hat{\mathbf{V}}_t{}^i)^T (\hat{\mathbf{V}}_t^i)(\hat{\boldsymbol{\Sigma}}_t^i)^T (\hat{\mathbf{U}}_t^i)^T = \sum_{i=1}^{M} \hat{\mathbf{U}}_t^i \hat{\boldsymbol{\Sigma}}_t^i (\hat{\boldsymbol{\Sigma}}_t^i)^T (\mathbf{U}_t^i)^T \tag{B.45}$$

Equivalently, the singular values of $\mathbf{Z}_t$ are similarly defined as the square root of the eigenvalues of $\mathbf{Z}_t\mathbf{Z}_t^T$.

$$\mathbf{Z}\mathbf{Z}^T = \sum_{i=1}^{M} (\hat{\mathbf{U}}_t^i \hat{\boldsymbol{\Sigma}}_t^i)(\hat{\mathbf{U}}_t^i \hat{\boldsymbol{\Sigma}}_t^i)^T = \sum_{i=1}^{M} \hat{\mathbf{U}}_t^i \hat{\boldsymbol{\Sigma}}_t^i (\hat{\boldsymbol{\Sigma}}_t^i)^T (\hat{\mathbf{U}}_t^i)^T \tag{B.46}$$

Thus $\mathbf{Y}_t\mathbf{Y}_t^T = \mathbf{Z}_t\mathbf{Z}_t^T$ at any t, hence the singular values of matrix $\mathbf{Z}_t$ must surely equal to those of matrix $\mathbf{Y}_t$. Moreover, since the left singular vectors of both $\mathbf{Y}_t$ and $\mathbf{Z}_t$ will be also eigenvectors of $\mathbf{Y}_t\mathbf{Y}_t^T$ and $\mathbf{Z}_t\mathbf{Z}_t^T$, respectively; then the eigenspaces associated with each - possibly repeated - eigenvalue will also be equal thus $\hat{\mathbf{U}}_t = \hat{\mathbf{U}}_t'\mathbf{B}_t$. The block diagonal unitary matrix $\mathbf{B}_t$ which has p unitary blocks of size $p \times p$ for each repeated eigenvalue; this enables the singular vectors which are associated with each repeated singular value to be rotated in the desired matrix representation $\hat{\mathbf{U}}_t$. In case of different update chunk sizes per worker the result is unaffected as long as the requirement for their size (b) mentioned above is kept and their rank r is the same. □

Time Order Independence

Further, a natural extension to Lemma 1 which is pivotal to a successful federated scheme is the ability to guarantee that our result will be the same regardless of the merging order in the case there are no input perturbation masks.

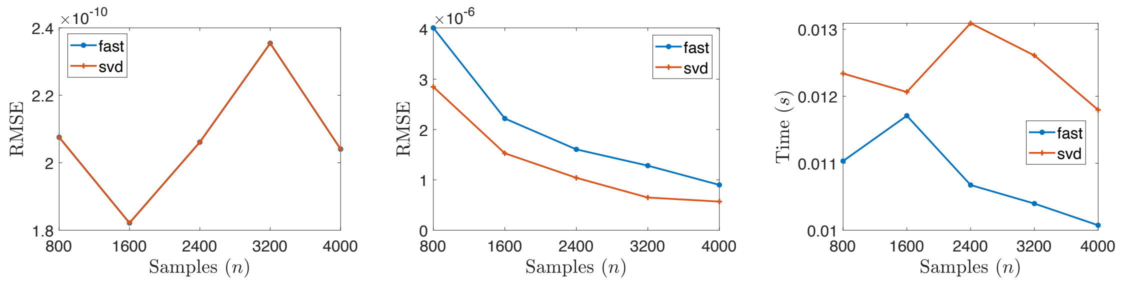
Lemma B.3.1 (Time independence). *Let* $\mathbf{Y} \in \mathbb{R}^{d\times n}$. *Then, if* $\mathbf{P} \in \mathbb{R}^{n\times n}$ *is a row permutation of the identity. Then, in the absence of input-perturbation masks,* $\mathrm{FPCA}(\mathbf{Y}) = \mathrm{FPCA}(\mathbf{YP})$.

Proof. If $\mathbf{Y} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T$ is the Singular Value Decomposition (SVD) of $\mathbf{Y}$, then $\mathbf{YP} = \mathbf{U}\boldsymbol{\Sigma}\left(\mathbf{V}^T\mathbf{P}\right)$. Since $\mathbf{V}' = \mathbf{P}^T\mathbf{V}$ is orthogonal, $\mathbf{U}\boldsymbol{\Sigma}(\mathbf{V}')^T$ is the SVD of $\mathbf{YP}$. Hence, both $\mathbf{Y}$ and $\mathbf{YP}$ have the same singular values and left principal subspaces. □

Notably, by formally proving the above Lemmas we can now exploit the following important properties: i) that we can create a block decomposition of $\mathbf{Y}_t$ for every t without fully materialising the block matrices while being able to obtain their truncated r-SVD incrementally, and ii) that the result will hold regardless of the arrival order.

B.3.2 Subspace Merging

Complimentary to the material shown in Section 4.2 and to illustrate the practical benefits of the merging algorithm we conducted an experiment in order to evaluate if the proposed algorithms' empirical performance is as expected. Concretely, we created synthetic data using $\text{Synth}(1)^{d\times n}$ function[1] with $d = 800$ and $n \in \{800, 1.6k, 2.4k, 3.2k, 4k\}$; then we split each dataset into two equal chunks each of which was processed using Federated-PCA with a target rank of 100. Then we proceeded to merge the two resulting subspaces with two different techniques, namely, with the Algorithm 4 and Algorithm 6 as well as find the offline subspace using traditionally SVD. We then show in Figure B.1 the errors incurred with respect to the offline SVD against the resulting merged subspaces and singular values of the two techniques used, as well as their execution. We can clearly see that the resulting subspaces are *identical* in all cases and that the error penalty in the singular values is minimal when compared to Algorithm 4. As expected, we also observe that derived algorithm is faster while consuming less memory. Critically speaking, the speed benefit is not significant in the single case as presented. However, these benefits can be additive in the presence of thousands of merges that would likely occur in a federated setting.



(a) **U** errors. (b) Singular Value errors. (c) Execution time.

Fig. B.1 Illustration of the benefits of Algorithm 6, in of errors of subspace (fig. B.1a), singular values (fig. B.1b), and its execution speed (fig. B.1c).

[1]If $\mathbf{Y} \sim \text{Synth}(\alpha)^{d\times n}$ iff $\mathbf{Y} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T$ with $[\mathbf{U}, \sim] = \text{QR}(\mathbf{N}^{d\times d})$, $[\mathbf{V}, \sim] = \text{QR}(\mathbf{N}^{d\times n})$, and $\boldsymbol{\Sigma}_{i,i} = i^{-\alpha}$, and $\mathbf{N}^{m\times n}$ is an $m \times n$ matrix with i.i.d. entries drawn from $\mathcal{N}(0, 1)$.

B.3.3 Federated Error Analysis

In this section we will give a lower and a upper bound of our federated approach. This is also based on the mathematical toolbox we previously used [96] but is adapted in the case of streaming block matrices.

Lemma B.3.2. *Let $\mathbf{Y}_t^i \in \mathbb{R}^{d \times tMb}, i = [M]$ for a any time t and a fixed update chunk size b. Furthermore, suppose matrix $\mathbf{Y}_t^i$ at time t has block matrices defined as $\mathbf{Y}_t^i = \left[\mathbf{Y}_t^1 | \mathbf{Y}_t^2 | \cdots | \mathbf{Y}_t^M\right]$, and $\mathbf{Z_t}$ at the same time has blocks defined as $\mathbf{Z}_t = \left[(\mathbf{Y}_t^1)_r | (\mathbf{Y}_t^2)_r | \cdots | (\mathbf{Y}_t^M)_r\right]$, where $r \leq d$. Then, $\|(\mathbf{Z}_t)_r - \mathbf{Y}_t\|_{\mathrm{F}} \leq \|(\mathbf{Z})_r - \mathbf{Z}_t\|_{\mathrm{F}} + \|\mathbf{Z}_t - \mathbf{Y}_t\|_{\mathrm{F}} \leq 3\|(\mathbf{Y}_t)_r - \mathbf{Y}_t\|_{\mathrm{F}}$ holds for all $r \in [d]$.*

Proof. We base our proof on an invariant at each time t the matrix $\mathbf{Y}_t$, although not kept in memory, due to the approximation described in chapter B can be treated as such for the purposes of this proof. Thus, we have the following:

$$\begin{aligned}\|(\mathbf{Z}_t)_r - \mathbf{Y}_t\|_{\mathrm{F}} &\leq \|(\mathbf{Z}_t)_r - \mathbf{Z_t}\|_{\mathrm{F}} + \|\mathbf{Z}_t - \mathbf{Y}_t\|_{\mathrm{F}} \\ &\leq \|(\mathbf{Y}_t)_r - \mathbf{Z}_t\|_{\mathrm{F}} + \|\mathbf{Z}_t - \mathbf{Y}_t\|_{\mathrm{F}} \\ &\leq \|(\mathbf{Y}_t)_r - \mathbf{Y}_t\|_{\mathrm{F}} + 2\|\mathbf{Z}_t - \mathbf{Y}_t\|_{\mathrm{F}}.\end{aligned}$$

We let $(\mathbf{Y}_t^i)_r \in \mathbb{R}^{d \times tMb}, i = 1, 2, \ldots, M$ denote the i^{th} block of $(\mathbf{Y}_t)_r$, we can see that

$$\|\mathbf{Z}_t - \mathbf{Y}_t\|_{\mathrm{F}}^2 = \sum_{i=1}^{M} \|(\mathbf{Y}_t^i)_d - \mathbf{Y}_t^i\|_{\mathrm{F}}^2 \leq \sum_{i=1}^{M} \|(\mathbf{Y}_t^i)_r - \mathbf{Y}_t^i\|_{\mathrm{F}}^2 = \|(\mathbf{Y}_t)_r - \mathbf{Y}_t\|_{\mathrm{F}}^2.$$

Hence, if we combine these two estimates we complete our proof. □

To bound the error of the federated algorithm, we use Lemma B.3.2 to derive a lower and an upper bound of the error. Suppose that we choose a $r \leq d$ which is a truncated version of $\mathbf{Y}_t$ while also having the depth equal to 1. We can improve over Lemma B.3.2 in this particular setting by requiring no access on the right singular vectors of any given block - *e.g.* the $\mathbf{V_t^i}^T$. Furthermore, it is possible to also show that this method is stable with respect to (small) additive errors. We represent this mathematically with a noise matrix Ψ.

Theorem B.3.3. *Let $\mathbf{Y}_t \in \mathbb{R}^{d \times tMb}$ at time t has its blocks defined as $\mathbf{Y}_t^i \in \mathbb{R}^{d \times tMb}, i = [M]$, so that $\mathbf{Y}_t = \left[\mathbf{Y}_t^1 | \mathbf{Y}_t^2 | \cdots | \mathbf{Y}_t^M\right]$. Now, also let $\mathbf{Z_t} = \left[\overline{(\mathbf{Y}_t^1)_r} \mid \overline{(\mathbf{Y}_t^2)_r} \mid \cdots \mid \overline{(\mathbf{Y}_t^M)_r}\right]$, $\Psi_t \in \mathbb{R}^{d \times tMb}$, and $\mathbf{Z_t}' = \mathbf{Z_t} + \Psi_t$. Then, there exists a unitary matrix $\mathbf{B}_t$ such that*

$$\left\|\overline{(\mathbf{Z_t}')_r} - \mathbf{Y}_t \mathbf{B_t}_t\right\|_{\mathrm{F}} \leq 3\sqrt{2}\|(\mathbf{Y}_t)_r - \mathbf{Y}_t\|_{\mathrm{F}} + \left(1 + \sqrt{2}\right)\|\Psi_t\|_{\mathrm{F}}$$

holds for all $r \in [d]$.

Proof. Let $\mathbf{Y}_t' = \left[\overline{\mathbf{Y}_t^1} \mid \overline{\mathbf{Y}_t^2} \mid \cdots \mid \overline{\mathbf{Y}_t^M}\right]$. Note that $\overline{\mathbf{Y}_t'} = \overline{\mathbf{Y}_t}$ by Lemma 4.2.1. Thus, there exists a unitary matrix $\mathbf{B_t}''$ such that $\mathbf{Y}_t' = \overline{\mathbf{Y}_t}\mathbf{B_t}''$. Using this fact in combination with the unitary invariance of the Frobenius norm, one can now see that

$$\|(\mathbf{Z_t}')_r - \mathbf{Y}_t'\|_{\mathrm{F}} = \left\|(\mathbf{Z_t}')_r - \overline{\mathbf{Y}_t}\mathbf{B_t}''\right\|_{\mathrm{F}} = \left\|\overline{(\mathbf{Z_t}')_r} - \overline{\mathbf{Y}}_t\mathbf{B_t}'\right\|_{\mathrm{F}} = \left\|\overline{(\mathbf{Z_t}')_r} - \mathbf{Y}_t\mathbf{B_t}\right\|_{\mathrm{F}}$$

for some (random) unitary matrices $\mathbf{B_t}'$ and $\mathbf{B_t}$. Hence, it suffices to bound the norm of $\|(\mathbf{Z_t}')_r - \mathbf{Y}_t'\|_{\mathrm{F}}$. Having said that, we can now do,

$$\begin{aligned}
\|(\mathbf{Z_t}')_r - \mathbf{Y}_t'\|_{\mathrm{F}} &\leq \|(\mathbf{Z_t}')_r - \mathbf{Z_t}'\|_{\mathrm{F}} + \|\mathbf{Z_t}' - \mathbf{Z_t}\|_{\mathrm{F}} + \|\mathbf{Z_t} - \mathbf{Y}_t'\|_{\mathrm{F}} \\
&= \sqrt{\sum_{j=r+1}^{d} \sigma_j^2(\mathbf{Z_t} + \Psi_t)} + \|\Psi_t\|_{\mathrm{F}} + \|\mathbf{Z_t} - \mathbf{Y}_t'\|_{\mathrm{F}} \\
&= \sqrt{\sum_{j=1}^{\lceil \frac{d-r}{2} \rceil} \sigma_{r+2j-1}^2(\mathbf{Z_t} + \Psi_t) + \sigma_{r+2j}^2(\mathbf{Z_t} + \Psi_t)} + \|\Psi_t\|_{\mathrm{F}} + \|\mathbf{Z_t} - \mathbf{Y}_t'\|_{\mathrm{F}} \\
&\leq \sqrt{\sum_{j=1}^{\lceil \frac{d-r}{2} \rceil} (\sigma_{r+j}(\mathbf{Z_t}) + \sigma_j(\Psi_t))^2 + (\sigma_{r+j}(\mathbf{Z_t}) + \sigma_{j+1}(\Psi_t))^2} + \|\Psi_t\|_{\mathrm{F}} + \|\mathbf{Z_t} - \mathbf{Y}_t'\|_{\mathrm{F}}
\end{aligned}$$

the result follows from applying Weyl's inequality in the first term [91]. By the application of the triangle inequality on the first term we now have the following,

$$\begin{aligned}
\|(\mathbf{Z_t}')_r - \mathbf{Y}_t'\|_{\mathrm{F}} &\leq \sqrt{\sum_{j=r+1}^{d} 2\sigma_j^2(\mathbf{Z_t})} + \sqrt{\sum_{j=1}^{d} 2\sigma_j^2(\Psi_t)} + \|\Psi_t\|_{\mathrm{F}} + \|\mathbf{Z_t} - \mathbf{Y}_t'\|_{\mathrm{F}} \\
&\leq \sqrt{2}\left(\|(\mathbf{Z_t})_r - \mathbf{Z_t}\|_{\mathrm{F}} + \|\mathbf{Z_t} - \mathbf{Y}_t'\|_{\mathrm{F}}\right) + \left(1 + \sqrt{2}\right)\|\Psi_t\|_{\mathrm{F}}.
\end{aligned}$$

Finally, Lemma B.3.2 for bounding the first two terms concludes the proof if we note that $\|(\mathbf{Y}_t')_r - \mathbf{Y}_t'\|_{\mathrm{F}} = \|(\mathbf{Y}_t)_r - \mathbf{Y}_t\|_{\mathrm{F}}$. □

Now, we introduce the final theorem which bounds the general error of Federated-PCA with respect to the data matrix $\mathbf{Y}_t$ and up to multiplication by a unitary matrix.

Theorem B.3.4. *Let $\mathbf{Y}_t \in \mathbb{R}^{d \times tMb}$ and $q \geq 1$. Then,* Federated-PCA *is guaranteed to recover an $\mathbf{Y}_t^{q+1,1} \in \mathbb{R}^{d \times tMb}$ for any t such that $\left(\mathbf{Y}_t^{q+1,1}\right)_r = \mathbf{Y}_t\mathbf{B_t} + \Psi_t$, where $\mathbf{B_t}$ is a unitary matrix, and $\|\Psi_t\|_{\mathrm{F}} \leq \left(\left(1 + \sqrt{2}\right)^{q+1} - 1\right)\|(\mathbf{Y}_t)_r - \mathbf{Y}_t\|_{\mathrm{F}}$.*

Proof. For the purposes of this proof we will refer to the approximate subspace result for $\mathbf{Y}_t^{p+1,i}$ from the merging chunks as

$$\mathbf{Z_t}^{p+1,i} := \left[\overline{\left(\mathbf{Z_t}^{p,(i-1)tMb+1}\right)_r} \,\Big|\cdots\Big|\, \overline{\left(\mathbf{Z_t}^{p,itMb}\right)_r}\right],$$

for $p \in [q]$, and $i \in [M/(tMb)^p]$. Which, as previously proved is equivalent to $\mathbf{Y}_t$, for any t and up to a unitary transform. Moreover, $\mathbf{Y}_t$ will refer to the original - and, potentially full rank - matrix with block components defined as $\mathbf{Y}_t = \left[\mathbf{Y}_t^1|\mathbf{Y}_t^2|\cdots|\mathbf{Y}_t^M\right]$, where $M = (tMb)^q$. Additionally, $\mathbf{Y}_t^{p,i}$ will refer to the respective uncorrupted block part of the original matrix $\mathbf{Y}_t$ whose values correspond to the ones of $\mathbf{Z_t}^{p,i}$. [2]

Hence, it follows that $\mathbf{Y}_t = \left[\mathbf{Y}_t^{p,1}|\mathbf{Y}_t^{p,2}|\cdots|\mathbf{Y}_t^{p,M/(tMb)^{(p-1)}}\right]$ holds for all $p \in [q+1]$, in which

$$\mathbf{Y}_t^{p+1,i} := \left[\mathbf{Y}_t^{p,(i-1)tMb+1} \,\Big|\cdots\Big|\, \mathbf{Y}_t^{p,itMb}\right]$$

for all $p \in [q]$, and $i \in [M/(tMb)^p]$. For $p = 1$ we have $\mathbf{Z_t}^{1,i} = \mathbf{Y}_t^i = \mathbf{Y}_t^{1,i}$ for $i \in [M]$ by definition. Our target is to bound $\overline{\left(\mathbf{Z_t}^{q+1,1}\right)_d}$ matrix with respect to the original matrix $\mathbf{Y}_t$, which can be done by induction on the level p. Concretely, we have to formally prove the following for all $p \in [q+1]$, and $i \in [M/(tMb)^{(p-1)}]$,

1. $\overline{\left(\mathbf{Z_t}^{p,i}\right)_r} = \mathbf{Y}_t^{p,i} W^{p,i} + \Psi_t^{p,i}$, where
2. $\mathbf{B_t}^{p,i}$ is always a unitary matrix, and
3. $\|\Psi_t^{p,i}\|_\mathrm{F} \leq \left(\left(1+\sqrt{2}\right)^p - 1\right)\left\|(\mathbf{Y}_t^{p,i})_d - \mathbf{Y}_t^{p,i}\right\|_\mathrm{F}$.

Notably, requirements $1-3$ are always satisfied when $p = 1$ since $\mathbf{Z_t}^{1,i} = \mathbf{Y}_t^i = \mathbf{Y}_t^{1,i}$ for all $i \in [M]$ by definition. Hence, we can claim that a unitary matrix $\mathbf{B_t}^{1,i}$ for all $i \in [M]$ satisfying

$$\overline{\left(\mathbf{Z_t}^{1,i}\right)_d} = \overline{\left(\mathbf{Y}_t^{1,i}\right)_r} = \left(\mathbf{Y}_t^{1,i}\right)_r \mathbf{Z_t}^{1,i} = \mathbf{Y}_t^{1,i}\mathbf{B_t}^{1,i} + \left(\left(\mathbf{Y}_t^{1,i}\right)_r - \mathbf{Y}_t^{1,i}\right)\mathbf{B_t}^{1,i},$$

where $\Psi^{1,i} := \left(\left(\mathbf{Y}_t^{1,i}\right)_r - \mathbf{Y}_t^{1,i}\right) W^{1,i}$ has

$$\|\Psi_t^{1,i}\|_\mathrm{F} = \left\|\left(\mathbf{Y}_t^{1,i}\right)_r - \mathbf{Y}_t^{1,i}\right\|_\mathrm{F} \leq \sqrt{2}\left\|\left(\mathbf{Y}_t^{1,i}\right)_r - \mathbf{Y}_t^{1,i}\right\|_\mathrm{F}. \tag{B.47}$$

Moreover, let's assume that conditions $1-3$ hold for some $p \in [q]$. In which case, we can see see from condition 1 that

[2]Meaning, $\mathbf{Z_t}^{p,i}$ is used to estimate the approximate singular values and left singular vectors of $\mathbf{Y}_t^{p,i}$ for all $p \in [q+1]$, and $i \in [M/(tMb)^{p-1}]$

$$\begin{aligned}
\mathbf{Z_t}^{p+1,i} &:= \left[\overline{\left(\mathbf{Z_t}^{p,(i-1)tMb+1}\right)_r} \,\middle|\, \cdots \,\middle|\, \overline{\left(\mathbf{Z_t}^{p,itMb}\right)_r}\right] \\
&= \left[\mathbf{Y}_t^{p,(i-1)tMb+1}\mathbf{B_t}^{p,(i-1)tMb+1} + \Psi_t^{p,(i-1)tMb+1} \,\middle|\, \cdots \,\middle|\, \mathbf{Y}_t^{p,itMb}\mathbf{B_t}^{p,itMb} + \Psi_t^{p,itMb}\right] \\
&= \left[\mathbf{Y}_t^{p,(i-1)tMb+1}\mathbf{B_t}^{p,(i-1)tMb+1} \,\middle|\, \cdots \,\middle|\, \mathbf{Y}_t^{p,itMb}\mathbf{B_t}^{p,itMb}\right] + \left[\Psi_t^{p,(i-1)tMb+1} \,\middle|\, \cdots \,\middle|\, \Psi_t^{p,itMb}\right] \\
&= \left[\mathbf{Y}_t^{p,(i-1)tMb+1} \,\middle|\, \cdots \,\middle|\, \mathbf{Y}_t^{p,itMb}\right] \tilde{\mathbf{B_t}} + \tilde{\Psi}_t,
\end{aligned}$$

where $\tilde{\Psi}_t := \left[\Psi_t^{p,(i-1)tMb+1} \,\middle|\, \cdots \,\middle|\, \Psi_t^{p,itMb)}\right]$, and

$$\tilde{\mathbf{B_t}} := \left(\begin{array}{c:c:c:c}
\mathbf{B_t}^{p,(i-1)tMb+1} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \hdashline
\mathbf{0} & \mathbf{B_t}^{p,(i-1)tMb+2} & \mathbf{0} & \mathbf{0} \\ \hdashline
\mathbf{0} & \mathbf{0} & \ddots & \mathbf{0} \\ \hdashline
\mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{B_t}^{p,i(tMb)}
\end{array}\right).$$

Of note is that $\tilde{\mathbf{B_t}}$ is always unitary due to its diagonal blocks all being unitary by condition 2 (and hence, by construction). Hence, we can claim that $\mathbf{Z_t}^{p+1,i} = \mathbf{Y}_t^{p+1,i}\tilde{\mathbf{B_t}} + \tilde{\Psi}_t$. Following this, we can now bound the quantity $\left\|\left(\mathbf{Z_t}^{p+1,i}\right)_r - \mathbf{Y}_t^{p+1,i}\tilde{\mathbf{B_t}}\right\|_{\mathrm{F}}$ by the use of a similar argument to the one we employed during the the proof of Theorem B.3.3.

$$\begin{aligned}
\left\|\left(\mathbf{Z_t}^{p+1,i}\right)_r - \mathbf{Y}_t^{p+1,i}\tilde{\mathbf{B_t}}\right\|_{\mathrm{F}} &\leq \left\|\left(\mathbf{Z_t}^{p+1,i}\right)_r - \mathbf{Z_t}^{p+1,i}\right\|_{\mathrm{F}} + \left\|\mathbf{Z_t}^{p+1,i} - \mathbf{Y}_t^{p+1,i}\tilde{\mathbf{B_t}}\right\|_{\mathrm{F}} \\
&= \sqrt{\sum_{j=r+1}^{d} \sigma_j^2\left(\mathbf{Y}_t^{p+1,i}\tilde{\mathbf{B_t}} + \tilde{\Psi}_t\right)} + \|\tilde{\Psi}_t\|_{\mathrm{F}} \\
&\leq \sqrt{\sum_{j=r+1}^{d} 2\sigma_j^2\left(\mathbf{Y}_t^{p+1,i}\tilde{\mathbf{B_t}}\right)} + \sqrt{\sum_{j=1}^{d} 2\sigma_j^2(\tilde{\Psi}_t)} + \|\tilde{\Psi}_t\|_{\mathrm{F}} \\
&= \sqrt{2}\left\|\mathbf{Y}_t^{p+1,i} - \left(\mathbf{Y}_t^{p+1,i}\right)_r\right\|_{\mathrm{F}} + \left(1+\sqrt{2}\right)\|\tilde{\Psi}_t\|_{\mathrm{F}}. \qquad \text{(B.48)}
\end{aligned}$$

Appealing to condition 3 in order to bound $\|\tilde{\Psi}_t\|_{\mathrm{F}}$ we obtain,

$$\begin{aligned}\|\tilde{\Psi}_t\|_{\mathrm{F}}^2 &= \sum_{j=1}^{tMb} \|\Psi_t^{p,(i-1)tMb+j}\|_{\mathrm{F}}^2 \leq \left(\left(1+\sqrt{2}\right)^p - 1\right)^2 \sum_{j=1}^{tMb} \left\|(\mathbf{Y}_t^{p,(i-1)tMb+j})_r - \mathbf{Y}_t^{p,(i-1)tMb+j}\right\|_{\mathrm{F}}^2 \\ &\leq \left(\left(1+\sqrt{2}\right)^p - 1\right)^2 \sum_{j=1}^{tMb} \left\|(\mathbf{Y}_t^{p+1,i})_d^j - \mathbf{Y}_t^{p,(i-1)n+j}\right\|_{\mathrm{F}}^2,\end{aligned}$$

where $(\mathbf{Y}_t^{p+1,i})_r^j$ denotes the block of $(\mathbf{Y}_t^{p+1,i})_d$ corresponding to $\mathbf{Y}_t^{p,(i-1)n+j}$ for $j \in [tMb]$. Hence,

$$\begin{aligned}\|\tilde{\Psi}_t\|_{\mathrm{F}}^2 &\leq \left(\left(1+\sqrt{2}\right)^p - 1\right)^2 \sum_{j=1}^{tMb} \left\|(\mathbf{Y}_t^{p+1,i})_d^j - \mathbf{Y}_t^{p,(i-1)tMb+j}\right\|_{\mathrm{F}}^2 \\ &= \left(\left(1+\sqrt{2}\right)^p - 1\right)^2 \left\|(\mathbf{Y}_t^{p+1,i})_r - \mathbf{Y}_t^{p+1,i}\right\|_{\mathrm{F}}^2. \qquad \text{(B.49)}\end{aligned}$$

By using both (B.48) and (B.49) we can claim that,

$$\begin{aligned}\left\|\left(\mathbf{Z_t}^{p+1,i}\right)_r - \mathbf{Y}_t^{p+1,i}\tilde{\mathbf{B}}_\mathbf{t}\right\|_{\mathrm{F}} &\leq \left[\sqrt{2} + (1+\sqrt{2})\left(\left(1+\sqrt{2}\right)^p - 1\right)\right] \left\|\left(\mathbf{Y}_t^{p+1,i}\right)_r - \mathbf{Y}_t^{p+1,i}\right\|_{\mathrm{F}} \\ &= \left(\left(1+\sqrt{2}\right)^{p+1} - 1\right) \left\|\left(\mathbf{Y}_t^{p+1,i}\right)_r - \mathbf{Y}_t^{p+1,i}\right\|_{\mathrm{F}}. \qquad \text{(B.50)}\end{aligned}$$

In the above, of note is that $\left\|\left(\mathbf{Z_t}^{p+1,i}\right)_r - \mathbf{Y}_t^{p+1,i}\tilde{\mathbf{B}}_\mathbf{t}\right\|_{\mathrm{F}} = \left\|\overline{\left(\mathbf{Z_t}^{p+1,i}\right)_r} - \mathbf{Y}_t^{p+1,i}\mathbf{B_t}^{p+1,i}\right\|_{\mathrm{F}}$ where $\mathbf{B_t}^{p+1,i}$ is always unitary. Hence, we can see that conditions 1 - 3 hold at any t and any $p+1$ with $\Psi_t^{p+1,i} := \overline{\left(\mathbf{Z_t}^{p+1,i}\right)_r} - \mathbf{Y}_t^{p+1,i}\mathbf{B_t}^{p+1,i}$. □

Theorem B.3.4 proves that at any given time t, Federated-PCA will accurately compute low rank approximations $\overline{\mathbf{Y}}_t$ of the data seen so up to time t so long as the depth of the tree is relatively small. This is a valid assumption in our setting since we expect federated deployments to be shallow and have a large fanout. That is, we expect that the depth of the tree will be low and that many nodes will be using the same aggregator for their merging procedures. It is also worth mentioning that the proof of Theorem B.3.4 can tolerate small additive noise (e.g. round-off and approximation errors) in the input matrix $\mathbf{Y}_t$ at time t. Finally, we fully expect that, at any t, the resulting error will be no higher than $\min \operatorname{rank}(\mathbf{Y}_t^i)\ \forall i \in [M]$ and no lower than $\max \operatorname{rank}(\mathbf{Y}_t^i)\ \forall i \in [M]$

B.4 Further Evaluation Details

In addition to the traditional MNIST results presented in the main paper, we further evaluate FPCA against other competing methods which show that it performs favourably both in terms of accuracy and time when using synthetic and real datasets.

B.4.1 Synthetic Datasets

For the tests on synthetic datasets, the vectors $\{\mathbf{y}_t\}_{t=1}^{\tau}$ are drawn independently from a zero-mean Gaussian distribution with the covariance matrix $\mathbf{\Xi} = \mathbf{S}\mathbf{\Lambda}\mathbf{S}^T$, where $\mathbf{S} \in \mathcal{O}(d)$ is a generic basis obtained by orthogonalising a standard random Gaussian matrix. The entries of the diagonal matrix $\mathbf{\Lambda} \in \mathbb{R}^{d\times d}$ (the eigenvalues of the covariance matrix $\mathbf{\Xi}$) are selected according to the power law, namely, $\lambda_i = i^{-\alpha}$, for a positive α. To be more succinct, wherever possible we employ MATLAB's notation for specifying the value ranges in this section.

To assess the performance of Federated-PCA, we let $\mathbf{Y}_t = [\mathbf{y}_1, \cdots, \mathbf{y}_t] \in \mathbb{R}^{d\times t}$ be the data received by time t and $\widehat{\mathbf{Y}}_{t,r}^{\text{FPCA}}$ be the output of FPCA at time t. [3] Then, the error incurred by FPCA is

$$\frac{1}{t}\|\mathbf{Y}_t - \widehat{\mathbf{Y}}_{t,r}^{\text{FPCA}}\|_F^2, \tag{B.51}$$

Recall, that the above error is always larger than the residual of $\mathbf{Y}_t$, namely,

$$\|\mathbf{Y}_t - \widehat{\mathbf{Y}}_{t,r}^{\text{FPCA}}\|_F^2 \geq \|\mathbf{Y}_t - \mathbf{Y}_{t,r}\|_F^2 = \rho_r^2(\mathbf{Y}_t). \tag{B.52}$$

In the expression above, $\mathbf{Y}_{t,r} = \text{SVD}_r(\mathbf{Y}_t)$ is a rank-r truncated SVD of $\mathbf{Y}_t$ and $\rho_r^2(\mathbf{Y}_t)$ is the corresponding residual. Additionally, we compare our proposed scheme against GROUSE [13], FD [47], PM [130] and a version of PAST [142, 194]. Interestingly and contrary to FPCA, the aforementioned algorithms are *only* able to estimate the principal components of the data and *not* their projected data on-the-fly. Although, it has to noted that in this setup we are only interested in the resulting subspace $\mathcal{U}$ along with its singular values Σ but is worth mentioning that the projected data, if desired, can be kept as well. More specifically, let $\widehat{\mathcal{S}}_{t,r}^g \in \text{G}(d,r)$ be the span of the output of GROUSE, with the outputs of the other algorithms defined similarly. Then, these algorithms incur errors

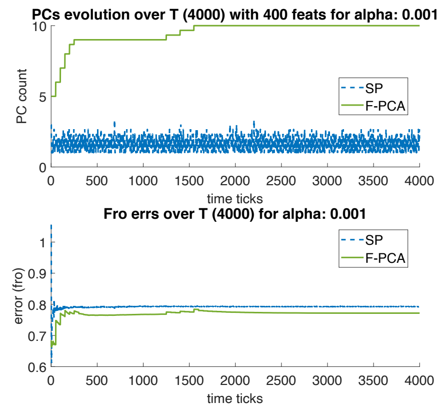
$$\frac{1}{t}\|\mathbf{Y}_t - \mathbf{P}_{\widehat{\mathcal{S}}_{t,r}^v}\mathbf{Y}_t\|_F^2, \; v \in g, f, p, \text{FPCA},$$

where we have used the notation $\mathbf{P}_{\mathcal{A}} \in \mathbb{R}^{d\times d}$ to denote the orthogonal projection onto the subspace $\mathcal{A}$. Even though robust FD [120] improves over FD in the quality of matrix sketching, since the subspaces produced by FD and robust FD coincide, there is no need here for computing a separate error for robust FD.

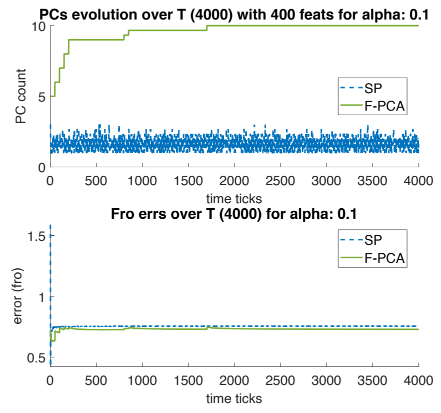
[3] Recall, since *block*-based algorithms like Federated-PCA, do not update their estimate after receiving feature vector but per each block for convenience in with respect to the evaluation against other algorithms (which might have different block sizes or singular updates), we properly *interpolate* their outputs over time.

Throughout our synthetic dataset experiments we have used an ambient dimension $d = 400$, and for each $a \in (0.001, 0.1, 0.5, 1, 2, 3)$ generated $N = 4000$ feature vectors in $\mathbb{R}^d$ using the method above. This results in a set of with four datasets of size $\mathbb{R}^{d \times N}$. Furthermore, in our experiments we used a block size of $b = 50$ for FPCA, while for PM we chose $b = d$. FD & GROUSE perform singular updates and do not need a block-size value. Additionally, the step size for GROUSE was set to 2 and the total sketch size for FD was set $2r$. In all cases, unless otherwise noted in the respective graphs the starting rank for all methods in the synthetic dataset experiments was set to $r = 10$.

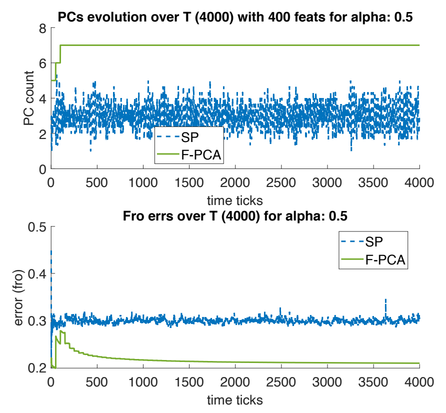
We evaluated our algorithm using the aforementioned error metrics on a set of datasets generated as described above. The results for the different a values are shown in Figure B.3, which shows FPCA can achieve an error that is significantly smaller than SP while maintaining a small number of principal components throughout the evolution of the algorithms in the absence of a forgetting factor λ. When a forgetting factor is used, as is shown in B.2 then the performance of the two methods is similar. This figure was produced on pathological datasets generated with an adversarial spectrum. It can be seen that in SPIRIT the need for PC's increases dramatically for no apparent reason, whereas Federated-PCA behaves favourably.



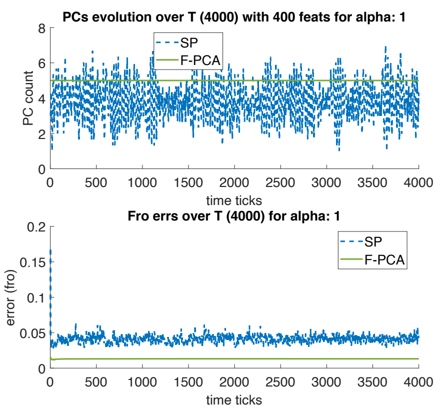
(a) $\alpha = 0.001$.



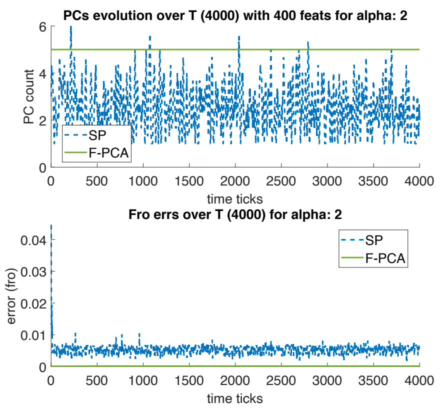
(b) $\alpha = 0.1$.



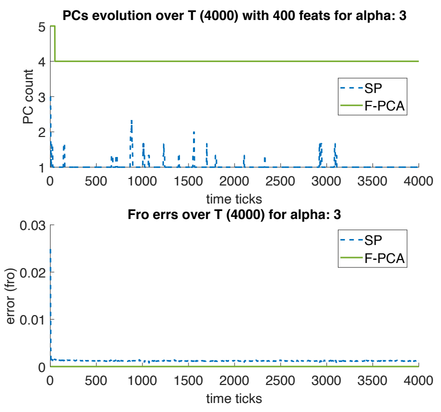
(c) $\alpha = 0.5$.



(d) $\alpha = 1$.



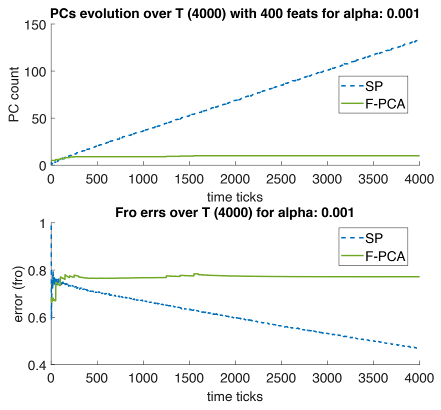
(e) $\alpha = 2$.



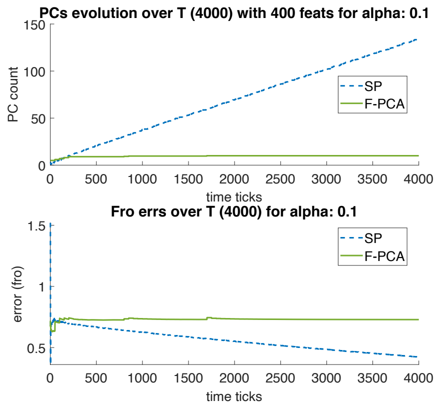
(f) $\alpha = 3$.

Fig. B.2 Performance measurements across the spectrum (when using forgetting factor $\lambda = 0.9$).

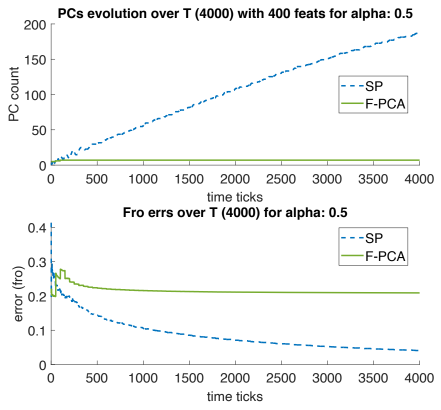
Additionally, in order to bound our algorithm in terms of the expected error, we used a *fixed rank* version with a low and high bound which fixed its rank value r to the lowest and highest estimated r-rank during its normal execution. We fully expect the incurred error of our adaptive scheme to fall within these bounds. On the other hand, Figure B.2 shows that a drastic performance improvement occurs when using an exponential forgetting factor for SPIRIT with value $\lambda = 0.9$, but the generated subspace is of inferior quality when compared to the one produced by FPCA.



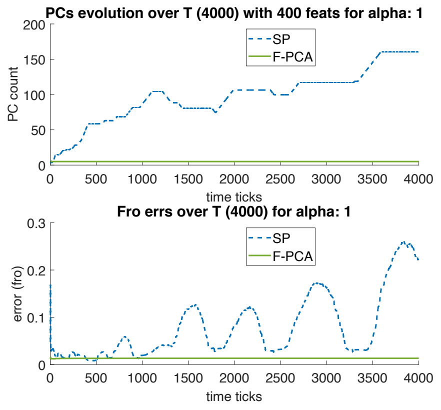
(a) $\alpha = 0.001$.



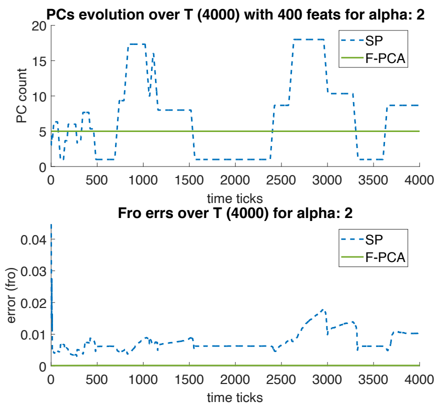
(b) $\alpha = 0.1$.



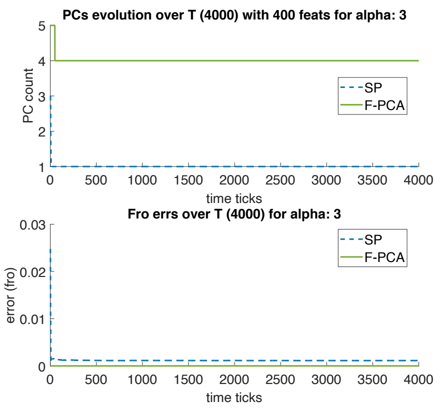
(c) $\alpha = 0.5$.



(d) $\alpha = 1$.



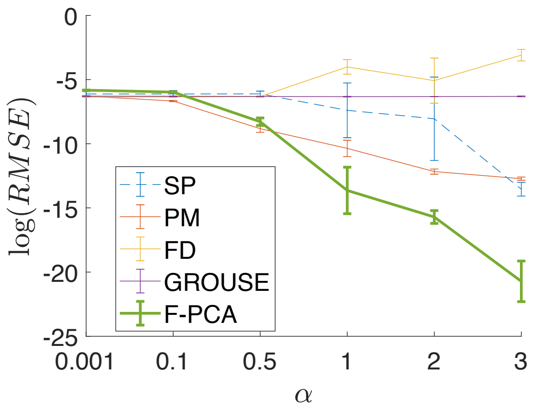
(e) $\alpha = 2$.



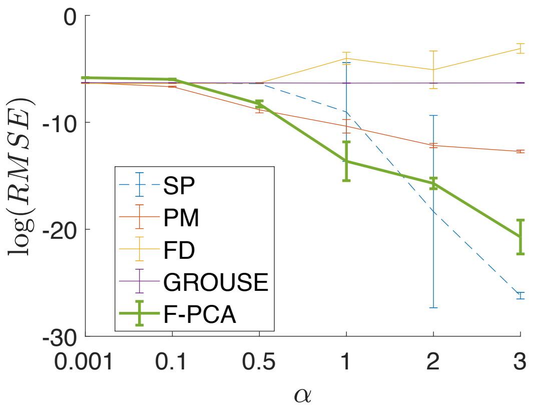
(f) $\alpha = 3$.

Fig. B.3 Pathological examples for adversarial Spectrums.

Figures B.4a and B.4b show the results of our experiments on synthetic data $\text{Synth}(\alpha)^{d\times n} \subset \mathbb{R}^{d\times n}$ with $(d, n) = (400, 4000)$ generated as described above. In the experiments, we let λ be the forgetting factor of SP. Figure B.2 compares FPCA with SP when $(\alpha, \lambda) = (1, 0.9)$ and Figure B.3 when $(\alpha, \lambda) = (2, 1)$. While Federated-PCA exhibits relative stability in both cases with respect to the incurred $||\cdot||_F$ error, SP exhibits a monotonic increase in the number of principal components estimated, in most cases, when $\lambda = 1$. This behaviour is replicated in Figures B.4a and B.4b where `RMSE` subspace error is computed across the evaluated methods; thus, we can see while SP has better performance when $\lambda = 1$ the number of principal components kept in most cases is unusually high.



(a) $\lambda = 0.9$.



(b) $\lambda = 1$.

Fig. B.4 Resulting subspace $\mathbf{U}$ comparison across different spectrums generated using different α values.

B.4.2 Real Datasets

To further evaluate our method against real datasets we also report in addition to the final subspace errors the Frobenious norm errors over time for all datasets and methods we used in the main paper. Namely, we used one that contains *light*, *volt*, and *temperature* readings gathered over a significant period of time, each of which exhibiting different noteworthy characteristics[4]. These datasets are used in addition to the MNIST and Wine quality datasets discussed in the main paper. As with the synthetic datasets, across all real dataset experiments we used an ambient dimension d and N equal to the dimensions of each dataset. For the configuration parameters we elected to use a block size of $b = 50$ for FPCA and $b = d$ for PM. The step size for GROUSE was again set to 2 and the total sketch size for FD equal to $2r$. Additionally, we used the same bounding technique as with the synthetic datasets to bound the error of FPCA using a fixed r with lowest and highest estimation of the r-rank and note that we fully expect FPCA to fall again within these bounds. Note, that most reported errors are logarithmic; this

[4]Source of data: https://www.cs.cmu.edu/afs/cs/project/spirit-1/www/data/Motes.zip

was done in order for better readability and to be able to fit in the same plot most methods - of course, this is also reflected on the y-axis label as well. We elected to do this as a number of methods, had errors orders of magnitude higher which posed a challenge when trying to plot them in the same figure.

Motes datasets

In this we elaborate on the findings with respect to the Motes dataset; below we present each of the measurements included along with discussion on the findings.

Humidity readings sensor node dataset evaluation. Firstly, we evaluate against the motes dataset which has an ambient dimension $d = 48$ and is comprised out of $N = 7712$ total feature vectors thus its total size being $\mathbb{R}^{48\times7712}$. This dataset is highly periodic in nature and has a larger lower/higher value deltas when compared to the other datasets. The initial rank used for all algorithms was $r = 10$. The errors are plotted in logarithmic scale and can be seen in Figure B.5a and we can clearly see that FPCA outperforms the competing algorithms while being within the expected $\text{FPCA}_{(\text{low})}$ & $\text{FPCA}_{(\text{high})}$ bounds.

Light readings sensor node dataset evaluation. Secondly, we evaluate against a motes dataset that has an ambient dimension $d = 48$ and is comprised out of $N = 7712$ feature vectors thus making its total size $\mathbb{R}^{48\times7712}$. It contains mote light readings can be characterised as a much more volatile dataset when compared to the Humidity one as it contains much more frequent and rapid value changes while also having the highest value delta of all mote datasets evaluated. Again, as with Humidity dataset we used an initial seed rank $r = 10$ while keeping the rest of the parameters as described above, the errors over time for all algorithms is shown in Figure B.5d plotted logarithmic scale. As before, FPCA outperforms the other algorithms while being again within the expected $\text{FPCA}_{(\text{low})}$ & $\text{FPCA}_{(\text{high})}$ bounds.

Temperature readings sensor node dataset evaluation. The third motes dataset we evaluate contains temperature readings from the mote sensors and has an ambient dimension $d = 56$ containing $N = 7712$ feature vectors thus making its total size $\mathbb{R}^{56\times7712}$. Like the humidity dataset the temperature readings exhibit periodicity in their value change and rarely have spikes. As previously we used a seed rank of $r = 20$ and the rest of the parameters as described in the synthetic comparison above, the errors over time for all algorithms is shown in Figure B.5b plotted in logarithmic scale. It is again evident that FPCA outperforms the other algorithms while being within the $\text{FPCA}_{(\text{low})}$ & $\text{FPCA}_{(\text{high})}$ bounds.

Voltage readings sensor node dataset evaluation. Finally, the fourth and final motes dataset we consider has an ambient dimension of $d = 46$ contains $N = 7712$ feature vectors thus making its size $\mathbb{R}^{46\times7712}$. Similar to the Light dataset this is an contains very frequent value changes, has large value delta which can be expected during operation of the nodes due to various reasons (one being duty cycling). As with the previous datasets we use a seed rank of $r = 10$ and leave the rest of the parameters as described previously. Finally, the errors over time for all algorithms is shown in Figure B.5c and are plotted in logarithmic scale. As expected, Federated-PCA here outperforms the competing algorithms while being within the required error bounds.

MNIST

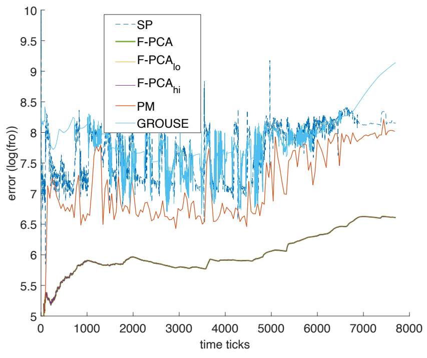
To evaluate more concretely the performance of our algorithm in a streaming setting and how the errors evolve over time rather than just reporting the result we plot the logarithm of the Frobenious norm error over time while using the MNIST dataset used in the main manuscript. From our results as can be seen from Figure B.5e Federated-PCA consistently outperforms competing methods and exhibits state of the art performance throughout.

Wine

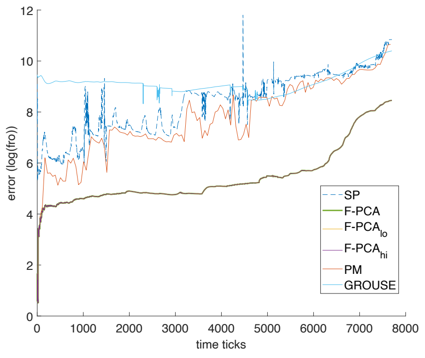
The final real dataset we consider to evaluate and plot the evolving errors is the (red) Wine quality dataset, in which we also used in the main manuscript albeit, as with MNIST, we only reported the resulting subspace quality error. Again, as we can see from Figure B.5f Federated-PCA performs again remarkably, besting all other methods in this test as well.

Real dataset evaluation remarks

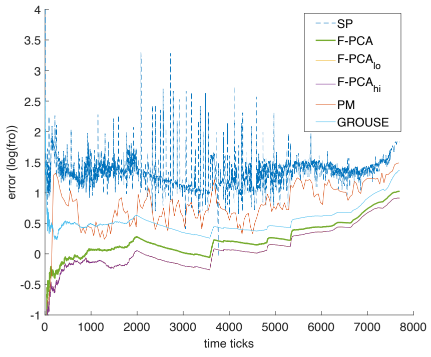
One strength of our algorithm is that it has the flexibility of not having its incremental updates to be bounded by the ambient dimension d - *i.e.* its merges. This is especially true when operating on a memory limited scenario as the minimum number of feature vectors that need to be kept has to be a multiple of the ambient dimension d in order to provide their theoretical guarantees (such as in [130]). Moreover, in the case of having an adversarial spectrum (*e.g.* $\alpha > 1$), energy thresholding can quickly overestimates the number of required principal components, unless a forgetting factor is used, but at the cost of approximation quality and robustness as it can be seen through our experiments. Notably, in a number of runs SP ended up with linearly dependent columns in the generated subspace and failed to complete. This is an inherent limitation of Gram-Schmidt orthonormalisation procedure used in the reference implementation and substituting it with a more robust one (such as QR) decreased its efficiency throughout our experiments.



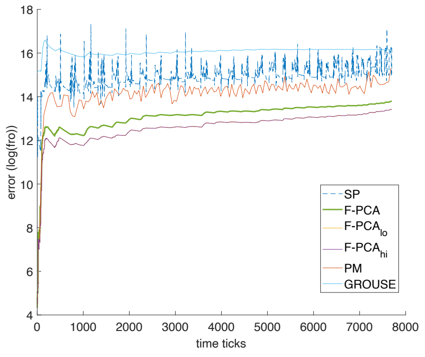
(a) Humidity.



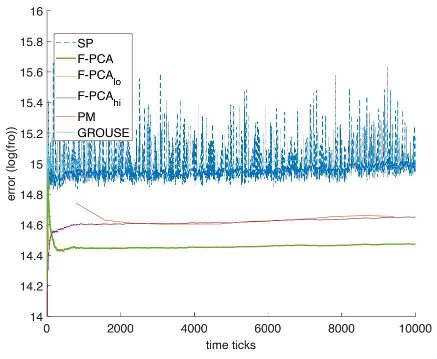
(b) Temperature.



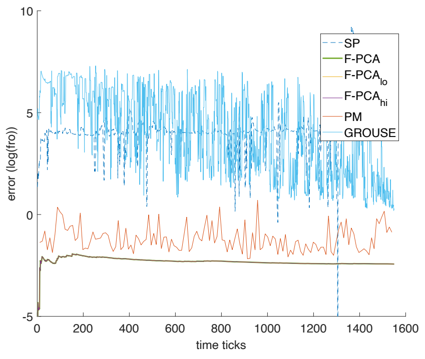
(c) Volt.



(d) Light.



(e) MNIST.

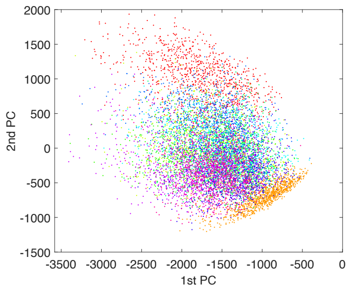


(f) (red) Wine Quality.

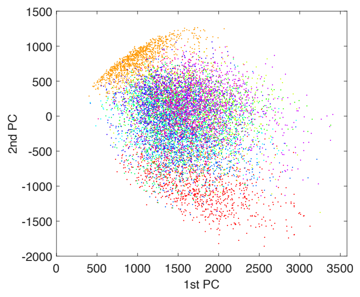
Fig. B.5 Comparisons against the Motes dataset containing Humidity (fig. B.5a), Temperature (fig. B.5b), Volt (fig. B.5c), and Light (fig. B.5d) datasets with respect to the Frobenious norm error over time; further, we compare the same error over time for the MNIST (fig. B.5e) and (red) Wine quality (fig. B.5f) datasets. We compare against SPIRIT (SP), FPCA, non-adaptive FPCA (low/high bounds), PM, & GROUSE; Frequent directions was excluded due to exploding errors.

B.4.3 Differential Privacy

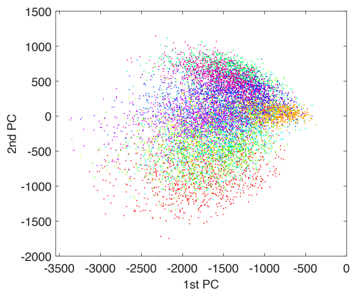
Due to spacing limitation we refrained from showing the projections using a variety of differential privacy budgets for the evaluated datasets; in this section we will show how the projections behave for two additional DP budgets, namely for: $\varepsilon \in \{0.6, 1\}$ and $\delta = 0.1$ for both datasets. The projections for MNIST can be seen in Figure B.6; the quality of the projections produced by Federated-PCA appear to be *closer* to the offline ones Figure B.6a than the ones produced by MOD-SuLQ for both DP budgets considered.



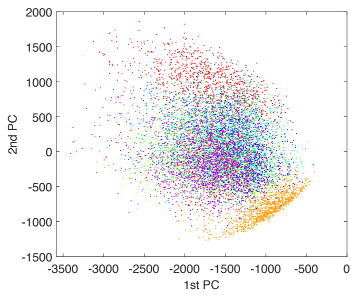
(a) Offline.



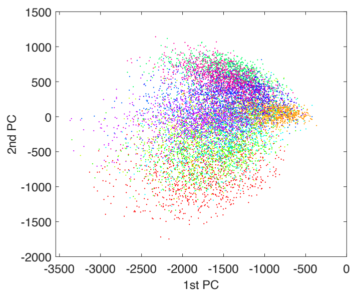
(b) FPCA (with masks), $(\varepsilon, \delta) = (0.6, 0.1)$.



(c) MOD-SuLQ, $(\varepsilon, \delta) = (0.6, 0.1)$.



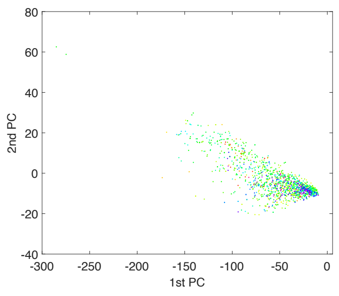
(d) FPCA (with masks), $(\varepsilon, \delta) = (1, 0.1)$.



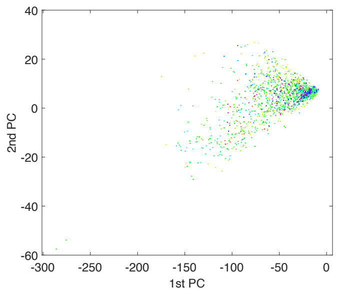
(e) MOD-SuLQ, $(\varepsilon, \delta) = (1, 0.1)$.

Fig. B.6 MNIST projections using different differential privacy budgets, at the top (fig. B.6a) is the full rank PCA while on the left column is Federated-PCA with perturbation masks and on the right column MOD-SuLQ using DP budget of $\varepsilon \in \{0.6, 1\}$ and $\delta = 0.1$ while starting from a recovery rank of 6. Note here that Federated-PCA exhibits remarkable performance producing higher quality projections than MOD-SuLQ in both cases.

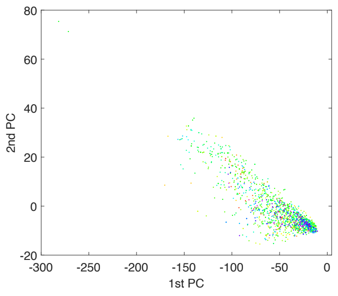
However, on the Wine quality dataset projections seen in Figure B.7 it seems that MOD-SuLQ can produce projection that are *closer* to the offline ones than Federated-PCA but not too far apart. Notably, this can be attributed to the higher sample complexity required by Federated-PCA as it is an inherently *streaming* method and the (red) Wine dataset is *considerably* smaller than MNIST.



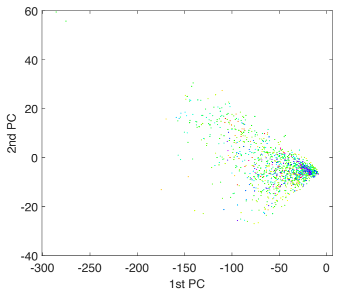
(a) Offline.



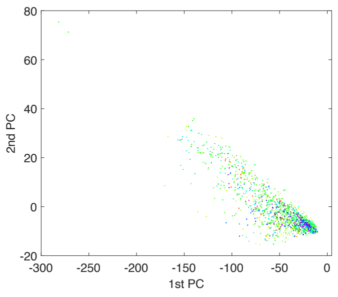
(b) FPCA (with masks), $(\varepsilon, \delta) = (0.6, 0.1)$.



(c) MOD-SuLQ, $(\varepsilon, \delta) = (0.6, 0.1)$.



(d) FPCA (with masks), $(\varepsilon, \delta) = (1, 0.1)$.



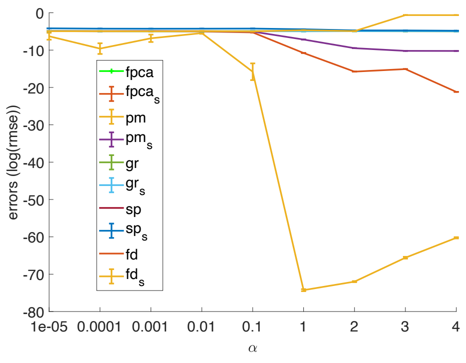
(e) MOD-SuLQ, $(\varepsilon, \delta) = (1, 0.1)$.

Fig. B.7 (red) Wine quality projections using different differential privacy budgets, at the top (fig. B.7a) is the full rank PCA while on the left column is Federated-PCA with perturbation masks and on the right column MOD-SuLQ using DP budget of $\varepsilon \in \{0.6, 1\}$ and $\delta = 0.1$ while starting from a recovery rank of 6. Note here that due to the higher sample complexity requirements of Federated-PCA the projections appear slighly worse.

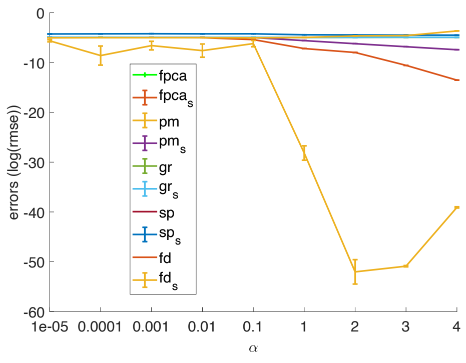
B.4.4 Extended Time-Order Independence Empirical Evaluation

The figures show the errors for recovery ranks r equal to 5 (B.8a), 20 (B.8b), 40 (B.8c), 60 (B.8d), and 80 (B.8e). It has to be noted, that legends which are subscripted with s (e.g. gr_s) compare against the SVD output while the others against its own output of the perturbation against the original $\mathbf{Y}$. As in our previous examples, we created synthetic data using $\text{Synth}(1)^{d \times n}$ function[5] We remark that when trying a full rank recovery (i.e. $r = 100$), SPIRIT failed to complete the full run as it ended up in some instances with linearly dependent columns, while the other methods perform similarly to the previous examples.

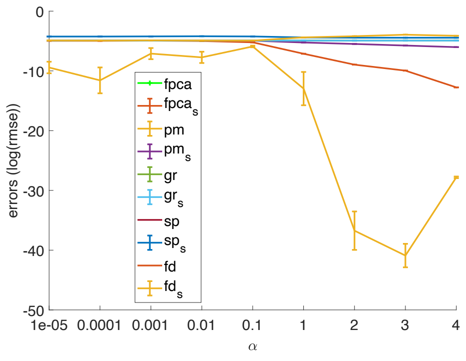
[5]If $\mathbf{Y} \sim \text{Synth}(\alpha)^{d\times n}$ iff $\mathbf{Y} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T$ with $[\mathbf{U}, \sim] = \text{QR}(\mathbf{N}^{d\times d})$, $[\mathbf{V}, \sim] = \text{QR}(\mathbf{N}^{d\times n})$, and $\boldsymbol{\Sigma}_{i,i} = i^{-\alpha}$, and $\mathbf{N}^{m\times n}$ is an $m \times n$ matrix with i.i.d. entries drawn from $\mathcal{N}(0, 1)$.



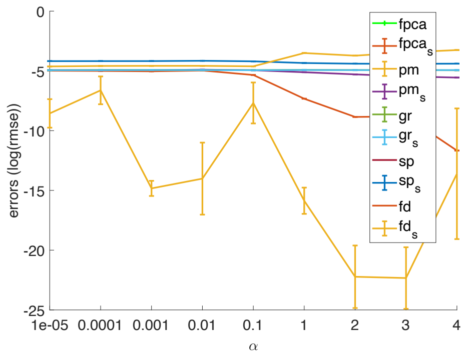
(a) Permutation errors for recovery rank $r = 5$.



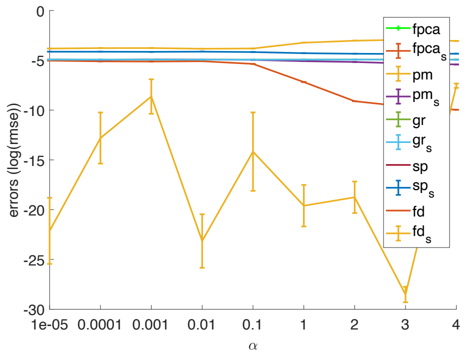
(b) Permutation errors for recovery rank $r = 20$.



(c) Permutation errors for recovery rank $r = 40$.



(d) Permutation errors for recovery rank $r = 60$.



(e) Permutation errors for recovery rank $r = 80$.

Fig. B.8 Mean Subspace errors over 20 permutations of $Y \in \mathbb{R}^{100\times10000}$ for recovery rank r equals 5 (a), 20 (b), 40 (c), 60 (d), and 80 (e).

References

[1] Abraham, G. and Inouye, M. (2014). Fast principal component analysis of large-scale genome-wide data. *PloS one*, 9(4):e93766. Publisher: Public Library of Science San Francisco, USA. (see page 3.)

[2] Abraham, G., Qiu, Y., and Inouye, M. (2017). FlashPCA2: principal component analysis of Biobank-scale genotype datasets. *Bioinformatics*. (see page 3.)

[3] Adamic, L. A. and Huberman, B. A. (2002). Zipf's law and the Internet. *Glottometrics*, 3(1):143–150. (see page 2.)

[4] Albrecht, J. P. (2016). How the GDPR will change the world. *Eur. Data Prot. L. Rev.*, 2:287. Publisher: HeinOnline. (see page 2.)

[5] Apple (2018). *Apple Differential Privacy Technical Overview*. Apple. (see page 89.)

[6] Ardekani, B. A., Kershaw, J., Kashikura, K., and Kanno, I. (1999). Activation detection in functional MRI using subspace modeling and maximum likelihood estimation. *IEEE Transactions on Medical Imaging*, 18(2):101–114. Publisher: IEEE. (see pages 3 and 44.)

[7] Arora, R., Cotter, A., Livescu, K., and Srebro, N. (2012). Stochastic optimization for PCA and PLS. In *Communication, Control, and Computing (Allerton), 2012 50th Annual Allerton Conference on*, pages 861–868. IEEE. (see pages 71 and 75.)

[8] Arora, R., Mianjy, P., and Marinov, T. (2016). Stochastic optimization for multiview representation learning using partial least squares. In *International Conference on Machine Learning*, pages 1786–1794. PMLR. (see pages 75 and 97.)

[9] Baglama, J. and Reichel, L. (2005). Augmented implicitly restarted Lanczos bidiagonalization methods. *SIAM Journal on Scientific Computing*, 27(1):19–42. Publisher: SIAM. (see page 53.)

[10] Balsubramani, A., Dasgupta, S., and Freund, Y. (2013). The fast convergence of incremental pca. In *Advances in Neural Information Processing Systems*, pages 3174–3182. (see pages 45 and 70.)

[11] Balzano, L., Chi, Y., and Lu, Y. M. (2018). Streaming pca and subspace tracking: The missing data case. *Proceedings of the IEEE*, 106(8):1293–1310. Publisher: IEEE. (see pages 3, 12, 44, 124, and 125.)

[12] Balzano, L., Nowak, R., and Recht, B. (2010). Online identification and tracking of subspaces from highly incomplete information. In *Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 704–711. IEEE. (see page 44.)

[13] Balzano, L. and Wright, S. J. (2013). On GROUSE and incremental SVD. In *IEEE International Workshop on Computational Advances in Multi-Sensor Adaptive Processing (CAMSAP)*, pages 1–4. IEEE. (see pages 5, 63, 64, 65, 70, 93, 97, and 162.)

[14] Barroso, Luiz Andraand Halzle, U. and Ranganathan, P. (2018). The Datacenter as a Computer: Designing Warehouse-Scale Machines, Third Edition. *Synthesis Lectures on Computer Architecture*, 13(3):i–189. (see page 101.)

[15] Bartels, R. H. and Golub, G. H. (1969). The simplex method of linear programming using LU decomposition. *Communications of the ACM*, 12(5):266–268. Publisher: ACM New York, NY, USA. (see page 21.)

[16] Benford, F. (1938). The law of anomalous numbers. *Proceedings of the American philosophical society*, pages 551–572. Publisher: JSTOR. (see page 32.)

[17] Bhatt, C., Desai, A., Kambo, R., Li, Z., and Zadok, E. (2013). vATM: VMware vSphere Adaptive Task Management. In *Proceedings of the VMware Technical Journal*, pages 29–33. (see page 118.)

[18] Blum, A., Dwork, C., McSherry, F., and Nissim, K. (2005). Practical privacy: the SuLQ framework. In *Proceedings of the twenty-fourth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 128–138. ACM. (see pages 5, 75, 76, 88, 98, and 99.)

[19] Bonawitz, K., Eichner, H., Grieskamp, W., Huba, D., Ingerman, A., Ivanov, V., Kiddon, C., Konečný, J., Mazzocchi, S., McMahan, H. B., and others (2019). Towards federated learning at scale: System design. *arXiv preprint arXiv:1902.01046*. (see page 103.)

[20] Boutin, E., Ekanayake, J., Lin, W., Shi, B., Zhou, J., Qian, Z., Wu, M., and Zhou, L. (2014). Apollo: Scalable and coordinated scheduling for cloud-scale computing. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 285–300. (see pages 101, 102, and 119.)

[21] Boutsidis, C., Garber, D., Karnin, Z., and Liberty, E. (2015). Online principal components analysis. In *Proceedings of the twenty-sixth annual ACM-SIAM symposium on Discrete algorithms*, pages 887–901. Society for Industrial and Applied Mathematics. (see pages 71, 75, and 97.)

[22] Boutsidis, C. and Woodruff, D. P. (2017). Optimal CUR matrix decompositions. *SIAM Journal on Computing*, 46(2):543–589. Publisher: SIAM. (see page 22.)

[23] Bouwmans, T. and Zahzah, E. H. (2014). Robust PCA via principal component pursuit: A review for a comparative evaluation in video surveillance. *Computer Vision and Image Understanding*, 122:22–34. Publisher: Elsevier. (see page 75.)

[24] Brakel, J.-P. v. (2014). z-score based streaming peak detection. (see page 110.)

[25] Brand, M. (2002). Incremental singular value decomposition of uncertain data with missing values. *ECCV 2002*, pages 707–720. Publisher: Springer. (see pages 5, 44, 50, and 69.)

[26] Brand, M. (2006). Fast low-rank modifications of the thin singular value decomposition. *Linear algebra and its applications*, 415(1):20–30. Publisher: Elsevier. (see pages 44, 50, and 69.)

[27] Brauckhoff, D., Salamatian, K., and May, M. (2009). Applying PCA for traffic anomaly detection: Problems and solutions. In *IEEE INFOCOM 2009*, pages 2866–2870. IEEE. (see page 3.)

[28] Brown, C., Chauhan, J., Grammenos, A., Han, J., Hasthanasombat, A., Spathis, D., Xia, T., Cicuta, P., and Mascolo, C. (2020). Exploring automatic diagnosis of covid-19 from crowdsourced respiratory sound data. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 3474–3484. (see page 8.)

[29] Bunch, J. R., Nielsen, C. P., and Sorensen, D. C. (1978). Rank-one modification of the symmetric eigenproblem. *Numerische Mathematik*, 31(1):31–48. Publisher: Springer. (see pages 44, 50, and 69.)

[30] Burns, B., Grant, B., Oppenheimer, D., Brewer, E., and Wilkes, J. (2016). Borg, Omega, and Kubernetes. *ACM Queue*, 14:70–93. (see page 119.)

[31] Charalambous, T. and Kalyvianaki, E. (2010). A Min-Max Framework for CPU Resource Provisioning in Virtualized Servers using $\mathcalH_infty$ Filters. In *Proceedings of the 49th IEEE Conference on Decision and Control (CDC)*, pages 3778–3783. (see page 118.)

[32] Chaudhuri, K., Sarwate, A., and Sinha, K. (2012). Near-optimal differentially private principal components. In *Advances in Neural Information Processing Systems*, pages 989–997. (see pages 5, 40, 75, 76, 87, 88, 90, 98, 99, 148, 151, 152, and 153.)

[33] Chaudhuri, K., Sarwate, A. D., and Sinha, K. (2013). A near-optimal algorithm for differentially-private principal components. *The Journal of Machine Learning Research*, 14(1):2905–2943. Publisher: JMLR. org. (see pages 5, 40, and 99.)

[34] Chiu, J. and Demanet, L. (2013). Sublinear randomized algorithms for skeleton decompositions. *SIAM Journal on Matrix Analysis and Applications*, 34(3):1361–1383. Publisher: SIAM. (see page 71.)

[35] Cho, Y. S., Go, M. J., Kim, Y. J., Heo, J. Y., Oh, J. H., Ban, H.-J., Yoon, D., Lee, M. H., Kim, D.-J., Park, M., and others (2009). A large-scale genome-wide association study of Asian populations uncovers genetic factors influencing eight quantitative traits. *Nature genetics*, 41(5):527–534. Publisher: Nature Publishing Group. (see page 3.)

[36] CodeNotary (2015). The good the bad and the ugly about CPU Ready. (see page 105.)

[37] Comon, P. and Golub, G. H. (1990). Tracking a few extreme singular values and vectors in signal processing. *Proceedings of the IEEE*, 78(8):1327–1343. Publisher: IEEE. (see pages 44, 50, and 69.)

[38] Cormode, G., Dickens, C., and Woodruff, D. (2018). Leveraging well-conditioned bases: Streaming and distributed summaries in minkowski p-norms. In *International Conference on Machine Learning*, pages 1048–1056. (see page 71.)

[39] Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine learning*, 20(3):273–297. Publisher: Springer. (see page 1.)

[40] Cortez, E., Bonde, A., Muzio, A., Russinovich, M., Fontoura, M., and Bianchini, R. (2017). Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 153–167. (see page 119.)

[41] Cortez, P., Cerdeira, A., Almeida, F., Matos, T., and Reis, J. (2009). Modeling wine preferences by data mining from physicochemical properties. *Decision Support Systems*, 47(4):547–553. Publisher: Elsevier. (see pages 90 and 102.)

[42] Dang, T., Han, J., Xia, T., Spathis, D., Bondareva, E., Brown, C., Chauhan, J., Grammenos, A., Hasthanasombat, A., Cicuta, P., and Mascolo, C. (2021). *COVID-19 Disease Progression Prediction via AudioSignals: A Longitudinal Study*. arXiv. (see page 9.)

[43] Davenport, M. A. and Romberg, J. (2016). An overview of low-rank matrix recovery from incomplete observations. *IEEE Journal of Selected Topics in Signal Processing*, 10(4):608–622. Publisher: IEEE. (see page 72.)

[44] Davis, D. M. (2002). Demystifying CPU Ready (%RDY) as a Performance Metric.Don't Trust Available CPU. (see page 103.)

[45] De la Torre, F. (2012). A least-squares framework for component analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(6):1041–1055. Publisher: IEEE. (see pages 36 and 71.)

[46] De Sa, C., Olukotun, K., and Ré, C. (2014). Global convergence of stochastic gradient descent for some non-convex matrix problems. *arXiv preprint arXiv:1411.1134*. (see page 71.)

[47] Desai, A., Ghashami, M., and Phillips, J. M. (2016). Improved practical matrix sketching with guarantees. *IEEE Transactions on Knowledge and Data Engineering*, 28(7):1678–1690. Publisher: IEEE. (see pages 63, 64, 93, and 162.)

[48] Deshpande, A., Guestrin, C., Madden, S. R., Hellerstein, J. M., and Hong, W. (2004). Model-driven data acquisition in sensor networks. In *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*, pages 588–599. VLDB Endowment. (see pages 67 and 94.)

[49] Drineas, P., Magdon-Ismail, M., Mahoney, M. W., and Woodruff, D. P. (2012). Fast approximation of matrix coherence and statistical leverage. *The Journal of Machine Learning Research*, 13(1):3475–3506. Publisher: JMLR. org. (see page 71.)

[50] Dwork, C., McSherry, F., Nissim, K., and Smith, A. (2006). Calibrating noise to sensitivity in private data analysis. In *Theory of cryptography conference*, pages 265–284. Springer. (see pages 2, 36, and 74.)

[51] Dwork, C., Roth, A., and others (2014a). The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science*, 9(3–4):211–407. Publisher: Now Publishers, Inc. (see pages 36, 37, 38, 74, and 99.)

[52] Dwork, C., Smith, A., Steinke, T., and Ullman, J. (2017). Exposed! a survey of attacks on private data. *Annual Review of Statistics and Its Application*, 4:61–84. Publisher: Annual Reviews. (see pages 2 and 37.)

[53] Dwork, C., Talwar, K., Thakurta, A., and Zhang, L. (2014b). Analyze gauss: optimal bounds for privacy-preserving principal component analysis. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, pages 11–20. ACM. (see pages 75 and 98.)

[54] Eckart, C.\ and Young, G. (1936). The approximation of one matrix by another of lower rank. *Psychometrika*, 1:211–218. (see pages 1, 22, 30, 31, 74, and 98.)

[55] Eftekhari, A., Balzano, L., and Wakin, M. B. (2016a). What to Expect When You Are Expecting on the Grassmannian. *arXiv preprint arXiv:1611.07216*. (see page 70.)

[56] Eftekhari, A., Hauser, R. A., and Grammenos, A. (2019a). MOSES: A streaming algorithm for linear dimensionality reduction. *IEEE transactions on pattern analysis and machine intelligence*, 42(11):2901–2911. Publisher: IEEE. (see pages 7 and 8.)

[57] Eftekhari, A., Li, P., Wakin, M. B., and Ward, R. A. (2016b). Learning the Differential Correlation Matrix of a Smooth Function From Point Samples. *arXiv preprint arXiv:1612.06339*. (see page 58.)

[58] Eftekhari, A., Ongie, G., Balzano, L., and Wakin, M. B. (2019b). Streaming Principal Component Analysis From Incomplete Data. *J. Mach. Learn. Res.*, 20:86–1. (see pages 12, 70, 124, 137, and 145.)

[59] Eftekhari, A., Wakin, M. B., and Ward, R. A. (2016c). MC\$^2\$: A two-phase algorithm for leveraged matrix Completion. *arXiv preprint arXiv:1609.01795*. (see page 72.)

[60] Ferlini, A., Montanari, A., Grammenos, A., Harle, R., and Mascolo, C. (2021). Enabling In-Ear Magnetic Sensing: Automatic and User Transparent Magnetometer Calibration. In *2021 IEEE International Conference on Pervasive Computing and Communications (PerCom)*. (see page 9.)

[61] Francis, S., Tenison, I., and Rish, I. (2021). Towards Causal Federated Learning For Enhanced Robustness and Privacy. *arXiv preprint arXiv:2104.06557*. (see page 125.)

[62] Gabaix, X. (1999). Zipf's law for cities: an explanation. *The Quarterly journal of economics*, 114(3):739–767. Publisher: MIT Press. (see page 2.)

[63] Garefalakis, P., Karanasos, K., Pietzuch, P., Suresh, A., and Rao, S. (2018). Medea: Scheduling of Long Running Applications in Shared Production Clusters. In *Proceedings of the 13th EuroSys Conference*. (see page 119.)

[64] Ge, J., Wang, Z., Wang, M., and Liu, H. (2018). Minimax-optimal privacy-preserving sparse pca in distributed systems. In *International Conference on Artificial Intelligence and Statistics*, pages 1589–1598. (see pages 75 and 99.)

[65] Geyer, R. C., Klein, T., and Nabi, M. (2017). Differentially private federated learning: A client level perspective. *arXiv preprint arXiv:1712.07557*. (see pages 40 and 74.)

[66] Ghashami, M., Liberty, E., Phillips, J. M., and Woodruff, D. P. (2016). Frequent directions: Simple and deterministic matrix sketching. *SIAM Journal on Computing*, 45(5):1762–1792. Publisher: SIAM. (see pages 65, 71, and 98.)

[67] Gilbert, A. C., Park, J. Y., and Wakin, M. B. (2012). Sketched SVD: Recovering spectral features from compressive measurements. *arXiv preprint arXiv:1211.0361*. (see page 71.)

[68] Gittens, A. and Mahoney, M. W. (2016). Revisiting the Nystrom method for improved large-scale machine learning. *The Journal of Machine Learning Research*, 17(1):3977–4041. Publisher: JMLR. org. (see page 71.)

[69] Gog, I., Schwarzkopf, M., Gleave, A., Watson, R. N. M., and Hand, S. (2016). Firmament: Fast, Centralized Cluster Scheduling at Scale. In *Operating Systems Design and Implementation (OSDI)*. (see pages 101, 102, and 119.)

[70] Golub, G. H. and Van Loan, C. F. (2013). *Matrix computations*. Johns Hopkins Studies in the Mathematical Sciences. Johns Hopkins University Press. (see pages 21, 25, and 53.)

[71] Gonen, A., Hazan, E., and Moran, S. (2019). Private learning implies online learning: An efficient reduction. *arXiv preprint arXiv:1905.11311*. (see page 89.)

[72] Gong, Z., Gu, X., and Wilkes, J. (2010). PRESS: Predictive Elastic Resource Scaling for Cloud Systems. In *International Conference on Network and Service Management (CNSM)*, pages 9–16. IEEE Computer Society. (see page 118.)

[73] Grammenos, A., Charalambous, T., and Kalyvianaki, E. (2021a). CPU Scheduling in Data Centers Using Asynchronous Finite-Time Distributed Coordination Mechanisms. *arXiv preprint arXiv:2101.06139*. (see page 9.)

[74] Grammenos, A., Kalyvianaki, E., and Pietzuch, P. (2021b). *Pronto: Federated Task Scheduling*. arXiv. _eprint: arXiv:2104.13429. (see pages 7, 8, and 105.)

[75] Grammenos, A., Mascolo, C., and Crowcroft, J. (2018a). Efficient, privacy aware federated model sharing. *First UK Mobile, Wearable and Ubiquitous Systems Research Symposium*. (see pages 7 and 8.)

[76] Grammenos, A., Mascolo, C., and Crowcroft, J. (2018b). Online pattern discovery in distributed, high-dimensional, streaming data under the YOLO principle. In *EuroSys 2018 Doctoral Workshop*. (see page 7.)

[77] Grammenos, A., Mascolo, C., and Crowcroft, J. (2018c). You are sensing, but are you biased? a user unaided sensor calibration approach for mobile sensing. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 2(1):1–26. Publisher: ACM New York, NY, USA. (see page 8.)

[78] Grammenos, A., Mascolo, C., and Crowcroft, J. (2019a). On Device Federated PCA & Subspace Tracking. In *Second UK Mobile, Wearable and Ubiquitous Systems Research Symposium*. (see page 8.)

[79] Grammenos, A., Mendoza Smith, R., Crowcroft, J., and Mascolo, C. (2020a). Federated Principal Component Analysis. In *Advances in Neural Information Processing Systems*, volume 33. (see pages 7, 8, and 104.)

[80] Grammenos, A., Mendoza-Smith, R., Mascolo, C., and Crowcroft, J. (2019b). Federated PCA with Adaptive Rank Estimation. *arXiv preprint arXiv:1907.08059*. (see pages 7 and 8.)

[81] Grammenos, A., Raman, A., Böttger, T., Gilani, Z., and Tyson, G. (2020b). Dissecting the Workload of a Major Adult Video Portal. In *International Conference on Passive and Active Network Measurement*, pages 267–279. Springer. (see page 8.)

[82] Han, J., Brown, C., Chauhan, J., Grammenos, A., Hasthanasombat, A., Spathis, D., Xia, T., Cicuta, P., and Mascolo, C. (2021a). Exploring Automatic COVID-19 Diagnosis via voice and symptoms from Crowdsourced Data. In *ICASSP 2021*. (see page 8.)

[83] Han, J., Spathis, D., Xia, T., Bondareva, E., Brown, C., Chauhan, J., Grammenos, A., Han, J., Hasthanasombat, A., Cicuta, P., and Mascolo, C. (2021b). *Sounds of COVID-19: from digital screening to disease progression*. arXiv. (see page 9.)

[84] Hardt, M. and Price, E. (2014). The noisy power method: A meta algorithm with applications. In *Advances in Neural Information Processing Systems*, pages 2861–2869. (see pages 71, 75, and 99.)

[85] Hardt, M. and Roth, A. (2012). Beating randomized response on incoherent matrices. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, pages 1255–1268. ACM. (see pages 75 and 99.)

[86] Hardt, M. and Roth, A. (2013). Beyond worst-case analysis in private singular vector computation. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 331–340. ACM. (see pages 75 and 99.)

[87] Hastie, T., Tibshirani, R., and Friedman, J. (2013). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Springer Series in Statistics. Springer New York. (see page 44.)

[88] He, L., Bian, A., and Jaggi, M. (2018). Cola: Decentralized linear learning. In *Advances in Neural Information Processing Systems*, pages 4536–4546. (see pages 40, 74, and 102.)

[89] Hindman, B., Konwinski, A., Zaharia, M., Ghodsi, A., Joseph, A. D., Katz, R., Shenker, S., and Stoica, I. (2011). Mesos: A Platform for Fine-grained Resource Sharing in the Data Center. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI).* (see page 119.)

[90] Hoang, D. H. and Nguyen, H. D. (2018). A PCA-based method for IoT network traffic anomaly detection. In *2018 20th International conference on advanced communication technology (ICACT)*, pages 381–386. IEEE. (see page 3.)

[91] Horn, R. A. and Johnson, C. R. (1994). *Topics in matrix analysis.* Cambridge University Press. (see pages 24 and 158.)

[92] Hotelling, H. (1933). Analysis of a complex of statistical variables into principal components. *Journal of educational psychology*, 24(6):417. Publisher: Warwick & York. (see page 32.)

[93] Huang, L., Nguyen, X., Garofalakis, M., Jordan, M. I., Joseph, A., and Taft, N. (2006). In-network PCA and anomaly detection. In *NIPS*, volume 2006, pages 617–624. (see page 3.)

[94] IBM (2019). Virtual machine CPU ready. (see page 105.)

[95] Isard, M., Prabhakaran, V., Currey, J., Wieder, U., Talwar, K., and Goldberg, A. (2009). Quincy: fair scheduling for distributed computing clusters. In *Proceedings of the ACM SIGOPS 22nd Symposium on Operating systems Principles*, pages 261–276. (see page 119.)

[96] Iwen, M. and Ong, B. (2016). A distributed and incremental svd algorithm for agglomerative data analysis on large networks. *SIAM Journal on Matrix Analysis and Applications*, 37(4):1699–1718. Publisher: SIAM. (see pages 78, 154, and 157.)

[97] Jain, P., Jin, C., Kakade, S. M., Netrapalli, P., and Sidford, A. (2016). Streaming PCA: Matching matrix Bernstein and near-optimal finite sample guarantees for Oja's algorithm. In *Conference on Learning Theory*, pages 1147–1164. (see page 71.)

[98] Jiang, W., Grammenos, A., Kalyvianaki, E., and Charalambous, T. (2021). *An Asynchronous Approximate Distributed Alternating Direction Method of Multipliers in Digraphs.* arXiv. _eprint: arXiv:2104.11866. (see page 9.)

[99] Johnstone, I. M. (2001). On the distribution of the largest eigenvalue in principal components analysis. *Annals of statistics*, pages 295–327. Publisher: JSTOR. (see pages 5 and 61.)

[100] Jolliffe, I. (2011). Principal component analysis. In *International encyclopedia of statistical science*, pages 1094–1096. Springer. (see pages 3, 36, 74, and 106.)

[101] Kadison, R. V. (1984). Diagonalizing matrices. *American Journal of Mathematics*, 106(6):1451–1468. Publisher: JSTOR. (see page 22.)

[102] Kairouz, P., Oh, S., and Viswanath, P. (2015). The composition theorem for differential privacy. In *International conference on machine learning*, pages 1376–1385. PMLR. (see page 99.)

[103] Kalyvianaki, E., Charalambous, T., and Hand, S. (2009). Self-adaptive and Self-configured CPU Resource Provisioning for Virtualized Servers Using Kalman Filters. In *ICAC*. (see page 118.)

[104] Kalyvianaki, E., Charalambous, T., and Hand, S. (2014). Adaptive Resource Provisioning for Virtualized Servers Using Kalman Filters. *Transactions on Autonomous and Adaptive Systems (TAAS) (to appear)*. Publisher: ACM. (see page 118.)

[105] Kannan, R., Vempala, S., and Woodruff, D. (2014). Principal component analysis and higher correlations for distributed data. In *Conference on Learning Theory*, pages 1040–1057. (see pages 75 and 97.)

[106] Karanasos, K., Rao, S., Curino, C., Douglas, C., Chaliparambil, K., Fumarola, G. M., Heddaya, S., Ramakrishnan, R., and Sakalanaga, S. (2015). Mercury: Hybrid Centralized and Distributed Scheduling in Large Shared Clusters. In *USENIX ATC*. (see page 101.)

[107] Kim, K. I., Franz, M. O., and Scholkopf, B. (2005). Iterative kernel principal component analysis for image modeling. *IEEE transactions on pattern analysis and machine intelligence*, 27(9):1351–1366. Publisher: IEEE. (see page 71.)

[108] Konečnỳ, J., McMahan, H. B., Yu, F. X., Richtárik, P., Suresh, A. T., and Bacon, D. (2016). Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*. (see pages 39, 74, and 102.)

[109] Krasulina, T. (1969). The method of stochastic approximation for the determination of the least eigenvalue of a symmetrical matrix. *USSR Computational Mathematics and Mathematical Physics*, 9(6):189–195. Publisher: Elsevier. (see pages 45 and 70.)

[110] Krim, H. and Viberg, M. (1996). Two decades of array signal processing research: The parametric approach. *IEEE Signal processing magazine*, 13(4):67–94. Publisher: IEEE. (see pages 3 and 44.)

[111] Larsen, R. M. (1998). Lanczos bidiagonalization with partial reorthogonalization. *DAIMI Report Series*, 1(537). (see page 53.)

[112] Läuchli, P. (1961). Jordan-elimination und Ausgleichung nach kleinsten Quadraten. *Numerische Mathematik*, 3(1):226–240. Publisher: Springer. (see page 33.)

[113] LeCun, Y., Cortes, C., and Burges, C. J. (2010). The MNIST database of handwritten digits, 2010. (see page 90.)

[114] Ledoux, M. and Talagrand, M. (2013). *Probability in Banach Spaces: Isoperimetry and Processes*. Classics in Mathematics. Springer Berlin Heidelberg. (see page 143.)

[115] Li, T., Sahu, A. K., Talwalkar, A., and Smith, V. (2020). Federated learning: Challenges, methods, and future directions. *IEEE Signal Processing Magazine*, 37(3):50–60. Publisher: IEEE. (see pages 74, 102, and 103.)

[116] Li, Y. (2004). On incremental and robust subspace learning. *Pattern recognition*, 37(7):1509–1518. Publisher: Elsevier. (see pages 44, 50, and 69.)

[117] Liang, Y., Balcan, M.-F. F., Kanchanapally, V., and Woodruff, D. (2014). Improved distributed principal component analysis. In *Advances in Neural Information Processing Systems*, pages 3113–3121. (see page 75.)

[118] Liberty, E. (2013). Simple and deterministic matrix sketching. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 581–588. ACM. (see page 114.)

[119] Liu, X., Zhu, X., Padala, P., Wang, Z., and Singhal, S. (2007). Optimal Multivariate Control for Differentiated Services on a Shared Hosting Platform. In *Proceedings of the 46th IEEE Conference on Decision and Control*, pages 3792 –3799. IEEE Computer Society. (see page 118.)

[120] Luo, L., Chen, C., Zhang, Z., Li, W.-J., and Zhang, T. (2017). Robust Frequent Directions with Application in Online Learning. *arXiv preprint arXiv:1705.05067*. (see pages 63, 64, 93, and 162.)

[121] Maaten, L. v. d. and Hinton, G. (2008). Visualizing data using t-SNE. *Journal of machine learning research*, 9(Nov):2579–2605. (see page 74.)

[122] Mahoney, M. W. and Drineas, P. (2009). CUR matrix decompositions for improved data analysis. *Proceedings of the National Academy of Sciences*, 106(3):697–702. Publisher: National Acad Sciences. (see page 22.)

[123] Mao, H., Schwarzkopf, M., Venkatakrishnan, S. B., Meng, Z., and Alizadeh, M. (2019). Learning Scheduling Algorithms for Data Processing Clusters. In *Proceedings of the ACM Special Interest Group on Data Communication*, SIGCOMM, pages 270–288. event-place: Beijing, China. (see pages 101, 102, and 119.)

[124] Mardani, M., Mateos, G., and Giannakis, G. B. (2015). Subspace learning and imputation for streaming big data matrices and tensors. *IEEE Transactions on Signal Processing*, 63(10):2663–2677. Publisher: IEEE. (see page 125.)

[125] Marinov, T. V., Mianjy, P., and Arora, R. (2018). Streaming Principal Component Analysis in Noisy Settings. In *International Conference on Machine Learning*, pages 3410–3419. (see page 75.)

[126] Markovsky, I. and Usevich, K. (2012). *Low rank approximation*, volume 139. Springer. (see page 31.)

[127] McInnes, L., Healy, J., and Melville, J. (2018). Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*. (see page 74.)

[128] McMahan, H. B., Moore, E., Ramage, D., Hampson, S., and others (2016). Communication-efficient learning of deep networks from decentralized data. *arXiv preprint arXiv:1602.05629*. (see pages 39, 74, and 102.)

[129] Mirsky, L. (1960). Symmetric gauge functions and unitarily invariant norms. *The quarterly journal of mathematics*, 11(1):50–59. Publisher: Oxford University Press. (see pages 1, 30, 31, 74, and 98.)

[130] Mitliagkas, I., Caramanis, C., and Jain, P. (2013). Memory limited, streaming PCA. In *Advances in Neural Information Processing Systems*, pages 2886–2894. (see pages 6, 12, 44, 63, 64, 65, 71, 75, 93, 97, 98, 114, 162, and 168.)

[131] Mitliagkas, I., Caramanis, C., and Jain, P. (2014). Streaming PCA with many missing entries. *Preprint*. (see pages 6, 63, 65, 75, and 97.)

[132] Moneta, A., Entner, D., Hoyer, P. O., and Coad, A. (2013). Causal inference by independent component analysis: Theory and applications. *Oxford Bulletin of Economics and Statistics*, 75(5):705–730. Publisher: Wiley Online Library. (see page 125.)

[133] Muthukrishnan, S. (2005). *Data streams: Algorithms and applications.* Now Publishers Inc. (see page 44.)

[134] Narayanamurthy, P., Vaswani, N., and Ramamoorthy, A. (2020). Federated Over-the-Air Subspace Learning from Incomplete Data. *arXiv preprint arXiv:2002.12873.* (see page 75.)

[135] Nelson, J. and Nguyên, H. L. (2013). OSNAP: Faster numerical linear algebra algorithms via sparser subspace embeddings. In *2013 ieee 54th annual symposium on foundations of computer science*, pages 117–126. IEEE. (see page 31.)

[136] Nguyen, H., Shen, Z., Gu, X., Subbiah, S., and Wilkes, J. (2013). AGILE: Elastic Distributed Resource Scaling for Infrastructure-as-a-Service. In *Proceedings of the 10th International Conference on Autonomic Computing (ICAC)*, pages 69–82. (see page 118.)

[137] Oja, E. (1983). *Subspace methods of pattern recognition.* Electronic & electrical engineering research studies. Research Studies Press. (see pages 45 and 70.)

[138] Oja, E. and Karhunen, J. (1985). On stochastic approximation of the eigenvectors and eigenvalues of the expectation of a random matrix. *Journal of mathematical analysis and applications*, 106(1):69–84. Publisher: Elsevier. (see page 71.)

[139] Ousterhout, K., Wendell, P., Zaharia, M., and Stoica, I. (2013). Sparrow: Distributed, Low Latency Scheduling. In *Symposium on Operating Systems Principles (SOSP).* (see pages 101, 102, and 119.)

[140] Padala, P., Hou, K.-Y., Shin, K. G., Zhu, X., Uysal, M., Wang, Z., Singhal, S., and Merchant, A. (2009). Automated Control of Multiple Virtualized Resources. In *EuroSys.* (see page 118.)

[141] Padala, P., Shin, K. G., Zhu, X., Uysal, M., Wang, Z., Singhal, S., Merchant, A., and Salem, K. (2007). Adaptive Control of Virtualized Resources in Utility Computing Environments. In *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, EuroSys '07, pages 289–302. (see page 118.)

[142] Papadimitriou, S., Sun, J., and Faloutsos, C. (2005). Streaming pattern discovery in multiple time-series. In *Proceedings of the 31st international conference on Very large data bases*, pages 697–708. VLDB Endowment. (see pages 93, 114, and 162.)

[143] Pearson, K. (1901). LIII. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572. Publisher: Taylor & Francis. (see pages 32, 74, and 106.)

[144] Pourkamali-Anaraki, F. and Becker, S. (2016). Randomized Clustered Nystrom for Large-Scale Kernel Machines. *arXiv preprint arXiv:1612.06470.* (see page 71.)

[145] Powell, W. B. (2007). *Approximate Dynamic Programming: Solving the curses of dimensionality*, volume 703. John Wiley & Sons. (see page 1.)

[146] Qu, Y., Ostrouchov, G., Samatova, N., and Geist, A. (2002). Principal component analysis for dimension reduction in massive distributed data sets. In *Proceedings of IEEE International Conference on Data Mining (ICDM).* (see page 75.)

[147] Raniwala, A. and Chiueh, T.-c. (2005). Architecture and algorithms for an IEEE 802.11-based multi-channel wireless mesh network. In *Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies.*, volume 3, pages 2223–2234. IEEE. (see page 126.)

[148] Rehurek, R. (2011). Subspace tracking for latent semantic analysis. In *European Conference on Information Retrieval*, pages 289–300. Springer. (see pages 80 and 81.)

[149] Rikos, A. I., Grammenos, A., Kalyvianaki, E., Hadjicostis, C. N., Charalambous, T., and Johansson, K. H. (2021). *Optimal CPU Scheduling in Data Centers via a Finite-Time Distributed Quantized Coordination Mechanism.* arXiv. arXiv:2104.03126. (see page 9.)

[150] Ringberg, H., Soule, A., Rexford, J., and Diot, C. (2007). Sensitivity of PCA for traffic anomaly detection. In *Proceedings of the 2007 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 109–120. (see page 3.)

[151] Roweis, S. T. (1998). EM algorithms for PCA and SPCA. In *Advances in neural information processing systems*, pages 626–632. (see page 71.)

[152] Roy, A. G., Siddiqui, S., Pölsterl, S., Navab, N., and Wachinger, C. (2019). Braintorrent: A peer-to-peer environment for decentralized federated learning. *arXiv preprint arXiv:1905.06731.* (see page 126.)

[153] Rubin, D. B. (1996). Multiple imputation after 18+ years. *Journal of the American statistical Association*, 91(434):473–489. Publisher: Taylor & Francis Group. (see page 125.)

[154] Rudelson, M., Vershynin, R., and others (2013). Hanson-Wright inequality and sub-gaussian concentration. *Electronic Communications in Probability*, 18. Publisher: The Institute of Mathematical Statistics and the Bernoulli Society. (see pages 62 and 129.)

[155] Ruder, S. (2016). An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747.* (see page 98.)

[156] Rudin, C. (2019). Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, 1(5):206–215. Publisher: Nature Publishing Group. (see page 33.)

[157] Rzadca, K., Findeisen, P., Swiderski, J., Zych, P., Broniek, P., Kusmierek, J., Nowak, P., Strack, B., Witusowski, P., Hand, S., and Wilkes, J. (2020). Autopilot: Workload Autoscaling at Google. In *Proceedings of the 15th European Conference on Computer Systems (EuroSys).* (see page 119.)

[158] Sanger, T. D. (1989). Optimal unsupervised learning in a single-layer linear feedforward neural network. *Neural networks*, 2(6):459–473. Publisher: Elsevier. (see page 71.)

[159] Scharf, L. L. (1991). The SVD and reduced rank signal processing. *Signal processing*, 25(2):113–133. Publisher: Elsevier. (see page 3.)

[160] Schwarzkopf, M., Konwinski, A., Abd-El-Malek, M., and Wilkes, J. (2013). Omega: Flexible, Scalable Schedulers for Large Compute Clusters. In *EuroSys.* (see page 119.)

[161] Shen, Z., Subbiah, S., Gu, X., and Wilkes, J. (2011). CloudScale: Elastic Resource Scaling for Multi-Tenant Cloud Systems. In *Proceedings of the 2nd ACM Symposium on Cloud Computing (SOCC)*, pages 5:1–5:14. (see page 118.)

[162] Smith, V., Chiang, C.-K., Sanjabi, M., and Talwalkar, A. S. (2017). Federated multi-task learning. In *Advances in Neural Information Processing Systems*, pages 4424–4434. (see pages 40, 74, and 102.)

[163] Stoica, I., Morris, R., Karger, D., Kaashoek, M. F., and Balakrishnan, H. (2001). Chord: A scalable peer-to-peer lookup service for internet applications. *ACM SIGCOMM Computer Communication Review*, 31(4):149–160. Publisher: ACM New York, NY, USA. (see page 126.)

[164] Strang, G. (1993). The fundamental theorem of linear algebra. *The American Mathematical Monthly*, 100(9):848–855. Publisher: Taylor & Francis. (see pages 17, 22, and 31.)

[165] Strang, G. (2016). *Introduction to linear algebra*. Wellesley-Cambridge Press Wellesley, MA, 5 edition. (see pages 22, 27, 31, and 86.)

[166] Strang, G. (2019). *Linear algebra and learning from data*. Wellesley-Cambridge Press Cambridge. (see pages 17, 27, 70, and 79.)

[167] Tanenbaum, A. S. and Van Steen, M. (2007). *Distributed systems: principles and paradigms*. Prentice-Hall. (see page 76.)

[168] Tipping, M. E. and Bishop, C. M. (1999). Probabilistic principal component analysis. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 61(3):611–622. Publisher: Wiley Online Library. (see page 71.)

[169] Tong, L. and Perreau, S. (1998). Multichannel blind identification: From subspace to maximum likelihood methods. *Proceedings of IEEE*, 86:1951–1968. Publisher: IEEE INSTITUTE OF ELECTRICAL AND ELECTRONICS. (see pages 3 and 44.)

[170] Trefethen, L. N. and Bau III, D. (1997). *Numerical linear algebra*, volume 50. Siam. (see page 27.)

[171] Tropp, J. A., Yurtsever, A., Udell, M., and Cevher, V. (2017). Fixed-Rank Approximation of a Positive-Semidefinite Matrix from Streaming Data. In *Advances in Neural Information Processing Systems*, pages 1225–1234. (see page 71.)

[172] Trunk, G. V. (1979). A problem of dimensionality: A simple example. *IEEE Transactions on pattern analysis and machine intelligence*, 3:306–307. Publisher: IEEE. (see page 1.)

[173] Tumanov, A., Zhu, T., Park, J. W., Kozuch, M. A., Harchol-Balter, M., and Ganger, G. R. (2016). TetriSched: Global Rescheduling with Adaptive Plan-Ahead in Dynamic Heterogeneous Clusters. In *Proceedings of the Eleventh European Conference on Computer Systems*, EuroSys. (see page 119.)

[174] Vadhan, S. (2017). The complexity of differential privacy. In *Tutorials on the Foundations of Cryptography*, pages 347–450. Springer. (see page 2.)

[175] van Overschee, P. and de Moor, B. L. (2012). *Subspace identification for linear systems: Theory, implementation, applications*. Springer US. (see page 44.)

[176] Varian, H. R. (2014). Big data: New tricks for econometrics. *Journal of Economic Perspectives*, 28(2):3–28. (see page 1.)

[177] Vavilapalli, V. K., Murthy, A. C., Douglas, C., Agarwal, S., Konar, M., Evans, R., Graves, T., Lowe, J., Shah, H., Seth, S., and others (2013). Apache hadoop yarn: Yet another resource negotiator. In *Proceedings of the 4th annual Symposium on Cloud Computing*, pages 1–16. (see page 119.)

[178] Verma, A., Pedrosa, L., Korupolu, M., Oppenheimer, D., Tune, E., and Wilkes, J. (2015). Large-scale cluster management at Google with Borg. In *Proceedings of the Tenth European Conference on Computer Systems*, pages 1–17. (see pages 101, 102, and 119.)

[179] Vershynin, R. (2012a). How close is the sample covariance matrix to the actual covariance matrix? *Journal of Theoretical Probability*, 25(3):655–686. Publisher: Springer. (see pages 58, 61, and 62.)

[180] Vershynin, R. (2012b). Introduction to the non-asymptotic analysis of random matrices. In Eldar, Y. C. and Kutyniok, G., editors, *Compressed Sensing: Theory and Applications*, pages 95–110. Cambridge University Press. (see pages 62, 129, and 143.)

[181] Vidal, R., Ma, Y., and Sastry, S. (2016). *Generalized Principal Component Analysis*. Interdisciplinary Applied Mathematics. Springer New York. (see pages 3 and 44.)

[182] Vladan (2017). What is vmware CPU ready. (see page 105.)

[183] VMWare (2008). *Server Consolidation and Containment, With Virtual Infrastructure*. arXiv. (see page 103.)

[184] VMWare (2020). *vSphere Datacenter Administration Guide*. VMware. (see page 103.)

[185] VMWare, L. (2018). *PERFORMANCE TROUBLESHOOTING – CPU READY TIME*. VMware. (see page 105.)

[186] Wang, H., Lee, M. K., and Wang, C. (1998). Consumer privacy concerns about Internet marketing. *Communications of the ACM*, 41(3):63–70. Publisher: ACM New York, NY, USA. (see page 2.)

[187] Wang, Z., Zhu, X., and Singhal, S. (2005). Utilization and SLO-Based Control for Dynamic Sizing of Resource Partitions. In *Proceedings of the 16th IFIP/IEEE Ambient Networks International Conference on Distributed Systems: Operations and Management*, DSOM'05, pages 133–144. (see page 118.)

[188] Warmuth, M. K. and Kuzmin, D. (2008). Randomized online PCA algorithms with regret bounds that are logarithmic in the dimension. *Journal of Machine Learning Research*, 9(Oct):2287–2320. (see page 71.)

[189] Wedin, P. (1972). Perturbation bounds in connection with singular value decomposition. *BIT Numerical Mathematics*, 12(1):99–111. Publisher: Springer. (see page 137.)

[190] Wohwe Sambo, D., Yenke, B. O., Förster, A., and Dayang, P. (2019). Optimized clustering algorithms for large wireless sensor networks: A review. *Sensors*, 19(2):322. Publisher: Multidisciplinary Digital Publishing Institute. (see pages 76 and 107.)

[191] Xia, T., Spathis, D., Ch, J., Grammenos, A., Han, J., Hasthanasombat, A., Bondareva, E., Dang, T., Floto, A., Cicuta, P., and others (2021). COVID-19 Sounds: A Large-Scale Audio Dataset for Digital Respiratory Screening. In *35th Conference on Neural Information Processing Systems (NeurIPS)*. (see page 9.)

[192] Xie, Y., Huang, J., and Willett, R. (2013). Change-point detection for high-dimensional time series with missing data. *IEEE Journal of Selected Topics in Signal Processing*, 7(1):12–27. Publisher: IEEE. (see page 70.)

[193] Xu, W., Zhu, X., Singhal, S., and Wang, Z. (2006). Predictive Control for Dynamic Resource Allocation in Enterprise Data Centers. In *Proceedings of the IEEE/IFIP Network Operations and Management Symposium (NOMS)*, pages 115–126. (see page 118.)

[194] Yang, B. (1995). Projection approximation subspace tracking. *IEEE Transactions on Signal processing*, 43(1):95–107. Publisher: IEEE. (see pages 93 and 162.)

[195] Yang, Q., Liu, Y., Chen, T., and Tong, Y. (2019). Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 10(2):1–19. Publisher: ACM New York, NY, USA. (see page 102.)

[196] Young, A. L. and Quan-Haase, A. (2009). Information revelation and internet privacy concerns on social network sites: a case study of facebook. In *Proceedings of the fourth international conference on Communities and technologies*, pages 265–274. (see page 2.)

[197] Yu, B. (1997). Assouad, fano, and le cam. In *Festschrift for Lucien Le Cam*, pages 423–435. Springer. (see page 151.)

[198] Zhou, S., Ligett, K., and Wasserman, L. (2009). Differential privacy with compression. In *2009 IEEE International Symposium on Information Theory*, pages 2718–2722. IEEE. (see pages 75 and 99.)

[199] Zhu, X., Wang, Z., and Singhal, S. (2006). Utility-Driven Workload Management using Nested Control Design. In *Proceedings of the American Control Conference (ACC)*, page 6. (see page 118.)

[200] Zipf, G. and Behavior, H. (1950). The principle of least effort. *Massachusetts: Addison*. (see pages 2 and 32.)