Imperial College London

Department of Computing

MSc in Advanced Computing

Robust Low-Rank Modelling on Matrices and Tensors

by

Georgios Papamakarios

Submitted in partial fulfilment of the requirements for the MSc Degree in Advanced Computing of Imperial College London

September 2014

Abstract

Robust low-rank modelling has recently emerged as a family of powerful methods for recovering the low-dimensional structure of grossly corrupted data, and has become successful in a wide range of applications in signal processing and computer vision. In principle, robust low-rank modelling focuses on decomposing a given data matrix into a low-rank and a sparse component, by minimising the rank of the former and maximising the sparsity of the latter. In several practical cases, however, tensors can be a more suitable representation than matrices for higher-order data. Nevertheless, existing work on robust low-rank modelling has mostly focused on matrices and has largely neglected tensors.

In this thesis we aim to bridge the gap between matrix and tensor methods for robust low-rank modelling. We conduct a thorough survey on matrix methods, in which we discuss and analyse the state-of-the-art algorithms. We take one step further and we present—for the first time in the literature—a set of novel algorithms for robust low-rank modelling on tensors. We show how both matrix and tensor methods can be extended to deal with missing values and we discuss non-convex generalisations thereof. Finally, we demonstrate the applicability of all methods, both those already existing and those proposed herein, to a range of computer vision problems of practical interest.

Acknowledgements

Dr Stefanos Zafeiriou has been excellent not only as a lecturer but also as a supervisor. Dr Yannis Panagakis has been an invaluable source of help and encouragement in all aspects of this project. Working with them has been nothing less than a great pleasure.

Christos Sagonas did the important job of preparing the Multi-PIE dataset—used in the image reconstruction experiment—to be nicely cropped and aligned. Mihaela Rosca has been kind enough to read parts of the report along its preparation and provide helpful suggestions.

Finally, the Lilian Voudouri foundation has supported my studies this year through its postgraduate scholarships programme.

Contents

Chapter 1

Introduction

Real-life data are rarely, if ever, pure. They are often *corrupted by noise* or *incomplete.* In contrast, high data quality is essential for several practical applications. To bridge the gap between reality and need, it is necessary to be able to *recover* the pure and uncorrupted information which remains hidden within the available, possibly corrupted or incomplete, data.

Recovering information from noisy data is a problem simple to describe, yet fundamentally difficult to solve. In recent years, a family of powerful approaches has emerged, collectively referred to as *robust low-rank modelling.* These approaches are based on the observation that, in several real-life cases, uncorrupted information is of *low rank* whereas noise is *sparse.* This simple but powerful idea has allowed robust low-rank modelling to become the state of the art in several signal processing and data mining applications, such as denoising, reconstruction, completion, and many more.

Robust low-rank modelling typically treats data in the form of *matrices.* Matrices however are inadequate in representing the *higher-order structure* of data. This can be achieved by employing *tensors*, the higher-order generalisation of matrices. Nevertheless, robust low-rank modelling with tensors is still not well studied.

In this thesis, we provide a thorough survey of robust low-rank modelling for matrices and we review the state-of-the-art algorithms. Most importantly, we show how the matrix methods can be extended to tensors, introducing, for the first time in the literature, several new algorithms for robust low-rank modelling with tensors. Finally, we demonstrate how robust low-rank modelling, both with matrices and tensors, can be successfully applied to real-life problems, with an emphasis on image analysis and computer vision applications.

1.1 Overview of Robust Low-Rank Modelling

1.1.1 The Information Recovery Problem

Let $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$ be a set of n vectors in $\mathbb{R}^m$. In practice they can represent any measurement or observation, such as images, videos, waveforms and so on. We assume that each vector $\mathbf{x}_i \in \mathbb{R}^m$ was generated by corrupting $\mathbf{a}_i \in \mathbb{R}^m$ with noise $\mathbf{e}_i \in \mathbb{R}^m$, which we express as follows

$$\mathbf{x}_i = \mathbf{a}_i + \mathbf{e}_i \tag{1.1}$$

for $i \in \{1, 2, \ldots, n\}$. For our purposes, it is more convenient to write the above in matrix form. Let $\mathbf{X} = \begin{bmatrix}\mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_n\end{bmatrix}$, $\mathbf{A} = \begin{bmatrix}\mathbf{a}_1 & \mathbf{a}_2 & \cdots & \mathbf{a}_n\end{bmatrix}$ and $\mathbf{E} = \begin{bmatrix}\mathbf{e}_1 & \mathbf{e}_2 & \cdots & \mathbf{e}_n\end{bmatrix}$ be matrices in $\mathbb{R}^{m\times n}$ that contain vectors $\mathbf{x}_i$, $\mathbf{a}_i$ and $\mathbf{e}_i$ as columns respectively. Then we can equivalently write equation (1.1) as follows

$$\mathbf{X} = \mathbf{A} + \mathbf{E} \tag{1.2}$$

The *information recovery problem* asks the following; given a data matrix $\mathbf{X}$, can we recover the uncorrupted data $\mathbf{A}$ and the noise $\mathbf{E}$? It immediately becomes apparent that the problem is *ill-posed*, as by knowing only mn values we require to compute $2mn$ values.

1.1.2 The Low-Rank and Sparse Assumption

In order to overcome the ill-posedness of the naively-stated information recovery problem, we need to impose certain conditions on the form of $\mathbf{A}$ and $\mathbf{E}$. Such conditions shall come from our *prior knowledge* of their nature. The central idea of robust low-rank modelling is to model $\mathbf{A}$ as *low-rank* and $\mathbf{E}$ as *sparse.*

The low-rank assumption for $\mathbf{A}$ comes from the fact that, in many real-life cases, uncorrupted data (which $\mathbf{A}$ represents) appear to be *similar* or *correlated* to a certain degree. For instance, an image of the same person's face, taken under various illumination conditions, is expected to have much fewer *degrees of freedom* than a general image with m pixels does. Robust low-rank modelling expresses this form of prior knowledge by requiring that all $\mathbf{a}_i \in \mathbb{R}^m$ live in a *low-dimensional subspace.* This is equivalent to requiring that the rank of $\mathbf{A}$, which is equal to the dimensionality of the subspace spanned by vectors $\mathbf{a}_i$, be small.

As for $\mathbf{E}$, a traditional approach would be to assume that it is *small* and *Gaussian distributed.* This assumption in fact leads to one of the most popular and widely used methods, the well-known *Principal Component Analysis* [39, 43, 71]. Introduced as early as 1901, PCA can be considered as one of the earliest approaches towards low-rank modelling. Nevertheless, PCA is very sensitive to noise that invalidates the Gaussian assumption; and in practice such noise can be rather common, as real-life datasets are often contaminated by gross errors and outliers.

In order to make low-rank modelling *robust* to a wide range of corruptions (such as cases (c),(d) and (e) in Fig. 1.1), we need a better and more general description for noise than naively assuming it is Gaussian. To achieve such a description, we shall note that gross errors *do not occur too often*, that is, they typically corrupt a small portion of the data. This can be expressed by requiring that the noise term $\mathbf{E}$ consist mostly of *zeros* or, equivalently, requiring that $\mathbf{E}$ be *sparse.*

1.1.3 The Formulation as Optimisation Problem

So far, it should have become obvious that robust low-rank modelling is all about *decomposing* a given data matrix $\mathbf{X} \in \mathbb{R}^{m\times n}$ into a low-rank component $\mathbf{A} \in \mathbb{R}^{m\times n}$ and a sparse component $\mathbf{E} \in \mathbb{R}^{m\times n}$. In fact, robust low-rank modelling seeks, among all possible decompositions, the component $\mathbf{A}$ with the *lowest rank* and the *sparsest* $\mathbf{E}$. Using $\operatorname{rank}(\mathbf{A})$ to represent the rank of $\mathbf{A}$ and the so-called ℓ_0-norm $\|\mathbf{E}\|_0$ to represent the number of non-zero entries of $\mathbf{E}$, the above can be simply expressed by the following optimisation problem

$$\min_{\mathbf{A},\mathbf{E}} \operatorname{rank}(\mathbf{A}) + \lambda \|\mathbf{E}\|_0 \quad \text{s.t.} \quad \mathbf{X} = \mathbf{A} + \mathbf{E} \tag{1.3}$$

where λ is a positive regularisation parameter balancing the two terms in problem (1.3).

Essentially, robust low-rank modelling reduces to solving problem (1.3). Nevertheless, due to the discrete nature of the rank and the ℓ_0-norm, this problem is *NP-hard* and therefore *intractable.* Luckily though, the last few years have witnessed the development of a multitude of powerful methods capable of circumventing the intractability of problem (1.3). Most of such methods are based on replacing problem (1.3) with an approximately equivalent tractable problem using *convex relaxation* and/or *matrix factorisation.* In chapter 3 we will review the most important of those methods and describe in full detail the state-of-the-art algorithms that accompany them.

1.1.4 The Extension to Tensors

Several forms of data have some kind of spatial, temporal or other *higher-order structure.* For instance, images have a 2-dimensional spatial structure and videos have a 3-dimensional spatial and temporal structure. By representing such forms of data by vectors, their inherent structure is ultimately *destroyed.*

Tensors on the other hand are capable of preserving such structure. A tensor can be thought of as a *multidimensional array.* In this sense, vectors and matrices are simply tensors with one and two dimensions respectively. As an example, a two-dimensional tensor may represent an image and a three-dimensional tensor may represent a video. In section 2.2 we will describe tensors and their mathematical properties in detail.

Tensors have been successfully used in the past in the analysis of structured data, typically by extending some relevant concept from linear algebra. Notable examples include *TensorFaces* [82], which is the tensor extension of the well-known concept of *EigenFaces* [77], and *Multilinear Principal Component Analysis* [58], which is the tensor extension of classical *Principal Component Analysis* [39, 43, 71].

Using robust low-rank modelling in combination with tensors is an ongoing research challenge and a development of both theoretical and practical significance. In principle, it consists in decomposing a given data tensor $\boldsymbol{\mathcal{X}}$ into a low-rank tensor $\boldsymbol{\mathcal{A}}$ and a sparse tensor $\boldsymbol{\mathcal{E}}$ by solving a problem of the following kind

$$\min_{\boldsymbol{\mathcal{A}},\boldsymbol{\mathcal{E}}} \operatorname{rank}(\boldsymbol{\mathcal{A}}) + \lambda \|\boldsymbol{\mathcal{E}}\|_0 \quad \text{s.t.} \quad \boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{A}} + \boldsymbol{\mathcal{E}} \tag{1.4}$$

There are several non-trivial challenges that need to be addressed for this to be made possible. Tractable measures for tensor rank and sparsity need to be defined and algorithms for solving problem (1.4) need to be developed. So far, important work has been done in [32, 49, 56, 76, 85, 90], however most methods for robust low-rank modelling available for matrices are still not directly applicable to tensors.

This thesis comes to fill the above gap in the literature. In chapter 4 we show how most of the matrix methods presented in chapter 3 can be extended to tensors and we introduce tractable algorithms for solving problem (1.4).

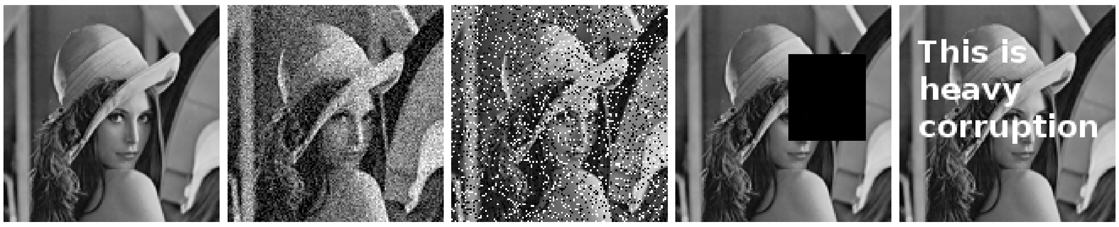
1.2 Summary of Contributions

Our work presented in this thesis contributes to the field of robust low-rank modelling in the following ways.

(i) **Complete survey on matrix methods**. We provide a thorough survey on robust low-rank modelling for matrices (chapter 3). We present the state-of-the-art methods and describe in detail the available algorithms for solving them. Since robust low-rank modelling is a relatively recent topic, it is the first time that such a survey is presented in the literature.

(ii) **Extensions to tensors**. We extend to tensors the most important methods of robust low-rank modelling that are currently only applicable to matrices and we introduce novel algorithms for solving them (chapter 4). As part of these extensions, we introduce robust formulations of the most important tensor decompositions, the CP decomposition and the HOSVD. To the best of our knowledge, this is the first time that such extensions are proposed.

(iii) **Comparative evaluation**. We make a thorough experimental evaluation of the available matrix algorithms and the herein developed tensor algorithms (chapter 7). We compare them in terms of both performance and efficiency using synthetic and real benchmarks, with an emphasis on image analysis and computer vision applications.



(a) No noise (b) Gaussian (c) Salt & pepper (d) Area (e) Text

Figure 1.1: Various types of noise in a greyscale image. Cases (c), (d) and (e) correspond to heavy, but sparse, non-Gaussian noise.

(iv) **Software toolbox release**. We make available a MATLAB toolbox which implements all the methods described herein, both the existing and the newly proposed ones. To the best of our knowledge, this is the most comprehensive implementation of robust low-rank modelling currently available. Our hope is that this toolbox will promote and facilitate future research and development in this promising field.

1.3 Applications of Robust Low-Rank Modelling

Robust low-rank modelling is particularly useful in several applications, typically as a data preprocessing step. Its applicability relies on the data under consideration fitting in some way the low-rank and sparse assumption described in section 1.1.2. Several such cases can be found in practice, the most representative of which are outlined in the following sections.

1.3.1 Denoising and Reconstruction

Denoising refers to reconstruction of data corrupted by noise. If the data are of low rank and the corruption is sparsely supported, then robust low-rank modelling can be successfully used for denoising and reconstruction (see e.g. [5, 69, 93]). Some examples of corrupted images are depicted in Fig. 1.1. Note that the noise in cases (c), (d) and (e) can be effectively modelled as sparse.

1.3.2 Missing Data Completion

A common case in real scenarios is that a part of the data is missing and needs to be recovered. If the data can be modelled as low-rank, the missing value problem can be cast as a robust low-rank modelling problem. In this case, the low-rank component contains the full dataset, whereas the sparse component accounts for the missing values (see e.g. [16, 29, 42, 50, 56, 89]).

An example of missing data completion can be found in *recommender systems* [45]. Such systems record the preferences of users for certain selections of products (e.g. films, songs, books) and based on them they make recommendations to other users. Such a dataset of preferences can be modelled as low-rank, based on the fact that certain groups of users have similar preferences (e.g. similar movies may be liked by the same age group). In other words, the columns of the preference matrix are *linearly correlated.* A recommendation to a new user is made by essentially recovering their missing preferences. The *Netflix prize problem*[1] is a famous instance of the above.

Another example from the field of image processing is *image inpainting* [9]. Image inpainting refers to the case where part of an image is missing and the information from the rest of the image is used to fill in the unknown region, as the example in Fig. 1.2 demonstrates.

[1]http://www.netflixprize.com/

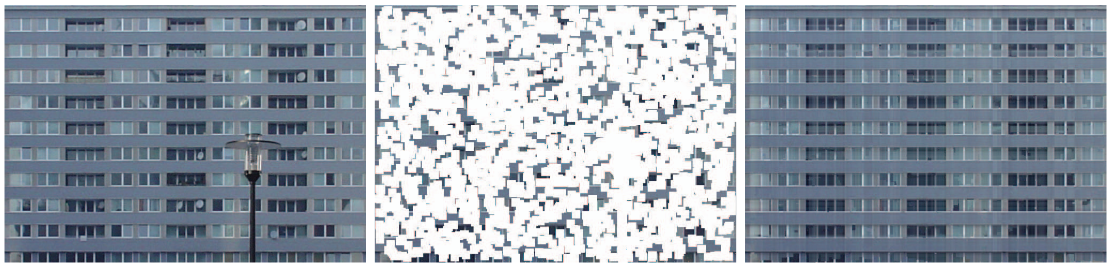

(a) Original (b) Missing values (c) Reconstructed

Figure 1.2: Example of inpainting. In (b) the white regions are the missing values. In (c) the image is reconstructed only by the observed values in (b). Notice that the lamp and the satellite dishes have been removed. Image taken from [56].

1.3.3 Face Recognition

Face recognition refers to the automatic identification of a person from an image of their face. It is known [6] that an image of a convex Lambertian[2] object, under various illuminations, lies approximately on a 9-dimensional subspace. Since a face is approximately convex and Lambertian, face images typically are of low rank. This low-rank property of face images has been exploited for face recognition as early as 1991, with the usage of *EigenFaces* [81].

Face images often include corruptions that compromise the low-rank assumption and therefore the performance of recognition algorithms that rely on it. Such corruptions may include facial expressions, glasses, specularities, shadows, beards, scarfs, and so on. Robust low-rank modelling can be used to restore the low-rank property of the face images by removing such corruptions (as shown in Fig. 1.3) and thus improve performance of traditional low-rank-based systems [21].

1.3.4 Background Subtraction and Foreground Segmentation

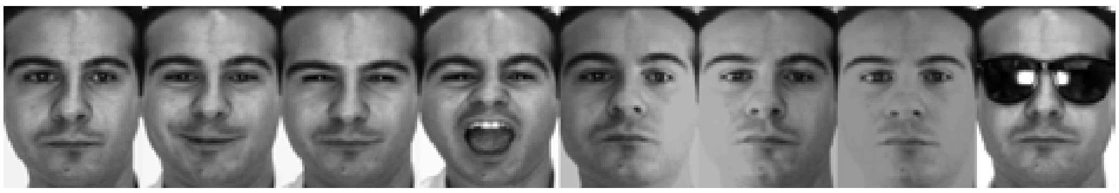
Given a set of consecutive video frames, the goal of *background subtraction* is to separate the background from moving foreground objects. Background subtraction is a fundamental preprocessing step in computer vision applications such as video surveillance, object tracking, motion estimation and so on. Since the background of a video sequence changes only slightly between consecutive video frames, it can be successfully modelled as low-rank, whereas moving foreground objects can be thought of as sparse corruption (see e.g. [5, 12, 14, 18, 75]). A single frame from a surveillance video is shown in Fig. 1.4, where robust low-rank modelling has been used to separate the background from the foreground.

1.3.5 Rectification and Alignment

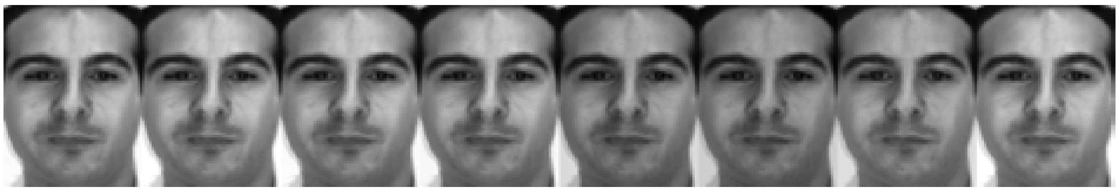
Data alignment refers to the process of transforming the data such that their point-to-point correspondence is maximised. It is mostly used with respect to images (as in [72, 90, 91]), however it can also be used with respect to data sequences, such as video sequences or trajectory data (see e.g. [68]).

The idea behind the usage of robust low-rank modelling for data alignment is that the set of aligned data points, due to the resulting point-to-point similarity among them, should be low-rank, up to possible local differences. Practical systems that use robust low-rank modelling for alignment purposes typically work by seeking the alignment which yields the best low-rank/sparse decomposition. At the end, the low-rank component will contain the aligned data points and the sparse component will contain the local differences between them, as well as possible corruptions by noise. An example of image alignment is shown in Fig. 1.5.

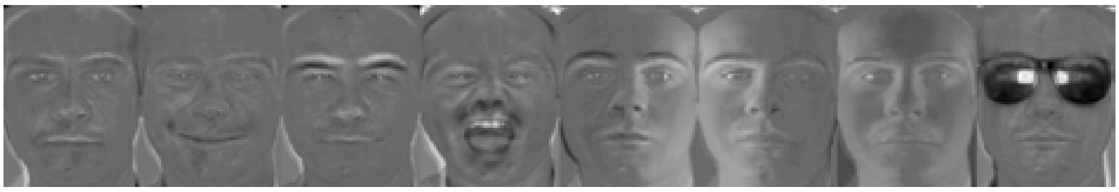
[2]Lambertian is an object that reflects light equally to all directions, i.e. it exhibits no specular reflection.



(a) Original, including corruptions

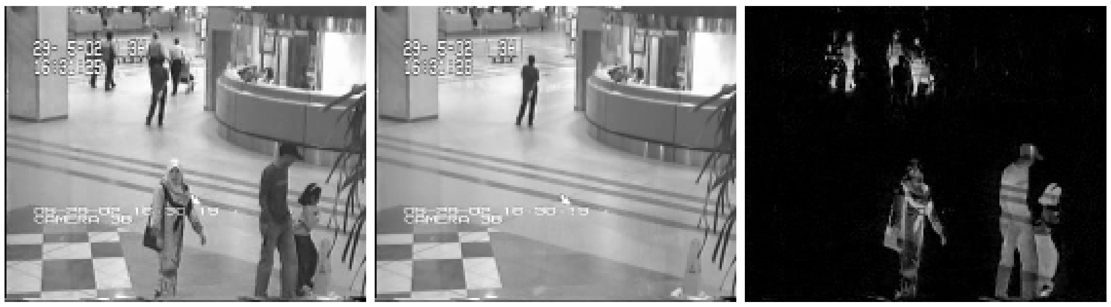


(b) Reconstructed low-rank component



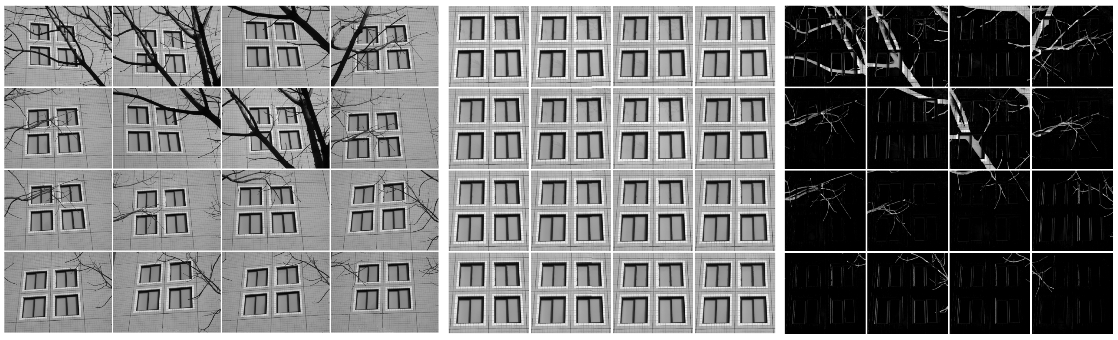
(c) Sparse error component

Figure 1.3: Example of corruption removal from face images for the purpose of face recognition. Image taken from [21].



(a) Original (b) Low-rank component (c) Sparse component

Figure 1.4: Example of background subtraction from a surveillance video. A single video frame is shown. The low-rank component (b) contains the background and the sparse component (c) contains the foreground. The man in (b) was not moving and hence was incorporated in the background. Image taken from [18].



(a) Original (b) Low-rank component (c) Sparse component

Figure 1.5: Example of aligning a set of 16 window images. The low-rank component (b) contains the aligned images and the sparse component (c) contains the corruptions by tree branches. Image taken from [72].

1.3.6 Other Applications

Other applications of robust low-rank modelling include structure from motion and photometric stereo [14], image classification and segmentation [15], gait recognition from gait energy images [38], video stabilisation and moving object detection in turbulence [67], and automatic construction of person-specific facial deformable models [74].

Chapter 2

Mathematical Preliminaries

In this chapter we provide the necessary mathematical background required for the rest of the thesis. We discuss matrix norms, provide an overview of multilinear algebra, including tensor decompositions, and describe the optimisation algorithms that will be used in the chapters to follow. Although familiarity with basic concepts from linear algebra is assumed, a comprehensive introduction to linear algebra can be found in [78].

2.1 Matrix Norms

Since matrix norms will play a central role in robust low-rank modelling for matrices, we shall discuss them in detail here. We begin by the general definition of a norm in a real vector space.

Definition 2.1 (Norm). *Let $\mathcal{V}$ be a vector space on the field of real numbers $\mathbb{R}$. A norm on $\mathcal{V}$ is any function $\|\cdot\| : \mathcal{V} \to [0, +\infty)$ which satisfies the following three properties*

(i) $\|\mathbf{v}\| = 0 \;\Rightarrow\; \mathbf{v} = \mathbf{0}$

(ii) $\|a\mathbf{v}\| = |a| \, \|\mathbf{v}\|$

(iii) $\|\mathbf{v} + \mathbf{w}\| \leq \|\mathbf{v}\| + \|\mathbf{w}\|$

for all $\mathbf{v}, \mathbf{w} \in \mathcal{V}$ and $a \in \mathbb{R}$.

The following theorem follows directly from the definition of the norm.

Theorem 2.1 (Convexity of norms). *A norm $\|\cdot\| : \mathcal{V} \to [0, +\infty)$ is a convex function.*

Proof. Let $\mathbf{v}, \mathbf{w} \in \mathcal{V}$ and $\lambda \in [0, 1]$. Notice that $\lambda\mathbf{v} + (1 - \lambda)\,\mathbf{w} \in \mathcal{V}$. By the norm properties we have

$$\|\lambda\mathbf{v} + (1-\lambda)\,\mathbf{w}\| \leq \|\lambda\mathbf{v}\| + \|(1-\lambda)\,\mathbf{w}\| = \lambda\,\|\mathbf{v}\| + (1-\lambda)\,\|\mathbf{w}\| \tag{2.1}$$

which concludes the proof. □

The notion of a matrix norm arises when we identify $\mathcal{V} = \mathbb{R}^{m \times n}$, i.e. the set of real matrices of size $m \times n$. In this thesis, we will focus our attention on two matrix norms, namely the *elementwise ℓ_p-norm* and the *Schatten p-norm.*[1]

Definition 2.2 (Matrix elementwise ℓ_p-norm). *Let $\mathbf{X} \in \mathbb{R}^{m \times n}$ be a matrix of size $m \times n$. For $p \geq 1$, the elementwise ℓ_p-norm of $\mathbf{X}$ is defined as*

$$\|\mathbf{X}\|_p = \left(\sum_{i=1}^{m} \sum_{j=1}^{n} |X_{ij}|^p \right)^{1/p} \tag{2.2}$$

[1]Another common matrix norm is the *operator* or *induced norm.*

Definition 2.3 (Matrix Schatten p-norm). *Let $\mathbf{X} \in \mathbb{R}^{m \times n}$ be a matrix of size $m \times n$. Let σ_i be the i^{th} singular value of $\mathbf{X}$, for $i \in \{1, 2, \dots, \min(m, n)\}$. For $p \geq 1$, the Schatten p-norm of $\mathbf{X}$ is defined as*

$$\|\mathbf{X}\|_{S_p} = \left(\sum_{i=1}^{\min(m,n)} \sigma_i^p \right)^{1/p} \tag{2.3}$$

Notice that the elementwise ℓ_p-norm is simply the ℓ_p-norm of the vectorised matrix and the Schatten p-norm is the ℓ_p-norm of the vector consisting of the singular values. For $p = 1$, the Schatten p-norm becomes simply the sum of the singular values and it is specifically known as the *nuclear* or *trace norm*.

Definition 2.4 (Matrix nuclear norm). *Let $\mathbf{X} \in \mathbb{R}^{m \times n}$ be a matrix of size $m \times n$. Let σ_i be the i^{th} singular value of $\mathbf{X}$, for $i \in \{1, 2, \dots, \min(m, n)\}$. The nuclear norm of $\mathbf{X}$ is defined as*

$$\|\mathbf{X}\|_* = \sum_{i=1}^{\min(m,n)} \sigma_i \tag{2.4}$$

The following theorem holds in general for Schatten p-norms.

Theorem 2.2 (Unitary invariance of Schatten p-norms). *Schatten p-norms are unitary invariant, i.e. for any matrices $\mathbf{X}, \mathbf{U}, \mathbf{V}$ such that $\mathbf{U}^T\mathbf{U} = \mathbf{I}$ and $\mathbf{V}^T\mathbf{V} = \mathbf{I}$ we have*

$$\left\|\mathbf{U}\mathbf{X}\mathbf{V}^T\right\|_{S_p} = \|\mathbf{X}\|_{S_p} \tag{2.5}$$

Proof. Assume the Singular Value Decomposition $\mathbf{X} = \mathbf{U}_x\mathbf{S}_x\mathbf{V}_x^T$. Then

$$\mathbf{U}\mathbf{X}\mathbf{V}^T = (\mathbf{U}\mathbf{U}_x)\,\mathbf{S}_x\,(\mathbf{V}\mathbf{V}_x)^T \tag{2.6}$$

is a valid Singular Value Decomposition, since

$$(\mathbf{U}\mathbf{U}_x)^T(\mathbf{U}\mathbf{U}_x) = \mathbf{U}_x^T\mathbf{U}^T\mathbf{U}\mathbf{U}_x = \mathbf{U}_x^T\mathbf{U}_x = \mathbf{I} \tag{2.7}$$

$$(\mathbf{V}\mathbf{V}_x)^T(\mathbf{V}\mathbf{V}_x) = \mathbf{V}_x^T\mathbf{V}^T\mathbf{V}\mathbf{V}_x = \mathbf{V}_x^T\mathbf{V}_x = \mathbf{I} \tag{2.8}$$

and, therefore, $\mathbf{X}$ and $\mathbf{U}\mathbf{X}\mathbf{V}^T$ share the same matrix of singular values $\mathbf{S}_x$. Hence

$$\left\|\mathbf{U}\mathbf{X}\mathbf{V}^T\right\|_{S_p} = \|\mathbf{S}_x\|_p = \|\mathbf{X}\|_{S_p} \tag{2.9}$$

□

However, unlike the Schatten p-norm, the elementwise ℓ_p-norm is not unitary invariant for all p.

The *standard inner product* between real vectors[2] can be easily extended to matrices as follows.

Definition 2.5 (Matrix inner product). *Let $\mathbf{X}, \mathbf{Y} \in \mathbb{R}^{m \times n}$ be matrices of size $m \times n$. Their inner product is defined by*

$$\langle \mathbf{X}, \mathbf{Y} \rangle = \sum_{i=1}^{m}\sum_{j=1}^{n} X_{ij}Y_{ij} \tag{2.10}$$

The following properties of the matrix inner product follow easily from the definition:

(i) $\langle \mathbf{X}, \mathbf{Y} \rangle = \langle \mathbf{Y}, \mathbf{X} \rangle$

[2]Also known as the *dot product*.

(ii) $\langle a\mathbf{X}, \mathbf{Y}\rangle = \langle \mathbf{X}, a\mathbf{Y}\rangle = a\langle \mathbf{X}, \mathbf{Y}\rangle$

(iii) $\langle \mathbf{X}_1 + \mathbf{X}_2, \mathbf{Y}\rangle = \langle \mathbf{X}_1, \mathbf{Y}\rangle + \langle \mathbf{X}_2, \mathbf{Y}\rangle$

(iv) $\langle \mathbf{X}, \mathbf{Y}_1 + \mathbf{Y}_2\rangle = \langle \mathbf{X}, \mathbf{Y}_1\rangle + \langle \mathbf{X}, \mathbf{Y}_2\rangle$

(v) $\langle \mathbf{A}\mathbf{X}, \mathbf{Y}\rangle = \left\langle \mathbf{X}, \mathbf{A}^T\mathbf{Y}\right\rangle$

(vi) $\langle \mathbf{X}, \mathbf{Y}\mathbf{B}\rangle = \left\langle \mathbf{X}\mathbf{B}^T, \mathbf{Y}\right\rangle$

Another important matrix norm, which can be thought of as the natural generalisation of the Euclidean norm to matrices, is the *Frobenius norm.*

Definition 2.6 (Matrix Frobenius norm). *Let* $\mathbf{X} \in \mathbb{R}^{m\times n}$ *be a matrix of size* $m \times n$. *The Frobenius norm of* $\mathbf{X}$ *is defined as*

$$\|\mathbf{X}\|_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}(X_{ij})^2} \tag{2.11}$$

The Frobenius norm is induced by the inner product, since $\|\mathbf{X}\|_F = \sqrt{\langle \mathbf{X}, \mathbf{X}\rangle}$. It is also straightforward to see that $\|\mathbf{X}\|_F = \|\mathbf{X}\|_2$. Although less obvious, the same is also true for the Schatten p-norm, as the following theorem states.

Theorem 2.3 (Relation between the Frobenius and the Schatten p-norm). *For* $p = 2$, *the Schatten* p*-norm is equal to the Frobenius norm, i.e.* $\forall\, \mathbf{X} \in \mathbb{R}^{m\times n}$

$$\|\mathbf{X}\|_{S_2} = \|\mathbf{X}\|_F \tag{2.12}$$

Proof. Assume the Singular Value Decomposition $\mathbf{X} = \mathbf{U}_x\mathbf{S}_x\mathbf{V}_x^T$. Then we have

$$\begin{aligned}\|\mathbf{X}\|_F^2 &= \langle \mathbf{X}, \mathbf{X}\rangle \\ &= \left\langle \mathbf{U}_x\mathbf{S}_x\mathbf{V}_x^T, \mathbf{U}_x\mathbf{S}_x\mathbf{V}_x^T\right\rangle \\ &= \left\langle \mathbf{S}_x\mathbf{V}_x^T\mathbf{V}_x, \mathbf{U}_x^T\mathbf{U}_x\mathbf{S}_x\right\rangle \\ &= \langle \mathbf{S}_x, \mathbf{S}_x\rangle \\ &= \|\mathbf{X}\|_{S_2}^2 \end{aligned} \tag{2.13}$$

□

From the above result it follows that the Frobenius norm is unitary invariant, as a special case of the Schatten p-norm.

In this thesis, we will mostly use norms as a measure for the *rank* and the *sparsity* of matrices. The rank of a matrix $\mathbf{X} \in \mathbb{R}^{m\times n}$, denoted by $\operatorname{rank}(\mathbf{X})$, is the maximum number of columns which, if seen as vectors, are linearly independent. It can easily be shown that the rank is always equal to the number of non-zero singular values. The sparsity of $\mathbf{X}$ is quantified by the number of its non-zero entries, denoted by $\|\mathbf{X}\|_0$ and often referred to as ℓ_0-norm, for reasons that will soon become clear. Note however that $\|\cdot\|_0$ is not a norm in the strict sense, as it does not satisfy definition 2.1.

In order to closely approximate $\operatorname{rank}(\cdot)$ and $\|\cdot\|_0$, we will consider the elementwise p-norm and the Schatten p-norm when $0 < p < 1$. For these values of p, property (iii) of Definition 2.1 does not hold anymore,[3] which makes $\|\cdot\|_p$ and $\|\cdot\|_{S_p}$ non-convex and not norms in the strict sense. Nevertheless, the following important property holds.

Theorem 2.4 (Approximation of rank and sparsity by norms). *Let* $\mathbf{X} \in \mathbb{R}^{m\times n}$. *Then*

[3]This property is known as *subadditivity* or *triangle inequality* and is necessary for convexity.

(i) $\lim_{p\to 0^+} \|\mathbf{X}\|_p^p = \|\mathbf{X}\|_0$

(ii) $\lim_{p\to 0^+} \|\mathbf{X}\|_{S_p}^p = \mathit{rank}(\mathbf{X})$

Proof. First notice that for any $x \in \mathbb{R}$

$$\lim_{p\to 0^+} x^p = \begin{cases} 1 & x \neq 0 \\ 0 & x = 0 \end{cases} \tag{2.14}$$

Then, (i) is shown as follows

$$\lim_{p\to 0^+} \|\mathbf{X}\|_p^p = \sum_{i=1}^{m}\sum_{j=1}^{n} \lim_{p\to 0^+} |X_{ij}|^p = \|\mathbf{X}\|_0 \tag{2.15}$$

As for (ii), given that the rank is the number of non-zero singular values we have

$$\lim_{p\to 0^+} \|\mathbf{X}\|_{S_p}^p = \sum_{i=1}^{\min(m,n)} \lim_{p\to 0^+} \sigma_i^p = \operatorname{rank}(\mathbf{X}) \tag{2.16}$$

□

We may observe that, even though $\|\cdot\|_p$ and $\|\cdot\|_{S_p}$ become non-convex for $p < 1$, they can arbitrarily approximate $\|\cdot\|_0$ and $\operatorname{rank}(\mathbf{X})$ respectively as $p \to 0$.

Definition 2.7 (Convex envelope)**.** *Let $f : \mathbb{R}^n \to \mathbb{R}$ be a real function. The convex envelope of f, represented by $\mathcal{C}_f$, is the pointwise largest function $\mathcal{C}_f : \mathbb{R}^n \to \mathbb{R}$ which is pointwise less than f. In other words*

$$\mathcal{C}_f = \mathit{sup}\{g : \mathbb{R}^n \to \mathbb{R} \mid g \text{ is convex and } g(\mathbf{x}) \le f(\mathbf{x}) \ \forall \mathbf{x} \in \mathbb{R}^n\} \tag{2.17}$$

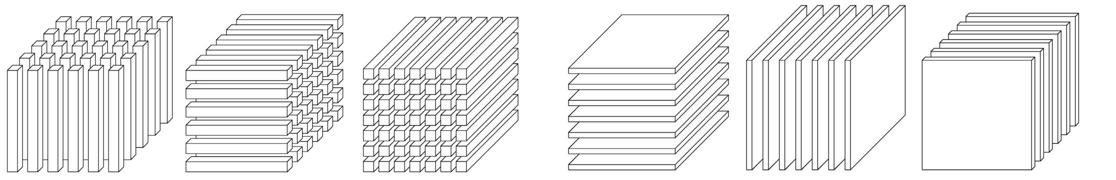
The above definition means that the convex envelope of a function is its *tightest convex underestimate*. Interestingly, $\|\cdot\|_1$ and $\|\cdot\|_*$ have been found to be the convex envelopes of $\|\cdot\|_0$ and $\operatorname{rank}(\cdot)$ respectively. This property will be particularly useful in the development of algorithms in chapters 3 and 4. For more details and relevant proofs, the reader may refer to [24, 28, 73].

2.2 Multilinear Algebra Basics

2.2.1 Tensor Basics

A *tensor* is a multidimensional array. The number of array dimensions is known as the *tensor order*. Alternatively, the tensor order can be described as the necessary number of indices to specify each of the tensor's entries. In that sense, a tensor is a natural extension of scalars, vectors and matrices to a higher order; a scalar is a 0th-order tensor, a vector is a 1st-order tensor and a matrix is a 2nd-order tensor. Mathematically, an N^{th}-order tensor is defined as an element of a *tensor space*, which in turn is given by the tensor product of N vector spaces. The study of tensors and their applications is known as *multilinear algebra*. For a thorough discussion on tensors, their properties and their applications, the reader may refer to [1, 44].

Following the usual conventions found in the literature, in this thesis we will denote tensors by boldface calligraphic letters. For instance, $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1\times I_2\times\cdots\times I_N}$ will be an N^{th}-order tensor of size $I_1 \times I_2 \times \cdots \times I_N$. For specifying each of its entries, an n-tuple index is needed. The entry of $\boldsymbol{\mathcal{X}}$ corresponding to index $(i_1, i_2, \ldots, i_N)$, with $i_n \in \{1, 2, \ldots, I_n\}$, will be denoted as $X_{i_1 i_2 \cdots i_N}$. If all but one of the indices are kept fixed and the remaining index is left to vary, we get a vector called *fibre*. Fibres can be thought of as the tensor equivalent of matrix rows and columns. For



(a) Tensor fibres (b) Tensor slices

Figure 2.1: Illustration of a 3rd-order tensor, showing its fibres and slices along various modes. Image taken from [44].

instance, the fibres of length I_n formed by letting index i_n vary while keeping all other indices fixed are called *n-mode fibres*. Similarly to fibres, when all but two indices are fixed and the rest are left to vary, we get a matrix called *slice*. Fig. 2.1 illustrates the various fibres and slices of a 3rd-order tensor.

In order to work with tensors, it is often convenient to transform them to vectors or matrices, a process known as *vectorisation* and *matricisation* respectively.[4]

Definition 2.8 (Tensor vectorisation). *Let $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ be an N^{th}-order tensor. The vectorised tensor, denoted as $\mathrm{vec}(\boldsymbol{\mathcal{X}})$, is a vector formed by the tensor entries, such that tensor entry $(i_1, i_2, \ldots, i_N)$ is mapped to vector entry j, where*

$$j = 1 + \sum_{k=1}^{N} (i_k - 1) J_k \quad \textit{and} \quad J_k = \prod_{m=1}^{k-1} I_m \tag{2.18}$$

Definition 2.9 (Tensor matricisation). *Let $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ be an N^{th}-order tensor. The n-mode tensor matricisation, with $n \in \{1, 2, \ldots, N\}$, produces a matrix of size $I_n \times \prod_{k \neq n} I_k$ which is denoted as $\mathbf{X}_{[n]}$ and is formed by the tensor entries, such that tensor entry $(i_1, i_2, \ldots, i_N)$ is mapped to matrix entry (i_n, j), where*

$$j = 1 + \sum_{\substack{k=1 \\ k \neq n}}^{N} (i_k - 1) J_k \quad \textit{and} \quad J_k = \prod_{\substack{m=1 \\ m \neq n}}^{k-1} I_m \tag{2.19}$$

Even though the above definitions seem complicated, in essence vectorisation and matricisation are conceptually simple processes. Vector $\mathrm{vec}(\boldsymbol{\mathcal{X}})$ is simply formed by stacking the entries of $\boldsymbol{\mathcal{X}}$ in a column-major order. That is, the fastest varying index is taken to be i_1, then i_2 and so on up to the slowest varying index i_N. Matrix $\mathbf{X}_{[n]}$ is similarly formed but instead of stacking entries, we stack fibres. In other words, the columns of $\mathbf{X}_{[n]}$ are the n-mode fibres of $\boldsymbol{\mathcal{X}}$ taken in a column-major order, by first varying i_1, then i_2 up to i_N, excluding of course i_n. The above should be clarified by the following example.

Example 2.1 (Tensor matricisation and vectorisation). *Consider the $2 \times 3 \times 2$ tensor formed by the following two slices*

$$\mathbf{X}(:,:,1) = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix} \quad \mathbf{X}(:,:,2) = \begin{bmatrix} 7 & 9 & 11 \\ 8 & 10 & 12 \end{bmatrix} \tag{2.20}$$

[4]Matricisation is also known as *unfolding*.

The three matricisations of $\boldsymbol{\mathcal{X}}$ are

$$\mathbf{X}_{[1]} = \begin{bmatrix} 1 & 3 & 5 & 7 & 9 & 11 \\ 2 & 4 & 6 & 8 & 10 & 12 \end{bmatrix} \tag{2.21}$$

$$\mathbf{X}_{[2]} = \begin{bmatrix} 1 & 2 & 7 & 8 \\ 3 & 4 & 9 & 10 \\ 5 & 6 & 11 & 12 \end{bmatrix} \tag{2.22}$$

$$\mathbf{X}_{[3]} = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 7 & 8 & 9 & 10 & 11 & 12 \end{bmatrix} \tag{2.23}$$

The vectorisation of $\boldsymbol{\mathcal{X}}$ is

$$vec(\boldsymbol{\mathcal{X}}) = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \end{bmatrix}^T \tag{2.24}$$

Similarly to vectors and matrices, addition and subtraction between two tensors are defined elementwise. However, multiplication between tensors is a more complicated procedure. In this thesis, we will mainly make use of the *n-mode product* between a tensor and a matrix.

Definition 2.10 (n-mode product)**.** *Let $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ be an N^{th}-order tensor and $\mathbf{U} \in \mathbb{R}^{J \times I_n}$ be a matrix of size $J \times I_n$. The n-mode product between $\boldsymbol{\mathcal{X}}$ and $\mathbf{U}$, denoted by $\boldsymbol{\mathcal{X}} \times_n \mathbf{U}$, is a tensor of size $I_1 \times \cdots \times I_{n-1} \times J \times I_{n+1} \times \cdots \times I_N$ whose entries are given by*

$$(\boldsymbol{\mathcal{X}} \times_n \mathbf{U})_{i_1 \cdots i_{n-1} j i_{n+1} \cdots i_N} = \sum_{i_n=1}^{I_n} X_{i_1 i_2 \cdots i_N} U_{j i_n} \tag{2.25}$$

Conceptually, $\boldsymbol{\mathcal{X}} \times_n \mathbf{U}$ is a tensor whose fibres are given by applying $\mathbf{U}$ on the fibres of $\boldsymbol{\mathcal{X}}$. By the definition, the following properties can be easily proven.

(i) If $m \neq n$, $(\boldsymbol{\mathcal{X}} \times_n \mathbf{U}_1) \times_m \mathbf{U}_2 = (\boldsymbol{\mathcal{X}} \times_m \mathbf{U}_2) \times_n \mathbf{U}_1$

(ii) $(\boldsymbol{\mathcal{X}} \times_n \mathbf{U}_1) \times_n \mathbf{U}_2 = \boldsymbol{\mathcal{X}} \times_n (\mathbf{U}_2 \mathbf{U}_1)$

(iii) $(\boldsymbol{\mathcal{X}} \times_n \mathbf{U})_{[n]} = \mathbf{U}\mathbf{X}_{[n]}$

Property (i) implies that when cascading n-mode products along different modes the order is unimportant, and therefore the parentheses are not necessary. From now on and for the rest of this thesis, we shall use $\boldsymbol{\mathcal{X}} \times_{i=1}^{N} \mathbf{U}_i$ as shorthand for $\boldsymbol{\mathcal{X}} \times_1 \mathbf{U}_1 \times_2 \mathbf{U}_2 \times_3 \cdots \times_N \mathbf{U}_N$.

Extending the definitions of the *elementwise ℓ_p-norm* and the *standard inner product* to tensors is rather straightforward.

Definition 2.11 (Tensor elementwise ℓ_p-norm)**.** *Let $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ be an N^{th}-order tensor. For $p \geq 1$, the elementwise ℓ_p-norm of $\boldsymbol{\mathcal{X}}$ is defined as*

$$\|\boldsymbol{\mathcal{X}}\|_p = \left(\sum_{i_i=1}^{I_1} \sum_{i_2=1}^{I_2} \cdots \sum_{i_N=1}^{I_N} |X_{i_1 i_2 \cdots i_N}|^p \right)^{1/p} \tag{2.26}$$

Definition 2.12 (Tensor inner product)**.** *Let $\boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{Y}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ be N^{th}-order tensors. Their inner product is defined by*

$$\langle \boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{Y}} \rangle = \sum_{i_1=1}^{I_1} \sum_{i_2=1}^{I_2} \cdots \sum_{i_N=1}^{I_N} X_{i_1 i_2 \cdots i_N} Y_{i_1 i_2 \cdots i_N} \tag{2.27}$$

Since the tensor elementwise ℓ_p-norm and the tensor inner product are defined in the same elementwise fashion as with matrices, they have the same properties. For instance, notice that the elementwise ℓ_2-norm is induced by the inner product, i.e. $\|\boldsymbol{\mathcal{X}}\|_2 = \sqrt{\langle \boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{X}} \rangle}$. Similarly to matrices, we refer to $\|\boldsymbol{\mathcal{X}}\|_2$ as the *Frobenius norm* of $\boldsymbol{\mathcal{X}}$ and denote it as $\|\boldsymbol{\mathcal{X}}\|_F$. Also, similarly to matrices, the number of non-zero entries of $\boldsymbol{\mathcal{X}}$, also referred to as the ℓ_0-norm of $\boldsymbol{\mathcal{X}}$, can be obtained as $\|\boldsymbol{\mathcal{X}}\|_0 = \lim_{p \to 0^+} \|\boldsymbol{\mathcal{X}}\|_p^p$. However, extending the Schatten p-norm to tensors is not straightforward, since this norm requires a notion of singular values.

Finally, for simplifying tensor expressions, we will make use of two important matrix products, the *Kronecker product* and the *Khatri-Rao product.*

Definition 2.13 (Kronecker product)**.** *Let* $\mathbf{A} \in \mathbb{R}^{m \times n}$ *and* $\mathbf{B} \in \mathbb{R}^{k \times l}$ *be two matrices of sizes* $m \times n$ *and* $k \times l$ *respectively. Their Kronecker product is a matrix of size* $mk \times nl$*, which, in block matrix form, is given by*

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} A_{11}\mathbf{B} & A_{12}\mathbf{B} & \cdots & A_{1n}\mathbf{B} \\ A_{21}\mathbf{B} & A_{22}\mathbf{B} & \cdots & A_{2n}\mathbf{B} \\ \vdots & \vdots & \ddots & \vdots \\ A_{m1}\mathbf{B} & A_{m2}\mathbf{B} & \cdots & A_{mn}\mathbf{B} \end{bmatrix} \tag{2.28}$$

Definition 2.14 (Khatri-Rao product)**.** *Let* $\mathbf{A} \in \mathbb{R}^{m \times n}$ *and* $\mathbf{B} \in \mathbb{R}^{k \times n}$ *be two matrices of sizes* $m \times n$ *and* $k \times n$ *respectively. Their Khatri-Rao product is a matrix of size* $mk \times n$*, which, in block matrix form, is given by*

$$\mathbf{A} \odot \mathbf{B} = \begin{bmatrix} \mathbf{a}_1 \otimes \mathbf{b}_1 & \mathbf{a}_2 \otimes \mathbf{b}_2 & \cdots & \mathbf{a}_n \otimes \mathbf{b}_n \end{bmatrix} \tag{2.29}$$

where $\mathbf{A} = \begin{bmatrix} \mathbf{a}_1 & \mathbf{a}_2 & \cdots & \mathbf{a}_n \end{bmatrix}$ *and* $\mathbf{B} = \begin{bmatrix} \mathbf{b}_1 & \mathbf{b}_2 & \cdots & \mathbf{b}_n \end{bmatrix}$.

By the above definitions, the following properties can be easily proven.

(i) $(\mathbf{A} \otimes \mathbf{B}) \otimes \mathbf{C} = \mathbf{A} \otimes (\mathbf{B} \otimes \mathbf{C})$

(ii) $(\mathbf{A} \odot \mathbf{B}) \odot \mathbf{C} = \mathbf{A} \odot (\mathbf{B} \odot \mathbf{C})$

(iii) $(\mathbf{A} \otimes \mathbf{B})^T = \mathbf{A}^T \otimes \mathbf{B}^T$

(iv) $(\mathbf{A} \otimes \mathbf{B})(\mathbf{C} \otimes \mathbf{D}) = (\mathbf{AC}) \otimes (\mathbf{BD})$

(v) $\operatorname{vec}\left(\boldsymbol{\mathcal{X}} \times_{i=1}^{N} \mathbf{U}_i\right) = \left(\bigotimes_{i=N}^{1} \mathbf{U}_i\right) \operatorname{vec}(\boldsymbol{\mathcal{X}})$

(vi) $\left(\boldsymbol{\mathcal{X}} \times_{i=1}^{N} \mathbf{U}_i\right)_{[n]} = \mathbf{U}_n \mathbf{X}_{[n]} \left(\bigotimes_{\substack{i=N \\ i \neq n}}^{1} \mathbf{U}_i\right)^T$

Properties (v) and (vi) above demonstrate the usefulness of the Kronecker product in tensor expressions. Note finally that neither the Kronecker nor the Khatri-Rao product are commutative.

2.2.2 Tensor Decompositions

One of the main goals of multilinear algebra is to generalise the techniques of linear algebra, such as matrix decompositions, to tensors. In this section, we present the two perhaps most important tensor decompositions, the *CANDECOMP/PARAFAC decomposition* and the *Higher Order Singular Value Decomposition.*

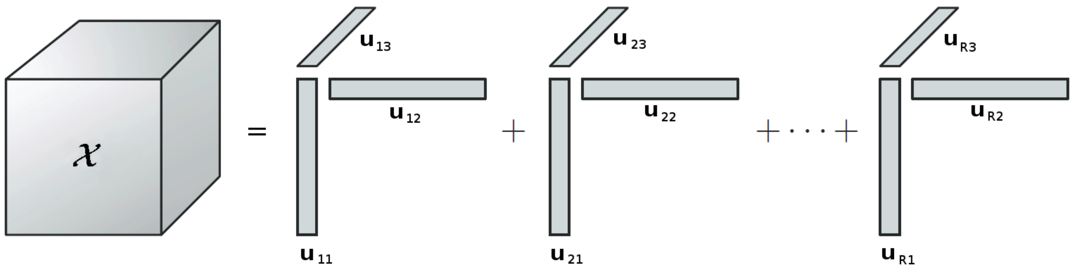


Figure 2.2: Illustration of the CP decomposition of the 3rd-order tensor $\boldsymbol{\mathcal{X}}$ as the sum of rank-1 tensors $\mathbf{u}_{r1} \circ \mathbf{u}_{r2} \circ \mathbf{u}_{r3}$, for $r \in \{1, 2, \ldots, R\}$. Image adapted from [44].

CANDECOMP/PARAFAC Decomposition

The CANDECOMP/PARAFAC (or simply CP) decomposition was first introduced in 1970 in the psychometrics community by Carrol and Chang [19] under the name CANDECOMP (CANonical DECOMPosition) and simultaneously by Harshman [36] under the name PARAFAC (PARAllel FACtors). In the same way the SVD decomposes a matrix as a sum of rank-1 matrices, the CP decomposition expresses a tensor as the sum of *rank*-1 *tensors*.

Definition 2.15 (Rank-1 tensor)**.** *A tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ is said to be rank-*1 *if it can be written as the outer product of N vectors $\mathbf{u}_n \in \mathbb{R}^{I_n}$, for $i \in \{1, 2, \ldots, N\}$, as follows*

$$\boldsymbol{\mathcal{X}} = \mathbf{u}_1 \circ \mathbf{u}_2 \circ \cdots \circ \mathbf{u}_N \tag{2.30}$$

In other words, the entries of $\boldsymbol{\mathcal{X}}$ are given by

$$X_{i_1 i_2 \cdots i_N} = u_{i_1} u_{i_2} \cdots u_{i_N} \tag{2.31}$$

Definition 2.16 (CP decomposition)**.** *Let $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ be an N^{th}-order tensor. The CP decomposition of $\boldsymbol{\mathcal{X}}$ is*

$$\boldsymbol{\mathcal{X}} = \sum_{r=1}^{R} \mathbf{u}_{r1} \circ \mathbf{u}_{r2} \circ \cdots \circ \mathbf{u}_{rN} \tag{2.32}$$

If we define $\mathbf{U}_n = \begin{bmatrix} \mathbf{u}_{1n} & \mathbf{u}_{2n} & \cdots & \mathbf{u}_{Rn} \end{bmatrix}$ for $n \in \{1, 2, \ldots, N\}$, the CP decomposition can be symbolically written as

$$\boldsymbol{\mathcal{X}} = \mathbf{U}_1 \circ \mathbf{U}_2 \circ \cdots \circ \mathbf{U}_N \tag{2.33}$$

In the above definition, matrices $\mathbf{U}_n \in \mathbb{R}^{I_n \times R}$ are known as *factor matrices*. Often, vectors $\mathbf{u}_{rn}$ are chosen such that $\|\mathbf{u}_{rn}\| = 1$. In this case, the CP decomposition is written as

$$\boldsymbol{\mathcal{X}} = \sum_{r=1}^{R} s_r \mathbf{u}_{r1} \circ \mathbf{u}_{r2} \circ \cdots \circ \mathbf{u}_{rN} \tag{2.34}$$

where s_r is a scalar that compensates for the magnitudes of vectors $\mathbf{u}_{rn}$. Fig. 2.2 illustrates the CP decomposition of a 3rd-order tensor. Finally, a computationally useful property of the CP decomposition is the following

$$(\mathbf{U}_1 \circ \mathbf{U}_2 \circ \cdots \circ \mathbf{U}_N)_{[n]} = \mathbf{U}_n \left(\bigodot_{\substack{i=N \\ i \neq n}}^{1} \mathbf{U}_i \right)^T \tag{2.35}$$

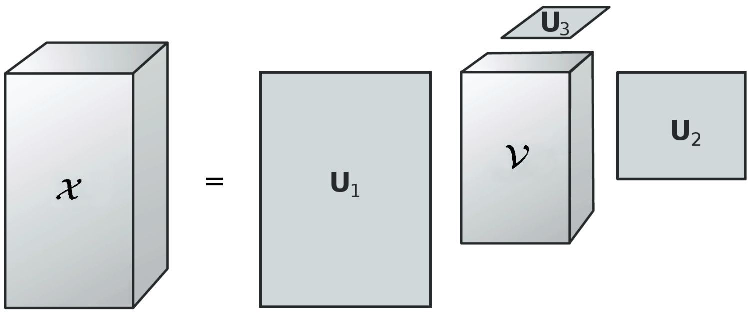


Figure 2.3: Illustration of the HOSVD of the 3rd-order tensor $\boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{V}} \times_1 \mathbf{U}_1 \times_2 \mathbf{U}_2 \times_3 \mathbf{U}_3$. Image adapted from [44].

Higher Order Singular Value Decomposition

The other significant tensor decomposition is the Higher Order Singular Value Decomposition (or simply HOSVD), also known as Tucker decomposition. It was first introduced in 1966 by Tucker [80] in the field of psychometrics. In the same way the matrix SVD computes the left and right singular vectors, HOSVD aims to recover orthonormal bases for the spaces spanned by the n-mode fibres.

Definition 2.17 (HOSVD). *Let $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ be an N^{th}-order tensor. The HOSVD of $\boldsymbol{\mathcal{X}}$ is*

$$\boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{V}} \times_{i=1}^{N} \mathbf{U}_i \tag{2.36}$$

where $\boldsymbol{\mathcal{V}} \in \mathbb{R}^{J_1 \times J_2 \times \cdots \times J_N}$, $J_i \leq I_i$ and all $\mathbf{U}_i \in \mathbb{R}^{I_i \times J_i}$ are column-orthonormal matrices such that $\mathbf{U}_i^T \mathbf{U}_i = \mathbf{I}$.

In the above definition, $\boldsymbol{\mathcal{V}}$ is called the *core tensor*. Unlike the matrix SVD, however, $\boldsymbol{\mathcal{V}}$ does not necessarily result to be diagonal. The HOSVD can be thought of as a higher order form of Principal Component Analysis, where for each $n \in \{1, 2, \ldots, N\}$ the columns of matrix $\mathbf{U}_n$ are the principal components of the n-mode fibres of $\boldsymbol{\mathcal{X}}$. Fig. 2.3 illustrates the HOSVD of a 3rd-order tensor.

The HOSVD and the CP decomposition are related to a certain degree. Notice that if we take $J_1 = J_2 = \cdots = J_N$ and $\boldsymbol{\mathcal{V}}$ to be diagonal (i.e. $V_{i_1 i_2 \cdots i_N} = 0$ if it is not the case that $i_1 = i_2 = \cdots = i_N$) the HOSVD becomes a CP decomposition with column-orthonormal factor matrices. In general however the CP decomposition does not require the factor matrices to be column-orthonormal.

Other Decompositions

In this thesis, we focus our attention on the CP decomposition and the HOSVD. However, in the literature a variety of other decompositions have been proposed, most of which are simply a variation of either the CP decomposition or the HOSVD. Such decompositions include INDSCAL, PARAFAC2, CANDELINC, DEDICOM and PARATUCK2. We refer the interested reader to [44], where further information can be found on all the above decompositions (including the CP and the HOSVD), their properties, algorithms for their computation and their applications.

2.2.3 Tensor Rank

The rank of a matrix is the number of linearly independent column vectors, or, equivalently, the number of non-zero singular values. However, this definition of the matrix rank is not directly extensible to tensors. In order to extend the notion of rank to tensors, we need to consider it from a different perspective. Let $\mathbf{X} \in \mathbb{R}^{m \times n}$ and assume its Singular Value Decomposition $\mathbf{X} = \mathbf{USV}^T$, where $\mathbf{U} = \begin{bmatrix} \mathbf{u}_1 & \mathbf{u}_2 & \cdots & \mathbf{u}_m \end{bmatrix}$ and $\mathbf{V} = \begin{bmatrix} \mathbf{v}_1 & \mathbf{v}_2 & \cdots & \mathbf{v}_n \end{bmatrix}$ are the matrices of the left and right singular vectors respectively and $\mathbf{S} = \operatorname{diag}\left(\sigma_1, \sigma_2, \ldots, \sigma_{\min(m,n)}\right)$ is the diagonal matrix of ordered singular values. Since exactly the first $\operatorname{rank}(\mathbf{X})$ singular values are non-zero, the SVD can also be written as

$$\mathbf{X} = \sum_{i=1}^{\operatorname{rank}(\mathbf{X})} \sigma_i \mathbf{u}_i \mathbf{v}_i^T \tag{2.37}$$

Since each $\mathbf{u}_i \mathbf{v}_i^T$ is a matrix of rank 1, we see that any matrix $\mathbf{X}$ can be written as a sum of exactly $\operatorname{rank}(\mathbf{X})$ matrices of rank 1. Motivated by the above observation, we can formulate a definition of the tensor rank, based on the CP decomposition.

Definition 2.18 (Tensor rank)**.** *Let $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ be an N^{th}-order tensor. The rank of $\boldsymbol{\mathcal{X}}$, denoted as rank$(\boldsymbol{\mathcal{X}})$, is defined as the smallest positive integer R for which we can write $\boldsymbol{\mathcal{X}}$ as a sum of R rank-1 tensors*

$$\boldsymbol{\mathcal{X}} = \sum_{r=1}^{R} \mathbf{u}_{r1} \circ \mathbf{u}_{r2} \circ \cdots \circ \mathbf{u}_{rN} \tag{2.38}$$

Nevertheless, the above definition of tensor rank, however elegant it may be, is known to be NP-hard to compute [37], which makes it impractical for most applications. Instead, in this thesis we will focus our attention on the so-called *tensor n-rank*.

Definition 2.19 (Tensor n-rank)**.** *Let $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ be an N^{th}-order tensor. The n-rank of $\boldsymbol{\mathcal{X}}$, denoted as rank$_n(\boldsymbol{\mathcal{X}})$, is defined as the number of linearly independent n-mode fibres of $\boldsymbol{\mathcal{X}}$. Equivalently*

$$\mathit{rank}_n(\boldsymbol{\mathcal{X}}) = \mathit{rank}\left(\mathbf{X}_{[n]}\right) \tag{2.39}$$

In the matrix case, since $\operatorname{rank}(\mathbf{X}) = \operatorname{rank}\left(\mathbf{X}^T\right)$, we have $\operatorname{rank}_1(\mathbf{X}) = \operatorname{rank}_2(\mathbf{X})$, or that the number of linearly independent column vectors is always equal to the number of linearly independent row vectors. However, for a general N^{th}-order tensor this is not true, i.e. in general $\operatorname{rank}_n(\boldsymbol{\mathcal{X}}) \neq \operatorname{rank}_m(\boldsymbol{\mathcal{X}})$ for $n \neq m$.

The following theorems provide useful upper bounds of the n-rank.

Theorem 2.5 (Upper bound of the n-rank based on CP decomposition)**.** *Let $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ be a tensor having the following CP decomposition*

$$\boldsymbol{\mathcal{X}} = \mathbf{U}_1 \circ \mathbf{U}_2 \circ \cdots \circ \mathbf{U}_N \tag{2.40}$$

Then, the following inequality holds

$$\mathit{rank}_n(\boldsymbol{\mathcal{X}}) \leq \mathit{rank}(\mathbf{U}_n) \tag{2.41}$$

Proof.

$$\begin{aligned}
\operatorname{rank}_n(\boldsymbol{\mathcal{X}}) &= \operatorname{rank}\left(\mathbf{X}_{[n]}\right) \\
&= \operatorname{rank}\left(\left(\mathbf{U}_1 \circ \mathbf{U}_2 \circ \cdots \circ \mathbf{U}_N\right)_{[n]}\right) \\
&= \operatorname{rank}\left(\mathbf{U}_n \left(\bigodot_{\substack{i=N \\ i \neq n}}^{1} \mathbf{U}_i\right)^T\right) \\
&\leq \operatorname{rank}(\mathbf{U}_n)
\end{aligned} \tag{2.42}$$

□

Theorem 2.6 (Relation between the rank and the n-rank)**.** *For any* $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ *the following inequality holds* $\forall n \in \{1, 2, \dots, N\}$

$$\mathit{rank}_n\left(\boldsymbol{\mathcal{X}}\right) \leq \mathit{rank}\left(\boldsymbol{\mathcal{X}}\right) \tag{2.43}$$

Proof. From the definition of the tensor rank, we can always write $\boldsymbol{\mathcal{X}}$ as

$$\boldsymbol{\mathcal{X}} = \mathbf{U}_1 \circ \mathbf{U}_2 \circ \cdots \circ \mathbf{U}_N \tag{2.44}$$

where $\mathbf{U}_n \in \mathbb{R}^{I_n \times \mathrm{rank}(\boldsymbol{\mathcal{X}})}$. Therefore, $\forall n \in \{1, 2, \dots, N\}$ we have

$$\mathrm{rank}_n\left(\boldsymbol{\mathcal{X}}\right) \leq \mathrm{rank}\left(\mathbf{U}_n\right) \leq \mathrm{rank}\left(\boldsymbol{\mathcal{X}}\right) \tag{2.45}$$

□

Theorem 2.7 (Upper bound of the n-rank based on n-mode product)**.** *Let* $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ *be a tensor which can be written as*

$$\boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{V}} \times_{i=1}^{N} \mathbf{U}_i \tag{2.46}$$

where $\boldsymbol{\mathcal{V}} \in \mathbb{R}^{J_1 \times J_2 \times \cdots \times J_N}$ *and* $\mathbf{U}_n \in \mathbb{R}^{I_n \times J_n}$ *for* $n \in \{1, 2, \dots, N\}$*. Then, the following inequalities hold*

$$\mathit{rank}_n\left(\boldsymbol{\mathcal{X}}\right) \leq \mathit{rank}\left(\mathbf{U}_n\right) \tag{2.47}$$

$$\mathit{rank}_n\left(\boldsymbol{\mathcal{X}}\right) \leq \mathit{rank}_n\left(\boldsymbol{\mathcal{V}}\right) \tag{2.48}$$

Proof. For the first inequality we have

$$\begin{aligned}
\mathrm{rank}_n\left(\boldsymbol{\mathcal{X}}\right) &= \mathrm{rank}\left(\mathbf{X}_{[n]}\right) \\
&= \mathrm{rank}\left(\left(\boldsymbol{\mathcal{V}} \times_{i=1}^{N} \mathbf{U}_i\right)_{[n]}\right) \\
&= \mathrm{rank}\left(\mathbf{U}_n \mathbf{V}_{[n]} \left(\bigotimes_{\substack{i=N \\ i \neq n}}^{1} \mathbf{U}_i\right)^T\right) \\
&\leq \mathrm{rank}\left(\mathbf{U}_n\right)
\end{aligned} \tag{2.49}$$

Similarly, for the second inequality we have

$$\begin{aligned}
\mathrm{rank}_n\left(\boldsymbol{\mathcal{X}}\right) &= \mathrm{rank}\left(\mathbf{X}_{[n]}\right) \\
&= \mathrm{rank}\left(\left(\boldsymbol{\mathcal{V}} \times_{i=1}^{N} \mathbf{U}_i\right)_{[n]}\right) \\
&= \mathrm{rank}\left(\mathbf{U}_n \mathbf{V}_{[n]} \left(\bigotimes_{\substack{i=N \\ i \neq n}}^{1} \mathbf{U}_i\right)^T\right) \\
&\leq \mathrm{rank}\left(\mathbf{V}_{[n]} \left(\bigotimes_{\substack{i=N \\ i \neq n}}^{1} \mathbf{U}_i\right)^T\right) \\
&\leq \mathrm{rank}\left(\mathbf{V}_{[n]}\right) \\
&= \mathrm{rank}_n\left(\boldsymbol{\mathcal{V}}\right)
\end{aligned} \tag{2.50}$$

□

2.3 Optimisation Methods

2.3.1 Proximal Operators

In this thesis, *proximal operators* will play a major role in optimisation algorithms as building blocks, therefore we shall discuss them in detail here. For a thorough discussion on proximal operators, their properties and their applications, the reader may refer to [70].

Definition 2.20 (Proximal operator)**.** *Let $f : \mathbb{R}^n \to \mathbb{R}$ be a real function. The proximal operator of f at $\mathbf{x} \in \mathbb{R}^n$ is defined as*

$$\mathcal{P}_f\left\{\mathbf{x}\right\} = \arg\min_{\mathbf{y}} f\left(\mathbf{y}\right) + \frac{1}{2}\left\|\mathbf{x}-\mathbf{y}\right\|_2^2 \tag{2.51}$$

provided that the solution to the above optimisation problem exists and it is unique.

In a way, $\mathcal{P}_f\left\{\mathbf{x}\right\}$ is computed by trying to minimise f without getting too far from $\mathbf{x}$ (with the distance being measured by the ℓ_2-norm). Depending on the form of f, $\mathcal{P}_f\left\{\cdot\right\}$ may or may not have a closed form solution. In the following, we shall discuss $\mathcal{P}_f\left\{\cdot\right\}$ for certain cases of f which will be used in the optimisation algorithms discussed later in this thesis.

Theorem 2.8 (Shrinkage operator)**.** *If $x \in \mathbb{R}$ and $f\left(x\right) = a\left|x\right|$ with $a > 0$, the proximal operator becomes*

$$\mathcal{P}_f\left\{x\right\} = \begin{cases} 0 & -a \le x \le a \\ x-a & x > a \\ x+a & x < -a \end{cases} \tag{2.52}$$

The above proximal operator is known as the shrinkage operator *and is represented by $\mathcal{S}_a\left\{x\right\}$.*

Proof. The objective function becomes $h\left(y\right) = a\left|y\right| + \frac{1}{2}\left(x-y\right)^2$. We have the following two cases.

(i) $y \ge 0$. In this case, $h\left(y\right) = ay + \frac{1}{2}\left(x-y\right)^2$ and $h'\left(y\right) = a - x + y$. Studying the monotony of h, we get that h is decreasing in $[0, x-a]$ and increasing in $[x-a, +\infty)$. Therefore

$$y_+ = \arg\min_{y\ge 0} h\left(y\right) = \begin{cases} 0 & x \le a \\ x-a & x > a \end{cases} \tag{2.53}$$

(ii) $y < 0$. In this case, $h\left(y\right) = -ay + \frac{1}{2}\left(x-y\right)^2$ and $h'\left(y\right) = -a - x + y$. Studying the monotony of h, we get that h is decreasing in $(-\infty, x+a]$ and increasing in $[x+a, 0]$. Therefore

$$y_- = \arg\min_{y<0} h\left(y\right) = \begin{cases} x+a & x < -a \\ 0 & x \ge -a \end{cases} \tag{2.54}$$

Finally, by simply evaluating h at y_+ and y_-, we get that the shrinkage operator is given by

$$\mathcal{S}_a\left\{x\right\} = \arg\min_{y\in\{y_+, y_-\}} h\left(y\right) = \begin{cases} 0 & -a \le x \le a \\ x-a & x > a \\ x+a & x < -a \end{cases} \tag{2.55}$$

□

The shrinkage operator extends naturally to vectors, matrices and, in general, tensors, as the following theorem states.

Theorem 2.9 (Shrinkage operator for tensors)**.** *Let $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ be an N^{th}-order tensor and $f(\boldsymbol{\mathcal{X}}) = a \|\boldsymbol{\mathcal{X}}\|_1$ with $a > 0$. Then, the proximal operator reduces to elementwise application of the shrinkage operator*

$$\left(\mathcal{P}_f\{\boldsymbol{\mathcal{X}}\}\right)_{i_1 i_2 \cdots i_N} = \mathcal{S}_a\{X_{i_1 i_2 \cdots i_N}\} \tag{2.56}$$

For convenience, we will represent $\mathcal{P}_f\{\boldsymbol{\mathcal{X}}\}$ as $\mathcal{S}_a\{\boldsymbol{\mathcal{X}}\}$ and we will imply that $\mathcal{S}_a\{\cdot\}$ is applied elementwise.

Proof. The proximal operator can be written as

$$\begin{aligned}\mathcal{P}_f\{\boldsymbol{\mathcal{X}}\} &= \arg\min_{\boldsymbol{\mathcal{Y}}} a\|\boldsymbol{\mathcal{Y}}\|_1 + \frac{1}{2}\|\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{Y}}\|_F^2 \\ &= \arg\min_{\boldsymbol{\mathcal{Y}}} \sum_{i_i=1}^{I_1}\sum_{i_2=1}^{I_2}\cdots\sum_{i_N=1}^{I_N}\left(a|Y_{i_1 i_2\cdots i_N}| + \frac{1}{2}\left(X_{i_1 i_2 \cdots i_N} - Y_{i_1 i_2\cdots i_N}\right)^2\right)\end{aligned} \tag{2.57}$$

Since the objective function is separable in the elements of $\boldsymbol{\mathcal{Y}}$, we have

$$\begin{aligned}\left(\mathcal{P}_f\{\boldsymbol{\mathcal{X}}\}\right)_{i_1 i_2\cdots i_N} &= \arg\min_{Y_{i_1 i_2\cdots i_N}} a|Y_{i_1 i_2\cdots i_N}| + \frac{1}{2}\left(X_{i_1 i_2\cdots i_N} - Y_{i_1 i_2 \cdots i_N}\right)^2 \\ &= \mathcal{S}_a\{X_{i_1 i_2\cdots i_N}\}\end{aligned} \tag{2.58}$$

□

Finally, the following theorem describes the proximal operator when f becomes the nuclear norm.

Theorem 2.10 (Singular value thresholding)**.** *Let $\mathbf{X} \in \mathbb{R}^{m\times n}$ be a matrix of size $m \times n$ and $f(\mathbf{X}) = a\|\mathbf{X}\|_*$ with $a > 0$. Assume the Singular Value Decomposition $\mathbf{X} = \mathbf{U}\mathbf{S}\mathbf{V}^T$. Then, the proximal operator becomes*

$$\mathcal{P}_f\{\mathbf{X}\} = \mathbf{U}\mathcal{S}_a\{\mathbf{S}\}\mathbf{V}^T \tag{2.59}$$

We shall refer to the above proximal operator as singular value thresholding *and represent it by $\mathcal{D}_a\{\mathbf{X}\}$.*

Proof. This is Theorem 2.1 in [16]. See proof therein. □

2.3.2 Accelerated Proximal Gradient

Consider the following unconstrained optimisation problem

$$\min_{\mathbf{x}} g(\mathbf{x}) + f(\mathbf{x}) \tag{2.60}$$

where $f : \mathbb{R}^n \to \mathbb{R}$, $g : \mathbb{R}^n \to \mathbb{R}$ and f is differentiable in $\mathbb{R}^n$. We allow g to be possibly non-smooth, therefore techniques from smooth optimisation may not be always applicable. We will describe a method for solving the above optimisation problem known as *Proximal Gradient*.

Proximal Gradient is an iterative first-order optimisation method, which is based on replacing the differentiable part of the objective, $f(\mathbf{x})$, by its following quadratic approximation around some fixed point $\mathbf{y} \in \mathbb{R}$

$$Q_n(\mathbf{x}, \mathbf{y}) = f(\mathbf{y}) + \langle\nabla f(\mathbf{y}), \mathbf{x} - \mathbf{y}\rangle + \frac{n}{2}\|\mathbf{x} - \mathbf{y}\|_2^2 \tag{2.61}$$

where $n > 0$ is a carefully chosen positive parameter. We will describe two versions of Proximal Gradient, the classic version which can be viewed as a descent algorithm and its accelerated version which makes use of Nesterov's acceleration method.

Proximal Gradient as Descent Algorithm

To justify the usage of Q_n and the selection of n, we will make use of the following results.

Definition 2.21 (Lipschitz continuous function)**.** *A function $f : \mathbb{R}^n \to \mathbb{R}^m$ is called Lipschitz continuous if and only if there exists some non-negative constant $L \geq 0$ such that $\forall\, \mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ the following holds*[5]

$$\|f(\mathbf{x}) - f(\mathbf{y})\|_2 \leq L \|\mathbf{x} - \mathbf{y}\|_2 \tag{2.62}$$

The smallest such L is called the Lipschitz constant *of f.*

Theorem 2.11 (Quadratic approximation as upper bound)**.** *Let $f : \mathbb{R}^n \to \mathbb{R}$ be a real function, whose gradient ∇f is Lipschitz continuous with a Lipschitz constant $L \geq 0$. Then, for $n \geq L$ the following inequality holds $\forall\, \mathbf{x}, \mathbf{y} \in \mathbb{R}^n$*

$$f(\mathbf{x}) \leq f(\mathbf{y}) + \langle \nabla f(\mathbf{y}), \mathbf{x} - \mathbf{y} \rangle + \frac{n}{2} \|\mathbf{x} - \mathbf{y}\|_2^2 \tag{2.63}$$

Proof. It suffices to prove the inequality for $n = L$. Since ∇f is a conservative vector field, for any line integral from $\mathbf{y}$ to $\mathbf{x}$ we have

$$f(\mathbf{x}) - f(\mathbf{y}) = \int_{\mathbf{y}}^{\mathbf{x}} \langle \nabla f(\mathbf{t}), \mathrm{d}\mathbf{t} \rangle \tag{2.64}$$

If we parameterise $\mathbf{t}$ to follow the line segment from $\mathbf{y}$ to $\mathbf{x}$ we have

$$\int_{\mathbf{y}}^{\mathbf{x}} \langle \nabla f(\mathbf{t}), \mathrm{d}\mathbf{t} \rangle = \int_0^1 \langle \nabla f(\theta \mathbf{x} + (1-\theta)\mathbf{y}), \mathbf{x} - \mathbf{y} \rangle \,\mathrm{d}\theta \tag{2.65}$$

We then have the following

$$\begin{aligned}
f(\mathbf{x}) - f(\mathbf{y}) - \langle \nabla f(\mathbf{y}), \mathbf{x} - \mathbf{y} \rangle &= \int_0^1 \langle \nabla f(\theta \mathbf{x} + (1-\theta)\mathbf{y}) - \nabla f(\mathbf{y}), \mathbf{x} - \mathbf{y} \rangle \,\mathrm{d}\theta \\
&\leq \|\mathbf{x} - \mathbf{y}\|_2 \int_0^1 \|\nabla f(\theta \mathbf{x} + (1-\theta)\mathbf{y}) - \nabla f(\mathbf{y})\|_2 \,\mathrm{d}\theta \\
&\leq L \|\mathbf{x} - \mathbf{y}\|_2 \int_0^1 \|\theta \mathbf{x} + (1-\theta)\mathbf{y} - \mathbf{y}\|_2 \,\mathrm{d}\theta \\
&= L \|\mathbf{x} - \mathbf{y}\|_2^2 \int_0^1 \theta \,\mathrm{d}\theta \\
&= \frac{L}{2} \|\mathbf{x} - \mathbf{y}\|_2^2
\end{aligned} \tag{2.66}$$

which concludes the proof. □

The above theorem says that if ∇f is Lipschitz continuous then $Q_n(\mathbf{x}, \mathbf{y})$ is an upper bound on $f(\mathbf{x})$, as long as n is no less than the Lipschitz constant of ∇f. Based on the above, Proximal Gradient proceeds by computing the following at each iteration $k \in \{0, 1, 2, \ldots\}$

$$\begin{aligned}
\mathbf{x}_{k+1} &= \arg\min_{\mathbf{x}} g(\mathbf{x}) + Q_n(\mathbf{x}, \mathbf{y}_k) \\
&= \arg\min_{\mathbf{x}} g(\mathbf{x}) + \langle \nabla f(\mathbf{y}_k), \mathbf{x} - \mathbf{y}_k \rangle + \frac{n}{2} \|\mathbf{x} - \mathbf{y}_k\|_2^2 \\
&= \arg\min_{\mathbf{x}} n^{-1} g(\mathbf{x}) + \frac{1}{2} \left\|\mathbf{x} - \left(\mathbf{y}_k - n^{-1} \nabla f(\mathbf{y}_k)\right)\right\|_2^2 \\
&= \mathcal{P}_{(n^{-1} g)} \left\{\mathbf{y}_k - n^{-1} \nabla f(\mathbf{y}_k)\right\}
\end{aligned} \tag{2.67}$$

[5]Here the definition is given in terms of the ℓ_2-norm, however in general any norms on $\mathbb{R}^n$ and $\mathbb{R}^m$ (even different ones) can be used instead.

The above is the proximal operator of function $n^{-1}g$, which, as we have seen in section 2.3.1, it is often easy to compute. It is easy to see that if we choose $\mathbf{y}_k = \mathbf{x}_k$ then the above can be interpreted as a descent algorithm, as the following theorem states.

Theorem 2.12 (Proximal Gradient as descent algorithm)**.** *Let $f : \mathbb{R}^n \to \mathbb{R}$, $g : \mathbb{R}^n \to \mathbb{R}$ and f be differentiable with a Lipschitz continuous gradient. If $\mathbf{x}_{k+1} = \arg\min_{\mathbf{x}} g(\mathbf{x}) + Q_n(\mathbf{x}, \mathbf{x}_k)$ with n no less than the Lipschitz constant of ∇f, then the following holds*

$$g(\mathbf{x}_{k+1}) + f(\mathbf{x}_{k+1}) \leq g(\mathbf{x}_k) + f(\mathbf{x}_k) \tag{2.68}$$

Proof. By Theorem 2.11 we have

$$g(\mathbf{x}_{k+1}) + f(\mathbf{x}_{k+1}) \leq g(\mathbf{x}_{k+1}) + Q_n(\mathbf{x}_{k+1}, \mathbf{x}_k) \tag{2.69}$$

By the definition of $\mathbf{x}_{k+1}$ we have

$$g(\mathbf{x}_{k+1}) + Q_n(\mathbf{x}_{k+1}, \mathbf{x}_k) \leq g(\mathbf{x}_k) + Q_n(\mathbf{x}_k, \mathbf{x}_k) \tag{2.70}$$

Finally, by the definition of Q_n we have

$$Q_n(\mathbf{x}_k, \mathbf{x}_k) = f(\mathbf{x}_k) \tag{2.71}$$

which concludes the proof. □

Proximal Gradient is fully described in Algorithm 2.1. It can be shown [7] that this algorithm converges with a rate of $\mathcal{O}\left(k^{-1}\right)$ in the sense that

$$[g(\mathbf{x}_k) + f(\mathbf{x}_k)] - [g(\mathbf{x}^*) + f(\mathbf{x}^*)] \in \mathcal{O}\left(k^{-1}\right) \tag{2.72}$$

where $\mathbf{x}^*$ is the true minimiser and k is the iteration number. Finally, it is interesting to see that in the special case where $\forall \mathbf{x} \in \mathbb{R}^n$ we have $g(\mathbf{x}) = 0$, the proximal operator $\mathcal{P}_{(n^{-1}g)}\{\cdot\}$ reduces to the identity operator and therefore Algorithm 2.1 reduces to *gradient descent* [11]. In this sense, Proximal Gradient can be viewed as a generalisation of gradient descent.

Algorithm 2.1: Proximal Gradient

```
  Initialise: $\mathbf{x}_0$, $k = 0$
1 while not converged do
2     $\mathbf{x}_{k+1} = \mathcal{P}_{(n^{-1}g)}\left\{\mathbf{x}_k - n^{-1}\nabla f(\mathbf{x}_k)\right\}$
3     $k \leftarrow k + 1$
4 end while
  Output: $\mathbf{x}_k$
```

Nesterov's Acceleration Method

As we have seen, Proximal Gradient in its classic form has a convergence rate of $\mathcal{O}\left(k^{-1}\right)$. It has been shown that it is possible to improve this, by simply selecting points $\mathbf{y}_k$, around which the quadratic approximation $Q_n(\mathbf{x}, \mathbf{y}_k)$ is made, in a more efficient way. Nesterov [64] showed that in the special case where $\forall \mathbf{x} \in \mathbb{R}^n$ we have $g(\mathbf{x}) = 0$ (i.e. in the case of gradient descent) the following selection of points $\mathbf{y}_k$ can improve the convergence rate to $\mathcal{O}\left(k^{-2}\right)$

$$\mathbf{y}_k = \mathbf{x}_k + \frac{t_{k-1} - 1}{t_k}(\mathbf{x}_k - \mathbf{x}_{k-1}) \tag{2.73}$$

as long as the sequence $\{t_k\}$ satisfies $t_{k+1}^2 - t_{k+1} \leq t_k^2$. Beck and Teboulle [7] extended the above convergence result to a general non-smooth function g.

The *Accelerated Proximal Gradient* method incorporating the above scheme is described in algorithm 2.2. It can be seen that the computational effort per iteration is not significantly greater than in its classic form, while its convergence rate is superior. For this reason, Accelerated Proximal Gradient is usually preferred in practice than classic Proximal Gradient.

Algorithm 2.2: Accelerated Proximal Gradient

```
   Initialise: $\mathbf{x}_0$, $\mathbf{x}_{-1}$, $t_0 = t_{-1} = 1$, $k = 0$
1  while not converged do
2      $\mathbf{y}_k = \mathbf{x}_k + \frac{t_{k-1}-1}{t_k}\left(\mathbf{x}_k - \mathbf{x}_{k-1}\right)$
3      $\mathbf{x}_{k+1} = \mathcal{P}_{(n^{-1}g)}\left\{\mathbf{y}_k - n^{-1}\nabla f\left(\mathbf{y}_k\right)\right\}$
4      $t_{k+1} = \frac{1+\sqrt{1+4t_k^2}}{2}$
5      $k \leftarrow k + 1$
6  end while
   Output: $\mathbf{x}_k$
```

2.3.3 Method of Augmented Lagrange Multipliers

Consider the following equality-constrained optimisation problem

$$\min_{\mathbf{x}} f\left(\mathbf{x}\right) \quad \text{s.t.} \quad \mathbf{h}\left(\mathbf{x}\right) = \mathbf{0} \tag{2.74}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ and $\mathbf{h} : \mathbb{R}^n \to \mathbb{R}^m$.

A common approach towards solving such problems is relaxing them to an unconstrained form, such as the following

$$\min_{\mathbf{x}} f\left(\mathbf{x}\right) + \frac{\mu}{2}\left\|\mathbf{h}\left(\mathbf{x}\right)\right\|_2^2 \tag{2.75}$$

Approaches of this type are known as *penalty methods* [59]. Here, $\mu > 0$ is called the *penalty parameter* and as $\mu \to +\infty$ the relaxed problem becomes equivalent to the original constrained problem. Typically, penalty methods proceed with the following updates

$$\mathbf{x}_{k+1} = \arg\min_{\mathbf{x}} f\left(\mathbf{x}\right) + \frac{\mu_k}{2}\left\|\mathbf{h}\left(\mathbf{x}\right)\right\|_2^2 \tag{2.76}$$

$$\mu_{k+1} = \rho\mu_k \tag{2.77}$$

where $\rho > 1$. However, a drawback of penalty methods is that they typically require μ to attain very large values, which can potentially make the problem ill-conditioned.

Another approach is transforming the original problem to its dual form. The *Lagrangian* of the original problem can be written as follows

$$\mathcal{L}\left(\mathbf{x}, \mathbf{y}\right) = f\left(\mathbf{x}\right) + \left\langle \mathbf{h}\left(\mathbf{x}\right), \mathbf{y}\right\rangle \tag{2.78}$$

where $\mathbf{y} \in \mathbb{R}^m$ is the vector of *Lagrange multipliers*. The so-called *Lagrangian methods* [59] attempt to find a saddle point of $\mathcal{L}$, using the following updates

$$\mathbf{x}_{k+1} = \arg\min_{\mathbf{x}} \mathcal{L}\left(\mathbf{x}, \mathbf{y}_k\right) \tag{2.79}$$

$$\mathbf{y}_{k+1} = \mathbf{y}_k + a_k\,\mathbf{h}\left(\mathbf{x}_{k+1}\right) \tag{2.80}$$

In the update for $\mathbf{y}$, $a_k \geq 0$ is known as the *step size*. A drawback of such approaches is that they require tuning for the right step size a_k in each iteration.

The method of *Augmented Lagrange Multipliers*, or simply ALM, combines both approaches. The method starts by forming the following *augmented Lagrangian*

$$\mathcal{L}_\mu\left(\mathbf{x}, \mathbf{y}\right) = f\left(\mathbf{x}\right) + \left\langle \mathbf{h}\left(\mathbf{x}\right), \mathbf{y}\right\rangle + \frac{\mu}{2}\left\|\mathbf{h}\left(\mathbf{x}\right)\right\|_2^2 \tag{2.81}$$

where $\mathbf{y} \in \mathbb{R}^m$ is the vector of Lagrange multipliers and $\mu > 0$ is the penalty parameter. The objective is finding a saddle point of $\mathcal{L}$, while at the same time increasing μ so as to enforce the constraint. The method is fully described as Algorithm 2.3.

Algorithm 2.3: Method of Augmented Lagrange Multipliers

```
Initialise: $\mathbf{x}_0$, $\mathbf{y}_0$, $\mu_0 > 0$, $k = 0$
1 while not converged do
2     $\mathbf{x}_{k+1} = \arg\min_{\mathbf{x}} \mathcal{L}_{\mu_k}(\mathbf{x}, \mathbf{y}_k)$
3     $\mathbf{y}_{k+1} = \mathbf{y}_k + \mu_k \mathbf{h}(\mathbf{x}_{k+1})$
4     $\mu_{k+1} = \min(\rho\mu_k, \mu_{max})$
5     $k \leftarrow k + 1$
6 end while
Output: $\mathbf{x}_k$
```

The ALM method combines the merits of both penalty and Lagrangian methods while avoiding some of their drawbacks. For instance, ALM does not require $\mu \to +\infty$ in order to converge, thus avoiding ill-conditioning. The only requirement is that sequence $\{\mu_k\}$ is non-decreasing. In Algorithm 2.3, μ is geometrically increased up to a preset upper limit μ_{max}. Furthermore, in the update for $\mathbf{y}$, the optimal step size is known to be the penalty parameter μ, therefore tuning is not required. For more information on ALM and its convergence, the reader may refer to [10, 11, 13].

Chapter 3

Robust Low-Rank Modelling on Matrices

Let $\mathbf{X} \in \mathbb{R}^{m\times n}$ be a data matrix, containing in its columns the observed data points. Our aim is to decompose $\mathbf{X}$ into a *low-rank component* $\mathbf{A} \in \mathbb{R}^{m\times n}$ and a *sparse component* $\mathbf{E} \in \mathbb{R}^{m\times n}$, such that

$$\mathbf{X} = \mathbf{A} + \mathbf{E} \tag{3.1}$$

In order to do this, from all possible decomposition of the above form, we seek to find the one which minimises the *rank* of $\mathbf{A}$ and maximises the *sparsity* of $\mathbf{E}$. As we have seen in section 2.1, the rank of $\mathbf{A}$ is quantified by the number of linearly independent columns and is represented by $\operatorname{rank}(\mathbf{A})$ and the sparsity of $\mathbf{E}$ is quantified by the number of non-zero elements and is represented by $\|\mathbf{E}\|_0$.[1] Based on the above, in order to find the desired decomposition, we need to solve the following optimisation problem

$$\min_{\mathbf{A},\mathbf{E}} \operatorname{rank}(\mathbf{A}) + \lambda \|\mathbf{E}\|_0 \quad \text{s.t.} \quad \mathbf{X} = \mathbf{A} + \mathbf{E} \tag{3.2}$$

where $\lambda > 0$ is a positive regulariser that determines the significance of minimising $\|\mathbf{E}\|_0$ compared to minimising $\operatorname{rank}(\mathbf{A})$. That is, for a larger λ the optimal solution will contain a sparser $\mathbf{E}$ and a less low-rank $\mathbf{A}$ whereas a smaller λ will result to a denser $\mathbf{E}$ and a lower-rank $\mathbf{A}$.

Due to the discrete nature of the rank and the ℓ_0-norm, the above problem is an NP-hard combinatorial problem. In order to address this, most approaches attempt to replace the rank and the ℓ_0-norm with appropriate *approximant functions* that are easier to minimise. We shall represent such approximants of the rank and the ℓ_0-norm by $r(\cdot)$ and $s(\cdot)$ respectively. In this case, the original NP-hard optimisation problem is relaxed to the following

$$\min_{\mathbf{A},\mathbf{E}} r(\mathbf{A}) + \lambda\, s(\mathbf{E}) \quad \text{s.t.} \quad \mathbf{X} = \mathbf{A} + \mathbf{E} \tag{3.3}$$

Depending on the choice of $r(\cdot)$ and $s(\cdot)$, different instantiations of the above problem can be produced. In this chapter, we will examine the optimisation problems that result from various choices for $r(\cdot)$ and $s(\cdot)$ and we will describe the algorithms that have been developed in the literature for solving them. We will conclude the chapter with a discussion over the advantages and disadvantages of each approach and an analysis of their computational cost.

3.1 Robust Principal Component Analysis

An instance of problem (3.3) involves choosing $r(\cdot)$ and $s(\cdot)$ to be the *convex envelopes* of the rank and the ℓ_0-norm respectively. The convex envelope of some function f is the largest convex

[1] To be precise, $\|\mathbf{E}\|_0$ quantifies the *non-sparsity* of $\mathbf{E}$, as the larger $\|\mathbf{E}\|_0$ is, the less sparse $\mathbf{E}$ is.

function which is pointwise less than f. As we have seen in section 2.1, the convex envelopes of the rank and the ℓ_0-norm are the nuclear norm and the ℓ_1-norm respectively. Under these substitutions, problem 3.3 becomes as follows

$$\min_{\mathbf{A},\mathbf{E}} \|\mathbf{A}\|_* + \lambda \|\mathbf{E}\|_1 \quad \text{s.t.} \quad \mathbf{X} = \mathbf{A} + \mathbf{E} \tag{3.4}$$

We shall refer to the above problem as *Robust Principal Component Analysis.*[2] RPCA has the advantage of being a convex problem, which means that it can easily be *globally* optimised. Perhaps the most interesting fact about RPCA, though, is that, under some rather weak conditions, the solution of problem (3.4) is, with overwhelming probability, the same as the solution of the original NP-hard optimisation problem (3.2). This result was proved by Candès et al. [18] and it practically means that, in most cases, RPCA is theoretically *guaranteed* to yield the correct solution to the problem.

Since RPCA is a convex problem, convex optimisation methods such as interior point methods [59] can in principle be used to find the (global) solution. However, as it is indicated in [18, 52], interior point methods use second-order information and therefore, even though they have a superior rate of convergence, computing the step direction is expensive. In fact, for an input matrix $\mathbf{X} \in \mathbb{R}^{n\times n}$ each step takes $\mathcal{O}\left(n^6\right)$ to compute, which limits interior point methods to matrices of size $n < 10^2$ on a typical desktop computer. In contrast, applications in computer vision, web search or bioinformatics often require matrices of size $n > 10^4$, making interior point methods impractical for many real-life applications.

To cope with the scalability requirements, recent approaches focus on first-order optimisation methods instead. In the following sections, we describe the two state-of-the-art methods for RPCA, which are based on Accelerated Proximal Gradient and the method of Augmented Lagrange Multipliers.

3.1.1 Solution Based on Accelerated Proximal Gradient

The APG algorithm for RPCA was first suggested in [51, 84]. As we have seen in section 2.3.2, APG is primarily a method for unconstrained optimisation. Therefore, in order to apply APG to RPCA, the RPCA problem is relaxed to the following unconstrained form

$$\min_{\mathbf{A},\mathbf{E}} \|\mathbf{A}\|_* + \lambda \|\mathbf{E}\|_1 + \frac{1}{2\mu} \|\mathbf{X} - \mathbf{A} - \mathbf{E}\|_F^2 \tag{3.5}$$

or equivalently

$$\min_{\mathbf{A},\mathbf{E}} \mu \|\mathbf{A}\|_* + \lambda\mu \|\mathbf{E}\|_1 + \frac{1}{2} \|\mathbf{X} - \mathbf{A} - \mathbf{E}\|_F^2 \tag{3.6}$$

where $\mu > 0$. Notice that as $\mu \to 0$ the unconstrained relaxation becomes equivalent to the original constrained problem. If we write the above minimisation separately for $\mathbf{A}$ and for $\mathbf{E}$, we get the following two problems

$$\min_{\mathbf{A}} \mu \|\mathbf{A}\|_* + \frac{1}{2} \|\mathbf{X} - \mathbf{A} - \mathbf{E}\|_F^2 \tag{3.7}$$

$$\min_{\mathbf{E}} \lambda\mu \|\mathbf{E}\|_1 + \frac{1}{2} \|\mathbf{X} - \mathbf{A} - \mathbf{E}\|_F^2 \tag{3.8}$$

In both cases, the smooth term is $\frac{1}{2} \|\mathbf{X} - \mathbf{A} - \mathbf{E}\|_F^2$, whose gradients with respect to $\mathbf{A}$ and $\mathbf{E}$ are

$$\nabla_{\mathbf{A}} \left(\frac{1}{2} \|\mathbf{X} - \mathbf{A} - \mathbf{E}\|_F^2\right) = \nabla_{\mathbf{E}} \left(\frac{1}{2} \|\mathbf{X} - \mathbf{A} - \mathbf{E}\|_F^2\right) = \mathbf{A} + \mathbf{E} - \mathbf{X} \tag{3.9}$$

It is easy to see that the Lipschitz constant of the above gradients is $L = 1$.

[2] In [18] it is also referred to as *Principal Component Pursuit.*

Based on the above, APG for RPCA is presented in detail in Algorithm 3.1. The proximal parameter n is typically set to $n = 2$. Note that a continuation scheme is used for μ, i.e. μ is initialised with a typically large value and at each iteration it is geometrically decreased by $\rho < 1$ down to a lower limit μ_{min}. This continuation scheme has been found [51, 84] to accelerate convergence.

Algorithm 3.1: Matrix Robust PCA via APG

```
  Input: Data matrix $\mathbf{X}$, regulariser $\lambda > 0$
  Initialise: $\mathbf{A}_0 = \mathbf{A}_{-1} = \mathbf{0}$, $\mathbf{E}_0 = \mathbf{E}_{-1} = \mathbf{0}$, $\mu_0 > 0$, $t_0 = t_{-1} = 1$, $k = 0$
1 while not converged do
2     $\mathbf{Y}_k^A = \mathbf{A}_k + \frac{t_{k-1}-1}{t_k}\left(\mathbf{A}_k - \mathbf{A}_{k-1}\right)$
3     $\mathbf{Y}_k^E = \mathbf{E}_k + \frac{t_{k-1}-1}{t_k}\left(\mathbf{E}_k - \mathbf{E}_{k-1}\right)$
4     $\mathbf{A}_{k+1} = \mathcal{D}_{\mu_k n^{-1}}\left\{\mathbf{Y}_k^A - n^{-1}\left(\mathbf{Y}_k^A + \mathbf{Y}_k^E - \mathbf{X}\right)\right\}$
5     $\mathbf{E}_{k+1} = \mathcal{S}_{\lambda\mu_k n^{-1}}\left\{\mathbf{Y}_k^E - n^{-1}\left(\mathbf{Y}_k^A + \mathbf{Y}_k^E - \mathbf{X}\right)\right\}$
6     $\mu_{k+1} = \max\left(\rho\mu_k, \mu_{min}\right)$
7     $t_{k+1} = \frac{1+\sqrt{1+4t_k^2}}{2}$
8     $k \leftarrow k + 1$
9 end while
  Output: Low-rank component $\mathbf{A}_k$, sparse component $\mathbf{E}_k$
```

Recently, a generalisation of the above algorithm was introduced by He et al. [38] by replacing problem (3.8) by the following generalised version instead

$$\phi\left(\mathbf{X} - \mathbf{A}\right) = \min_{\mathbf{E}} \varphi\left(\mathbf{E}\right) + \frac{1}{2}\left\|\mathbf{X} - \mathbf{A} - \mathbf{E}\right\|_F^2 \tag{3.10}$$

Here, $\varphi(\cdot)$ is chosen such that $\phi(\cdot)$ be a robust *M-estimator*, which allows the method to be robust to outliers. In this case, the proximal operator $\mathcal{P}_\phi\{\cdot\}$ is equal to the *minimiser function* of the M-estimator $\phi(\cdot)$ and typically has a closed algebraic form. The only difference from Algorithm 3.1 is that, in the update of $\mathbf{E}$, the shrinkage operator $\mathcal{S}_{\lambda\mu n^{-1}}\{\cdot\}$ is replaced by the more general $\mathcal{P}_\phi\{\cdot\}$. The reader may refer to [38] for further details.

3.1.2 Solution Based on Augmented Lagrange Multipliers

The ALM algorithm for RPCA was introduced in [52, 87] and was subsequently used in [18]. The augmented Lagrangian of the RPCA problem can be written as follows

$$\mathcal{L}_\mu\left(\mathbf{A}, \mathbf{E}, \mathbf{Y}\right) = \left\|\mathbf{A}\right\|_* + \lambda\left\|\mathbf{E}\right\|_1 + \left\langle \mathbf{X} - \mathbf{A} - \mathbf{E}, \mathbf{Y}\right\rangle + \frac{\mu}{2}\left\|\mathbf{X} - \mathbf{A} - \mathbf{E}\right\|_F^2 \tag{3.11}$$

where $\mathbf{Y}$ is the matrix of Lagrange multipliers and $\mu > 0$ is the penalty parameter. At each iteration, ALM requires the minimisation of $\mathcal{L}_\mu$ with respect to $\mathbf{A}$ and $\mathbf{E}$. Keeping $\mathbf{E}$ fixed, the minimisation with respect to $\mathbf{A}$ is done as follows

$$\begin{aligned}
&\arg\min_{\mathbf{A}} \mathcal{L}_\mu\left(\mathbf{A}, \mathbf{E}, \mathbf{Y}\right) \\
&= \arg\min_{\mathbf{A}} \left\|\mathbf{A}\right\|_* + \left\langle \mathbf{X} - \mathbf{A} - \mathbf{E}, \mathbf{Y}\right\rangle + \frac{\mu}{2}\left\|\mathbf{X} - \mathbf{A} - \mathbf{E}\right\|_F^2 \\
&= \arg\min_{\mathbf{A}} \mu^{-1}\left\|\mathbf{A}\right\|_* + \frac{1}{2}\left\|\mathbf{A} - \left(\mathbf{X} - \mathbf{E} + \mu^{-1}\mathbf{Y}\right)\right\|_F^2 \\
&= \mathcal{D}_{\mu^{-1}}\left\{\mathbf{X} - \mathbf{E} + \mu^{-1}\mathbf{Y}\right\}
\end{aligned} \tag{3.12}$$

Keeping **A** fixed, the minimisation with respect to **E** is done as follows

$$
\begin{aligned}
&\arg\min_{\mathbf{E}} \mathcal{L}_\mu(\mathbf{A},\mathbf{E},\mathbf{Y}) \\
&= \arg\min_{\mathbf{E}} \lambda \|\mathbf{E}\|_1 + \langle \mathbf{X}-\mathbf{A}-\mathbf{E},\mathbf{Y}\rangle + \frac{\mu}{2}\|\mathbf{X}-\mathbf{A}-\mathbf{E}\|_F^2 \\
&= \arg\min_{\mathbf{E}} \lambda\mu^{-1} \|\mathbf{E}\|_1 + \frac{1}{2}\left\|\mathbf{E}-\left(\mathbf{X}-\mathbf{A}+\mu^{-1}\mathbf{Y}\right)\right\|_F^2 \\
&= \mathcal{S}_{\lambda\mu^{-1}}\left\{\mathbf{X}-\mathbf{A}+\mu^{-1}\mathbf{Y}\right\}
\end{aligned}
\tag{3.13}
$$

The minimisation of $\mathcal{L}_\mu$ with respect to both **A** and **E** can be achieved by alternating between the two above steps until convergence. This has to be done in each iteration of ALM. In [52] this is referred to as *Exact ALM*. However, the authors show that running each step only once per ALM iteration is still sufficient for convergence, while requiring much less computation. They refer to this algorithm as *Inexact ALM*. We presented Inexact ALM in detail in Algorithm 3.2. For simplicity, for the remaining of this thesis, we shall refer to Inexact ALM simply as ALM.

Algorithm 3.2: Matrix Robust PCA via ALM

```
  Input: Data matrix $\mathbf{X}$, regulariser $\lambda > 0$
  Initialise: $\mathbf{E}_0 = \mathbf{0}$, $\mathbf{Y}_0 = \mathbf{0}$, $\mu_0 > 0$, $k = 0$
1 while not converged do
2     $\mathbf{A}_{k+1} = \mathcal{D}_{\mu_k^{-1}}\left\{\mathbf{X}-\mathbf{E}_k+\mu_k^{-1}\mathbf{Y}_k\right\}$
3     $\mathbf{E}_{k+1} = \mathcal{S}_{\lambda\mu_k^{-1}}\left\{\mathbf{X}-\mathbf{A}_{k+1}+\mu_k^{-1}\mathbf{Y}_k\right\}$
4     $\mathbf{Y}_{k+1} = \mathbf{Y}_k + \mu_k\left(\mathbf{X}-\mathbf{A}_{k+1}-\mathbf{E}_{k+1}\right)$
5     $\mu_{k+1} = \min\left(\rho\mu_k, \mu_{max}\right)$
6     $k \leftarrow k+1$
7 end while
  Output: Low-rank component $\mathbf{A}_k$, sparse component $\mathbf{E}_k$
```

3.2 Bilinear Robust Principal Component Analysis

Solving the RPCA optimisation problem, using either APG or ALM, requires at each iteration the computation of an SVD of size $m \times n$, for an input matrix $\mathbf{X} \in \mathbb{R}^{m\times n}$. This is a consequence of the usage of the nuclear norm and, thus, the application of the singular value thresholding operator. While this is acceptable for many cases, as **X** becomes larger or performance requirements become higher, an SVD per iteration becomes too computationally expensive.

To alleviate this problem, Cabral et al. [14] use ideas from bilinear factorisation in order to avoid the explicit usage of the nuclear norm and therefore remove the need for singular value thresholding. Their approach is based on the following theorem.

Theorem 3.1 (Factorisation-based formulation of the nuclear norm)**.** *Let* $\mathbf{A} \in \mathbb{R}^{m\times n}$,$\mathbf{U} \in \mathbb{R}^{m\times r}$ *and* $\mathbf{V} \in \mathbb{R}^{r\times n}$*, such that* $\mathbf{A} = \mathbf{UV}$*. The nuclear norm of* **A** *is the tightest lower bound of the average of the square Frobenius norms of* **U** *and* **V***. In other words*

$$
\|\mathbf{A}\|_* = \min_{\mathbf{A},\mathbf{U},\mathbf{V}} \frac{1}{2}\left(\|\mathbf{U}\|_F^2 + \|\mathbf{V}\|_F^2\right) \quad s.t. \quad \mathbf{A} = \mathbf{UV} \tag{3.14}
$$

Proof. This theorem is a special case of Lemma 5.1 in [73]. See proof therein. □

The above theorem says that $\frac{1}{2}\left(\|\mathbf{U}\|_F^2 + \|\mathbf{V}\|_F^2\right)$ is an attainable upper bound for $\|\mathbf{A}\|_*$, where $\mathbf{A} = \mathbf{UV}$. Using this fact, the RPCA optimisation problem (3.4) can be relaxed to the following

$$\min_{\mathbf{U},\mathbf{V},\mathbf{E}} \frac{1}{2}\left(\|\mathbf{U}\|_F^2 + \|\mathbf{V}\|_F^2\right) + \lambda\|\mathbf{E}\|_1 \quad \text{s.t.} \quad \mathbf{X} = \mathbf{U}\mathbf{V} + \mathbf{E} \tag{3.15}$$

We will refer to the above optimisation problem as *Bilinear Robust Principal Component Analysis.* Notice the absence of nuclear norm in the objective of BRPCA. Cabral et al. [14] developed an ALM algorithm for BRPCA, a slight variation of which we describe in the following.

The augmented Lagrangian of the BRPCA problem can be written as follows

$$\mathcal{L}_\mu(\mathbf{U},\mathbf{V},\mathbf{E},\mathbf{Y}) = \frac{1}{2}\left(\|\mathbf{U}\|_F^2 + \|\mathbf{V}\|_F^2\right) + \lambda\|\mathbf{E}\|_1 + \langle \mathbf{X}-\mathbf{U}\mathbf{V}-\mathbf{E}, \mathbf{Y}\rangle + \frac{\mu}{2}\|\mathbf{X}-\mathbf{U}\mathbf{V}-\mathbf{E}\|_F^2 \tag{3.16}$$

Similarly to Inexact ALM for RPCA, we will minimise $\mathcal{L}_\mu$ with respect to $\mathbf{U}$,$\mathbf{V}$ and $\mathbf{E}$ separately while keeping all other arguments fixed. The minimisation with respect to $\mathbf{U}$ becomes

$$\begin{aligned}
&\arg\min_{\mathbf{U}} \mathcal{L}_\mu(\mathbf{U},\mathbf{V},\mathbf{E},\mathbf{Y}) \\
&= \arg\min_{\mathbf{U}} \frac{1}{2}\|\mathbf{U}\|_F^2 + \langle \mathbf{X}-\mathbf{U}\mathbf{V}-\mathbf{E}, \mathbf{Y}\rangle + \frac{\mu}{2}\|\mathbf{X}-\mathbf{U}\mathbf{V}-\mathbf{E}\|_F^2
\end{aligned} \tag{3.17}$$

Since the above objective is differentiable, we can easily minimise it by forcing its derivative equal to zero. We have

$$\begin{aligned}
&\frac{\partial}{\partial \mathbf{U}}\left(\frac{1}{2}\|\mathbf{U}\|_F^2 + \langle \mathbf{X}-\mathbf{U}\mathbf{V}-\mathbf{E}, \mathbf{Y}\rangle + \frac{\mu}{2}\|\mathbf{X}-\mathbf{U}\mathbf{V}-\mathbf{E}\|_F^2\right) = \mathbf{0} && \Rightarrow \\
&\mathbf{U} - \mathbf{Y}\mathbf{V}^T - \mu(\mathbf{X}-\mathbf{U}\mathbf{V}-\mathbf{E})\mathbf{V}^T = \mathbf{0} && \Rightarrow \\
&\mathbf{U}\left(\mathbf{V}\mathbf{V}^T + \mu^{-1}\mathbf{I}\right) - \left(\mathbf{X}-\mathbf{E}+\mu^{-1}\mathbf{Y}\right)\mathbf{V}^T = \mathbf{0} && \Rightarrow \\
&\mathbf{U} = \left(\mathbf{X}-\mathbf{E}+\mu^{-1}\mathbf{Y}\right)\mathbf{V}^T\left(\mathbf{V}\mathbf{V}^T + \mu^{-1}\mathbf{I}\right)^{-1}
\end{aligned} \tag{3.18}$$

Next, we minimise $\mathcal{L}_\mu$ with respect to $\mathbf{V}$ as follows

$$\begin{aligned}
&\arg\min_{\mathbf{V}} \mathcal{L}_\mu(\mathbf{U},\mathbf{V},\mathbf{E},\mathbf{Y}) \\
&= \arg\min_{\mathbf{V}} \frac{1}{2}\|\mathbf{V}\|_F^2 + \langle \mathbf{X}-\mathbf{U}\mathbf{V}-\mathbf{E}, \mathbf{Y}\rangle + \frac{\mu}{2}\|\mathbf{X}-\mathbf{U}\mathbf{V}-\mathbf{E}\|_F^2
\end{aligned} \tag{3.19}$$

Similarly, we take the derivative of the above objective and force it equal to zero.

$$\begin{aligned}
&\frac{\partial}{\partial \mathbf{V}}\left(\frac{1}{2}\|\mathbf{V}\|_F^2 + \langle \mathbf{X}-\mathbf{U}\mathbf{V}-\mathbf{E}, \mathbf{Y}\rangle + \frac{\mu}{2}\|\mathbf{X}-\mathbf{U}\mathbf{V}-\mathbf{E}\|_F^2\right) = \mathbf{0} && \Rightarrow \\
&\mathbf{V} - \mathbf{U}^T\mathbf{Y} - \mu\mathbf{U}^T(\mathbf{X}-\mathbf{U}\mathbf{V}-\mathbf{E}) = \mathbf{0} && \Rightarrow \\
&\left(\mathbf{U}^T\mathbf{U} + \mu^{-1}\mathbf{I}\right)\mathbf{V} - \mathbf{U}^T\left(\mathbf{X}-\mathbf{E}+\mu^{-1}\mathbf{Y}\right) = \mathbf{0} && \Rightarrow \\
&\mathbf{V} = \left(\mathbf{U}^T\mathbf{U} + \mu^{-1}\mathbf{I}\right)^{-1}\mathbf{U}^T\left(\mathbf{X}-\mathbf{E}+\mu^{-1}\mathbf{Y}\right)
\end{aligned} \tag{3.20}$$

Finally, we minimise $\mathcal{L}_\mu$ with respect to $\mathbf{E}$ as follows

$$\begin{aligned}
&\arg\min_{\mathbf{E}} \mathcal{L}_\mu(\mathbf{U},\mathbf{V},\mathbf{E},\mathbf{Y}) \\
&= \arg\min_{\mathbf{E}} \lambda\|\mathbf{E}\|_1 + \langle \mathbf{X}-\mathbf{U}\mathbf{V}-\mathbf{E}, \mathbf{Y}\rangle + \frac{\mu}{2}\|\mathbf{X}-\mathbf{U}\mathbf{V}-\mathbf{E}\|_F^2 \\
&= \arg\min_{\mathbf{E}} \lambda\mu^{-1}\|\mathbf{E}\|_1 + \frac{1}{2}\left\|\mathbf{E} - \left(\mathbf{X}-\mathbf{U}\mathbf{V}+\mu^{-1}\mathbf{Y}\right)\right\|_F^2 \\
&= \mathcal{S}_{\lambda\mu^{-1}}\left\{\mathbf{X}-\mathbf{U}\mathbf{V}+\mu^{-1}\mathbf{Y}\right\}
\end{aligned} \tag{3.21}$$

The full ALM algorithm is presented in Algorithm 3.3. Notice that the size of factorisation r, i.e. the number of columns of $\mathbf{U}$ or the number of rows of $\mathbf{V}$, is required as input. In [14] no particular suggestion is made on how to initialise $\mathbf{U}$ or $\mathbf{V}$. In our implementation, we initialise $\mathbf{U}$ to have as columns the r first left singular vectors of $\mathbf{X}$.

Algorithm 3.3: Matrix Bilinear Robust PCA

```
  Input: Data matrix $\mathbf{X}$, regulariser $\lambda > 0$, factorisation size $r$
  Initialise: $\mathbf{U}_0 = [r$ first left singular vectors of $\mathbf{X}]$, $\mathbf{E}_0 = \mathbf{0}$, $\mathbf{Y}_0 = \mathbf{0}$, $\mu_0 > 0$, $k = 0$
1 while not converged do
2     $\mathbf{V}_{k+1} = \left(\mathbf{U}_k^T \mathbf{U}_k + \mu_k^{-1}\mathbf{I}\right)^{-1} \mathbf{U}_k^T \left(\mathbf{X} - \mathbf{E}_k + \mu_k^{-1}\mathbf{Y}_k\right)$
3     $\mathbf{U}_{k+1} = \left(\mathbf{X} - \mathbf{E}_k + \mu_k^{-1}\mathbf{Y}_k\right) \mathbf{V}_{k+1}^T \left(\mathbf{V}_{k+1}\mathbf{V}_{k+1}^T + \mu_k^{-1}\mathbf{I}\right)^{-1}$
4     $\mathbf{E}_{k+1} = \mathcal{S}_{\lambda\mu_k^{-1}} \left\{\mathbf{X} - \mathbf{U}_{k+1}\mathbf{V}_{k+1} + \mu_k^{-1}\mathbf{Y}_k\right\}$
5     $\mathbf{Y}_{k+1} = \mathbf{Y}_k + \mu_k \left(\mathbf{X} - \mathbf{U}_{k+1}\mathbf{V}_{k+1} - \mathbf{E}_{k+1}\right)$
6     $\mu_{k+1} = \min\left(\rho\mu_k, \mu_{max}\right)$
7     $k \leftarrow k + 1$
8 end while
  Output: Low-rank component $\mathbf{A} = \mathbf{U}_k\mathbf{V}_k$, sparse component $\mathbf{E}_k$
```

3.3 Inductive Robust Principal Component Analysis

A major limitation of RPCA and BRPCA is that they are essentially *transductive* methods in that they are more suitable for *batch* than *online* computation. Suppose that the input data matrix is $\mathbf{X}$ and RPCA (or BRPCA) has been used to recover its low-rank and sparse components $\mathbf{A}$ and $\mathbf{E}$ respectively. Now, suppose that a new data point $\mathbf{x}_{new}$ is observed, which we also wish to decompose as $\mathbf{x}_{new} = \mathbf{a}_{new} + \mathbf{e}_{new}$, with $\mathbf{a}_{new}$ being the true signal and $\mathbf{e}_{new}$ being sparse noise. In order to do this, we need to perform RPCA (or BRPCA) on data matrix $\mathbf{X}' = \begin{bmatrix}\mathbf{X} & \mathbf{x}_{new}\end{bmatrix}$, which includes the whole data matrix $\mathbf{X}$ that has previously been processed. It becomes apparent that having to run the algorithm over all data every time a new data point is observed is particularly unsuitable for applications where online—instead of batch—computation is required.

In order to overcome the above limitation of RPCA, Bao et al. [5] proposed *Inductive Robust Principal Component Analysis*. Given an initial data matrix $\mathbf{X}$, rather than learning its low-rank component $\mathbf{A}$, IRPCA attempts to learn a *projection matrix* $\mathbf{P}$ which projects $\mathbf{X}$ onto the low-dimensional subspace. In other words, $\mathbf{P}$ is such that $\mathbf{A} = \mathbf{PX}$. Subsequently, if a new data point $\mathbf{x}_{new}$ is observed, it can be easily projected onto the low-dimensional subspace in order to recover $\mathbf{a}_{new}$, or, in other words, $\mathbf{a}_{new} = \mathbf{P}\mathbf{x}_{new}$.

The optimisation problem defining Inductive RPCA can be easily obtained by the RPCA problem (3.4) by rewriting $\mathbf{A} = \mathbf{PX}$. Furthermore, notice that

$$\operatorname{rank}(\mathbf{A}) = \operatorname{rank}(\mathbf{PX}) \leq \operatorname{rank}(\mathbf{P}) \tag{3.22}$$

therefore minimising the rank of $\mathbf{P}$ implies minimising an upper bound on the rank of $\mathbf{A}$. The above motivates the formulation of IRPCA as the following optimisation problem

$$\min_{\mathbf{P},\mathbf{E}} \|\mathbf{P}\|_* + \lambda \|\mathbf{E}\|_1 \quad \text{s.t.} \quad \mathbf{X} = \mathbf{PX} + \mathbf{E} \tag{3.23}$$

However, due to the existence of the multiplicative term $\mathbf{PX}$, the above problem cannot be directly solved in the same way as RPCA. In the following, we will describe two algorithms for solving the above problem. The first one, based on substitution and using the ALM method, was proposed in the original IRPCA paper [5]. The second one is introduced in this thesis and it is based on ALM with an extra linearisation step.

3.3.1 Solution Based on Substitution

In order to solve the IRPCA problem, Bao et al. [5] rewrite it in the following equivalent form

$$\min_{\mathbf{J},\mathbf{P},\mathbf{E}} \|\mathbf{J}\|_* + \lambda \|\mathbf{E}\|_1 \quad \text{s.t.} \quad \begin{array}{l} \mathbf{X} = \mathbf{PX} + \mathbf{E} \\ \mathbf{P} = \mathbf{J} \end{array} \tag{3.24}$$

The developed solution is based on ALM. The augmented Lagrangian of the above problem is

$$\begin{aligned}\mathcal{L}_\mu\left(\mathbf{J},\mathbf{P},\mathbf{E},\mathbf{Y}_1,\mathbf{Y}_2\right) = \left\|\mathbf{J}\right\|_* + \lambda\left\|\mathbf{E}\right\|_1 + \langle\mathbf{X}-\mathbf{PX}-\mathbf{E},\mathbf{Y}_1\rangle + \langle\mathbf{P}-\mathbf{J},\mathbf{Y}_2\rangle \\ + \frac{\mu}{2}\left\|\mathbf{X}-\mathbf{PX}-\mathbf{E}\right\|_F^2 + \frac{\mu}{2}\left\|\mathbf{P}-\mathbf{J}\right\|_F^2 \qquad (3.25)\end{aligned}$$

We shall minimise the above with respect to each argument separately. Fixing **P** and **E**, minimisation with respect to **J** is done as follows

$$\begin{aligned}&\arg\min_{\mathbf{J}} \mathcal{L}_\mu\left(\mathbf{J},\mathbf{P},\mathbf{E},\mathbf{Y}_1,\mathbf{Y}_2\right)\\ &= \arg\min_{\mathbf{J}} \left\|\mathbf{J}\right\|_* + \langle\mathbf{P}-\mathbf{J},\mathbf{Y}_2\rangle + \frac{\mu}{2}\left\|\mathbf{P}-\mathbf{J}\right\|_F^2\\ &= \arg\min_{\mathbf{J}} \mu^{-1}\left\|\mathbf{J}\right\|_* + \frac{1}{2}\left\|\mathbf{J}-\left(\mathbf{P}+\mu^{-1}\mathbf{Y}_2\right)\right\|_F^2\\ &= \mathcal{D}_{\mu^{-1}}\left\{\mathbf{P}+\mu^{-1}\mathbf{Y}_2\right\} \qquad (3.26)\end{aligned}$$

Fixing **J** and **E**, we next minimise with respect to **P**. Since $\mathcal{L}_\mu$ is differentiable with respect to **P**, we can easily minimise it by forcing its derivative with respect to **P** equal to zero. Using the following derivatives

$$\frac{\partial}{\partial\mathbf{P}}\langle\mathbf{X}-\mathbf{PX}-\mathbf{E},\mathbf{Y}_1\rangle = -\mathbf{Y}_1\mathbf{X}^T \qquad (3.27)$$

$$\frac{\partial}{\partial\mathbf{P}}\langle\mathbf{P}-\mathbf{J},\mathbf{Y}_2\rangle = \mathbf{Y}_2 \qquad (3.28)$$

$$\frac{\partial}{\partial\mathbf{P}}\left(\frac{\mu}{2}\left\|\mathbf{X}-\mathbf{PX}-\mathbf{E}\right\|_F^2\right) = -\mu\left(\mathbf{X}-\mathbf{PX}-\mathbf{E}\right)\mathbf{X}^T \qquad (3.29)$$

$$\frac{\partial}{\partial\mathbf{P}}\left(\frac{\mu}{2}\left\|\mathbf{P}-\mathbf{J}\right\|_F^2\right) = \mu\left(\mathbf{P}-\mathbf{J}\right) \qquad (3.30)$$

we obtain the following solution for **P**

$$\begin{aligned}&\frac{\partial\mathcal{L}_\mu}{\partial\mathbf{P}} = \mathbf{0} && \Rightarrow\\ &\mathbf{P}\left(\mathbf{XX}^T+\mathbf{I}\right) - \left(\mathbf{X}-\mathbf{E}+\mu^{-1}\mathbf{Y}_1\right)\mathbf{X}^T - \mathbf{J} + \mu^{-1}\mathbf{Y}_2 = \mathbf{0} && \Rightarrow\\ &\mathbf{P} = \left[\left(\mathbf{X}-\mathbf{E}+\mu^{-1}\mathbf{Y}_1\right)\mathbf{X}^T + \mathbf{J} - \mu^{-1}\mathbf{Y}_2\right]\left(\mathbf{XX}^T+\mathbf{I}\right)^{-1} && (3.31)\end{aligned}$$

Then, keeping **J** and **P** fixed, we minimise with respect to **E** as follows

$$\begin{aligned}&\arg\min_{\mathbf{E}} \mathcal{L}_\mu\left(\mathbf{J},\mathbf{P},\mathbf{E},\mathbf{Y}_1,\mathbf{Y}_2\right)\\ &= \arg\min_{\mathbf{E}} \lambda\left\|\mathbf{E}\right\|_1 + \langle\mathbf{X}-\mathbf{PX}-\mathbf{E},\mathbf{Y}_1\rangle + \frac{\mu}{2}\left\|\mathbf{X}-\mathbf{PX}-\mathbf{E}\right\|_F^2\\ &= \arg\min_{\mathbf{E}} \lambda\mu^{-1}\left\|\mathbf{E}\right\|_1 + \frac{1}{2}\left\|\mathbf{E}-\left(\mathbf{X}-\mathbf{PX}+\mu^{-1}\mathbf{Y}_1\right)\right\|_F^2\\ &= \mathcal{S}_{\lambda\mu^{-1}}\left\{\mathbf{X}-\mathbf{PX}+\mu^{-1}\mathbf{Y}_1\right\} \qquad (3.32)\end{aligned}$$

The full algorithm is presented in detail in Algorithm 3.4.

3.3.2 Solution Based on Linearisation

The drawback of the substitution-based ALM solution for IRPCA is that, by introducing auxiliary matrix **J**, it increases the number of variables to be optimised. We now introduce an alternative ALM solution which does not require auxiliary variables, but instead relies on an intermediate *linearisation step*. This solution follows the so-called *Linearised Alternating Direction Method* proposed by Lin et al. [53] for the problem of Low-Rank Representation.

Algorithm 3.4: Matrix Inductive Robust PCA via substitution

```
    Input: Data matrix $\mathbf{X}$, regulariser $\lambda > 0$
    Initialise: $\mathbf{P}_0 = \mathbf{0}$, $\mathbf{E}_0 = \mathbf{0}$, $\mathbf{Y}_1^0 = \mathbf{0}$, $\mathbf{Y}_2^0 = \mathbf{0}$, $\mu_0 > 0$, $k = 0$
1   while not converged do
2       $\mathbf{J}_{k+1} = \mathcal{D}_{\mu_k^{-1}} \left\{\mathbf{P}_k + \mu_k^{-1}\mathbf{Y}_2^k\right\}$
3       $\mathbf{P}_{k+1} = \left[\left(\mathbf{X} - \mathbf{E}_k + \mu_k^{-1}\mathbf{Y}_1^k\right)\mathbf{X}^T + \mathbf{J}_{k+1} - \mu_k^{-1}\mathbf{Y}_2^k\right]\left(\mathbf{X}\mathbf{X}^T + \mathbf{I}\right)^{-1}$
4       $\mathbf{E}_{k+1} = \mathcal{S}_{\lambda\mu_k^{-1}} \left\{\mathbf{X} - \mathbf{P}_{k+1}\mathbf{X} + \mu_k^{-1}\mathbf{Y}_1^k\right\}$
5       $\mathbf{Y}_1^{k+1} = \mathbf{Y}_1^k + \mu_k\left(\mathbf{X} - \mathbf{P}_{k+1}\mathbf{X} - \mathbf{E}_{k+1}\right)$
6       $\mathbf{Y}_2^{k+1} = \mathbf{Y}_2^k + \mu_k\left(\mathbf{P}_{k+1} - \mathbf{J}_{k+1}\right)$
7       $\mu_{k+1} = \min\left(\rho\mu_k, \mu_{max}\right)$
8       $k \leftarrow k + 1$
9   end while
    Output: Projection matrix $\mathbf{P}_k$, sparse component $\mathbf{E}_k$
```

We begin by considering the augmented Lagrangian of the original IRPCA problem

$$\mathcal{L}_\mu\left(\mathbf{P}, \mathbf{E}, \mathbf{Y}\right) = \left\|\mathbf{P}\right\|_* + \lambda\left\|\mathbf{E}\right\|_1 + \left\langle \mathbf{X} - \mathbf{P}\mathbf{X} - \mathbf{E}, \mathbf{Y}\right\rangle + \frac{\mu}{2}\left\|\mathbf{X} - \mathbf{P}\mathbf{X} - \mathbf{E}\right\|_F^2 \tag{3.33}$$

We shall minimise $\mathcal{L}_\mu$ with respect to each parameter separately. Fixing $\mathbf{E}$, the minimisation problem with respect to $\mathbf{P}$ becomes

$$\begin{aligned}
&\arg\min_{\mathbf{P}} \mathcal{L}_\mu\left(\mathbf{P}, \mathbf{E}, \mathbf{Y}\right) \\
&= \arg\min_{\mathbf{P}} \left\|\mathbf{P}\right\|_* + \left\langle \mathbf{X} - \mathbf{P}\mathbf{X} - \mathbf{E}, \mathbf{Y}\right\rangle + \frac{\mu}{2}\left\|\mathbf{X} - \mathbf{P}\mathbf{X} - \mathbf{E}\right\|_F^2 \\
&= \arg\min_{\mathbf{P}} \mu^{-1}\left\|\mathbf{P}\right\|_* + \frac{1}{2}\left\|\mathbf{P}\mathbf{X} - \left(\mathbf{X} - \mathbf{E} + \mu^{-1}\mathbf{Y}\right)\right\|_F^2
\end{aligned} \tag{3.34}$$

Notice that the form of problem (3.34) is similar to problem (3.12). However, due to the presence of the term $\mathbf{P}\mathbf{X}$, problem (3.34) is no longer a proximal operator. Instead, we shall replace it with a *proximal gradient update*, which we will refer to as *linearisation step.*

To better understand the linearisation step, consider first the more general case

$$\min_{\mathbf{X}} g\left(\mathbf{X}\right) + f\left(\mathbf{X}\right) \tag{3.35}$$

where $g\left(\mathbf{X}\right) = a\left\|\mathbf{X}\right\|_*$ and $f\left(\mathbf{X}\right) = \frac{1}{2}\left\|\mathbf{X}\mathbf{A} - \mathbf{B}\right\|_F^2$. Notice that problem (3.34) is a special case of the above. The derivative of the smooth term $f\left(\mathbf{X}\right)$ is the following

$$\nabla f\left(\mathbf{X}\right) = \frac{\partial}{\partial \mathbf{X}}\left(\frac{1}{2}\left\|\mathbf{X}\mathbf{A} - \mathbf{B}\right\|_F^2\right) = \left(\mathbf{X}\mathbf{A} - \mathbf{B}\right)\mathbf{A}^T \tag{3.36}$$

Note that $\forall\, \mathbf{X}_1, \mathbf{X}_2$

$$\begin{aligned}
\left\|\nabla f\left(\mathbf{X}_1\right) - \nabla f\left(\mathbf{X}_2\right)\right\|_F &= \left\|\mathbf{X}_1\mathbf{A}\mathbf{A}^T - \mathbf{B}\mathbf{A}^T - \mathbf{X}_2\mathbf{A}\mathbf{A}^T + \mathbf{B}\mathbf{A}^T\right\|_F \\
&= \left\|\left(\mathbf{X}_1 - \mathbf{X}_2\right)\mathbf{A}\mathbf{A}^T\right\|_F \\
&\leq \left\|\mathbf{A}\mathbf{A}^T\right\|_F \left\|\mathbf{X}_1 - \mathbf{X}_2\right\|_F
\end{aligned} \tag{3.37}$$

therefore ∇f is Lipschitz continuous with Lipschitz constant $L = \left\|\mathbf{A}\mathbf{A}^T\right\|_F$. If we applied the proximal gradient algorithm in order to solve problem (3.35), the update step (2.67) would become

$$\begin{aligned}
\mathbf{X}_{k+1} &= \mathcal{P}_{\left(n^{-1}g\right)}\left\{\mathbf{X}_k - n^{-1}\nabla f\left(\mathbf{X}_k\right)\right\} \\
&= \mathcal{D}_{an^{-1}}\left\{\mathbf{X}_k - n^{-1}\left(\mathbf{X}_k\mathbf{A} - \mathbf{B}\right)\mathbf{A}^T\right\}
\end{aligned} \tag{3.38}$$

By Theorem 2.12, we know that if $n \geq \left\|\mathbf{A}\mathbf{A}^T\right\|_F$ the above update is *guaranteed* to not increase the objective function. We shall refer to the above update as *linearisation step*.

Based on the above discussion, in order to solve problem (3.34) we would have to iterate the linearisation step (3.38) until convergence. This would have to be done at each ALM iteration. Fortunately, it can be shown that running step (3.38) only *once* at each ALM iteration is sufficient for convergence. This version of ALM is known in the literature as *Linearised Alternating Direction Method* or simply LADM. For more information and proofs of convergence, the reader may refer to [53, 57].

Going back to problem (3.34), the linearisation step for $\mathbf{P}$ can be obtained by substituting $a = \mu^{-1}$, $\mathbf{A} = \mathbf{X}$ and $\mathbf{B} = \mathbf{X} - \mathbf{E} + \mu^{-1}\mathbf{Y}$ in equation (3.38), which yields

$$\mathbf{P}_{k+1} = \mathcal{D}_{(\mu n)^{-1}}\left\{\mathbf{P}_k - n^{-1}\left(\mathbf{P}_k\mathbf{X} - \mathbf{X} + \mathbf{E} - \mu^{-1}\mathbf{Y}\right)\mathbf{X}^T\right\} \tag{3.39}$$

At the next step, we keep $\mathbf{P}$ fixed and minimise $\mathcal{L}_\mu$ with respect to $\mathbf{E}$, which is done as follows

$$\begin{aligned}
&\arg\min_{\mathbf{E}} \mathcal{L}_\mu(\mathbf{P}, \mathbf{E}, \mathbf{Y}) \\
&= \arg\min_{\mathbf{E}} \lambda\|\mathbf{E}\|_1 + \langle \mathbf{X} - \mathbf{P}\mathbf{X} - \mathbf{E}, \mathbf{Y}\rangle + \frac{\mu}{2}\|\mathbf{X} - \mathbf{P}\mathbf{X} - \mathbf{E}\|_F^2 \\
&= \arg\min_{\mathbf{E}} \lambda\mu^{-1}\|\mathbf{E}\|_1 + \frac{1}{2}\left\|\mathbf{E} - \left(\mathbf{X} - \mathbf{P}\mathbf{X} + \mu^{-1}\mathbf{Y}\right)\right\|_F^2 \\
&= \mathcal{S}_{\lambda\mu^{-1}}\left\{\mathbf{X} - \mathbf{P}\mathbf{X} + \mu^{-1}\mathbf{Y}\right\}
\end{aligned} \tag{3.40}$$

The procedure is presented in detail in Algorithm 3.5.

Algorithm 3.5: Matrix Inductive Robust PCA via linearisation

```
  Input: Data matrix $\mathbf{X}$, regulariser $\lambda > 0$
  Initialise: $\mathbf{P}_0 = \mathbf{0}$, $\mathbf{E}_0 = \mathbf{0}$, $\mathbf{Y}_0 = \mathbf{0}$, $\mu_0 > 0$, $n = \max\left(\left\|\mathbf{X}\mathbf{X}^T\right\|_F, 1.0\right)$, $k = 0$
1 while not converged do
2     $\mathbf{P}_{k+1} = \mathcal{D}_{(\mu_k n)^{-1}}\left\{\mathbf{P}_k - n^{-1}\left(\mathbf{P}_k\mathbf{X} - \mathbf{X} + \mathbf{E}_k - \mu_k^{-1}\mathbf{Y}_k\right)\mathbf{X}^T\right\}$
3     $\mathbf{E}_{k+1} = \mathcal{S}_{\lambda\mu_k^{-1}}\left\{\mathbf{X} - \mathbf{P}_{k+1}\mathbf{X} + \mu_k^{-1}\mathbf{Y}_k\right\}$
4     $\mathbf{Y}_{k+1} = \mathbf{Y}_k + \mu_k\left(\mathbf{X} - \mathbf{P}_{k+1}\mathbf{X} - \mathbf{E}_{k+1}\right)$
5     $\mu_{k+1} = \min\left(\rho\mu_k, \mu_{max}\right)$
6     $k \leftarrow k + 1$
7 end while
  Output: Projection matrix $\mathbf{P}_k$, sparse component $\mathbf{E}_k$
```

3.3.3 Connection to Low-Rank Representation

From a computational perspective, as it is also acknowledged in its original paper [5], IRPCA has many similarities to *Low-Rank Representation*. LRR was proposed by Liu et al. [55] and it can be described by the following optimisation problem

$$\min_{\mathbf{Z},\mathbf{E}} \|\mathbf{Z}\|_* + \lambda\|\mathbf{E}\|_1 \quad \text{s.t.} \quad \mathbf{X} = \mathbf{D}\mathbf{Z} + \mathbf{E} \tag{3.41}$$

Here, the low-rank component is factorised as $\mathbf{A} = \mathbf{D}\mathbf{Z}$, where $\mathbf{D} \in \mathbb{R}^{m\times r}$ is a given "dictionary" and $\mathbf{Z} \in \mathbb{R}^{r\times n}$ is the representation of $\mathbf{A}$ in that dictionary. The objective of LRR is to find the lowest-rank representation of $\mathbf{A}$ in $\mathbf{D}$, in the presence of sparse corruptions. If we choose $\mathbf{D} = \mathbf{I}$, LRR reduces to RPCA. Liu et al. [55] showed that if the data points are approximately sampled from a union of low-dimensional linear subspaces, for certain choices of $\mathbf{D}$ (such as $\mathbf{D} = \mathbf{X}$) the optimal $\mathbf{Z}$ is approximately block-diagonal, with each block corresponding to a linear subspace. Hence, LRR is most commonly used in the problem of robust subspace clustering [83].

The connection between LRR and IRPCA is not conceptual, since they address different problems, but computational. That is, if we choose $\mathbf{D} = \mathbf{X}$, the only difference between them is in the factorisation of the low-rank component; in IRPCA $\mathbf{P}$ is *right-multiplied* by $\mathbf{X}$ whereas in LRR $\mathbf{Z}$ is *left-multiplied* by $\mathbf{X}$. Therefore, LRR can also be solved, with small modifications, by Algorithms 3.4 and 3.5. For details, see the original LRR paper [55], where LRR is solved by the equivalent of Algorithm 3.4, and the paper by Lin et al. [53], where LRR is solved by the equivalent of Algorithm 3.5.

3.4 Orthonormal Robust Principal Component Analysis

Given a data matrix $\mathbf{X} \in \mathbb{R}^{m\times n}$, RPCA attempts to recover the low-rank matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$ which can be thought of as the projection of $\mathbf{X}$ onto a low-dimensional subspace, which we shall refer to as the *principal subspace*. IRPCA takes this idea one step further and tries to learn the *projection matrix* $\mathbf{P} \in \mathbb{R}^{m\times m}$ which projects $\mathbf{X}$ onto the principal subspace in order to retrieve $\mathbf{A}$ (that is, $\mathbf{A} = \mathbf{PX}$). It would be interesting however to be able to retrieve the principal subspace *itself*. Assuming that the principal subspace is r-dimensional, where $r \leq \min(m, n)$, that would require learning a set of basis vectors $\mathbf{U} = \begin{bmatrix}\mathbf{u}_1 & \mathbf{u}_2 & \cdots & \mathbf{u}_r\end{bmatrix} \in \mathbb{R}^{m\times r}$ which span the principal subspace. To restrict the degrees of freedom of $\mathbf{U}$, we may also require that this set of basis vectors be orthonormal, i.e. $\mathbf{U}^T\mathbf{U} = \mathbf{I}$. Classical Principal Component Analysis typically refers to such basis vectors as *principal components*. Having learned the principal components, the projected data points would be represented as linear combinations of the principal components. In other words, if $\mathbf{V} \in \mathbb{R}^{r\times n}$ is a matrix of appropriate coefficients, we can write $\mathbf{A} = \mathbf{UV}$.

In the following, we will modify the RPCA problem (3.4) so that a set of principal components can be recovered as well. From the above discussion, we can write $\mathbf{A} = \mathbf{UV}$, with $\mathbf{U}^T\mathbf{U} = \mathbf{I}$. Due to the unitary invariance of the nuclear norm (Theorem 2.2) we have the following[3]

$$\|\mathbf{A}\|_* = \|\mathbf{UV}\|_* = \|\mathbf{V}\|_* \tag{3.42}$$

The minimisation problem then becomes

$$\min_{\mathbf{V},\mathbf{E},\mathbf{U}} \|\mathbf{V}\|_* + \lambda \|\mathbf{E}\|_1 \quad \text{s.t.} \quad \begin{array}{l} \mathbf{X} = \mathbf{UV} + \mathbf{E} \\ \mathbf{U}^T\mathbf{U} = \mathbf{I} \end{array} \tag{3.43}$$

In this thesis, we will refer to the above problem as *Orthonormal Robust Principal Component Analysis*. This problem was introduced by Liu and Yan [54] as *active subspace learning* and was further generalised by Papamakarios et al. [69] as *Generalised Scalable Robust Principal Component Analysis*. In the following, we will describe the ALM algorithm that was presented in [54] for the solution of ORPCA.

We shall begin by forming the augmented Lagrangian for the ORPCA problem. Notice that this augmented Lagrangian is only partial, since it does not incorporate the constraint $\mathbf{U}^T\mathbf{U} = \mathbf{I}$.

$$\mathcal{L}_\mu(\mathbf{V}, \mathbf{E}, \mathbf{U}, \mathbf{Y}) = \|\mathbf{V}\|_* + \lambda\|\mathbf{E}\|_1 + \langle \mathbf{X} - \mathbf{UV} - \mathbf{E}, \mathbf{Y}\rangle + \frac{\mu}{2}\|\mathbf{X} - \mathbf{UV} - \mathbf{E}\|_F^2 \tag{3.44}$$

We will first minimise $\mathcal{L}_\mu$ with respect to $\mathbf{V}$ keeping all other variables fixed. Bearing in mind

[3]In fact, this would be true for any Schatten p-norm. Also, since $\mathbf{U}$ is of full column rank, we have that $\operatorname{rank}(\mathbf{A}) = \operatorname{rank}(\mathbf{UV}) = \operatorname{rank}(\mathbf{V})$.

that $\mathbf{U}^T\mathbf{U} = \mathbf{I}$ and that the Frobenius norm is unitary invariant, the minimisation becomes

$$\begin{aligned}
&\arg\min_{\mathbf{V}} \mathcal{L}_\mu(\mathbf{V},\mathbf{E},\mathbf{U},\mathbf{Y}) \\
&= \arg\min_{\mathbf{V}} \|\mathbf{V}\|_* + \langle \mathbf{X}-\mathbf{U}\mathbf{V}-\mathbf{E},\mathbf{Y}\rangle + \frac{\mu}{2}\|\mathbf{X}-\mathbf{U}\mathbf{V}-\mathbf{E}\|_F^2 \\
&= \arg\min_{\mathbf{V}} \mu^{-1}\|\mathbf{V}\|_* + \frac{1}{2}\left\|\mathbf{U}\mathbf{V}-\left(\mathbf{X}-\mathbf{E}+\mu^{-1}\mathbf{Y}\right)\right\|_F^2 \\
&= \arg\min_{\mathbf{V}} \mu^{-1}\|\mathbf{V}\|_* + \frac{1}{2}\|\mathbf{U}\mathbf{V}\|_F^2 - \left\langle \mathbf{U}\mathbf{V},\mathbf{X}-\mathbf{E}+\mu^{-1}\mathbf{Y}\right\rangle \\
&= \arg\min_{\mathbf{V}} \mu^{-1}\|\mathbf{V}\|_* + \frac{1}{2}\|\mathbf{V}\|_F^2 - \left\langle \mathbf{V},\mathbf{U}^T\left(\mathbf{X}-\mathbf{E}+\mu^{-1}\mathbf{Y}\right)\right\rangle \\
&= \arg\min_{\mathbf{V}} \mu^{-1}\|\mathbf{V}\|_* + \frac{1}{2}\left\|\mathbf{V}-\mathbf{U}^T\left(\mathbf{X}-\mathbf{E}+\mu^{-1}\mathbf{Y}\right)\right\|_F^2 \\
&= \mathcal{D}_{\mu^{-1}}\left\{\mathbf{U}^T\left(\mathbf{X}-\mathbf{E}+\mu^{-1}\mathbf{Y}\right)\right\}
\end{aligned} \tag{3.45}$$

We then minimise $\mathcal{L}_\mu$ with respect to $\mathbf{E}$ as follows

$$\begin{aligned}
&\arg\min_{\mathbf{E}} \mathcal{L}_\mu(\mathbf{V},\mathbf{E},\mathbf{U},\mathbf{Y}) \\
&= \arg\min_{\mathbf{E}} \lambda\|\mathbf{E}\|_1 + \langle \mathbf{X}-\mathbf{U}\mathbf{V}-\mathbf{E},\mathbf{Y}\rangle + \frac{\mu}{2}\|\mathbf{X}-\mathbf{U}\mathbf{V}-\mathbf{E}\|_F^2 \\
&= \arg\min_{\mathbf{E}} \lambda\mu^{-1}\|\mathbf{E}\|_1 + \frac{1}{2}\left\|\mathbf{E}-\left(\mathbf{X}-\mathbf{U}\mathbf{V}+\mu^{-1}\mathbf{Y}\right)\right\|_F^2 \\
&= \mathcal{S}_{\lambda\mu^{-1}}\left\{\mathbf{X}-\mathbf{U}\mathbf{V}+\mu^{-1}\mathbf{Y}\right\}
\end{aligned} \tag{3.46}$$

Next, we minimise $\mathcal{L}_\mu$ with respect to $\mathbf{U}$, subject to $\mathbf{U}^T\mathbf{U} = \mathbf{I}$. The minimisation problem becomes

$$\begin{aligned}
&\arg\min_{\mathbf{U}} \mathcal{L}_\mu(\mathbf{V},\mathbf{E},\mathbf{U},\mathbf{Y}) \\
&= \arg\min_{\mathbf{U}} \langle \mathbf{X}-\mathbf{U}\mathbf{V}-\mathbf{E},\mathbf{Y}\rangle + \frac{\mu}{2}\|\mathbf{X}-\mathbf{U}\mathbf{V}-\mathbf{E}\|_F^2 \\
&= \arg\min_{\mathbf{U}} \frac{1}{2}\left\|\left(\mathbf{X}-\mathbf{E}+\mu^{-1}\mathbf{Y}\right)-\mathbf{U}\mathbf{V}\right\|_F^2 \quad \text{s.t.} \quad \mathbf{U}^T\mathbf{U}=\mathbf{I}
\end{aligned} \tag{3.47}$$

Assume the following Singular Value Decomposition

$$\left(\mathbf{X}-\mathbf{E}+\mu^{-1}\mathbf{Y}\right)\mathbf{V}^T = \mathbf{U}_S\mathbf{D}_S\mathbf{V}_S^T \tag{3.48}$$

Here, we assume that $\mathbf{U}_S \in \mathbb{R}^{m\times r}$ and $\mathbf{D}_S, \mathbf{V}_S \in \mathbb{R}^{r\times r}$, where $r \leq m$ is the number of principal components. According to the *Reduced Rank Procrustes Theorem* (Theorem 4 in [94]), the optimiser $\mathbf{U}^*$ of problem (3.47) is given by

$$\mathbf{U}^* = \mathbf{U}_S\mathbf{V}_S^T \tag{3.49}$$

The full ALM algorithm for ORPCA is described in Algorithm 3.6. In our implementation, the number of principal components r is required as input. In ORPCA, this parameter serves as a controllable upper bound on the rank of $\mathbf{A}$, since

$$\operatorname{rank}(\mathbf{A}) = \operatorname{rank}(\mathbf{U}\mathbf{V}) \leq \operatorname{rank}(\mathbf{U}) = r \tag{3.50}$$

It is important also to note that the ORPCA problem, unlike RPCA and IRPCA, is non-convex, therefore the obtained solution depends on initialisation. In our implementation we initialise $\mathbf{U}$ with the r first left singular vectors of $\mathbf{X}$, which we have empirically found to be sufficient for most cases. This initialisation is equivalent to setting $\mathbf{U}$ to be the first r principal components that would be computed by classical PCA.

Algorithm 3.6: Matrix Orthonormal Robust PCA

```
    Input: Data matrix $\mathbf{X}$, regulariser $\lambda > 0$, number of principal components $r$
    Initialise: $\mathbf{U}_0 = [r$ first left singular vectors of $\mathbf{X}]$, $\mathbf{E}_0 = \mathbf{0}$, $\mathbf{Y}_0 = \mathbf{0}$, $\mu_0 > 0$, $k = 0$
1   while not converged do
2       $\mathbf{V}_{k+1} = \mathcal{D}_{\mu_k^{-1}}\left\{\mathbf{U}_k^T\left(\mathbf{X} - \mathbf{E}_k + \mu_k^{-1}\mathbf{Y}_k\right)\right\}$
3       $\mathbf{E}_{k+1} = \mathcal{S}_{\lambda\mu_k^{-1}}\left\{\mathbf{X} - \mathbf{U}_k\mathbf{V}_{k+1} + \mu_k^{-1}\mathbf{Y}_k\right\}$
4       $[\mathbf{U}_S, \mathbf{D}_S, \mathbf{V}_S] = \text{svd}\left\{\left(\mathbf{X} - \mathbf{E}_{k+1} + \mu_k^{-1}\mathbf{Y}_k\right)\mathbf{V}_{k+1}^T\right\}$
5       $\mathbf{U}_{k+1} = \mathbf{U}_S\mathbf{V}_S^T$
6       $\mathbf{Y}_{k+1} = \mathbf{Y}_k + \mu_k\left(\mathbf{X} - \mathbf{U}_{k+1}\mathbf{V}_{k+1} - \mathbf{E}_{k+1}\right)$
7       $\mu_{k+1} = \min\left(\rho\mu_k, \mu_{max}\right)$
8       $k \leftarrow k + 1$
9   end while
    Output: Principal components $\mathbf{U}_k$, coefficients $\mathbf{V}_k$, sparse component $\mathbf{E}_k$
```

3.5 Robust Orthonormal Subspace Learning

The computational cost of ORPCA is dominated by two SVDs per iteration, of size $r \times n$ and $m \times r$, where $m \times n$ is the size of the input matrix $\mathbf{X}$ and r is the specified number of principal components. In most cases r is small, therefore ORPCA is typically more efficient than RPCA or IRPCA. However, the total complexity is still quadratic to both m and n, due to the existence of the SVDs.

Recently, Shu et al. [75] introduced *Robust Orthonormal Subspace Learning*, with the aim of further improving computational efficiency by completely removing the need for SVDs. In its formulation, ROSL is quite similar to ORPCA in that it decomposes $\mathbf{A} = \mathbf{U}\mathbf{V}$ with $\mathbf{U}^T\mathbf{U} = \mathbf{I}$. However, ROSL avoids using the nuclear norm of $\mathbf{V}$ as a rank surrogate, so as to avoid singular value thresholding and therefore the need for SVD. Instead, ROSL stems from the observation that the rank of $\mathbf{A}$ is upper-bounded by the number of non-zero rows of $\mathbf{V}$. Using a similar notation as the one used for the ℓ_0-norm, let $\|\mathbf{V}\|_{\text{row-0}}$ be the number of non-zero rows of $\mathbf{V}$. We can easily see that

$$\text{rank}\left(\mathbf{A}\right) = \text{rank}\left(\mathbf{U}\mathbf{V}\right) = \text{rank}\left(\mathbf{V}\right) \leq \|\mathbf{V}\|_{\text{row-0}} \tag{3.51}$$

Therefore, minimising $\|\mathbf{V}\|_{\text{row-0}}$ results to minimising an upper bound on $\text{rank}\left(\mathbf{A}\right)$. In the same way that the ℓ_0 norm is surrogated by the convex ℓ_1-norm, Shu et al. [75] suggest surrogating $\|\mathbf{V}\|_{\text{row-0}}$ by the following convex norm

$$\|\mathbf{V}\|_{\text{row-1}} = \sum_{i=1}^{r} \|\mathbf{v}_i\|_2 \tag{3.52}$$

where $\mathbf{v}_i$ is the i^{th} row of $\mathbf{V} \in \mathbb{R}^{r\times n}$. Based on the above, the minimisation problem for ROSL becomes the following

$$\min_{\mathbf{U},\mathbf{V},\mathbf{E}} \|\mathbf{V}\|_{\text{row-1}} + \lambda\|\mathbf{E}\|_1 \quad \text{s.t.} \quad \begin{array}{l}\mathbf{X} = \mathbf{U}\mathbf{V} + \mathbf{E}\\ \mathbf{U}^T\mathbf{U} = \mathbf{I}\end{array} \tag{3.53}$$

In the following, we will describe the ALM algorithm that is used in [75] to minimise the above. We start by forming the partial augmented Lagrangian (not incorporating constraint $\mathbf{U}^T\mathbf{U} = \mathbf{I}$) as follows

$$\mathcal{L}_\mu\left(\mathbf{U}, \mathbf{V}, \mathbf{E}, \mathbf{Y}\right) = \|\mathbf{V}\|_{\text{row-1}} + \lambda\|\mathbf{E}\|_1 + \langle\mathbf{X} - \mathbf{U}\mathbf{V} - \mathbf{E}, \mathbf{Y}\rangle + \frac{\mu}{2}\|\mathbf{X} - \mathbf{U}\mathbf{V} - \mathbf{E}\|_F^2 \tag{3.54}$$

Algorithm 3.7: Robust Orthonormal Subspace Learning

```
   Input: Data matrix $\mathbf{X}$, regulariser $\lambda > 0$, number of principal components $r$
   Initialise: $\mathbf{U}_0 = \mathbf{0}$, $\mathbf{V}_0 = \text{rand}$, $\mathbf{E}_0 = \mathbf{0}$, $\mathbf{Y}_0 = \mathbf{0}$, $\mu_0 > 0$, $k = 0$
1  while not converged do
2      for $i \in \{1, 2, \ldots, r\}$ do
3          $\mathbf{R}_i = \mathbf{X} + \mu_k^{-1}\mathbf{Y}_k - \mathbf{E}_k - \sum_{j<i} \mathbf{u}_j^{k+1} \left(\mathbf{v}_j^{k+1}\right)^T - \sum_{j>i} \mathbf{u}_j^{k} \left(\mathbf{v}_j^{k}\right)^T$
4          $\mathbf{R}_i \leftarrow \mathbf{R}_i - \sum_{j=1}^{i-1} \mathbf{u}_j^{k+1} \left(\mathbf{u}_j^{k+1}\right)^T \mathbf{R}_j$
5          $\mathbf{u}_i^{k+1} = \mathbf{R}_i \mathbf{v}_i^k$
6          $\mathbf{u}_i^{k+1} \leftarrow \frac{\mathbf{u}_i^{k+1}}{\left\|\mathbf{u}_i^{k+1}\right\|_2}$
7          $\mathbf{v}_i^{k+1} = \mathcal{M}_{\mu_k^{-1}} \left\{\left(\mathbf{u}_i^{k+1}\right)^T \mathbf{R}_i\right\}$
8      end for
9      for $i \in \{1, 2, \ldots, r\}$ do
10         if $\left\|\mathbf{v}_i^{k+1}\right\|_2 = 0$ then
11             delete $\left(\mathbf{u}_i^{k+1}, \mathbf{v}_i^{k+1}\right)$
12             $r \leftarrow r - 1$
13         end if
14     end for
15     $\mathbf{E}_{k+1} = \mathcal{S}_{\lambda\mu_k^{-1}} \left\{\mathbf{X} - \mathbf{U}_{k+1}\mathbf{V}_{k+1} + \mu_k^{-1}\mathbf{Y}_k\right\}$
16     $\mathbf{Y}_{k+1} = \mathbf{Y}_k + \mu_k \left(\mathbf{X} - \mathbf{U}_{k+1}\mathbf{V}_{k+1} - \mathbf{E}_{k+1}\right)$
17     $\mu_{k+1} = \min\left(\rho\mu_k, \mu_{max}\right)$
18     $k \leftarrow k + 1$
19 end while
   Output: Principal components $\mathbf{U}_k$, coefficients $\mathbf{V}_k$, sparse component $\mathbf{E}_k$
```

ROSL begins by minimising $\mathcal{L}_\mu$ with respect to both $\mathbf{U}$ and $\mathbf{V}$ while keeping $\mathbf{E}$ fixed, by solving the following minimisation problem

$$\begin{aligned} &\arg\min_{\mathbf{U},\mathbf{V}} \mathcal{L}_\mu\left(\mathbf{U},\mathbf{V},\mathbf{E},\mathbf{Y}\right) \\ &= \arg\min_{\mathbf{U},\mathbf{V}} \|\mathbf{V}\|_{\text{row-1}} + \langle \mathbf{X} - \mathbf{U}\mathbf{V} - \mathbf{E}, \mathbf{Y}\rangle + \frac{\mu}{2}\|\mathbf{X} - \mathbf{U}\mathbf{V} - \mathbf{E}\|_F^2 \end{aligned} \tag{3.55}$$

In order to solve the above, ROSL attempts a *Block Coordinate Descent* (BCD) approach. Let $\mathbf{u}_i$ and $\mathbf{v}_i$ be the i^{th} column of $\mathbf{U}$ and the i^{th} row of $\mathbf{V}$ respectively, with $i \in \{1, 2, \ldots, r\}$. The BCD approach updates each pair $(\mathbf{u}_i, \mathbf{v}_i)$ sequentially, such that $\mathbf{u}_i\mathbf{v}_i^T$ is a good rank-1 approximation to the following residual

$$\mathbf{R}_i = \mathbf{X} + \mu^{-1}\mathbf{Y} - \mathbf{E} - \sum_{\substack{j=1 \\ j\neq i}}^{r} \mathbf{u}_j\mathbf{v}_j^T \tag{3.56}$$

Without the orthonormality constraint $\mathbf{U}^T\mathbf{U} = \mathbf{I}$, this can be achieved by the following updates

$$\mathbf{u}_i \leftarrow \mathbf{R}_i\mathbf{v}_i \tag{3.57}$$

$$\mathbf{v}_i \leftarrow \frac{1}{\|\mathbf{u}_i\|_2^2}\mathcal{M}_{\mu^{-1}}\left\{\mathbf{u}_i^T\mathbf{R}_i\right\} \tag{3.58}$$

Here, $\mathcal{M}_a\{\cdot\}$ is the *magnitude shrinkage* function which is defined as

$$\mathcal{M}_a\{\mathbf{X}\} = \mathcal{S}_a\left\{\|\mathbf{X}\|_F\right\}\mathbf{X} \tag{3.59}$$

In order to take into account $\mathbf{U}^T\mathbf{U} = \mathbf{I}$, $\mathbf{u}_i$ is orthonormalised using *Gram-Schmidt orthonormalisation*. For further information on the updates of $\mathbf{u}_i$ and $\mathbf{v}_i$, the reader may refer to [8, 75].

The next step of ALM is the minimisation of $\mathcal{L}_\mu$ with respect to $\mathbf{E}$, which is done in the same way as in ORPCA

$$\begin{aligned}
&\arg\min_{\mathbf{E}} \mathcal{L}_\mu(\mathbf{U}, \mathbf{V}, \mathbf{E}, \mathbf{Y}) \\
&= \arg\min_{\mathbf{E}} \lambda \|\mathbf{E}\|_1 + \langle \mathbf{X} - \mathbf{UV} - \mathbf{E}, \mathbf{Y}\rangle + \frac{\mu}{2} \|\mathbf{X} - \mathbf{UV} - \mathbf{E}\|_F^2 \\
&= \arg\min_{\mathbf{E}} \lambda\mu^{-1} \|\mathbf{E}\|_1 + \frac{1}{2} \left\|\mathbf{E} - \left(\mathbf{X} - \mathbf{UV} + \mu^{-1}\mathbf{Y}\right)\right\|_F^2 \\
&= \mathcal{S}_{\lambda\mu^{-1}} \left\{\mathbf{X} - \mathbf{UV} + \mu^{-1}\mathbf{Y}\right\} \qquad (3.60)
\end{aligned}$$

Finally, the full ALM algorithm for ROSL is given in Algorithm 3.7. Notice that $\mathbf{U}$ is initialised as the zero matrix, while the entries of $\mathbf{V}$ are initialised uniformly at random in the interval $[0, 1]$. The Gram-Schmidt orthonormalisation consists of step 4 (where each $\mathbf{R}_i$ is made orthogonal to the rest) and step 6 (where each $\mathbf{u}_i^{k+1}$ is normalised to unit length). Finally, steps 9 to 14 correspond to a pruning process, where the pair $\left(\mathbf{u}_i^{k+1}, \mathbf{v}_i^{k+1}\right)$ is removed from matrices $\mathbf{U}^{k+1}$ and $\mathbf{V}^{k+1}$, if $\mathbf{v}_i^{k+1} = \mathbf{0}$. In such a case, the number of principal components r is reduced by one. This ensures that, at the end, $\mathbf{U}$ will not contain principal components whose coefficients in $\mathbf{V}$ are zero.

3.6 Complexity Analysis and Discussion

We conclude this chapter with a discussion over the advantages and disadvantages of each method, and an analysis of each algorithm's computational cost.

3.6.1 Regularisation-Based and Factorisation-Based Methods

In retrospect, we can categorise the methods presented in this chapter in two main groups, based on the way they determine the rank of the low-rank component.

(i) **Regularisation-based methods**. These include RPCA and IRPCA. In these methods, the rank of the low-rank component $\mathbf{A}$ is determined by the regularisation parameter λ.

(ii) **Factorisation-based methods**. These include BRPCA, ORPCA and ROSL. In these methods, the low-rank component is factorised as $\mathbf{A} = \mathbf{UV}$, where the number of columns in $\mathbf{U}$ (and number or rows in $\mathbf{V}$) is r. The rank of $\mathbf{A}$ is implicitly determined by both λ and r.

The obvious advantage of regularisation-based methods is that they require only one parameter to tune, which is λ. Even better, there exist theoretical results for RPCA (but not for IRPCA) that provide strong suggestions on the choice of λ (see [18] for details). Also, these methods involve convex optimisation problems, therefore global optimality is *guaranteed* for any initialisation of the decision variables. Their main drawback is their computational cost, since they require the computation of one SVD per iteration.

The factorisation-based methods rewrite $\mathbf{A} = \mathbf{UV}$, with r being the size of the factorisation (i.e. the number of columns in $\mathbf{U}$ and number of rows in $\mathbf{V}$). The obvious disadvantage is the introduction of a new parameter r, which needs to be tuned. However, there is an attractive interpretation of r as a "hard-coded" upper bound on the rank of $\mathbf{A}$ since

$$\operatorname{rank}(\mathbf{A}) = \operatorname{rank}(\mathbf{UV}) \leq \operatorname{rank}(\mathbf{U}) \leq r \qquad (3.61)$$

In applications where it is known that the rank of $\mathbf{A}$ is low (and there are many such applications in computer vision, as we shall see in chapter 7) such an upper bound that is user-controllable can prove quite useful. What is more, in ORPCA and ROSL the columns of $\mathbf{U}$ can be conveniently interpreted as the *principal components*, which provides a useful insight about the form of the principal subspace. Finally, as we shall see in the next section, the factorisation allows for a significant reduction in computational cost. Contrary to one—typically large—SVD being required per iteration in regularisation-based methods, in ORPCA two—typically small—SVDs are required per iteration, whereas in BRPCA and ROSL the SVDs are avoided entirely. On the other hand, the main drawback of factorisation is that it makes the problem non-convex. As a result, global optimality is no longer guaranteed and the result relies heavily upon initialisation. In practice however, as we shall see in the experiments of chapter 7, our suggested initialisations yield results that are comparable to the convex regularisation-based methods.

3.6.2 Asymptotic Computational Complexity

As all methods are iterative, the total execution time depends both on the rate of convergence of the optimisation algorithm used and on the computational cost per iteration. All optimisation algorithms used (i.e. APG, ALM, ALM with linearisation) are first-order methods and have similar rates of convergence (as discussed in chapter 2 and will be experimentally verified in chapter 7). In this section, we will focus on the other aspect of total execution time and we will compare the algorithms with respect to their computational cost per iteration.

Since we are mainly interested in how the algorithms scale to large matrices, our calculation of computational cost will be based on asymptotic complexity using the "big O" notation, as described in [23]. Based on [33], the various building blocks of the algorithms have the following computational complexities.

(i) **Matrix addition and scaling**. For matrices $\mathbf{X}, \mathbf{Y} \in \mathbb{R}^{m\times n}$ and scalar $a \in \mathbb{R}$, matrix addition $\mathbf{X}+\mathbf{Y}$ and scaling $a\mathbf{X}$ cost $\mathcal{O}(mn)$.

(ii) **Matrix multiplication**. For matrices $\mathbf{X} \in \mathbb{R}^{m\times n}$ and $\mathbf{Y} \in \mathbb{R}^{n\times k}$, matrix multiplication $\mathbf{XY}$ costs $\mathcal{O}(mnk)$.

(iii) **Matrix inversion**. For a square matrix $\mathbf{X} \in \mathbb{R}^{n\times n}$, computing $\mathbf{X}^{-1}$ costs $\mathcal{O}\left(n^3\right)$

(iv) **Singular Value Decomposition**. For a rectangular matrix $\mathbf{X} \in \mathbb{R}^{m\times n}$, computing the SVD $\mathbf{X} = \mathbf{USV}^T$ with $\mathbf{U} \in \mathbb{R}^{m\times \min(m,n)}$, $\mathbf{S} \in \mathbb{R}^{\min(m,n)\times \min(m,n)}$ and $\mathbf{V} \in \mathbb{R}^{n\times \min(m,n)}$, costs $\mathcal{O}\left(mn \min(m,n) + (\min(m,n))^3\right)$. See also [20] for a detailed calculation.

(v) **Shrinkage operator on matrices**. For a matrix $\mathbf{X} \in \mathbb{R}^{m\times n}$, the shrinkage operator $\mathcal{S}_a\{\mathbf{X}\}$ performs a constant-time operation on each matrix entry, therefore its asymptotic cost is $\mathcal{O}(mn)$.

(vi) **Singular value thresholding**. For a matrix $\mathbf{X} \in \mathbb{R}^{m\times n}$, the singular value thresholding $\mathcal{D}_a\{\mathbf{X}\}$ requires computing the SVD for a cost of $\mathcal{O}\left(mn \min(m,n) + (\min(m,n))^3\right)$, shrinking the singular values for a cost of $\mathcal{O}(\min(m,n))$ and "reassembling" the matrix for a cost of $\mathcal{O}(mn \min(m,n))$. The total cost is that of the SVD.

Based on the above, Table 3.1 shows the asymptotic computational complexity per iteration for each matrix algorithm, where the size of the input data matrix is $m\times n$ and the factorisation size is r. The following important observations can be made.

(i) **RPCA and IRPCA**. If $m \le n$, the complexity of RPCA becomes $\mathcal{O}\left(m^2 n + m^3\right)$ which is the same as IRPCA. If however $m > n$, which means that the dimensionality of the data points is larger than their number, a rather common case in practice, then the complexity of RPCA becomes $\mathcal{O}\left(mn^2 + n^3\right)$, which is more efficient than IRPCA.

Algorithm	**Convex?**	**Computational cost per iteration**
RPCA (apg)	Yes	$\mathcal{O}\left(mn\ \min(m,n) + (\min(m,n))^3\right)$
RPCA (alm)	Yes	$\mathcal{O}\left(mn\ \min(m,n) + (\min(m,n))^3\right)$
BRPCA	No	$\mathcal{O}\left(rmn + r^2(m+n) + r^3\right)$
IRPCA (sub)	Yes	$\mathcal{O}\left(m^2n + m^3\right)$
IRPCA (lin)	Yes	$\mathcal{O}\left(m^2n + m^3\right)$
ORPCA	No	$\mathcal{O}\left(rmn + r^2(m+n) + r^3\right)$
ROSL	No	$\mathcal{O}\left(r^2mn\right)$

Table 3.1: Convexity and asymptotic computational complexity per iteration of all matrix algorithms. The size of the input data matrix $\mathbf{X}$ is assumed to be $m \times n$. In the factorisation-based algorithms (i.e. BRPCA, ORPCA and ROSL), r is the factorisation size.

(ii) **Factorisation-based methods**. Since r is an upper bound on the rank of the low-rank component, it is reasonable to assume that always $r \leq \min(m,n)$. In most applications, r is typically much less than both m and n, therefore factorisation-based methods have much lower complexity than regularisation-based methods. Among factorisation-based methods, BRPCA and ORPCA have the same complexity (even though in practice BRPCA is faster since it avoids computing the SVD), whereas ROSL has a higher complexity. In case no upper bound in the rank of the low-rank component is required and r is set equal to $\min(m,n)$, the complexity of BRPCA and ORPCA becomes the same as RPCA. Finally, it is interesting to note that ROSL is the only algorithm whose complexity is a polynomial of degree 4; in all other cases, the complexity is a polynomial of degree 3.

In chapter 7, the total execution time of all algorithms will be evaluated experimentally, and the above theoretical results will be confirmed.

Chapter 4

Robust Low-Rank Modelling on Tensors

In chapter 3, we discussed methods for decomposing a data matrix into a low-rank and a sparse component. In this chapter, we will extend such methods to the more general case of *tensors*. Such extensions are interesting not only from a theoretical perspective, since tensors are the higher-order generalisation of matrices, but also from a practical one.

To better understand the practical significance of using tensors, consider a data matrix $\mathbf{X} = \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_n \end{bmatrix} \in \mathbb{R}^{m \times n}$, where each column $\mathbf{x}_i$ is a data point in $\mathbb{R}^m$. In a real-life application, such data points would represent *observations*, which can be measurements of any kind, signals, images, videos and so on. It is often the case that such observations have some sort of inherent higher-order structure, which is inevitably *destroyed* when they are naively represented as vectors $\mathbf{x}_i \in \mathbb{R}^m$. For instance, images have a spatial structure, which might be better represented as a matrix, whereas a video has both spatial and temporal structure, which might be better represented as a 3^{rd}-order tensor. Therefore, it becomes evident that extending robust low-rank modelling to tensors would enable the application of low-rank/sparse decomposition while preserving the higher-order structure that certain observations may have.

Robust low-rank modelling on tensors consists in decomposing a general N^{th}-order tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ as follows

$$\boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{A}} + \boldsymbol{\mathcal{E}} \tag{4.1}$$

where $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ is *low-rank* and $\boldsymbol{\mathcal{E}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ is *sparse*. Before we are in a position to start developing methods for performing such a decomposition, we need to describe what a low-rank and a sparse tensor is and define measures for the rank and the sparsity of tensors. Similarly to matrices, it is straightforward to use the ℓ_0-norm $\|\cdot\|_0$, i.e. the number of non-zero entries, as a measure for tensor sparsity. However, defining a measure for the tensor rank is a more complicated issue than it is with matrices. As we have seen in section 2.2.3, the rank of a tensor $\boldsymbol{\mathcal{X}}$ is the minimum number of rank-1 tensors that generate $\boldsymbol{\mathcal{X}}$ as their sum. Unfortunately, this definition of the tensor rank is NP-hard to compute [37] and therefore impractical to use.

In this work, our measure of the tensor rank is based instead on the n-rank of $\boldsymbol{\mathcal{X}}$, represented by $\operatorname{rank}_n(\boldsymbol{\mathcal{X}})$, which is the number of linearly independent n-mode fibres and is much easier to compute than $\operatorname{rank}(\boldsymbol{\mathcal{X}})$. In particular, our measure consists of a convex combination of all n-ranks, as follows

$$\sum_{i=1}^{N} a_i \operatorname{rank}_i(\boldsymbol{\mathcal{X}}) \quad \text{where} \quad a_i \geq 0 \quad \text{and} \quad \sum_{i=1}^{N} a_i = 1 \tag{4.2}$$

The above approach for measuring the tensor rank has recently become a common choice, mainly in the literature of low-rank tensor recovery (in works such as [29, 56, 85]). The reason for using a convex combination is that it makes it consistent with the rank of a matrix. Indeed, in the special case where $N = 2$, i.e. the tensor is a matrix, we have that $\text{rank}_1(\mathbf{X}) = \text{rank}_2(\mathbf{X}) = \text{rank}(\mathbf{X})$ and therefore

$$a_1\text{rank}_1(\mathbf{X}) + a_2\text{rank}_2(\mathbf{X}) = (a_1 + a_2)\,\text{rank}(\mathbf{X}) = \text{rank}(\mathbf{X}) \tag{4.3}$$

Having defined appropriate measures for tensor rank and sparsity, the low-rank/sparse decomposition problem can be formulated as follows

$$\min_{\boldsymbol{\mathcal{A}},\boldsymbol{\mathcal{E}}} \sum_{i=1}^{N} a_i \text{rank}_i(\boldsymbol{\mathcal{A}}) + \lambda \|\boldsymbol{\mathcal{E}}\|_0 \quad \text{s.t.} \quad \boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{A}} + \boldsymbol{\mathcal{E}} \tag{4.4}$$

As it is, this problem is no easier than the equivalent NP-hard matrix problem in 3.2. To make it tractable, we need to appropriately relax it. In the remaining of this chapter, we will consider tractable relaxations that follow the same ideas used in the matrix case and we will show how the methods of the matrix case can be extended to solve the new problems. We will conclude the chapter with a discussion over the advantages and disadvantages of each method, a comparison with the matrix case and an analysis of the computational cost of each algorithm.

4.1 Robust Principal Component Analysis

Matrix RPCA was formulated by replacing the rank and the ℓ_0-norm by their convex envelopes, the nuclear norm and the elementwise ℓ_1-norm respectively. In order to do the same for tensors, we need a good convex surrogate for equation (4.2). Since the convex envelope of $\text{rank}_n(\boldsymbol{\mathcal{X}}) = \text{rank}\left(\mathbf{X}_{[n]}\right)$ is $\left\|\mathbf{X}_{[n]}\right\|_*$, we will use the following convex surrogate for equation (4.2)

$$r(\boldsymbol{\mathcal{X}}) = \sum_{i=1}^{N} a_i \left\|\mathbf{X}_{[i]}\right\|_* \tag{4.5}$$

The use of $r(\boldsymbol{\mathcal{X}})$ as a convex surrogate for equation (4.2) has been studied by Signoretto et al. [76], who proved (Theorem 4 in their paper) that in the case where $a_1 = a_2 = \cdots = a_N = 1/N$, if $\mathcal{C}_f(\boldsymbol{\mathcal{X}})$ is the true convex envelope of $\sum_{i=1}^{N} a_i \text{rank}_i(\boldsymbol{\mathcal{X}})$ then $r(\boldsymbol{\mathcal{X}}) \leq \mathcal{C}_f(\boldsymbol{\mathcal{X}})$. In other words, $r(\boldsymbol{\mathcal{X}})$ is known to underestimate the true convex envelope. However, whether the equality is strict and therefore $r(\boldsymbol{\mathcal{X}})$ is indeed the true convex envelope or there exists a tighter convex surrogate than $r(\boldsymbol{\mathcal{X}})$ is not known yet.

Using the above relaxation, together with the elementwise ℓ_1-norm as a surrogate for the ℓ_0-norm, tensor RPCA can be described by the following optimisation problem

$$\min_{\boldsymbol{\mathcal{A}},\boldsymbol{\mathcal{E}}} \sum_{i=1}^{N} a_i \left\|\mathbf{A}_{[i]}\right\|_* + \lambda \|\boldsymbol{\mathcal{E}}\|_1 \quad \text{s.t.} \quad \boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{A}} + \boldsymbol{\mathcal{E}} \tag{4.6}$$

where $a_i \geq 0$ and $\sum_{i=1}^{N} a_i = 1$. This problem was first studied by Li et al. [49], who developed a penalty-based algorithm for solving it. However, their method contains a lot of parameters that require tuning and is far from the state-of-the-art optimisation algorithms. Tensor RPCA was also studied by Zhang et al. [90] in the context of image alignment. The problem they consider is more general than the above, because they also consider affine transformations on $\boldsymbol{\mathcal{X}}$ for the purposes of alignment. Their method is based on ALM and it contains an intermediate proximal gradient step. Recently, tensor RPCA was extensively studied by Goldfarb and Qin [32], who proposed both ALM and APG algorithms for solving it and studied their convergence. They also consider non-convex variations of tensor RPCA with user-specified rank constraints. Finally, Huang et al. [40] provided sufficient conditions under which tensor RPCA is *guaranteed* to *exactly recover* the low-rank component, thus extending the work of Candès et al. [18] to the tensor case.

In this thesis, we describe an ALM algorithm for solving the tensor RPCA problem, similar to the one proposed in [32]. The difficulty in solving this problem stems from the fact that the terms $\mathbf{A}_{[i]}$ are interdependent, i.e. they contain the same entries in different arrangements. To remove this interdependence, the problem is written in the following equivalent form

$$\min_{\{\mathbf{J}_i\},\boldsymbol{\mathcal{A}},\boldsymbol{\mathcal{E}}} \sum_{i=1}^{N} a_i \left\|\mathbf{J}_i\right\|_* + \lambda \left\|\boldsymbol{\mathcal{E}}\right\|_1 \quad \text{s.t.} \quad \begin{array}{l} \boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{A}} + \boldsymbol{\mathcal{E}} \\ \mathbf{A}_{[i]} = \mathbf{J}_i \quad \forall i \in \{1,2,\ldots,N\} \end{array} \tag{4.7}$$

The corresponding augmented Lagrangian is

$$\begin{aligned} \mathcal{L}_\mu\left(\{\mathbf{J}_i\},\boldsymbol{\mathcal{A}},\boldsymbol{\mathcal{E}},\boldsymbol{\mathcal{Y}}_1,\{\mathbf{Y}_{2i}\}\right) = &\sum_{i=1}^{N} a_i \left\|\mathbf{J}_i\right\|_* + \lambda \left\|\boldsymbol{\mathcal{E}}\right\|_1 + \langle \boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{A}} - \boldsymbol{\mathcal{E}}, \boldsymbol{\mathcal{Y}}_1 \rangle + \sum_{i=1}^{N} \left\langle \mathbf{A}_{[i]} - \mathbf{J}_i, \mathbf{Y}_{2i} \right\rangle \\ &+ \frac{\mu}{2} \left\|\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{A}} - \boldsymbol{\mathcal{E}}\right\|_F^2 + \sum_{i=1}^{N} \frac{\mu}{2} \left\|\mathbf{A}_{[i]} - \mathbf{J}_i\right\|_F^2 \end{aligned} \tag{4.8}$$

In the above, $\boldsymbol{\mathcal{Y}}_1$ and $\{\mathbf{Y}_{2i}\}$ are the Lagrange multipliers and $\mu > 0$ is the penalty parameter. Notice that $\boldsymbol{\mathcal{Y}}_1$ is a tensor of multipliers whereas $\{\mathbf{Y}_{2i}\}$ is a set of N matrices of multipliers. The minimisation of $\mathcal{L}_\mu$ with respect to each $\mathbf{J}_i$ for $i \in \{1,2,\ldots,N\}$ is done as follows

$$\begin{aligned} &\arg\min_{\mathbf{J}_i} \mathcal{L}_\mu\left(\{\mathbf{J}_i\},\boldsymbol{\mathcal{A}},\boldsymbol{\mathcal{E}},\boldsymbol{\mathcal{Y}}_1,\{\mathbf{Y}_{2i}\}\right) \\ &= \arg\min_{\mathbf{J}_i} a_i \left\|\mathbf{J}_i\right\|_* + \left\langle \mathbf{A}_{[i]} - \mathbf{J}_i, \mathbf{Y}_{2i} \right\rangle + \frac{\mu}{2} \left\|\mathbf{A}_{[i]} - \mathbf{J}_i\right\|_F^2 \\ &= \arg\min_{\mathbf{J}_i} a_i \mu^{-1} \left\|\mathbf{J}_i\right\|_* + \frac{1}{2} \left\|\mathbf{J}_i - \left(\mathbf{A}_{[i]} + \mu^{-1}\mathbf{Y}_{2i}\right)\right\|_F^2 \\ &= \mathcal{D}_{a_i\mu^{-1}}\left\{\mathbf{A}_{[i]} + \mu^{-1}\mathbf{Y}_{2i}\right\} \end{aligned} \tag{4.9}$$

Then, we minimise $\mathcal{L}_\mu$ with respect to $\boldsymbol{\mathcal{A}}$ as follows

$$\begin{aligned} &\arg\min_{\boldsymbol{\mathcal{A}}} \mathcal{L}_\mu\left(\{\mathbf{J}_i\},\boldsymbol{\mathcal{A}},\boldsymbol{\mathcal{E}},\boldsymbol{\mathcal{Y}}_1,\{\mathbf{Y}_{2i}\}\right) \\ &= \arg\min_{\boldsymbol{\mathcal{A}}} \langle \boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{A}} - \boldsymbol{\mathcal{E}}, \boldsymbol{\mathcal{Y}}_1 \rangle + \sum_{i=1}^{N} \left\langle \mathbf{A}_{[i]} - \mathbf{J}_i, \mathbf{Y}_{2i} \right\rangle + \frac{\mu}{2} \left\|\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{A}} - \boldsymbol{\mathcal{E}}\right\|_F^2 + \sum_{i=1}^{N} \frac{\mu}{2} \left\|\mathbf{A}_{[i]} - \mathbf{J}_i\right\|_F^2 \\ &= \arg\min_{\boldsymbol{\mathcal{A}}} \langle \boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{A}} - \boldsymbol{\mathcal{E}}, \boldsymbol{\mathcal{Y}}_1 \rangle + \sum_{i=1}^{N} \langle \boldsymbol{\mathcal{A}} - \boldsymbol{\mathcal{J}}_i, \boldsymbol{\mathcal{Y}}_{2i} \rangle + \frac{\mu}{2} \left\|\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{A}} - \boldsymbol{\mathcal{E}}\right\|_F^2 + \sum_{i=1}^{N} \frac{\mu}{2} \left\|\boldsymbol{\mathcal{A}} - \boldsymbol{\mathcal{J}}_i\right\|_F^2 \end{aligned} \tag{4.10}$$

Here we have considered the tensorised forms of each $\mathbf{J}_i$ and $\mathbf{Y}_{2i}$, represented by $\boldsymbol{\mathcal{J}}_i$ and $\boldsymbol{\mathcal{Y}}_{2i}$ respectively, in order to avoid the repeated matricisations of $\boldsymbol{\mathcal{A}}$. By *tensorised form* we simply mean the rearrangement of the matrix entries such that they form a tensor of the appropriate dimensions, which can be thought of as the reverse process of matricisation. The above can be easily minimised by forcing its derivative with respect to $\boldsymbol{\mathcal{A}}$ equal to zero. Using the following derivatives

$$\frac{\partial}{\partial \boldsymbol{\mathcal{A}}} \langle \boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{A}} - \boldsymbol{\mathcal{E}}, \boldsymbol{\mathcal{Y}}_1 \rangle = -\boldsymbol{\mathcal{Y}}_1 \tag{4.11}$$

$$\frac{\partial}{\partial \boldsymbol{\mathcal{A}}} \left(\sum_{i=1}^{N} \langle \boldsymbol{\mathcal{A}} - \boldsymbol{\mathcal{J}}_i, \boldsymbol{\mathcal{Y}}_{2i} \rangle \right) = \sum_{i=1}^{N} \boldsymbol{\mathcal{Y}}_{2i} \tag{4.12}$$

$$\frac{\partial}{\partial \boldsymbol{\mathcal{A}}} \left(\frac{\mu}{2} \left\|\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{A}} - \boldsymbol{\mathcal{E}}\right\|_F^2 \right) = \mu\left(\boldsymbol{\mathcal{A}} + \boldsymbol{\mathcal{E}} - \boldsymbol{\mathcal{X}}\right) \tag{4.13}$$

$$\frac{\partial}{\partial \boldsymbol{\mathcal{A}}} \left(\sum_{i=1}^{N} \frac{\mu}{2} \left\|\boldsymbol{\mathcal{A}} - \boldsymbol{\mathcal{J}}_i\right\|_F^2 \right) = \mu N \boldsymbol{\mathcal{A}} - \mu \sum_{i=1}^{N} \boldsymbol{\mathcal{J}}_i \tag{4.14}$$

the minimisation becomes

$$\begin{aligned}
&\frac{\partial \mathcal{L}_\mu}{\partial \boldsymbol{\mathcal{A}}} = \mathbf{0} && \Rightarrow \\
&(N+1)\,\boldsymbol{\mathcal{A}} - \boldsymbol{\mathcal{X}} + \boldsymbol{\mathcal{E}} - \mu^{-1}\boldsymbol{\mathcal{Y}}_1 - \sum_{i=1}^{N}\left(\boldsymbol{\mathcal{J}}_i - \mu^{-1}\boldsymbol{\mathcal{Y}}_{2i}\right) = \mathbf{0} && \Rightarrow \\
&\boldsymbol{\mathcal{A}} = (N+1)^{-1}\left[\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{E}} + \mu^{-1}\boldsymbol{\mathcal{Y}}_1 + \sum_{i=1}^{N}\left(\boldsymbol{\mathcal{J}}_i - \mu^{-1}\boldsymbol{\mathcal{Y}}_{2i}\right)\right] && (4.15)
\end{aligned}$$

Next, we minimise $\mathcal{L}_\mu$ with respect to $\boldsymbol{\mathcal{E}}$ as follows

$$\begin{aligned}
&\arg\min_{\boldsymbol{\mathcal{E}}} \mathcal{L}_\mu\left(\{\mathbf{J}_i\}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{E}}, \boldsymbol{\mathcal{Y}}_1, \{\mathbf{Y}_{2i}\}\right) \\
&= \arg\min_{\boldsymbol{\mathcal{E}}} \lambda \left\|\boldsymbol{\mathcal{E}}\right\|_1 + \langle \boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{A}} - \boldsymbol{\mathcal{E}}, \boldsymbol{\mathcal{Y}}_1\rangle + \frac{\mu}{2}\left\|\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{A}} - \boldsymbol{\mathcal{E}}\right\|_F^2 \\
&= \arg\min_{\boldsymbol{\mathcal{E}}} \lambda\mu^{-1}\left\|\boldsymbol{\mathcal{E}}\right\|_1 + \frac{1}{2}\left\|\boldsymbol{\mathcal{E}} - \left(\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{A}} + \mu^{-1}\boldsymbol{\mathcal{Y}}_1\right)\right\|_F^2 \\
&= \mathcal{S}_{\lambda\mu^{-1}}\left\{\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{A}} + \mu^{-1}\boldsymbol{\mathcal{Y}}_1\right\} \qquad (4.16)
\end{aligned}$$

Finally, the full ALM algorithm for tensor RPCA is presented in Algorithm 4.1.

Algorithm 4.1: Tensor Robust PCA

```
   Input: Data tensor $\boldsymbol{\mathcal{X}}$, regulariser $\lambda > 0$, weights $\{a_i\}$
   Initialise: $\mathbf{J}_i^0 = \mathbf{0}$, $\boldsymbol{\mathcal{A}}_0 = \mathbf{0}$, $\boldsymbol{\mathcal{E}}_0 = \mathbf{0}$, $\boldsymbol{\mathcal{Y}}_1^0 = \mathbf{0}$, $\mathbf{Y}_{2i}^0 = \mathbf{0}$, $\mu_0 > 0$, $k = 0$
1  while not converged do
2      for $i \in \{1, 2, \ldots, N\}$ do
3          $\mathbf{J}_i^{k+1} = \mathcal{D}_{a_i\mu_k^{-1}}\left\{(\mathbf{A}_k)_{[i]} + \mu_k^{-1}\mathbf{Y}_{2i}^k\right\}$
4      end for
5      $\boldsymbol{\mathcal{A}}_{k+1} = (N+1)^{-1}\left[\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{E}}_k + \mu_k^{-1}\boldsymbol{\mathcal{Y}}_1^k + \sum_{i=1}^N\left(\boldsymbol{\mathcal{J}}_i^{k+1} - \mu_k^{-1}\boldsymbol{\mathcal{Y}}_{2i}^k\right)\right]$
6      $\boldsymbol{\mathcal{E}}_{k+1} = \mathcal{S}_{\lambda\mu_k^{-1}}\left\{\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{A}}_{k+1} + \mu_k^{-1}\boldsymbol{\mathcal{Y}}_1^k\right\}$
7      $\boldsymbol{\mathcal{Y}}_1^{k+1} = \boldsymbol{\mathcal{Y}}_1^k + \mu_k\left(\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{A}}_{k+1} - \boldsymbol{\mathcal{E}}_{k+1}\right)$
8      for $i \in \{1, 2, \ldots, N\}$ do
9          $\mathbf{Y}_{2i}^{k+1} = \mathbf{Y}_{2i}^k + \mu_k\left((\mathbf{A}_{k+1})_{[i]} - \mathbf{J}_i^{k+1}\right)$
10     end for
11     $\mu_{k+1} = \min\left(\rho\mu_k, \mu_{max}\right)$
12     $k \leftarrow k + 1$
13 end while
   Output: Low-rank component $\boldsymbol{\mathcal{A}}_k$, sparse component $\boldsymbol{\mathcal{E}}_k$
```

4.2 Bilinear Robust Principal Component Analysis

On matrices, BRPCA is a technique to avoid the nuclear norm term in the RPCA formulation, and hence the need for computing the SVD, using the following (Theorem 3.1)

$$\left\|\mathbf{A}\right\|_* = \min_{\mathbf{A},\mathbf{U},\mathbf{V}} \frac{1}{2}\left(\left\|\mathbf{U}\right\|_F^2 + \left\|\mathbf{V}\right\|_F^2\right) \quad \text{s.t.} \quad \mathbf{A} = \mathbf{UV} \qquad (4.17)$$

Using the same idea, in this section we will extend BRPCA to tensors. Recall the following equivalent formulation of tensor RPCA

$$\min_{\{\mathbf{J}_i\},\boldsymbol{\mathcal{A}},\boldsymbol{\mathcal{E}}} \sum_{i=1}^{N} a_i \|\mathbf{J}_i\|_* + \lambda \|\boldsymbol{\mathcal{E}}\|_1 \quad \text{s.t.} \quad \begin{array}{l} \boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{A}} + \boldsymbol{\mathcal{E}} \\ \mathbf{A}_{[i]} = \mathbf{J}_i \quad \forall i \in \{1,2,\ldots,N\} \end{array} \tag{4.18}$$

If we rewrite each $\mathbf{J}_i$ as $\mathbf{J}_i = \mathbf{U}_i\mathbf{V}_i$ and use the fact that $\|\mathbf{J}_i\|_* \leq \frac{1}{2}\left(\|\mathbf{U}_i\|_F^2 + \|\mathbf{V}_i\|_F^2\right)$, we can relax tensor RPCA as the following

$$\min_{\{\mathbf{U}_i\},\{\mathbf{V}_i\},\boldsymbol{\mathcal{A}},\boldsymbol{\mathcal{E}}} \sum_{i=1}^{N} \frac{a_i}{2}\left(\|\mathbf{U}_i\|_F^2 + \|\mathbf{V}_i\|_F^2\right) + \lambda \|\boldsymbol{\mathcal{E}}\|_1 \quad \text{s.t.} \quad \begin{array}{l} \boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{A}} + \boldsymbol{\mathcal{E}} \\ \mathbf{A}_{[i]} = \mathbf{U}_i\mathbf{V}_i \quad \forall i \in \{1,2,\ldots,N\} \end{array} \tag{4.19}$$

The above optimisation problem defines tensor BRPCA. Notice that it contains no nuclear norm terms. In the following, we will show how it can be solved using ALM.

We start by forming the augmented Lagrangian, which can be written as follows

$$\begin{aligned} \mathcal{L}_\mu\left(\{\mathbf{U}_i\},\{\mathbf{V}_i\},\boldsymbol{\mathcal{A}},\boldsymbol{\mathcal{E}},\boldsymbol{\mathcal{Y}}_1,\{\mathbf{Y}_{2i}\}\right) &= \sum_{i=1}^{N} \frac{a_i}{2}\left(\|\mathbf{U}_i\|_F^2 + \|\mathbf{V}_i\|_F^2\right) + \lambda \|\boldsymbol{\mathcal{E}}\|_1 + \langle \boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{A}} - \boldsymbol{\mathcal{E}}, \boldsymbol{\mathcal{Y}}_1 \rangle \\ &+ \sum_{i=1}^{N} \left\langle \mathbf{A}_{[i]} - \mathbf{U}_i\mathbf{V}_i, \mathbf{Y}_{2i} \right\rangle + \frac{\mu}{2}\|\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{A}} - \boldsymbol{\mathcal{E}}\|_F^2 + \sum_{i=1}^{N} \frac{\mu}{2}\left\|\mathbf{A}_{[i]} - \mathbf{U}_i\mathbf{V}_i\right\|_F^2 \end{aligned} \tag{4.20}$$

We will first minimise $\mathcal{L}_\mu$ with respect to each $\mathbf{U}_i$ for $i \in \{1,2,\ldots,N\}$ as follows

$$\begin{aligned} &\arg\min_{\mathbf{U}_i} \mathcal{L}_\mu\left(\{\mathbf{U}_i\},\{\mathbf{V}_i\},\boldsymbol{\mathcal{A}},\boldsymbol{\mathcal{E}},\boldsymbol{\mathcal{Y}}_1,\{\mathbf{Y}_{2i}\}\right) \\ &= \arg\min_{\mathbf{U}_i} \frac{a_i}{2}\|\mathbf{U}_i\|_F^2 + \left\langle \mathbf{A}_{[i]} - \mathbf{U}_i\mathbf{V}_i, \mathbf{Y}_{2i} \right\rangle + \frac{\mu}{2}\left\|\mathbf{A}_{[i]} - \mathbf{U}_i\mathbf{V}_i\right\|_F^2 \end{aligned} \tag{4.21}$$

Since all terms in the above objective are differentiable, it suffices to set its derivative with respect to $\mathbf{U}_i$ to zero.

$$\begin{aligned} &\frac{\partial}{\partial \mathbf{U}_i}\left(\frac{a_i}{2}\|\mathbf{U}_i\|_F^2 + \left\langle \mathbf{A}_{[i]} - \mathbf{U}_i\mathbf{V}_i, \mathbf{Y}_{2i} \right\rangle + \frac{\mu}{2}\left\|\mathbf{A}_{[i]} - \mathbf{U}_i\mathbf{V}_i\right\|_F^2\right) = \mathbf{0} && \Rightarrow \\ &a_i\mathbf{U}_i - \mathbf{Y}_{2i}\mathbf{V}_i^T - \mu\left(\mathbf{A}_{[i]} - \mathbf{U}_i\mathbf{V}_i\right)\mathbf{V}_i^T = \mathbf{0} && \Rightarrow \\ &\mathbf{U}_i\left(\mathbf{V}_i\mathbf{V}_i^T + a_i\mu^{-1}\mathbf{I}\right) - \left(\mathbf{A}_{[i]} + \mu^{-1}\mathbf{Y}_{2i}\right)\mathbf{V}_i^T = \mathbf{0} && \Rightarrow \\ &\mathbf{U}_i = \left(\mathbf{A}_{[i]} + \mu^{-1}\mathbf{Y}_{2i}\right)\mathbf{V}_i^T\left(\mathbf{V}_i\mathbf{V}_i^T + a_i\mu^{-1}\mathbf{I}\right)^{-1} \end{aligned} \tag{4.22}$$

Similarly, we minimise $\mathcal{L}_\mu$ with respect to each $\mathbf{V}_i$ for $i \in \{1,2,\ldots,N\}$ as follows

$$\begin{aligned} &\arg\min_{\mathbf{V}_i} \mathcal{L}_\mu\left(\{\mathbf{U}_i\},\{\mathbf{V}_i\},\boldsymbol{\mathcal{A}},\boldsymbol{\mathcal{E}},\boldsymbol{\mathcal{Y}}_1,\{\mathbf{Y}_{2i}\}\right) \\ &= \arg\min_{\mathbf{V}_i} \frac{a_i}{2}\|\mathbf{V}_i\|_F^2 + \left\langle \mathbf{A}_{[i]} - \mathbf{U}_i\mathbf{V}_i, \mathbf{Y}_{2i} \right\rangle + \frac{\mu}{2}\left\|\mathbf{A}_{[i]} - \mathbf{U}_i\mathbf{V}_i\right\|_F^2 \end{aligned} \tag{4.23}$$

Setting the derivative of the objective with respect to $\mathbf{V}_i$ to zero yields

$$\begin{aligned} &\frac{\partial}{\partial \mathbf{V}_i}\left(\frac{a_i}{2}\|\mathbf{V}_i\|_F^2 + \left\langle \mathbf{A}_{[i]} - \mathbf{U}_i\mathbf{V}_i, \mathbf{Y}_{2i} \right\rangle + \frac{\mu}{2}\left\|\mathbf{A}_{[i]} - \mathbf{U}_i\mathbf{V}_i\right\|_F^2\right) = \mathbf{0} && \Rightarrow \\ &a_i\mathbf{V}_i - \mathbf{U}_i^T\mathbf{Y}_{2i} - \mu\mathbf{U}_i^T\left(\mathbf{A}_{[i]} - \mathbf{U}_i\mathbf{V}_i\right) = \mathbf{0} && \Rightarrow \\ &\left(\mathbf{U}_i^T\mathbf{U}_i + a_i\mu^{-1}\mathbf{I}\right)\mathbf{V}_i - \mathbf{U}_i^T\left(\mathbf{A}_{[i]} + \mu^{-1}\mathbf{Y}_{2i}\right) = \mathbf{0} && \Rightarrow \\ &\mathbf{V}_i = \left(\mathbf{U}_i^T\mathbf{U}_i + a_i\mu^{-1}\mathbf{I}\right)^{-1}\mathbf{U}_i^T\left(\mathbf{A}_{[i]} + \mu^{-1}\mathbf{Y}_{2i}\right) \end{aligned} \tag{4.24}$$

The minimisation of $\mathcal{L}_\mu$ with respect to $\boldsymbol{\mathcal{A}}$ can be easily obtained by the equivalent step of tensor RPCA. Let $\mathbf{J}_i = \mathbf{U}_i\mathbf{V}_i$ and assume $\boldsymbol{\mathcal{J}}_i$ to be the appropriate tensorisation of $\mathbf{J}_i$ (i.e. the inverse of the i-mode factorisation of $\boldsymbol{\mathcal{A}}$). Then the minimisation becomes as follows

$$\begin{aligned}
&\arg\min_{\boldsymbol{\mathcal{A}}} \mathcal{L}_\mu\left(\{\mathbf{U}_i\},\{\mathbf{V}_i\},\boldsymbol{\mathcal{A}},\boldsymbol{\mathcal{E}},\boldsymbol{\mathcal{Y}}_1,\{\mathbf{Y}_{2i}\}\right)\\
&= \arg\min_{\boldsymbol{\mathcal{A}}} \langle \boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{A}} - \boldsymbol{\mathcal{E}}, \boldsymbol{\mathcal{Y}}_1\rangle + \sum_{i=1}^{N}\left\langle \mathbf{A}_{[i]} - \mathbf{J}_i, \mathbf{Y}_{2i}\right\rangle + \frac{\mu}{2}\|\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{A}} - \boldsymbol{\mathcal{E}}\|_F^2 + \sum_{i=1}^{N}\frac{\mu}{2}\left\|\mathbf{A}_{[i]} - \mathbf{J}_i\right\|_F^2\\
&= \arg\min_{\boldsymbol{\mathcal{A}}} \langle \boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{A}} - \boldsymbol{\mathcal{E}}, \boldsymbol{\mathcal{Y}}_1\rangle + \sum_{i=1}^{N}\langle \boldsymbol{\mathcal{A}} - \boldsymbol{\mathcal{J}}_i, \boldsymbol{\mathcal{Y}}_{2i}\rangle + \frac{\mu}{2}\|\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{A}} - \boldsymbol{\mathcal{E}}\|_F^2 + \sum_{i=1}^{N}\frac{\mu}{2}\|\boldsymbol{\mathcal{A}} - \boldsymbol{\mathcal{J}}_i\|_F^2
\end{aligned} \tag{4.25}$$

Notice that the above is identical to problem (4.10) and therefore its solution is given by

$$\boldsymbol{\mathcal{A}} = (N+1)^{-1}\left[\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{E}} + \mu^{-1}\boldsymbol{\mathcal{Y}}_1 + \sum_{i=1}^{N}\left(\boldsymbol{\mathcal{J}}_i - \mu^{-1}\boldsymbol{\mathcal{Y}}_{2i}\right)\right] \tag{4.26}$$

Finally, minimising $\mathcal{L}_\mu$ with respect to $\boldsymbol{\mathcal{E}}$ yields

$$\begin{aligned}
&\arg\min_{\boldsymbol{\mathcal{E}}} \mathcal{L}_\mu\left(\{\mathbf{U}_i\},\{\mathbf{V}_i\},\boldsymbol{\mathcal{A}},\boldsymbol{\mathcal{E}},\boldsymbol{\mathcal{Y}}_1,\{\mathbf{Y}_{2i}\}\right)\\
&= \arg\min_{\boldsymbol{\mathcal{E}}} \lambda\|\boldsymbol{\mathcal{E}}\|_1 + \langle \boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{A}} - \boldsymbol{\mathcal{E}}, \boldsymbol{\mathcal{Y}}_1\rangle + \frac{\mu}{2}\|\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{A}} - \boldsymbol{\mathcal{E}}\|_F^2\\
&= \arg\min_{\boldsymbol{\mathcal{E}}} \lambda\mu^{-1}\|\boldsymbol{\mathcal{E}}\|_1 + \frac{1}{2}\left\|\boldsymbol{\mathcal{E}} - \left(\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{A}} + \mu^{-1}\boldsymbol{\mathcal{Y}}_1\right)\right\|_F^2\\
&= \mathcal{S}_{\lambda\mu^{-1}}\left\{\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{A}} + \mu^{-1}\boldsymbol{\mathcal{Y}}_1\right\}
\end{aligned} \tag{4.27}$$

The full algorithm for tensor BRPCA is described in detail in Algorithm 4.2. Notice that the factorisation sizes $\{r_i\}$ are needed as input. Here, r_i is the number of columns of $\mathbf{U}_i$ and the number of rows of $\mathbf{V}_i$, for each $i \in \{1, 2, \dots, N\}$. It is important to understand the significance of r_i, since

$$\operatorname{rank}_i(\boldsymbol{\mathcal{A}}) = \operatorname{rank}\left(\mathbf{A}_{[i]}\right) = \operatorname{rank}(\mathbf{U}_i\mathbf{V}_i) \leq \operatorname{rank}(\mathbf{U}_i) \leq r_i \tag{4.28}$$

which means that r_i serves as a controllable upper bound for the n-rank of $\boldsymbol{\mathcal{A}}$. Furthermore, notice that in our implementation we initialise the entries of each $\mathbf{U}_i$ and of $\boldsymbol{\mathcal{A}}$ uniformly at random in the interval $[0, 1]$. Initialisation is important, since the problem is non-convex and therefore initialisation determines the final solution.

4.3 Inductive Robust Principal Component Analysis

The motivation for matrix IRPCA was the need for an *online equivalent* of RPCA. In order to achieve this, the low-rank component $\mathbf{A}$ was written as $\mathbf{A} = \mathbf{PX}$ and the algorithm was modified to learn the projection matrix $\mathbf{P}$ instead of the low-rank component $\mathbf{A}$. In this section, we will extend the IRPCA problem to tensors and develop algorithms to solve it.

The objective of matrix IRPCA is to learn a projection matrix $\mathbf{P} \in \mathbb{R}^{m\times m}$ which projects the columns of data matrix $\mathbf{X} \in \mathbb{R}^{m\times n}$ onto the principal subspace. In the case of an N^{th}-order data tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1\times I_2\times\cdots\times I_N}$, the equivalent objective becomes to learn a projection matrix $\mathbf{P}_i \in \mathbb{R}^{I_i\times I_i}$ for *each* mode of $\boldsymbol{\mathcal{X}}$; each projection matrix $\mathbf{P}_i$ will project the i-mode fibres of $\boldsymbol{\mathcal{X}}$, for $i \in \{1, 2, \dots, N\}$. Therefore, we shall rewrite the low-rank component $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{I_1\times I_2\times\cdots\times I_N}$ as follows

$$\boldsymbol{\mathcal{A}} = \boldsymbol{\mathcal{X}} \times_{i=1}^{N} \mathbf{P}_i \tag{4.29}$$

Algorithm 4.2: Tensor Bilinear Robust PCA

```
   Input: Data tensor $\boldsymbol{\mathcal{X}}$, regulariser $\lambda > 0$, weights $\{a_i\}$, factorisation sizes $\{r_i\}$
   Initialise: $\mathbf{U}_i^0 = \text{rand}$, $\boldsymbol{\mathcal{A}}_0 = \text{rand}$, $\boldsymbol{\mathcal{E}}_0 = \mathbf{0}$, $\boldsymbol{\mathcal{Y}}_1^0 = \mathbf{0}$, $\mathbf{Y}_{2i}^0 = \mathbf{0}$, $\mu_0 > 0$, $k = 0$
 1 while not converged do
 2     for $i \in \{1, 2, \dots, N\}$ do
 3         $\mathbf{V}_i^{k+1} = \left(\left(\mathbf{U}_i^k\right)^T \mathbf{U}_i^k + a_i\mu_k^{-1}\mathbf{I}\right)^{-1} \left(\mathbf{U}_i^k\right)^T \left((\mathbf{A}_k)_{[i]} + \mu_k^{-1}\mathbf{Y}_{2i}^k\right)$
 4         $\mathbf{U}_i^{k+1} = \left((\mathbf{A}_k)_{[i]} + \mu_k^{-1}\mathbf{Y}_{2i}^k\right) \left(\mathbf{V}_i^{k+1}\right)^T \left(\mathbf{V}_i^{k+1} \left(\mathbf{V}_i^{k+1}\right)^T + a_i\mu_k^{-1}\mathbf{I}\right)^{-1}$
 5         $\mathbf{J}_i^{k+1} = \mathbf{U}_i^{k+1}\mathbf{V}_i^{k+1}$
 6     end for
 7     $\boldsymbol{\mathcal{A}}_{k+1} = (N+1)^{-1} \left[\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{E}}_k + \mu_k^{-1}\boldsymbol{\mathcal{Y}}_1^k + \sum_{i=1}^N \left(\boldsymbol{\mathcal{J}}_i^{k+1} - \mu_k^{-1}\boldsymbol{\mathcal{Y}}_{2i}^k\right)\right]$
 8     $\boldsymbol{\mathcal{E}}_{k+1} = \mathcal{S}_{\lambda\mu_k^{-1}} \left\{\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{A}}_{k+1} + \mu_k^{-1}\boldsymbol{\mathcal{Y}}_1^k\right\}$
 9     $\boldsymbol{\mathcal{Y}}_1^{k+1} = \boldsymbol{\mathcal{Y}}_1^k + \mu_k \left(\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{A}}_{k+1} - \boldsymbol{\mathcal{E}}_{k+1}\right)$
10     for $i \in \{1, 2, \dots, N\}$ do
11         $\mathbf{Y}_{2i}^{k+1} = \mathbf{Y}_{2i}^k + \mu_k \left((\mathbf{A}_{k+1})_{[i]} - \mathbf{J}_i^{k+1}\right)$
12     end for
13     $\mu_{k+1} = \min\left(\rho\mu_k, \mu_{max}\right)$
14     $k \leftarrow k + 1$
15 end while
   Output: Low-rank component $\boldsymbol{\mathcal{A}}_k$, sparse component $\boldsymbol{\mathcal{E}}_k$
```

By Theorem 2.7, the following holds for the n-ranks of $\boldsymbol{\mathcal{A}}$

$$\text{rank}_i\left(\boldsymbol{\mathcal{A}}\right) \leq \text{rank}\left(\mathbf{P}_i\right) \tag{4.30}$$

which means that minimising $\text{rank}\left(\mathbf{P}_i\right)$ is equivalent to minimising an upper bound on $\text{rank}_i\left(\boldsymbol{\mathcal{A}}\right)$.

Based on the discussion above, we define tensor IRPCA as the following optimisation problem

$$\min_{\{\mathbf{P}_i\},\boldsymbol{\mathcal{E}}} \sum_{i=1}^N a_i \left\|\mathbf{P}_i\right\|_* + \lambda \left\|\boldsymbol{\mathcal{E}}\right\|_1 \quad \text{s.t.} \quad \boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{X}} \times_{i=1}^N \mathbf{P}_i + \boldsymbol{\mathcal{E}} \tag{4.31}$$

In the following subsections, we will derive two algorithms for solving the above problem. The first is based on ALM with substitution of variables and the second is based on ALM with a linearisation step.

4.3.1 Solution Based on Substitution

The IRPCA problem can equivalently be written as follows

$$\min_{\{\mathbf{J}_i\},\{\mathbf{P}_i\},\boldsymbol{\mathcal{E}}} \sum_{i=1}^N a_i \left\|\mathbf{J}_i\right\|_* + \lambda \left\|\boldsymbol{\mathcal{E}}\right\|_1 \quad \text{s.t.} \quad \begin{array}{l} \boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{X}} \times_{i=1}^N \mathbf{P}_i + \boldsymbol{\mathcal{E}} \\ \mathbf{P}_i = \mathbf{J}_i \quad \forall i \in \{1, 2, \dots, N\} \end{array} \tag{4.32}$$

We will derive an ALM algorithm for solving the above. The augmented Lagrangian can be written as

$$\begin{aligned}\mathcal{L}_\mu\left(\{\mathbf{J}_i\}, \{\mathbf{P}_i\}, \boldsymbol{\mathcal{E}}, \boldsymbol{\mathcal{Y}}_1, \{\mathbf{Y}_{2i}\}\right) &= \sum_{i=1}^N a_i \left\|\mathbf{J}_i\right\|_* + \lambda \left\|\boldsymbol{\mathcal{E}}\right\|_1 + \left\langle \boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{X}} \times_{i=1}^N \mathbf{P}_i - \boldsymbol{\mathcal{E}}, \boldsymbol{\mathcal{Y}}_1 \right\rangle \\ &+ \sum_{i=1}^N \left\langle \mathbf{P}_i - \mathbf{J}_i, \mathbf{Y}_{2i} \right\rangle + \frac{\mu}{2} \left\|\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{X}} \times_{i=1}^N \mathbf{P}_i - \boldsymbol{\mathcal{E}}\right\|_F^2 + \sum_{i=1}^N \frac{\mu}{2} \left\|\mathbf{P}_i - \mathbf{J}_i\right\|_F^2\end{aligned} \tag{4.33}$$

We will minimise $\mathcal{L}_\mu$ with respect to each parameter separately, keeping all others fixed. We start by minimising $\mathcal{L}_\mu$ with respect to each $\mathbf{J}_i$ for $i \in \{1, 2, \dots, N\}$ as follows

$$\begin{aligned}
&\arg\min_{\mathbf{J}_i} \mathcal{L}_\mu\left(\{\mathbf{J}_i\}, \{\mathbf{P}_i\}, \boldsymbol{\mathcal{E}}, \boldsymbol{\mathcal{Y}}_1, \{\mathbf{Y}_{2i}\}\right) \\
&= \arg\min_{\mathbf{J}_i} a_i \left\|\mathbf{J}_i\right\|_* + \langle \mathbf{P}_i - \mathbf{J}_i, \mathbf{Y}_{2i}\rangle + \frac{\mu}{2}\left\|\mathbf{P}_i - \mathbf{J}_i\right\|_F^2 \\
&= \arg\min_{\mathbf{J}_i} a_i \mu^{-1} \left\|\mathbf{J}_i\right\|_* + \frac{1}{2}\left\|\mathbf{J}_i - \left(\mathbf{P}_i + \mu^{-1}\mathbf{Y}_{2i}\right)\right\|_F^2 \\
&= \mathcal{D}_{a_i\mu^{-1}}\left\{\mathbf{P}_i + \mu^{-1}\mathbf{Y}_{2i}\right\}
\end{aligned} \tag{4.34}$$

We continue by minimising $\mathcal{L}_\mu$ with respect to each $\mathbf{P}_i$ for $i \in \{1, 2, \dots, N\}$ as follows

$$\begin{aligned}
&\arg\min_{\mathbf{P}_i} \mathcal{L}_\mu\left(\{\mathbf{J}_i\}, \{\mathbf{P}_i\}, \boldsymbol{\mathcal{E}}, \boldsymbol{\mathcal{Y}}_1, \{\mathbf{Y}_{2i}\}\right) \\
&= \arg\min_{\mathbf{P}_i} \left\langle \boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{X}} \times_{j=1}^N \mathbf{P}_j - \boldsymbol{\mathcal{E}}, \boldsymbol{\mathcal{Y}}_1\right\rangle + \langle \mathbf{P}_i - \mathbf{J}_i, \mathbf{Y}_{2i}\rangle \\
&\qquad\qquad + \frac{\mu}{2}\left\|\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{X}} \times_{j=1}^N \mathbf{P}_j - \boldsymbol{\mathcal{E}}\right\|_F^2 + \frac{\mu}{2}\left\|\mathbf{P}_i - \mathbf{J}_i\right\|_F^2 \\
&= \arg\min_{\mathbf{P}_i} \left\langle \mathbf{X}_{[i]} - \mathbf{P}_i\mathbf{X}_i - \mathbf{E}_{[i]}, \left(\mathbf{Y}_1\right)_{[i]}\right\rangle + \langle \mathbf{P}_i - \mathbf{J}_i, \mathbf{Y}_{2i}\rangle \\
&\qquad\qquad + \frac{\mu}{2}\left\|\mathbf{X}_{[i]} - \mathbf{P}_i\mathbf{X}_i - \mathbf{E}_{[i]}\right\|_F^2 + \frac{\mu}{2}\left\|\mathbf{P}_i - \mathbf{J}_i\right\|_F^2
\end{aligned} \tag{4.35}$$

where

$$\mathbf{X}_i = \mathbf{X}_{[i]}\left(\bigotimes_{\substack{j=N \\ j\neq i}}^{1} \mathbf{P}_j^T\right) = \left(\boldsymbol{\mathcal{X}} \times_{j\neq i} \mathbf{P}_j\right)_{[i]} \tag{4.36}$$

Notice that all the terms in the above objective are differentiable, therefore we can simply minimise it by forcing its derivative with respect to $\mathbf{P}_i$ equal to zero. Using the derivatives

$$\frac{\partial}{\partial \mathbf{P}_i}\left\langle \mathbf{X}_{[i]} - \mathbf{P}_i\mathbf{X}_i - \mathbf{E}_{[i]}, \left(\mathbf{Y}_1\right)_{[i]}\right\rangle = -\left(\mathbf{Y}_1\right)_{[i]}\mathbf{X}_i^T \tag{4.37}$$

$$\frac{\partial}{\partial \mathbf{P}_i}\langle \mathbf{P}_i - \mathbf{J}_i, \mathbf{Y}_{2i}\rangle = \mathbf{Y}_{2i} \tag{4.38}$$

$$\frac{\partial}{\partial \mathbf{P}_i}\left(\frac{\mu}{2}\left\|\mathbf{X}_{[i]} - \mathbf{P}_i\mathbf{X}_i - \mathbf{E}_{[i]}\right\|_F^2\right) = -\mu\left(\mathbf{X}_{[i]} - \mathbf{P}_i\mathbf{X}_i - \mathbf{E}_{[i]}\right)\mathbf{X}_i^T \tag{4.39}$$

$$\frac{\partial}{\partial \mathbf{P}_i}\left(\frac{\mu}{2}\left\|\mathbf{P}_i - \mathbf{J}_i\right\|_F^2\right) = \mu\left(\mathbf{P}_i - \mathbf{J}_i\right) \tag{4.40}$$

we get the following result

$$\begin{aligned}
&\frac{\partial \mathcal{L}_\mu}{\partial \mathbf{P}_i} = \mathbf{0} && \Rightarrow \\
&\mathbf{P}_i\left(\mathbf{X}_i\mathbf{X}_i^T + \mathbf{I}\right) - \left(\mathbf{X}_{[i]} - \mathbf{E}_{[i]} + \mu^{-1}\left(\mathbf{Y}_1\right)_{[i]}\right)\mathbf{X}_i^T - \mathbf{J}_i + \mu^{-1}\mathbf{Y}_{2i} = \mathbf{0} && \Rightarrow \\
&\mathbf{P}_i = \left[\left(\mathbf{X}_{[i]} - \mathbf{E}_{[i]} + \mu^{-1}\left(\mathbf{Y}_1\right)_{[i]}\right)\mathbf{X}_i^T + \mathbf{J}_i - \mu^{-1}\mathbf{Y}_{2i}\right]\left(\mathbf{X}_i\mathbf{X}_i^T + \mathbf{I}\right)^{-1}
\end{aligned} \tag{4.41}$$

Next, we minimise $\mathcal{L}_\mu$ with respect to $\boldsymbol{\mathcal{E}}$ as follows

$$\begin{aligned}
&\arg\min_{\boldsymbol{\mathcal{E}}} \mathcal{L}_\mu\left(\{\mathbf{J}_i\}, \{\mathbf{P}_i\}, \boldsymbol{\mathcal{E}}, \boldsymbol{\mathcal{Y}}_1, \{\mathbf{Y}_{2i}\}\right) \\
&= \arg\min_{\boldsymbol{\mathcal{E}}} \lambda\left\|\boldsymbol{\mathcal{E}}\right\|_1 + \left\langle \boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{X}} \times_{i=1}^N \mathbf{P}_i - \boldsymbol{\mathcal{E}}, \boldsymbol{\mathcal{Y}}_1\right\rangle + \frac{\mu}{2}\left\|\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{X}} \times_{i=1}^N \mathbf{P}_i - \boldsymbol{\mathcal{E}}\right\|_F^2 \\
&= \arg\min_{\boldsymbol{\mathcal{E}}} \lambda\mu^{-1}\left\|\boldsymbol{\mathcal{E}}\right\|_1 + \frac{1}{2}\left\|\boldsymbol{\mathcal{E}} - \left(\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{X}} \times_{i=1}^N \mathbf{P}_i + \mu^{-1}\boldsymbol{\mathcal{Y}}_1\right)\right\|_F^2 \\
&= \mathcal{S}_{\lambda\mu^{-1}}\left\{\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{X}} \times_{i=1}^N \mathbf{P}_i + \mu^{-1}\boldsymbol{\mathcal{Y}}_1\right\}
\end{aligned} \tag{4.42}$$

The substitution-based tensor IRPCA algorithm is presented in detail in Algorithm 4.3. Note that unlike matrix IRPCA, tensor IRPCA is a non-convex optimisation problem, therefore initialisation is important. In our implementation, we initialise projection matrices $\mathbf{P}_i$ as follows

$$\mathbf{P}_i^0 = \mathbf{U}_i \mathbf{U}_i^T \tag{4.43}$$

where $\mathbf{U}_i$ is formed by the first r_i left singular vectors of $\mathbf{X}_{[i]}$. Here, $\{r_i\}$ are pre-specified parameters. Empirically, we have found this "warm" initialisation to work best.

Algorithm 4.3: Tensor Inductive Robust PCA via substitution

```
   Input: Data tensor $\boldsymbol{\mathcal{X}}$, regulariser $\lambda > 0$, weights $\{a_i\}$, initialisation parameters $\{r_i\}$
   Initialise: $\mathbf{J}_i^0 = \mathbf{P}_i^0 = \mathbf{U}_i\mathbf{U}_i^T$, $\mathbf{U}_i = \left[r_i \text{ first left singular vectors of } \mathbf{X}_{[i]}\right]$, $\boldsymbol{\mathcal{E}}_0 = \mathbf{0}$, $\boldsymbol{\mathcal{Y}}_1^0 = \mathbf{0}$,
               $\mathbf{Y}_{2i}^0 = \mathbf{0}$, $\mu_0 > 0$, $k = 0$
1  while not converged do
2      for $i \in \{1, 2, \dots, N\}$ do
3          $\mathbf{J}_i^{k+1} = \mathcal{D}_{a_i\mu_k^{-1}}\left\{\mathbf{P}_i^k + \mu_k^{-1}\mathbf{Y}_{2i}^k\right\}$
4          $\mathbf{X}_i = \left(\boldsymbol{\mathcal{X}} \times_1 \mathbf{P}_1^{k+1} \times_2 \cdots \times_{i-1} \mathbf{P}_{i-1}^{k+1} \times_{i+1} \mathbf{P}_{i+1}^k \times_{i+2} \cdots \times_N \mathbf{P}_N^k\right)_{[i]}$
5          $\mathbf{P}_i^{k+1} = \left[\left(\mathbf{X}_{[i]} - (\mathbf{E}_k)_{[i]} + \mu_k^{-1}\left(\mathbf{Y}_1^k\right)_{[i]}\right)\mathbf{X}_i^T + \mathbf{J}_i^{k+1} - \mu_k^{-1}\mathbf{Y}_{2i}^k\right]\left(\mathbf{X}_i\mathbf{X}_i^T + \mathbf{I}\right)^{-1}$
6      end for
7      $\boldsymbol{\mathcal{E}}_{k+1} = \mathcal{S}_{\lambda\mu_k^{-1}}\left\{\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{X}} \times_{i=1}^N \mathbf{P}_i^{k+1} + \mu_k^{-1}\boldsymbol{\mathcal{Y}}_1^k\right\}$
8      $\boldsymbol{\mathcal{Y}}_1^{k+1} = \boldsymbol{\mathcal{Y}}_1^k + \mu_k\left(\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{X}} \times_{i=1}^N \mathbf{P}_i^{k+1} - \boldsymbol{\mathcal{E}}_{k+1}\right)$
9      for $i \in \{1, 2, \dots, N\}$ do
10         $\mathbf{Y}_{2i}^{k+1} = \mathbf{Y}_{2i}^k + \mu_k\left(\mathbf{P}_i^{k+1} - \mathbf{J}_i^{k+1}\right)$
11     end for
12     $\mu_{k+1} = \min\left(\rho\mu_k, \mu_{max}\right)$
13     $k \leftarrow k + 1$
14 end while
   Output: Projection matrices $\{\mathbf{P}_i^k\}$, sparse component $\boldsymbol{\mathcal{E}}_k$
```

4.3.2 Solution Based on Linearisation

Here, we will develop an algorithm for tensor IRPCA based on ALM with a linearisation step, similar to matrix IRPCA. We start by forming the augmented Lagrangian of the tensor IRPCA optimisation problem

$$\mathcal{L}_\mu\left(\{\mathbf{P}_i\}, \boldsymbol{\mathcal{E}}, \boldsymbol{\mathcal{Y}}\right) = \sum_{i=1}^N a_i \left\|\mathbf{P}_i\right\|_* + \lambda\left\|\boldsymbol{\mathcal{E}}\right\|_1 + \left\langle \boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{X}} \times_{i=1}^N \mathbf{P}_i - \boldsymbol{\mathcal{E}}, \boldsymbol{\mathcal{Y}}\right\rangle + \frac{\mu}{2}\left\|\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{X}} \times_{i=1}^N \mathbf{P}_i - \boldsymbol{\mathcal{E}}\right\|_F^2 \tag{4.44}$$

We will minimise the above with respect to each argument separately. Keeping all other arguments fixed, we minimise $\mathcal{L}_\mu$ with respect to each $\mathbf{P}_i$ for $i \in \{1, 2, \dots, N\}$ as follows

$$\begin{aligned}
&\arg\min_{\mathbf{P}_i} \mathcal{L}_\mu\left(\{\mathbf{P}_i\}, \boldsymbol{\mathcal{E}}, \boldsymbol{\mathcal{Y}}\right) \\
&= \arg\min_{\mathbf{P}_i} a_i\left\|\mathbf{P}_i\right\|_* + \left\langle \boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{X}} \times_{j=1}^N \mathbf{P}_j - \boldsymbol{\mathcal{E}}, \boldsymbol{\mathcal{Y}}\right\rangle + \frac{\mu}{2}\left\|\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{X}} \times_{j=1}^N \mathbf{P}_j - \boldsymbol{\mathcal{E}}\right\|_F^2 \\
&= \arg\min_{\mathbf{P}_i} a_i\mu^{-1}\left\|\mathbf{P}_i\right\|_* + \frac{1}{2}\left\|\boldsymbol{\mathcal{X}} \times_{j=1}^N \mathbf{P}_j - \left(\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{E}} + \mu^{-1}\boldsymbol{\mathcal{Y}}\right)\right\|_F^2 \\
&= \arg\min_{\mathbf{P}_i} a_i\mu^{-1}\left\|\mathbf{P}_i\right\|_* + \frac{1}{2}\left\|\mathbf{P}_i\mathbf{X}_i - \left(\mathbf{X}_{[i]} - \mathbf{E}_{[i]} + \mu^{-1}\mathbf{Y}_{[i]}\right)\right\|_F^2
\end{aligned} \tag{4.45}$$

where

$$\mathbf{X}_i = \mathbf{X}_{[i]} \left(\bigotimes_{\substack{j=N \\ j \neq i}}^{1} \mathbf{P}_j^T \right) = \left(\boldsymbol{\mathcal{X}} \times_{j \neq i} \mathbf{P}_j \right)_{[i]} \tag{4.46}$$

Notice that the above is a special case of problem (3.35). Therefore, we can apply in the same way a linearisation step to obtain a solution to the above problem. Using equation (3.38) and making the substitutions $a = a_i \mu^{-1}$, $\mathbf{A} = \mathbf{X}_i$ and $\mathbf{B} = \mathbf{X}_{[i]} - \mathbf{E}_{[i]} + \mu^{-1}\mathbf{Y}_{[i]}$, we get for $\mathbf{P}_i$ the following update

$$\mathbf{P}_i^{k+1} = \mathcal{D}_{a_i(\mu n)^{-1}} \left\{ \mathbf{P}_i^k - n^{-1} \left(\mathbf{P}_i^k \mathbf{X}_i - \mathbf{X}_{[i]} + \mathbf{E}_{[i]} - \mu^{-1}\mathbf{Y}_{[i]} \right) \mathbf{X}_i^T \right\} \tag{4.47}$$

Next, we minimise $\mathcal{L}_\mu$ with respect to $\boldsymbol{\mathcal{E}}$ as follows

$$\begin{aligned}
&\arg\min_{\boldsymbol{\mathcal{E}}} \mathcal{L}_\mu \left(\{\mathbf{P}_i\}, \boldsymbol{\mathcal{E}}, \boldsymbol{\mathcal{Y}} \right) \\
&= \arg\min_{\boldsymbol{\mathcal{E}}} \lambda \left\| \boldsymbol{\mathcal{E}} \right\|_1 + \left\langle \boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{X}} \times_{i=1}^N \mathbf{P}_i - \boldsymbol{\mathcal{E}}, \boldsymbol{\mathcal{Y}} \right\rangle + \frac{\mu}{2} \left\| \boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{X}} \times_{i=1}^N \mathbf{P}_i - \boldsymbol{\mathcal{E}} \right\|_F^2 \\
&= \arg\min_{\boldsymbol{\mathcal{E}}} \lambda \mu^{-1} \left\| \boldsymbol{\mathcal{E}} \right\|_1 + \frac{1}{2} \left\| \boldsymbol{\mathcal{E}} - \left(\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{X}} \times_{i=1}^N \mathbf{P}_i + \mu^{-1} \boldsymbol{\mathcal{Y}} \right) \right\|_F^2 \\
&= \mathcal{S}_{\lambda \mu^{-1}} \left\{ \boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{X}} \times_{i=1}^N \mathbf{P}_i + \mu^{-1} \boldsymbol{\mathcal{Y}} \right\}
\end{aligned} \tag{4.48}$$

The linearisation-based ALM algorithm for tensor IRPCA is presented in detail in Algorithm 4.4. Notice that the projection matrices $\mathbf{P}_i$ are initialised in the same way as in the substitution-based algorithm (Algorithm 4.3).

Algorithm 4.4: Tensor Inductive Robust PCA via linearisation

```
**Input**: Data tensor $\boldsymbol{\mathcal{X}}$, regulariser $\lambda > 0$, weights $\{a_i\}$, initialisation parameters $\{r_i\}$
**Initialise**: $\mathbf{P}_i^0 = \mathbf{U}_i \mathbf{U}_i^T$, $\mathbf{U}_i = [r_i$ first left singular vectors of $\mathbf{X}_{[i]}]$, $\boldsymbol{\mathcal{E}}_0 = \mathbf{0}$, $\boldsymbol{\mathcal{Y}}_0 = \mathbf{0}$,
    $\mu_0 > 0$, $k = 0$
1  while not converged do
2      for $i \in \{1, 2, \ldots, N\}$ do
3          $\mathbf{X}_i = \left( \boldsymbol{\mathcal{X}} \times_1 \mathbf{P}_1^{k+1} \times_2 \cdots \times_{i-1} \mathbf{P}_{i-1}^{k+1} \times_{i+1} \mathbf{P}_{i+1}^k \times_{i+2} \cdots \times_N \mathbf{P}_N^k \right)_{[i]}$
4          $n = \max \left( \left\| \mathbf{X}_i \mathbf{X}_i^T \right\|_F, 1.0 \right)$
5          $\mathbf{P}_i^{k+1} = \mathcal{D}_{a_i(\mu_k n)^{-1}} \left\{ \mathbf{P}_i^k - n^{-1} \left( \mathbf{P}_i^k \mathbf{X}_i - \mathbf{X}_{[i]} + (\mathbf{E}_k)_{[i]} - \mu_k^{-1} (\mathbf{Y}_k)_{[i]} \right) \mathbf{X}_i^T \right\}$
6      end for
7      $\boldsymbol{\mathcal{E}}_{k+1} = \mathcal{S}_{\lambda \mu_k^{-1}} \left\{ \boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{X}} \times_{i=1}^N \mathbf{P}_i^{k+1} + \mu_k^{-1} \boldsymbol{\mathcal{Y}}_k \right\}$
8      $\boldsymbol{\mathcal{Y}}_{k+1} = \boldsymbol{\mathcal{Y}}_k + \mu_k \left( \boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{X}} \times_{i=1}^N \mathbf{P}_i^{k+1} - \boldsymbol{\mathcal{E}}_{k+1} \right)$
9      $\mu_{k+1} = \min(\rho \mu_k, \mu_{max})$
10     $k \leftarrow k + 1$
11 end while
**Output**: Projection matrices $\{\mathbf{P}_i^k\}$, sparse component $\boldsymbol{\mathcal{E}}_k$
```

4.4 Robust Higher Order Singular Value Decomposition

The motivation behind matrix ORPCA was recovering a set of basis vectors for the principal subspace. In order to achieve this, the low-rank component $\mathbf{A} \in \mathbb{R}^{m \times n}$ was written as $\mathbf{A} = \mathbf{U}\mathbf{V}$ with $\mathbf{U}^T\mathbf{U} = \mathbf{I}$, where $\mathbf{U} \in \mathbb{R}^{m \times r}$ has as columns the r orthonormal basis vectors and $\mathbf{V} \in \mathbb{R}^{r \times n}$ has as columns the coefficients for each data point. In this section, we will show how the above can be extended to tensors.

Let $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ be an N^{th}-order data tensor and let $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ be its low-rank component. We shall refer to the space spanned by the n-mode fibres as *n-mode space.* The tensor equivalent of ORPCA is rewriting the low-rank component $\boldsymbol{\mathcal{A}}$ as follows

$$\boldsymbol{\mathcal{A}} = \boldsymbol{\mathcal{V}} \times_{i=1}^{N} \mathbf{U}_i \quad \text{where} \quad \mathbf{U}_i^T \mathbf{U}_i = \mathbf{I} \quad \forall i \in \{1, 2, \dots, N\} \tag{4.49}$$

In other words, we seek a set of basis vectors $\mathbf{U}_i$ for *each* n-mode space. The major difference from the matrix case is that the coefficients for the basis vectors are *shared* by all modes and form the *core tensor* $\boldsymbol{\mathcal{V}}$. It is easy to see that the above expression for $\boldsymbol{\mathcal{A}}$ is, in fact, its HOSVD. For this reason, we will refer to the tensor extension of ORPCA as *Robust Higher Order Singular Value Decomposition.*

Given the above expression for the low-rank component $\boldsymbol{\mathcal{A}}$, from Theorem 2.7 we have the following

$$\operatorname{rank}_i(\boldsymbol{\mathcal{A}}) \leq \operatorname{rank}_i(\boldsymbol{\mathcal{V}}) \tag{4.50}$$

Therefore, minimising the n-ranks of the core tensor $\boldsymbol{\mathcal{V}}$ is equivalent to minimising the n-ranks of the low-rank component $\boldsymbol{\mathcal{A}}$. Furthermore, the following theorem holds for the nuclear norm of each matricisation.[1]

Theorem 4.1 (Equality of nuclear norms in HOSVD)**.** *Let $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ be and N^{th}-order tensor with the following HOSVD*

$$\boldsymbol{\mathcal{A}} = \boldsymbol{\mathcal{V}} \times_{i=1}^{N} \mathbf{U}_i \tag{4.51}$$

with $\mathbf{U}_i^T \mathbf{U}_i = \mathbf{I}$ for all $i \in \{1, 2, \dots, N\}$. Then, for all matricisations, the following equality holds

$$\left\|\mathbf{A}_{[i]}\right\|_* = \left\|\mathbf{V}_{[i]}\right\|_* \tag{4.52}$$

Proof. We have that

$$\mathbf{A}_{[i]} = \mathbf{U}_i \mathbf{V}_{[i]} \mathbf{B}_i^T \tag{4.53}$$

where

$$\mathbf{B}_i = \bigotimes_{\substack{j=N \\ j \neq i}}^{1} \mathbf{U}_j \tag{4.54}$$

By definition, $\mathbf{U}_i^T \mathbf{U}_i = \mathbf{I}$. The same holds for $\mathbf{B}_i$, since

$$\mathbf{B}_i^T \mathbf{B}_i = \left(\bigotimes_{\substack{j=N \\ j \neq i}}^{1} \mathbf{U}_j^T \right) \left(\bigotimes_{\substack{j=N \\ j \neq i}}^{1} \mathbf{U}_j \right) = \bigotimes_{\substack{j=N \\ j \neq i}}^{1} \mathbf{U}_j^T \mathbf{U}_j = \bigotimes_{\substack{j=N \\ j \neq i}}^{1} \mathbf{I} = \mathbf{I} \tag{4.55}$$

Therefore, due to the unitary invariance of the nuclear norm (see Theorem 2.2) we have

$$\left\|\mathbf{A}_{[i]}\right\|_* = \left\|\mathbf{V}_{[i]}\right\|_* \tag{4.56}$$

□

Based on the above and using the nuclear norm as a convex relaxation for the rank, we can write the RHOSVD optimisation problem as follows

$$\min_{\boldsymbol{\mathcal{V}}, \boldsymbol{\mathcal{E}}, \{\mathbf{U}_i\}} \sum_{i=1}^{N} a_i \left\|\mathbf{V}_{[i]}\right\|_* + \lambda \left\|\boldsymbol{\mathcal{E}}\right\|_1 \quad \text{s.t.} \quad \begin{array}{l} \boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{V}} \times_{i=1}^{N} \mathbf{U}_i + \boldsymbol{\mathcal{E}} \\ \mathbf{U}_i^T \mathbf{U}_i = \mathbf{I} \quad \forall i \in \{1, 2, \dots, N\} \end{array} \tag{4.57}$$

[1]In fact, the theorem holds for any unitary invariant norm, including the more general Schatten p-norm.

As we did with tensor RPCA in order to remove the interdependencies between the nuclear norm terms, we rewrite the above optimisation problem in the following equivalent form

$$\min_{\{\mathbf{J}_i\},\boldsymbol{\mathcal{V}},\boldsymbol{\mathcal{E}},\{\mathbf{U}_i\}} \sum_{i=1}^{N} a_i \|\mathbf{J}_i\|_* + \lambda \|\boldsymbol{\mathcal{E}}\|_1 \quad \text{s.t.} \quad \begin{array}{l} \boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{V}} \times_{i=1}^{N} \mathbf{U}_i + \boldsymbol{\mathcal{E}} \\ \mathbf{V}_{[i]} = \mathbf{J}_i \\ \mathbf{U}_i^T \mathbf{U}_i = \mathbf{I} \end{array} \quad \forall i \in \{1,2,\dots,N\} \tag{4.58}$$

We are now in a position to solve the above optimisation problem. In the remaining of this section, we will develop an ALM algorithm for solving it. We shall start by forming the partial augmented Lagrangian for the problem (without incorporating the orthonormality constraints $\mathbf{U}_i^T\mathbf{U}_i = \mathbf{I}$) as follows

$$\begin{aligned}\mathcal{L}_\mu\left(\{\mathbf{J}_i\},\boldsymbol{\mathcal{V}},\boldsymbol{\mathcal{E}},\{\mathbf{U}_i\},\boldsymbol{\mathcal{Y}}_1,\{\mathbf{Y}_{2i}\}\right) &= \sum_{i=1}^{N} a_i \|\mathbf{J}_i\|_* + \lambda\|\boldsymbol{\mathcal{E}}\|_1 + \left\langle \boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{V}} \times_{i=1}^{N} \mathbf{U}_i - \boldsymbol{\mathcal{E}}, \boldsymbol{\mathcal{Y}}_1 \right\rangle \\ &+ \sum_{i=1}^{N} \left\langle \mathbf{V}_{[i]} - \mathbf{J}_i, \mathbf{Y}_{2i} \right\rangle + \frac{\mu}{2} \left\| \boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{V}} \times_{i=1}^{N} \mathbf{U}_i - \boldsymbol{\mathcal{E}} \right\|_F^2 + \sum_{i=1}^{N} \frac{\mu}{2} \left\| \mathbf{V}_{[i]} - \mathbf{J}_i \right\|_F^2 \end{aligned} \tag{4.59}$$

We first minimise $\mathcal{L}_\mu$ with respect to each $\mathbf{J}_i$ for $i \in \{1,2,\dots,N\}$ as follows

$$\begin{aligned} &\arg\min_{\mathbf{J}_i} \mathcal{L}_\mu\left(\{\mathbf{J}_i\},\boldsymbol{\mathcal{V}},\boldsymbol{\mathcal{E}},\{\mathbf{U}_i\},\boldsymbol{\mathcal{Y}}_1,\{\mathbf{Y}_{2i}\}\right) \\ &= \arg\min_{\mathbf{J}_i} a_i \|\mathbf{J}_i\|_* + \left\langle \mathbf{V}_{[i]} - \mathbf{J}_i, \mathbf{Y}_{2i} \right\rangle + \frac{\mu}{2} \left\| \mathbf{V}_{[i]} - \mathbf{J}_i \right\|_F^2 \\ &= \arg\min_{\mathbf{J}_i} a_i \mu^{-1} \|\mathbf{J}_i\|_* + \frac{1}{2} \left\| \mathbf{J}_i - \left(\mathbf{V}_{[i]} + \mu^{-1}\mathbf{Y}_{2i} \right) \right\|_F^2 \\ &= \mathcal{D}_{a_i\mu^{-1}} \left\{ \mathbf{V}_{[i]} + \mu^{-1}\mathbf{Y}_{2i} \right\} \end{aligned} \tag{4.60}$$

Keeping in mind that $\mathbf{U}_i^T\mathbf{U}_i = \mathbf{I}$ and that the Frobenius norm is unitary invariant, we minimise $\mathcal{L}_\mu$ with respect to $\boldsymbol{\mathcal{V}}$ as follows

$$\begin{aligned} &\arg\min_{\boldsymbol{\mathcal{V}}} \mathcal{L}_\mu\left(\{\mathbf{J}_i\},\boldsymbol{\mathcal{V}},\boldsymbol{\mathcal{E}},\{\mathbf{U}_i\},\boldsymbol{\mathcal{Y}}_1,\{\mathbf{Y}_{2i}\}\right) \\ &= \arg\min_{\boldsymbol{\mathcal{V}}} \left\langle \boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{V}} \times_{i=1}^{N} \mathbf{U}_i - \boldsymbol{\mathcal{E}}, \boldsymbol{\mathcal{Y}}_1 \right\rangle + \sum_{i=1}^{N} \left\langle \mathbf{V}_{[i]} - \mathbf{J}_i, \mathbf{Y}_{2i} \right\rangle \\ &\qquad + \frac{\mu}{2} \left\| \boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{V}} \times_{i=1}^{N} \mathbf{U}_i - \boldsymbol{\mathcal{E}} \right\|_F^2 + \sum_{i=1}^{N} \frac{\mu}{2} \left\| \mathbf{V}_{[i]} - \mathbf{J}_i \right\|_F^2 \\ &= \arg\min_{\boldsymbol{\mathcal{V}}} \left\langle \boldsymbol{\mathcal{V}}, -\boldsymbol{\mathcal{Y}}_1 \times_{i=1}^{N} \mathbf{U}_i^T \right\rangle + \sum_{i=1}^{N} \langle \boldsymbol{\mathcal{V}} - \boldsymbol{\mathcal{J}}_i, \boldsymbol{\mathcal{Y}}_{2i} \rangle + \frac{\mu}{2} \left\| \boldsymbol{\mathcal{V}} \times_{i=1}^{N} \mathbf{U}_i \right\|_F^2 \\ &\qquad + \left\langle \boldsymbol{\mathcal{V}}, -\mu\left(\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{E}}\right) \times_{i=1}^{N} \mathbf{U}_i^T \right\rangle + \sum_{i=1}^{N} \frac{\mu}{2} \|\boldsymbol{\mathcal{V}} - \boldsymbol{\mathcal{J}}_i\|_F^2 \\ &= \arg\min_{\boldsymbol{\mathcal{V}}} \left\langle \boldsymbol{\mathcal{V}}, -\left[\mu\left(\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{E}}\right) + \boldsymbol{\mathcal{Y}}_1\right] \times_{i=1}^{N} \mathbf{U}_i^T \right\rangle + \sum_{i=1}^{N} \langle \boldsymbol{\mathcal{V}} - \boldsymbol{\mathcal{J}}_i, \boldsymbol{\mathcal{Y}}_{2i} \rangle \\ &\qquad + \frac{\mu}{2} \|\boldsymbol{\mathcal{V}}\|_F^2 + \sum_{i=1}^{N} \frac{\mu}{2} \|\boldsymbol{\mathcal{V}} - \boldsymbol{\mathcal{J}}_i\|_F^2 \end{aligned} \tag{4.61}$$

Note that in the above expressions we have made use of the tensorised forms of $\mathbf{J}_i$ and $\mathbf{Y}_{2i}$, represented by $\boldsymbol{\mathcal{J}}_i$ and $\boldsymbol{\mathcal{Y}}_{2i}$. In this sense, the tensorisation is the exact reverse procedure of matricisation. The above is easy to minimise since all the terms are differentiable. Using the following derivatives

$$\frac{\partial}{\partial \boldsymbol{\mathcal{V}}} \left\langle \boldsymbol{\mathcal{V}}, -\left[\mu\left(\boldsymbol{\mathcal{X}}-\boldsymbol{\mathcal{E}}\right)+\boldsymbol{\mathcal{Y}}_1\right] \times_{i=1}^{N} \mathbf{U}_i^T \right\rangle = -\left[\mu\left(\boldsymbol{\mathcal{X}}-\boldsymbol{\mathcal{E}}\right)+\boldsymbol{\mathcal{Y}}_1\right] \times_{i=1}^{N} \mathbf{U}_i^T \tag{4.62}$$

$$\frac{\partial}{\partial \boldsymbol{\mathcal{V}}} \left(\sum_{i=1}^{N} \langle \boldsymbol{\mathcal{V}} - \boldsymbol{\mathcal{J}}_i, \boldsymbol{\mathcal{Y}}_{2i} \rangle \right) = \sum_{i=1}^{N} \boldsymbol{\mathcal{Y}}_{2i} \tag{4.63}$$

$$\frac{\partial}{\partial \boldsymbol{\mathcal{V}}} \left(\frac{\mu}{2} \left\| \boldsymbol{\mathcal{V}} \right\|_F^2 \right) = \mu \boldsymbol{\mathcal{V}} \tag{4.64}$$

$$\frac{\partial}{\partial \boldsymbol{\mathcal{V}}} \left(\sum_{i=1}^{N} \frac{\mu}{2} \left\| \boldsymbol{\mathcal{V}} - \boldsymbol{\mathcal{J}}_i \right\|_F^2 \right) = \mu N \boldsymbol{\mathcal{V}} - \mu \sum_{i=1}^{N} \boldsymbol{\mathcal{J}}_i \tag{4.65}$$

the minimisation becomes

$$\begin{aligned}
&\frac{\partial \mathcal{L}_\mu}{\partial \boldsymbol{\mathcal{V}}} = \mathbf{0} && \Rightarrow \\
&(N+1)\,\boldsymbol{\mathcal{V}} - \left(\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{E}} + \mu^{-1}\boldsymbol{\mathcal{Y}}_1\right) \times_{i=1}^{N} \mathbf{U}_i^T - \sum_{i=1}^{N} \left(\boldsymbol{\mathcal{J}}_i - \mu^{-1}\boldsymbol{\mathcal{Y}}_{2i}\right) = \mathbf{0} && \Rightarrow \\
&\boldsymbol{\mathcal{V}} = (N+1)^{-1} \left[\left(\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{E}} + \mu^{-1}\boldsymbol{\mathcal{Y}}_1\right) \times_{i=1}^{N} \mathbf{U}_i^T + \sum_{i=1}^{N} \left(\boldsymbol{\mathcal{J}}_i - \mu^{-1}\boldsymbol{\mathcal{Y}}_{2i}\right) \right] && (4.66)
\end{aligned}$$

Next, we minimise $\mathcal{L}_\mu$ with respect to $\boldsymbol{\mathcal{E}}$ as follows

$$\begin{aligned}
&\arg\min_{\boldsymbol{\mathcal{E}}} \mathcal{L}_\mu \left(\{\mathbf{J}_i\}, \boldsymbol{\mathcal{V}}, \boldsymbol{\mathcal{E}}, \{\mathbf{U}_i\}, \boldsymbol{\mathcal{Y}}_1, \{\mathbf{Y}_{2i}\}\right) \\
&= \arg\min_{\boldsymbol{\mathcal{E}}} \lambda \left\|\boldsymbol{\mathcal{E}}\right\|_1 + \left\langle \boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{V}} \times_{i=1}^{N} \mathbf{U}_i - \boldsymbol{\mathcal{E}}, \boldsymbol{\mathcal{Y}}_1 \right\rangle + \frac{\mu}{2} \left\| \boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{V}} \times_{i=1}^{N} \mathbf{U}_i - \boldsymbol{\mathcal{E}} \right\|_F^2 \\
&= \arg\min_{\boldsymbol{\mathcal{E}}} \lambda\mu^{-1} \left\|\boldsymbol{\mathcal{E}}\right\|_1 + \frac{1}{2} \left\| \boldsymbol{\mathcal{E}} - \left(\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{V}} \times_{i=1}^{N} \mathbf{U}_i + \mu^{-1}\boldsymbol{\mathcal{Y}}_1\right) \right\|_F^2 \\
&= \mathcal{S}_{\lambda\mu^{-1}} \left\{ \boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{V}} \times_{i=1}^{N} \mathbf{U}_i + \mu^{-1}\boldsymbol{\mathcal{Y}}_1 \right\} && (4.67)
\end{aligned}$$

Finally, we minimise $\mathcal{L}_\mu$ with respect to each $\mathbf{U}_i$ for $i \in \{1, 2, \ldots, N\}$, subject to $\mathbf{U}_i^T\mathbf{U}_i = \mathbf{I}$, as follows

$$\begin{aligned}
&\arg\min_{\mathbf{U}_i} \mathcal{L}_\mu \left(\{\mathbf{J}_i\}, \boldsymbol{\mathcal{V}}, \boldsymbol{\mathcal{E}}, \{\mathbf{U}_i\}, \boldsymbol{\mathcal{Y}}_1, \{\mathbf{Y}_{2i}\}\right) \\
&= \arg\min_{\mathbf{U}_i} \left\langle \boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{V}} \times_{j=1}^{N} \mathbf{U}_j - \boldsymbol{\mathcal{E}}, \boldsymbol{\mathcal{Y}}_1 \right\rangle + \frac{\mu}{2} \left\| \boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{V}} \times_{j=1}^{N} \mathbf{U}_j - \boldsymbol{\mathcal{E}} \right\|_F^2 \\
&= \arg\min_{\mathbf{U}_i} \frac{1}{2} \left\| \left(\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{E}} + \mu^{-1}\boldsymbol{\mathcal{Y}}_1\right) - \boldsymbol{\mathcal{V}} \times_{j=1}^{N} \mathbf{U}_j \right\|_F^2 \\
&= \arg\min_{\mathbf{U}_i} \frac{1}{2} \left\| \left(\mathbf{X}_{[i]} - \mathbf{E}_{[i]} + \mu^{-1}\left(\mathbf{Y}_1\right)_{[i]}\right) - \mathbf{U}_i\mathbf{B}_i \right\|_F^2 && (4.68)
\end{aligned}$$

where

$$\mathbf{B}_i = \mathbf{V}_{[i]} \left(\bigotimes_{\substack{j=N \\ j\neq i}}^{1} \mathbf{U}_j^T \right) = \left(\boldsymbol{\mathcal{V}} \times_{j\neq i} \mathbf{U}_j\right)_{[i]} \tag{4.69}$$

Assume the following Singular Value Decomposition

$$\left(\mathbf{X}_{[i]} - \mathbf{E}_{[i]} + \mu^{-1}\left(\mathbf{Y}_1\right)_{[i]}\right) \mathbf{B}_i^T = \mathbf{U}_{Si}\mathbf{D}_{Si}\mathbf{V}_{Si}^T \tag{4.70}$$

Algorithm 4.5: Robust Higher Order SVD

```
Input: Data tensor $\boldsymbol{\mathcal{X}}$, regulariser $\lambda > 0$, weights $\{a_i\}$, number of principal components $\{r_i\}$
Initialise: $\mathbf{J}_i^0 = \mathbf{0}$, $\boldsymbol{\mathcal{V}}_0 = \mathbf{0}$, $\boldsymbol{\mathcal{E}}_0 = \mathbf{0}$, $\mathbf{U}_i^0 = \left[r_i \text{ first left singular vectors of } \mathbf{X}_{[i]}\right]$, $\boldsymbol{\mathcal{Y}}_1^0 = \mathbf{0}$, $\mathbf{Y}_{2i}^0 = \mathbf{0}$, $\mu_0 > 0$, $k = 0$
1  while not converged do
2      for $i \in \{1, 2, \ldots, N\}$ do
3          $\mathbf{J}_i^{k+1} = \mathcal{D}_{a_i \mu_k^{-1}} \left\{ (\mathbf{V}_k)_{[i]} + \mu_k^{-1} \mathbf{Y}_{2i}^k \right\}$
4      end for
5      $\boldsymbol{\mathcal{V}}_{k+1} = (N+1)^{-1} \left[ \left( \boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{E}}_k + \mu_k^{-1} \boldsymbol{\mathcal{Y}}_1^k \right) \times_{i=1}^N \left( \mathbf{U}_i^k \right)^T + \sum_{i=1}^N \left( \boldsymbol{\mathcal{J}}_i^{k+1} - \mu_k^{-1} \boldsymbol{\mathcal{Y}}_{2i}^k \right) \right]$
6      $\boldsymbol{\mathcal{E}}_{k+1} = \mathcal{S}_{\lambda \mu_k^{-1}} \left\{ \boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{V}}_{k+1} \times_{i=1}^N \mathbf{U}_i^k + \mu_k^{-1} \boldsymbol{\mathcal{Y}}_1^k \right\}$
7      for $i \in \{1, 2, \ldots, N\}$ do
8          $\mathbf{B}_i = \left( \boldsymbol{\mathcal{V}}_{k+1} \times_1 \mathbf{U}_1^{k+1} \times_2 \cdots \times_{i-1} \mathbf{U}_{i-1}^{k+1} \times_{i+1} \mathbf{U}_{i+1}^k \times_{i+2} \cdots \times_N \mathbf{U}_N^k \right)_{[i]}$
9          $[\mathbf{U}_{Si}, \mathbf{D}_{Si}, \mathbf{V}_{Si}] = \text{svd} \left\{ \left( \mathbf{X}_{[i]} - (\mathbf{E}_{k+1})_{[i]} + \mu_k^{-1} \left( \mathbf{Y}_1^k \right)_{[i]} \right) \mathbf{B}_i^T \right\}$
10         $\mathbf{U}_i^{k+1} = \mathbf{U}_{Si} \mathbf{V}_{Si}^T$
11     end for
12     $\boldsymbol{\mathcal{Y}}_1^{k+1} = \boldsymbol{\mathcal{Y}}_1^k + \mu_k \left( \boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{V}}_{k+1} \times_{i=1}^N \mathbf{U}_i^{k+1} - \boldsymbol{\mathcal{E}}_{k+1} \right)$
13     for $i \in \{1, 2, \ldots, N\}$ do
14         $\mathbf{Y}_{2i}^{k+1} = \mathbf{Y}_{2i}^k + \mu_k \left( (\mathbf{V}_{k+1})_{[i]} - \mathbf{J}_i^{k+1} \right)$
15     end for
16     $\mu_{k+1} = \min \left( \rho \mu_k, \mu_{max} \right)$
17     $k \leftarrow k + 1$
18 end while
Output: Principal components $\{\mathbf{U}_i^k\}$, core tensor $\boldsymbol{\mathcal{V}}_k$, sparse component $\boldsymbol{\mathcal{E}}_k$
```

where $\mathbf{D}_{Si}$ is taken to be square. According to the *Reduced Rank Procrustes Theorem* (Theorem 4 in [94]), the optimiser $\mathbf{U}_i^*$ of problem (4.68) is given by

$$\mathbf{U}_i^* = \mathbf{U}_{Si} \mathbf{V}_{Si}^T \tag{4.71}$$

The ALM algorithm for RHOSVD is presented in detail in Algorithm 4.5. Notice that the algorithm requires as input the number of principal components r_i for each mode. That is, r_i is the number of columns of matrix $\mathbf{U}_i$, for $i \in \{1, 2, \ldots, N\}$. Since $\mathbf{U}_i^T \mathbf{U}_i = \mathbf{I}$, $\text{rank}(\mathbf{U}_i) = r_i$ and from Theorem 2.7 we have that

$$\text{rank}_i(\boldsymbol{\mathcal{A}}) = \text{rank}_i\left(\boldsymbol{\mathcal{V}} \times_{i=1}^N \mathbf{U}_i \right) \leq \text{rank}(\mathbf{U}_i) = r_i \tag{4.72}$$

Therefore, each r_i serves as a controllable upper bound of the corresponding n-rank of the low-rank component $\boldsymbol{\mathcal{A}}$. Finally, notice that each matrix $\mathbf{U}_i$ is initialised with the r_i first left singular vectors of $\mathbf{X}_{[i]}$. The RHOSVD is a non-convex problem and therefore initialisation is important. We have empirically found that this "warm" initialisation of each $\mathbf{U}_i$ performs well in most cases.

4.5 Robust CANDECOMP/PARAFAC Decomposition

In section 4.4 we modelled the low-rank component $\boldsymbol{\mathcal{A}}$ using the HOSVD. In this section, we follow a different approach and we model $\boldsymbol{\mathcal{A}}$ using the CP decomposition. In other words, we rewrite the low-rank component $\boldsymbol{\mathcal{A}}$ as follows

$$\boldsymbol{\mathcal{A}} = \mathbf{U}_1 \circ \cdots \circ \mathbf{U}_N \tag{4.73}$$

From theorem 2.5, we have that $\forall\, i \in \{1, 2, \dots, N\}$

$$\operatorname{rank}_i(\boldsymbol{\mathcal{A}}) \le \operatorname{rank}(\mathbf{U}_i) \tag{4.74}$$

Therefore, choosing to minimise the rank of each factor matrix $\mathbf{U}_i$ is equivalent to minimising an upper bound of the corresponding n-rank of the low-rank component $\boldsymbol{\mathcal{A}}$. Based on the above discussion and using the nuclear norm as a convex relaxation of the rank, we formulate the following optimisation problem

$$\min_{\{\mathbf{U}_i\},\boldsymbol{\mathcal{E}}} \sum_{i=1}^{N} a_i \|\mathbf{U}_i\|_* + \lambda \|\boldsymbol{\mathcal{E}}\|_1 \quad \text{s.t.} \quad \boldsymbol{\mathcal{X}} = \mathbf{U}_1 \circ \cdots \circ \mathbf{U}_N + \boldsymbol{\mathcal{E}} \tag{4.75}$$

We shall refer to the above problem as *Robust CANDECOMP/PARAFAC Decomposition.* In the remaining of this section, we derive two algorithms for solving the above. The first is based on ALM with substitution and the second is based on ALM with a linearisation step.

4.5.1 Solution Based on Substitution

In order to be able to develop an ALM algorithm for solving it, we rewrite the RCPD optimisation problem in the following equivalent form

$$\min_{\{\mathbf{J}_i\},\{\mathbf{U}_i\},\boldsymbol{\mathcal{E}}} \sum_{i=1}^{N} a_i \|\mathbf{J}_i\|_* + \lambda \|\boldsymbol{\mathcal{E}}\|_1 \quad \text{s.t.} \quad \begin{array}{l} \boldsymbol{\mathcal{X}} = \mathbf{U}_1 \circ \cdots \circ \mathbf{U}_N + \boldsymbol{\mathcal{E}} \\ \mathbf{U}_i = \mathbf{J}_i \quad \forall i \in \{1, 2, \dots, N\} \end{array} \tag{4.76}$$

We begin by formulating the augmented Lagrangian of the above as follows

$$\begin{aligned} \mathcal{L}_\mu(\{\mathbf{J}_i\}, \{\mathbf{U}_i\}, \boldsymbol{\mathcal{E}}, \boldsymbol{\mathcal{Y}}_1, \{\mathbf{Y}_{2i}\}) &= \sum_{i=1}^{N} a_i \|\mathbf{J}_i\|_* + \lambda \|\boldsymbol{\mathcal{E}}\|_1 + \langle \boldsymbol{\mathcal{X}} - \mathbf{U}_1 \circ \cdots \circ \mathbf{U}_N - \boldsymbol{\mathcal{E}}, \boldsymbol{\mathcal{Y}}_1 \rangle \\ + \sum_{i=1}^{N} \langle \mathbf{U}_i - \mathbf{J}_i, \mathbf{Y}_{2i} \rangle &+ \frac{\mu}{2} \|\boldsymbol{\mathcal{X}} - \mathbf{U}_1 \circ \cdots \circ \mathbf{U}_N - \boldsymbol{\mathcal{E}}\|_F^2 + \sum_{i=1}^{N} \frac{\mu}{2} \|\mathbf{U}_i - \mathbf{J}_i\|_F^2 \end{aligned} \tag{4.77}$$

We first minimise $\mathcal{L}_\mu$ with respect to each $\mathbf{J}_i$ for $i \in \{1, 2, \dots, N\}$ as follows

$$\begin{aligned} &\arg\min_{\mathbf{J}_i} \mathcal{L}_\mu(\{\mathbf{J}_i\}, \{\mathbf{U}_i\}, \boldsymbol{\mathcal{E}}, \boldsymbol{\mathcal{Y}}_1, \{\mathbf{Y}_{2i}\}) \\ &= \arg\min_{\mathbf{J}_i} a_i \|\mathbf{J}_i\|_* + \langle \mathbf{U}_i - \mathbf{J}_i, \mathbf{Y}_{2i} \rangle + \frac{\mu}{2} \|\mathbf{U}_i - \mathbf{J}_i\|_F^2 \\ &= \arg\min_{\mathbf{J}_i} a_i \mu^{-1} \|\mathbf{J}_i\|_* + \frac{1}{2} \left\|\mathbf{J}_i - \left(\mathbf{U}_i + \mu^{-1}\mathbf{Y}_{2i}\right)\right\|_F^2 \\ &= \mathcal{D}_{a_i\mu^{-1}} \left\{\mathbf{U}_i + \mu^{-1}\mathbf{Y}_{2i}\right\} \end{aligned} \tag{4.78}$$

We then minimise $\mathcal{L}_\mu$ with respect to each $\mathbf{U}_i$ for $i \in \{1, 2, \dots, N\}$ as follows

$$\begin{aligned} &\arg\min_{\mathbf{U}_i} \mathcal{L}_\mu(\{\mathbf{J}_i\}, \{\mathbf{U}_i\}, \boldsymbol{\mathcal{E}}, \boldsymbol{\mathcal{Y}}_1, \{\mathbf{Y}_{2i}\}) \\ &= \arg\min_{\mathbf{U}_i} \langle \boldsymbol{\mathcal{X}} - \mathbf{U}_1 \circ \cdots \circ \mathbf{U}_N - \boldsymbol{\mathcal{E}}, \boldsymbol{\mathcal{Y}}_1 \rangle + \langle \mathbf{U}_i - \mathbf{J}_i, \mathbf{Y}_{2i} \rangle \\ &\qquad + \frac{\mu}{2} \|\boldsymbol{\mathcal{X}} - \mathbf{U}_1 \circ \cdots \circ \mathbf{U}_N - \boldsymbol{\mathcal{E}}\|_F^2 + \frac{\mu}{2} \|\mathbf{U}_i - \mathbf{J}_i\|_F^2 \\ &= \arg\min_{\mathbf{U}_i} \left\langle \mathbf{X}_{[i]} - \mathbf{U}_i\tilde{\mathbf{U}}_i - \mathbf{E}_{[i]}, (\mathbf{Y}_1)_{[i]} \right\rangle + \langle \mathbf{U}_i - \mathbf{J}_i, \mathbf{Y}_{2i} \rangle \\ &\qquad + \frac{\mu}{2} \left\|\mathbf{X}_{[i]} - \mathbf{U}_i\tilde{\mathbf{U}}_i - \mathbf{E}_{[i]}\right\|_F^2 + \frac{\mu}{2} \|\mathbf{U}_i - \mathbf{J}_i\|_F^2 \end{aligned} \tag{4.79}$$

where

$$\tilde{\mathbf{U}}_i = \left(\bigodot_{\substack{j=N \\ j\neq i}}^{1} \mathbf{U}_j \right)^T \tag{4.80}$$

All the terms in the above objective are differentiable, therefore it can be easily minimised by forcing its derivative with respect to $\mathbf{U}_i$ equal to zero. Using the following derivatives

$$\frac{\partial}{\partial \mathbf{U}_i} \left\langle \mathbf{X}_{[i]} - \mathbf{U}_i \tilde{\mathbf{U}}_i - \mathbf{E}_{[i]}, (\mathbf{Y}_1)_{[i]} \right\rangle = -(\mathbf{Y}_1)_{[i]} \tilde{\mathbf{U}}_i^T \tag{4.81}$$

$$\frac{\partial}{\partial \mathbf{U}_i} \langle \mathbf{U}_i - \mathbf{J}_i, \mathbf{Y}_{2i} \rangle = \mathbf{Y}_{2i} \tag{4.82}$$

$$\frac{\partial}{\partial \mathbf{U}_i} \left(\frac{\mu}{2} \left\| \mathbf{X}_{[i]} - \mathbf{U}_i \tilde{\mathbf{U}}_i - \mathbf{E}_{[i]} \right\|_F^2 \right) = -\mu \left(\mathbf{X}_{[i]} - \mathbf{U}_i \tilde{\mathbf{U}}_i - \mathbf{E}_{[i]} \right) \tilde{\mathbf{U}}_i^T \tag{4.83}$$

$$\frac{\partial}{\partial \mathbf{U}_i} \left(\frac{\mu}{2} \| \mathbf{U}_i - \mathbf{J}_i \|_F^2 \right) = \mu (\mathbf{U}_i - \mathbf{J}_i) \tag{4.84}$$

the minimisation becomes

$$\begin{aligned}
&\frac{\partial \mathcal{L}_\mu}{\partial \mathbf{U}_i} = \mathbf{0} && \Rightarrow \\
&\mathbf{U}_i \left(\tilde{\mathbf{U}}_i \tilde{\mathbf{U}}_i^T + \mathbf{I} \right) - \left(\mathbf{X}_{[i]} - \mathbf{E}_{[i]} + \mu^{-1} (\mathbf{Y}_1)_{[i]} \right) \tilde{\mathbf{U}}_i^T - \mathbf{J}_i + \mu^{-1} \mathbf{Y}_{2i} = \mathbf{0} && \Rightarrow \\
&\mathbf{U}_i = \left[\left(\mathbf{X}_{[i]} - \mathbf{E}_{[i]} + \mu^{-1} (\mathbf{Y}_1)_{[i]} \right) \tilde{\mathbf{U}}_i^T + \mathbf{J}_i - \mu^{-1} \mathbf{Y}_{2i} \right] \left(\tilde{\mathbf{U}}_i \tilde{\mathbf{U}}_i^T + \mathbf{I} \right)^{-1} &&
\end{aligned} \tag{4.85}$$

Next, we minimise $\mathcal{L}_\mu$ with respect to $\boldsymbol{\mathcal{E}}$ as follows

$$\begin{aligned}
&\arg\min_{\boldsymbol{\mathcal{E}}} \mathcal{L}_\mu (\{\mathbf{J}_i\}, \{\mathbf{U}_i\}, \boldsymbol{\mathcal{E}}, \boldsymbol{\mathcal{Y}}_1, \{\mathbf{Y}_{2i}\}) \\
&= \arg\min_{\boldsymbol{\mathcal{E}}} \lambda \|\boldsymbol{\mathcal{E}}\|_1 + \langle \boldsymbol{\mathcal{X}} - \mathbf{U}_1 \circ \cdots \circ \mathbf{U}_N - \boldsymbol{\mathcal{E}}, \boldsymbol{\mathcal{Y}}_1 \rangle + \frac{\mu}{2} \|\boldsymbol{\mathcal{X}} - \mathbf{U}_1 \circ \cdots \circ \mathbf{U}_N - \boldsymbol{\mathcal{E}}\|_F^2 \\
&= \arg\min_{\boldsymbol{\mathcal{E}}} \lambda \mu^{-1} \|\boldsymbol{\mathcal{E}}\|_1 + \frac{1}{2} \left\| \boldsymbol{\mathcal{E}} - (\boldsymbol{\mathcal{X}} - \mathbf{U}_1 \circ \cdots \circ \mathbf{U}_N + \mu^{-1} \boldsymbol{\mathcal{Y}}_1) \right\|_F^2 \\
&= \mathcal{S}_{\lambda\mu^{-1}} \left\{ \boldsymbol{\mathcal{X}} - \mathbf{U}_1 \circ \cdots \circ \mathbf{U}_N + \mu^{-1} \boldsymbol{\mathcal{Y}}_1 \right\}
\end{aligned} \tag{4.86}$$

The substitution-based ALM algorithm for RCPD is presented in detail in Algorithm 4.6. The algorithm requires as input the number r of rank-1 tensors of the CP decomposition. In other words, the user needs to specify the number of columns r of each factor matrix $\mathbf{U}_i$. Notice that, due to Theorem 2.5, we have

$$\operatorname{rank}_i (\boldsymbol{\mathcal{A}}) = \operatorname{rank}_i (\mathbf{U}_1 \circ \cdots \circ \mathbf{U}_N) \leq \operatorname{rank} (\mathbf{U}_i) \leq r \tag{4.87}$$

therefore r serves as a controllable upper bound on each n-rank of the low-rank component $\boldsymbol{\mathcal{A}}$. Unlike RHOSVD, though, which requires the specification of N sizes $\{r_i\}$ as input, RCPD requires only one. Finally, it is important to mention that RCPD is a non-convex problem, therefore initialisation is important. In our implementation, we choose to initialise each entry of the factor matrices $\mathbf{U}_i$ uniformly at random in the interval $[0, 1]$.

4.5.2 Solution Based on Linearisation

In this section, we will derive an alternative ALM algorithm for RCPD which does not require any increase on the number of variables but instead it is based on a linearisation step. We begin

Algorithm 4.6: Robust CP Decomposition via substitution

```
   **Input**: Data tensor $\boldsymbol{\mathcal{X}}$, regulariser $\lambda > 0$, weights $\{a_i\}$, number of rank-1 tensors $r$
   **Initialise**: $\mathbf{J}_i^0 = \mathbf{U}_i^0 = \text{rand}$, $\boldsymbol{\mathcal{E}}_0 = \mathbf{0}$, $\boldsymbol{\mathcal{Y}}_1^0 = \mathbf{0}$, $\mathbf{Y}_{2i}^0 = \mathbf{0}$, $\mu_0 > 0$, $k = 0$
1  **while** *not converged* **do**
2      **for** $i \in \{1, 2, \dots, N\}$ **do**
3          $\mathbf{J}_i^{k+1} = \mathcal{D}_{a_i \mu_k^{-1}} \{\mathbf{U}_i^k + \mu_k^{-1} \mathbf{Y}_{2i}^k\}$
4          $\tilde{\mathbf{U}}_i = \left(\mathbf{U}_N^k \odot \cdots \odot \mathbf{U}_{i+1}^k \odot \mathbf{U}_{i-1}^{k+1} \odot \cdots \odot \mathbf{U}_1^{k+1}\right)^T$
5          $\mathbf{U}_i^{k+1} = \left[\left(\mathbf{X}_{[i]} - (\mathbf{E}_k)_{[i]} + \mu_k^{-1} \left(\mathbf{Y}_1^k\right)_{[i]}\right) \tilde{\mathbf{U}}_i^T + \mathbf{J}_i^{k+1} - \mu_k^{-1} \mathbf{Y}_{2i}^k\right] \left(\tilde{\mathbf{U}}_i \tilde{\mathbf{U}}_i^T + \mathbf{I}\right)^{-1}$
6      **end for**
7      $\boldsymbol{\mathcal{E}}_{k+1} = \mathcal{S}_{\lambda \mu_k^{-1}} \left\{\boldsymbol{\mathcal{X}} - \mathbf{U}_1^{k+1} \circ \cdots \circ \mathbf{U}_N^{k+1} + \mu_k^{-1} \boldsymbol{\mathcal{Y}}_1^k\right\}$
8      $\boldsymbol{\mathcal{Y}}_1^{k+1} = \boldsymbol{\mathcal{Y}}_1^k + \mu_k \left(\boldsymbol{\mathcal{X}} - \mathbf{U}_1^{k+1} \circ \cdots \circ \mathbf{U}_N^{k+1} - \boldsymbol{\mathcal{E}}_{k+1}\right)$
9      **for** $i \in \{1, 2, \dots, N\}$ **do**
10         $\mathbf{Y}_{2i}^{k+1} = \mathbf{Y}_{2i}^k + \mu_k \left(\mathbf{U}_i^{k+1} - \mathbf{J}_i^{k+1}\right)$
11     **end for**
12     $\mu_{k+1} = \min(\rho \mu_k, \mu_{max})$
13     $k \leftarrow k + 1$
14 **end while**
   **Output**: Factor matrices $\{\mathbf{U}_i^k\}$, sparse component $\boldsymbol{\mathcal{E}}_k$
```

by formulating the augmented Lagrangian of the RCPD optimisation problem as follows

$$\mathcal{L}_\mu (\{\mathbf{U}_i\}, \boldsymbol{\mathcal{E}}, \boldsymbol{\mathcal{Y}}) = \sum_{i=1}^N a_i \|\mathbf{U}_i\|_* + \lambda \|\boldsymbol{\mathcal{E}}\|_1 + \langle \boldsymbol{\mathcal{X}} - \mathbf{U}_1 \circ \cdots \circ \mathbf{U}_N - \boldsymbol{\mathcal{E}}, \boldsymbol{\mathcal{Y}} \rangle + \frac{\mu}{2} \|\boldsymbol{\mathcal{X}} - \mathbf{U}_1 \circ \cdots \circ \mathbf{U}_N - \boldsymbol{\mathcal{E}}\|_F^2 \tag{4.88}$$

Minimising $\mathcal{L}_\mu$ with respect to each $\mathbf{U}_i$ for $i \in \{1, 2, \dots, N\}$ is done as follows

$$\begin{aligned}
&\arg\min_{\mathbf{U}_i} \mathcal{L}_\mu (\{\mathbf{U}_i\}, \boldsymbol{\mathcal{E}}, \boldsymbol{\mathcal{Y}}) \\
&= \arg\min_{\mathbf{U}_i} a_i \|\mathbf{U}_i\|_* + \langle \boldsymbol{\mathcal{X}} - \mathbf{U}_1 \circ \cdots \circ \mathbf{U}_N - \boldsymbol{\mathcal{E}}, \boldsymbol{\mathcal{Y}} \rangle + \frac{\mu}{2} \|\boldsymbol{\mathcal{X}} - \mathbf{U}_1 \circ \cdots \circ \mathbf{U}_N - \boldsymbol{\mathcal{E}}\|_F^2 \\
&= \arg\min_{\mathbf{U}_i} a_i \mu^{-1} \|\mathbf{U}_i\|_* + \frac{1}{2} \left\|\mathbf{U}_1 \circ \cdots \circ \mathbf{U}_N - \left(\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{E}} + \mu^{-1} \boldsymbol{\mathcal{Y}}\right)\right\|_F^2 \\
&= \arg\min_{\mathbf{U}_i} a_i \mu^{-1} \|\mathbf{U}_i\|_* + \frac{1}{2} \left\|\mathbf{U}_i \tilde{\mathbf{U}}_i - \left(\mathbf{X}_{[i]} - \mathbf{E}_{[i]} + \mu^{-1} \mathbf{Y}_{[i]}\right)\right\|_F^2
\end{aligned} \tag{4.89}$$

where

$$\tilde{\mathbf{U}}_i = \left(\bigodot_{\substack{j=N \\ j \neq i}}^{1} \mathbf{U}_j\right)^T \tag{4.90}$$

The above minimisation problem is a special case of problem (3.35) and therefore we can use the method of linearisation to solve it. Using equation (3.38) and making the substitutions $a = a_i \mu^{-1}$, $\mathbf{A} = \tilde{\mathbf{U}}_i$ and $\mathbf{B} = \mathbf{X}_{[i]} - \mathbf{E}_{[i]} + \mu^{-1} \mathbf{Y}_{[i]}$, we get the following update for $\mathbf{U}_i$

$$\mathbf{U}_i^{k+1} = \mathcal{D}_{a_i (\mu n)^{-1}} \left\{\mathbf{U}_i^k - n^{-1} \left(\mathbf{U}_i^k \tilde{\mathbf{U}}_i - \mathbf{X}_{[i]} + \mathbf{E}_{[i]} - \mu^{-1} \mathbf{Y}_{[i]}\right) \tilde{\mathbf{U}}_i^T\right\} \tag{4.91}$$

Next, we minimise $\mathcal{L}_\mu$ with respect to $\boldsymbol{\mathcal{E}}$ as follows

$$\begin{aligned}
&\arg\min_{\boldsymbol{\mathcal{E}}} \mathcal{L}_\mu\left(\{\mathbf{U}_i\}, \boldsymbol{\mathcal{E}}, \boldsymbol{\mathcal{Y}}\right) \\
&= \arg\min_{\boldsymbol{\mathcal{E}}} \lambda \left\|\boldsymbol{\mathcal{E}}\right\|_1 + \langle \boldsymbol{\mathcal{X}} - \mathbf{U}_1 \circ \cdots \circ \mathbf{U}_N - \boldsymbol{\mathcal{E}}, \boldsymbol{\mathcal{Y}}\rangle + \frac{\mu}{2}\left\|\boldsymbol{\mathcal{X}} - \mathbf{U}_1 \circ \cdots \circ \mathbf{U}_N - \boldsymbol{\mathcal{E}}\right\|_F^2 \\
&= \arg\min_{\boldsymbol{\mathcal{E}}} \lambda\mu^{-1} \left\|\boldsymbol{\mathcal{E}}\right\|_1 + \frac{1}{2}\left\|\boldsymbol{\mathcal{E}} - \left(\boldsymbol{\mathcal{X}} - \mathbf{U}_1 \circ \cdots \circ \mathbf{U}_N + \mu^{-1}\boldsymbol{\mathcal{Y}}\right)\right\|_F^2 \\
&= \mathcal{S}_{\lambda\mu^{-1}}\left\{\boldsymbol{\mathcal{X}} - \mathbf{U}_1 \circ \cdots \circ \mathbf{U}_N + \mu^{-1}\boldsymbol{\mathcal{Y}}\right\} \qquad (4.92)
\end{aligned}$$

The linearisation-based ALM algorithm for RCPD is presented in detail in Algorithm 4.7. Similarly to the substitution-based algorithm presented in Algorithm 4.3, it requires as input the number r of rank-1 tensors forming the CP decomposition, which is the number of columns of factor matrices $\mathbf{U}_i$. Also, the entries of each factor matrix are initialised uniformly at random in the interval $[0, 1]$.

Algorithm 4.7: Robust CP Decomposition via linearisation

```
   Input: Data tensor $\boldsymbol{\mathcal{X}}$, regulariser $\lambda > 0$, weights $\{a_i\}$, number of rank-1 tensors $r$
   Initialise: $\mathbf{U}_i^0 = \text{rand}$, $\boldsymbol{\mathcal{E}}_0 = \mathbf{0}$, $\boldsymbol{\mathcal{Y}}_0 = \mathbf{0}$, $\mu_0 > 0$, $k = 0$
1  while not converged do
2      for $i \in \{1, 2, \dots, N\}$ do
3          $\tilde{\mathbf{U}}_i = \left(\mathbf{U}_N^k \odot \cdots \odot \mathbf{U}_{i+1}^k \odot \mathbf{U}_{i-1}^{k+1} \odot \cdots \odot \mathbf{U}_1^{k+1}\right)^T$
4          $n = \max\left(\left\|\tilde{\mathbf{U}}_i\tilde{\mathbf{U}}_i^T\right\|_F, 1.0\right)$
5          $\mathbf{U}_i^{k+1} = \mathcal{D}_{a_i(\mu_k n)^{-1}}\left\{\mathbf{U}_i^k - n^{-1}\left(\mathbf{U}_i^k\tilde{\mathbf{U}}_i - \mathbf{X}_{[i]} + (\mathbf{E}_k)_{[i]} - \mu_k^{-1}(\mathbf{Y}_k)_{[i]}\right)\tilde{\mathbf{U}}_i^T\right\}$
6      end for
7      $\boldsymbol{\mathcal{E}}_{k+1} = \mathcal{S}_{\lambda\mu_k^{-1}}\left\{\boldsymbol{\mathcal{X}} - \mathbf{U}_1^{k+1} \circ \cdots \circ \mathbf{U}_N^{k+1} + \mu_k^{-1}\boldsymbol{\mathcal{Y}}_k\right\}$
8      $\boldsymbol{\mathcal{Y}}_{k+1} = \boldsymbol{\mathcal{Y}}_k + \mu_k\left(\boldsymbol{\mathcal{X}} - \mathbf{U}_1^{k+1} \circ \cdots \circ \mathbf{U}_N^{k+1} - \boldsymbol{\mathcal{E}}_{k+1}\right)$
9      $\mu_{k+1} = \min\left(\rho\mu_k, \mu_{max}\right)$
10     $k \leftarrow k + 1$
11 end while
   Output: Factor matrices $\{\mathbf{U}_i^k\}$, sparse component $\boldsymbol{\mathcal{E}}_k$
```

4.6 Complexity Analysis and Discussion

We conclude this chapter with a discussion over the advantages and disadvantages of each method, a comparison with the matrix case and an analysis of the computational cost of each algorithm.

4.6.1 Regularisation-Based and Factorisation-Based Methods

Similarly to the matrix case, we categorise all methods in two main groups, based on the mechanism that determines the rank of the low-rank component.

(i) **Regularisation-based methods**. These include RPCA and IRPCA. The rank of the low-rank component is determined by the regularisation parameter λ and the weights $\{a_i\}$.

(ii) **Factorisation-based methods**. These include BRPCA, RHOSVD and RCPD. The common element among the methods of this group is that the low-rank component is factorised in some specific way. Its rank is determined not only by λ and $\{a_i\}$, but also by the factorisation size.

Each factorisation-based method relies on a different way of factorising the low-rank component $\boldsymbol{\mathcal{A}}$. BRPCA factorises each n-mode matricisation of $\boldsymbol{\mathcal{A}}$ as follows

$$\mathbf{A}_{[i]} = \mathbf{U}_i \mathbf{V}_i \tag{4.93}$$

for $i \in \{1, 2, \dots, N\}$, where the number of columns in $\mathbf{U}_i$ (and number of rows in $\mathbf{V}_i$) is r_i. RHOSVD factorises $\mathbf{A}$ based on the HOSVD as follows

$$\boldsymbol{\mathcal{A}} = \boldsymbol{\mathcal{V}} \times_{i=1}^{N} \mathbf{U}_i \tag{4.94}$$

where the number of columns in $\mathbf{U}_i$ is r_i. Finally, RCPD factorises $\boldsymbol{\mathcal{A}}$ based on the CP decomposition as follows

$$\boldsymbol{\mathcal{A}} = \mathbf{U}_1 \circ \cdots \circ \mathbf{U}_N \tag{4.95}$$

where the number of columns of all $\mathbf{U}_i$ is r. Note that BRPCA and RHOSVD involve N factorisation sizes whereas RCPD involves only one.

The significance of the factorisation sizes $\{r_i\}$ and r is that they provide a "hard-coded" controllable upper bound on each n-rank of $\boldsymbol{\mathcal{A}}$ as follows

$$\operatorname{rank}_i (\boldsymbol{\mathcal{A}}) \leq r_i \tag{4.96}$$
$$\operatorname{rank}_i (\boldsymbol{\mathcal{A}}) \leq r \tag{4.97}$$

It becomes obvious that, if the input data tensor is of size $I_1 \times I_2 \times \cdots \times I_N$, the only sensible choices for the factorisation sizes are $r_i \leq I_i$ and $r \leq \max_i (I_i)$. Being able to control the factorisation sizes allows for extra flexibility, however in practice they may not always be easy to tune. Tuning becomes easier in RCPD, since there is only one factorisation size. Finally, as we shall see later, factorisation allows for significant savings in computational cost, particularly when the factorisation sizes are small.

It is important to emphasise the theoretical significance of RHOSVD and RCPD, as a robustification of the classical HOSVD and CP decomposition respectively. Most approaches for fitting a HOSVD (such as those in [25, 44]) or a CP decomposition (such as those in [22, 26, 79]) operate by minimising a fitting cost based on the ℓ_2-norm, which is not robust [41]. Our approach, however, minimises instead the ℓ_1-norm of the error which makes it robust to gross corruptions and outliers. To the best of our knowledge, this is the first formulation of the HOSVD and the CP decomposition within the framework of robust low-rank modelling.

4.6.2 Comparison with the Matrix Case

The main conceptual difference between the tensor case and the matrix case is that tensor methods involve the simultaneous minimisation of more than one ranks. Given an N^{th}-order input data tensor $\boldsymbol{\mathcal{X}}$, whose low-rank component is $\boldsymbol{\mathcal{A}}$, tensor methods aim at minimising each and every $\operatorname{rank}_i (\boldsymbol{\mathcal{A}})$ for $i \in \{1, 2, \dots, N\}$. As will shall see shortly, this introduces increased flexibility but also makes the methods harder to tune.

The multiplicity of ranks to be minimised is perhaps mostly reflected in the introduction of a new set of parameters $\{a_i\}$, for $i \in \{1, 2, \dots, N\}$. This set of parameters is typically taken such that $\sum_{i=1}^{N} a_i = 1$, for consistency with the matrix case. The value of a_i controls the significance of minimising $\operatorname{rank}_i (\boldsymbol{\mathcal{A}})$ as opposed to minimising the other n-ranks of $\boldsymbol{\mathcal{A}}$. In practice, the choice of $\{a_i\}$ is driven by prior knowledge of the problem domain, typically as in the following two cases.

(i) **Minimisation of each n-rank is equally significant**. In this case, we can simply set $a_i = {}^{1}\!/_{N}$ for all $i \in \{1, 2, \dots, N\}$. This makes the problem simpler, especially for regularisation-based methods, where the only remaining parameter to tune is λ.

(ii) **Minimisation of certain n-ranks is not desired**. Say, for instance, that the rank of mode i is not desired to be minimised. Then we can simply set $a_i = 0$ and the corresponding n-rank will remain unaffected.

Regularisation-based methods are not always convex in the tensor case, as they are in the matrix case. RPCA is convex, IRPCA however is not, except in the trivial case where $N = 1$ (which is probably not useful in practice). This removes the guarantee of the global optimality of the solution of IRPCA and makes initialisation important. For this reason, we have proposed a more sophisticated initialisation scheme for the projection matrices of IRPCA, which uses the r_i first left singular vectors of $\mathbf{X}_{[i]}$ for $i \in \{1, 2, \dots, N\}$. Here, $\{r_i\}$ is a set of initialisation parameters controlled by the user. We have experimentally determined that this scheme is capable of providing a suitable initialisation in most practical cases.

4.6.3 Asymptotic Computational Complexity

Similarly to the discussion in section 3.6.2 about the computational cost of matrix methods, in this section we will calculate the asymptotic computational complexity per iteration for each tensor method. We begin by calculating the complexity of the various building blocks used in the tensor methods that did not exist in the matrix methods.

(i) **Tensor addition and scaling**. For tensors $\boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{Y}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ and scalar $a \in \mathbb{R}$, tensor addition $\boldsymbol{\mathcal{X}} + \boldsymbol{\mathcal{Y}}$ and scaling $a\boldsymbol{\mathcal{X}}$ cost $\mathcal{O}\left(\prod_{i=1}^{N} I_i\right)$.

(ii) **Shrinkage operator on tensors**. For a tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$, the calculation of the shrinkage operator $\mathcal{S}_a\{\boldsymbol{\mathcal{X}}\}$ requires the application of a constant-time operation on each tensor entry and therefore costs $\mathcal{O}\left(\prod_{i=1}^{N} I_i\right)$.

(iii) **n-mode product**. For a tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ and a matrix $\mathbf{U} \in \mathbb{R}^{J \times I_n}$, the calculation of the n-mode product $\boldsymbol{\mathcal{X}} \times_n \mathbf{U}$ involves the multiplication by $\mathbf{U}$ of each n-mode fibre of $\boldsymbol{\mathcal{X}}$. Since there are $\prod_{i \neq n} I_i$ such fibres and each multiplication costs $\mathcal{O}(JI_n)$, the total cost of the n-mode product is $\mathcal{O}\left(J \prod_{i=1}^{N} I_i\right)$.

(iv) **Khatri-Rao product**. For matrices $\mathbf{X} \in \mathbb{R}^{m \times n}$ and $\mathbf{Y} \in \mathbb{R}^{k \times n}$, the Khatri-Rao product $\mathbf{X} \odot \mathbf{Y}$ is a matrix of size $mk \times n$, each entry of which needs one multiplication to be computed. Therefore, the cost of the Khatri-Rao product is $\mathcal{O}(mnk)$.

(v) **CP decomposition**. For N factor matrices $\mathbf{U}_i \in \mathbb{R}^{I_i \times r}$, computing the CP decomposition $\mathbf{U}_1 \circ \cdots \circ \mathbf{U}_N$ yields a tensor of size $I_1 \times I_2 \times \cdots \times I_N$, each element of which is given by the addition of r factors, each of which takes $N-1$ multiplications to be computed. Therefore, the total cost of computing the CP decomposition is $\mathcal{O}\left(Nr \prod_{i=1}^{N} I_i\right)$.

Based on the above, Table 4.1 shows the total computational cost per iteration of each tensor algorithm. For simplicity, we have assumed that the input tensor is of size $I_1 \times I_2 \times \cdots \times I_N$ with $N \geq 2$ and $I_1 = I_2 = \cdots = I_N = I$. In the factorisation-based algorithms we have assumed that the factorisation sizes are $r_1 = r_2 = \cdots = r_N = r$. Based on this analysis, we make the following conclusions.

(i) **RPCA and IRPCA**. The complexity of RPCA and IRPCA is a polynomial of the length I of degree $N+1$. This is consistent with the matrix case, where for a matrix of size $n \times n$ the complexity of RPCA and IRPCA is $\mathcal{O}\left(n^3\right)$.

(ii) **Factorisation-based methods**. Since it will always be $r \leq I$, it becomes apparent that the factorisation-based methods, whose cost is dominated by the factor NrI^N, are more

Algorithm	**Convex?**	**Computational cost per iteration**
RPCA	Yes	$\mathcal{O}\left(NI^{N+1}\right)$
BRPCA	No	$\mathcal{O}\left(N\left(rI^{N}+r^2I^{N-1}+r^3\right)\right)$
IRPCA (sub)	No	$\mathcal{O}\left(N^2I^{N+1}\right)$
IRPCA (lin)	No	$\mathcal{O}\left(N^2I^{N+1}\right)$
RHOSVD	No	$\mathcal{O}\left(N\left(rI^{N}+r^2I^{N-1}+\cdots+r^NI+r^{N+1}\right)\right)$
RCPD (sub)	No	$\mathcal{O}\left(N^2rI^{N-1}+N\left(rI^{N}+r^2I^{N-1}+r^3\right)\right)$
RCPD (lin)	No	$\mathcal{O}\left(N^2rI^{N-1}+N\left(rI^{N}+r^2I^{N-1}+r^3\right)\right)$

Table 4.1: Convexity and asymptotic computational complexity per iteration of all tensor algorithms. The size of the input data tensor $\boldsymbol{\mathcal{X}}$ is assumed to be $I_1 \times I_2 \times \cdots \times I_N$, with $N \geq 2$ and $I_1 = I_2 = \cdots = I_N = I$. In the factorisation-based algorithms (i.e. BRPCA, RHOSVD and RCPD), the factorisation size is assumed to be $r_1 = r_2 = \cdots = r_N = r$.

efficient than the regularisation-based methods. In fact, their computational advantage increases as r becomes smaller, which can be the case in many practical applications, as we shall see in chapter 7. In case r is chosen to be equal to I, the cost of regularisation-based and factorisation-based methods becomes the same.

(iii) **Cost of the SVDs**. In the matrix case, the cost of RPCA and IRPCA was dominated by the cost of the SVD. In the tensor case, RPCA requires N SVDs of size $I \times I^{N-1}$ per iteration, which costs $\mathcal{O}\left(NI^{N+1}\right)$ and is indeed the dominant cost. In IRPCA however, N SVDs of size $I \times I$ are required per iteration, for a total cost of $\mathcal{O}\left(NI^3\right)$, which is not dominant, compared to the cost of the n-mode products. This illustrates that in tensor algorithms the SVDs are less of an issue compared to the matrix algorithms. Instead, the dominant costs of the tensor algorithms are mainly due to the several higher-order operations necessary.

(iv) **Scaling with respect to** N. The costs appear to scale exponentially with the tensor order N. However, this is somewhat misleading, as this is the case only if I remains constant. In practice, it would be fairer to consider the total number of elements $M = I^N$ as constant. In this case, the cost of e.g. RPCA becomes $\mathcal{O}\left(NM^{1+\frac{1}{N}}\right)$, which scales almost linearly with N. In any case, N is typically small in practice and thus not a major concern.

In chapter 7, we will evaluate the total execution time of each algorithm experimentally and we will confirm the above theoretical results.

Chapter 5

Extensions to Missing Values and Generalised Norms

In this chapter, we introduce two important extensions for the methods of chapters 3 and 4. Firstly, we show how the methods can be extended to be able to deal with missing values in the data, effectively making them capable of solving the problem of matrix and tensor completion. Secondly, we show how their optimisation problems can be reformulated using the more general Schatten p-norm and elementwise ℓ_q-norm, which allows for a closest approximation to the original intractable optimisation problems. For both extensions, we show how the algorithms described so far can be modified to accommodate them. We discuss *Generalised Scalable Robust Principal Component Analysis*, the extension of ORPCA to general norms, which we first introduced in our paper [69]. We conclude with a summary of the final extended methods, for both matrices and tensors.

5.1 Matrix and Tensor Completion with Missing Values

So far, we have considered the problem of low-rank/sparse decomposition of matrices and tensors that were *fully observed*, that is, all of their entries were known. Unfortunately, this is not always the case, as in many practical applications only a portion of the entries may be observed. In this section, we will show how the algorithms described in chapters 3 and 4 can be easily extended so as to accommodate *missing values*, i.e. unobserved entries, in the data. That is, given an input data matrix $\mathbf{X}$ or tensor $\boldsymbol{\mathcal{X}}$ which is only partially known, we will show how to decompose it into a *fully known* low-rank component and a sparse component accounting for *both* errors and missing values. As a result, we will have *reconstructed* the low-rank component not only from corruptions, but also from missing values.

For simplicity, we will focus on matrix methods first. We start by considering the slightly different but related problem of *low-rank matrix completion*. This problem can be stated as follows: given a data matrix $\mathbf{X} \in \mathbb{R}^{m \times n}$ with only a subset of its entries known, how can we fill in the unknown entries such that the resulting matrix be of the lowest rank possible? More formally, let $\Omega \subseteq \{(i,j) \mid i \in \{1,2,\ldots,m\}, j \in \{1,2,\ldots,n\}\}$ be an index set corresponding to the known entries of $\mathbf{X}$ and let $\pi_\Omega : \mathbb{R}^{m \times n} \to \mathbb{R}^{m \times n}$ be the *sampling operator*, which is defined as

$$(\pi_\Omega(\mathbf{X}))_{ij} = \begin{cases} X_{ij} & (i,j) \in \Omega \\ 0 & (i,j) \notin \Omega \end{cases} \tag{5.1}$$

The sampling operator π_Ω can be also thought of as the *projection operator* onto the linear subspace of matrices with non-zero entries only in Ω. It is trivial to show that π_Ω is also a linear operator, that is, for any $\mathbf{X}_1, \mathbf{X}_2 \in \mathbb{R}^{m \times n}$ and $a_1, a_2 \in \mathbb{R}$ we have

$$\pi_\Omega\left(a_1\mathbf{X}_1 + a_2\mathbf{X}_2\right) = a_1\pi_\Omega\left(\mathbf{X}_1\right) + a_2\pi_\Omega\left(\mathbf{X}_2\right) \tag{5.2}$$

Based on the above definitions, the low-rank matrix completion problem can be stated as follows

$$\min_{\mathbf{A}} \operatorname{rank}\left(\mathbf{A}\right) \quad \text{s.t.} \quad \pi_\Omega\left(\mathbf{X}\right) = \pi_\Omega\left(\mathbf{A}\right) \tag{5.3}$$

As stated previously in the thesis, the optimisation problems involving minimisation of the rank are *intractable*. Therefore, using the fact that the nuclear norm is the tightest convex surrogate of the rank function, the above problem can be relaxed to the following tractable convex problem

$$\min_{\mathbf{A}} \left\|\mathbf{A}\right\|_* \quad \text{s.t.} \quad \pi_\Omega\left(\mathbf{X}\right) = \pi_\Omega\left(\mathbf{A}\right) \tag{5.4}$$

Candès and Recht [17] proved the rather remarkable result that the above convex problem can, with high probability, *exactly recover* the complete matrix $\mathbf{X}$, provided that $\mathbf{X}$ is of low-rank and a sufficient number of entries has been observed (see their paper for more details).

Here, we will take advantage of the similarity between the relaxed low-rank matrix completion problem (5.4) and the matrix RPCA problem (3.4) in order to extend RPCA (and all other methods) to be able to handle missing values. The key idea is to consider that the missing values are themselves a form of *gross corruption*. Let $\mathbf{E} = \mathbf{X} - \mathbf{A}$ and notice that

$$\pi_\Omega\left(\mathbf{X}\right) = \pi_\Omega\left(\mathbf{A}\right) \quad \Rightarrow \quad \pi_\Omega\left(\mathbf{X} - \mathbf{A}\right) = \mathbf{0} \quad \Rightarrow \quad \pi_\Omega\left(\mathbf{E}\right) = \mathbf{0} \tag{5.5}$$

Hence, problem (5.4) can be equivalently written as

$$\min_{\mathbf{A},\mathbf{E}} \left\|\mathbf{A}\right\|_* \quad \text{s.t.} \quad \begin{array}{c} \mathbf{X} = \mathbf{A} + \mathbf{E} \\ \pi_\Omega\left(\mathbf{E}\right) = \mathbf{0} \end{array} \tag{5.6}$$

In the above, the unknown entries of $\mathbf{X}$ can be arbitrarily set to *any value* (e.g. to zero for simplicity), since this is compensated by the entries of $\mathbf{E}$ not in Ω. In order to allow for noise in the observed values, we can generalise the above to the following

$$\min_{\mathbf{A},\mathbf{E}} \left\|\mathbf{A}\right\|_* \quad \text{s.t.} \quad \begin{array}{c} \mathbf{X} = \mathbf{A} + \mathbf{E} \\ \left\|\pi_\Omega\left(\mathbf{E}\right)\right\|_1 \leq \epsilon \end{array} \tag{5.7}$$

where $\epsilon \geq 0$ is a parameter that controls the level of noise, as this is measured by the ℓ_1-norm over the observed entries. The usage of the ℓ_1-norm allows for robustness to gross corruptions. The above can be also seen in a "dual" form, by introducing a positive regulariser λ and rewriting it as

$$\min_{\mathbf{A},\mathbf{E}} \left\|\mathbf{A}\right\|_* + \lambda\left\|\pi_\Omega\left(\mathbf{E}\right)\right\|_1 \quad \text{s.t.} \quad \mathbf{X} = \mathbf{A} + \mathbf{E} \tag{5.8}$$

That is, for every value of $\epsilon \geq 0$, there exists a value $\lambda > 0$ for which the two above problems are *equivalent*, in the sense that they accept the same solution. In general, as $\epsilon \to 0^+$, we have that $\lambda \to +\infty$.

It now becomes obvious that problem (5.8) is the desired extension of matrix RPCA to handling missing values, as problem (5.8) reduces to the RPCA problem (3.4) if all entries of $\mathbf{X}$ are known. It is now straightforward to extend the remaining matrix methods to be able to handle missing values as well, by simply replacing $\left\|\mathbf{E}\right\|_1$ with $\left\|\pi_\Omega\left(\mathbf{E}\right)\right\|_1$ in their optimisation problems.

The same idea can be applied to tensor methods. Let $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ be an N^{th}-order input data tensor which is partially observed and let Ω be the index set of its observed entries

$$\Omega \subseteq \left\{\left(i_1, i_2, \ldots, i_N\right) \mid i_n \in \left\{1, 2, \ldots, I_n\right\}, n \in \left\{1, 2, \ldots, N\right\}\right\} \tag{5.9}$$

We can easily extend the sampling operator π_Ω to tensors as follows

$$\left(\pi_\Omega\left(\boldsymbol{\mathcal{X}}\right)\right)_{i_1 i_2 \cdots i_N} = \begin{cases} X_{i_1 i_2 \cdots i_N} & (i_1, i_2, \ldots, i_N) \in \Omega \\ 0 & (i_1, i_2, \ldots, i_N) \notin \Omega \end{cases} \tag{5.10}$$

Then, extending the tensor methods to be able to handle missing values is done simply by replacing $\|\boldsymbol{\mathcal{E}}\|_1$ with $\|\pi_\Omega(\boldsymbol{\mathcal{E}})\|_1$ in their optimisation problems. For instance, tensor RPCA with missing values will be written as

$$\min_{\boldsymbol{\mathcal{A}},\boldsymbol{\mathcal{E}}} \sum_{i=1}^{N} a_i \left\|\mathbf{A}_{[i]}\right\|_* + \lambda \left\|\pi_\Omega\left(\boldsymbol{\mathcal{E}}\right)\right\|_1 \quad \text{s.t.} \quad \boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{A}} + \boldsymbol{\mathcal{E}} \tag{5.11}$$

In order to solve the extended optimisation problems involving the sampling operator π_Ω, it is important to note that, in all the algorithms presented in chapters 3 and 4, the elementwise ℓ_1-norm of the sparse component *only* appears in the form of the shrinkage operator $\mathcal{S}_a\{\cdot\}$. Using for demonstration the more general case of tensors, in the update of the sparse component $\boldsymbol{\mathcal{E}}$, some problem of the following form had to be solved

$$\arg\min_{\boldsymbol{\mathcal{E}}} a \|\boldsymbol{\mathcal{E}}\|_1 + \frac{1}{2}\|\boldsymbol{\mathcal{E}} - \boldsymbol{\mathcal{E}}_c\|_F^2 \tag{5.12}$$

whose solution was given by $\mathcal{S}_a\{\boldsymbol{\mathcal{E}}_c\}$. In the missing value extension, we have to solve instead the following problem

$$\arg\min_{\boldsymbol{\mathcal{E}}} a \left\|\pi_\Omega\left(\boldsymbol{\mathcal{E}}\right)\right\|_1 + \frac{1}{2}\|\boldsymbol{\mathcal{E}} - \boldsymbol{\mathcal{E}}_c\|_F^2 \tag{5.13}$$

that is, we need to find the proximal operator $\mathcal{P}_f\{\cdot\}$ of the function $f(\boldsymbol{\mathcal{E}}) = a\|\pi_\Omega(\boldsymbol{\mathcal{E}})\|_1$. We refer to this proximal operator as *selective shrinkage* and we show how to calculate it in the following theorem.

Theorem 5.1 (Selective shrinkage)**.** *Let $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ be an N^{th}-order tensor, Ω be an index set of its entries and $f(\boldsymbol{\mathcal{X}}) = a\|\pi_\Omega(\boldsymbol{\mathcal{X}})\|_1$. The proximal operator of f, represented by $\mathcal{P}_f\{\cdot\}$, is a tensor of size $I_1 \times I_2 \times \cdots \times I_N$, whose entries are given by*

$$\left(\mathcal{P}_f\left\{\boldsymbol{\mathcal{X}}\right\}\right)_{i_1 i_2 \cdots i_N} = \begin{cases} \mathcal{S}_a\left\{X_{i_1 i_2 \cdots i_N}\right\} & (i_1, i_2, \ldots, i_N) \in \Omega \\ X_{i_1 i_2 \cdots i_N} & (i_1, i_2, \ldots, i_N) \notin \Omega \end{cases} \tag{5.14}$$

For convenience, we will refer to $\mathcal{P}_f\{\cdot\}$ as selective shrinkage and we will denote it by $\mathcal{S}_{a,\Omega}\{\cdot\}$.

Proof. The proximal operator can be written as

$$\begin{aligned} \mathcal{P}_f\left\{\boldsymbol{\mathcal{X}}\right\} &= \arg\min_{\boldsymbol{\mathcal{Y}}} a \left\|\pi_\Omega\left(\boldsymbol{\mathcal{Y}}\right)\right\|_1 + \frac{1}{2}\|\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{Y}}\|_F^2 \\ &= \arg\min_{\boldsymbol{\mathcal{Y}}} \sum_{(i_1,i_2,\ldots,i_N)\in\Omega} \left(a\left|Y_{i_1 i_2 \cdots i_N}\right| + \frac{1}{2}\left(X_{i_1 i_2 \cdots i_N} - Y_{i_1 i_2 \cdots i_N}\right)^2\right) \\ &\quad + \sum_{(i_1,i_2,\ldots,i_N)\notin\Omega} \left(\frac{1}{2}\left(X_{i_1 i_2 \cdots i_N} - Y_{i_1 i_2 \cdots i_N}\right)^2\right) \end{aligned} \tag{5.15}$$

Since the objective function is separable in the elements of $\boldsymbol{\mathcal{Y}}$, we have

(i) For $(i_1, i_2, \ldots, i_N) \in \Omega$

$$\left(\mathcal{P}_f\left\{\boldsymbol{\mathcal{X}}\right\}\right)_{i_1 i_2 \cdots i_N} = \arg\min_{Y_{i_1 i_2 \cdots i_N}} a\left|Y_{i_1 i_2 \cdots i_N}\right| + \frac{1}{2}\left(X_{i_1 i_2 \cdots i_N} - Y_{i_1 i_2 \cdots i_N}\right)^2 = \mathcal{S}_a\left\{X_{i_1 i_2 \cdots i_N}\right\} \tag{5.16}$$

(ii) For $(i_1, i_2, \dots, i_N) \notin \Omega$

$$\left(\mathcal{P}_f\left\{\boldsymbol{\mathcal{X}}\right\}\right)_{i_1 i_2 \cdots i_N} = \arg\min_{Y_{i_1 i_2 \cdots i_N}} \frac{1}{2}\left(X_{i_1 i_2 \cdots i_N} - Y_{i_1 i_2 \cdots i_N}\right)^2 = X_{i_1 i_2 \cdots i_N} \tag{5.17}$$

□

To conclude, in order to extend all algorithms presented in chapters 3 and 4, all that is necessary is to *replace* the shrinkage operator $\mathcal{S}_a\{\cdot\}$ used in the update of the sparse component with the selective shrinkage $\mathcal{S}_{a,\Omega}\{\cdot\}$ described in the above theorem. Of course, the index set Ω also needs to be given as input to the algorithm. The entries of the input data matrix/tensor that are not known can be set to an arbitrary value (such as zero) and the sparse component will account for them.

5.2 Generalisation to Schatten p-norms and Elementwise ℓ_q-norms

As we saw in the beginning of chapter 3, the optimisation problem which we primarily wish to solve is, in the matrix case, the following

$$\min_{\mathbf{A},\mathbf{E}} \operatorname{rank}(\mathbf{A}) + \lambda \|\mathbf{E}\|_0 \quad \text{s.t.} \quad \mathbf{X} = \mathbf{A} + \mathbf{E} \tag{5.18}$$

Similarly, for tensors, in the beginning of chapter 4 we formulated the following optimisation problem

$$\min_{\boldsymbol{\mathcal{A}},\boldsymbol{\mathcal{E}}} \sum_{i=1}^{N} a_i \operatorname{rank}_i(\boldsymbol{\mathcal{A}}) + \lambda \|\boldsymbol{\mathcal{E}}\|_0 \quad \text{s.t.} \quad \boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{A}} + \boldsymbol{\mathcal{E}} \tag{5.19}$$

Since problems directly involving the rank and the ℓ_0-norm are intractable, we approached the above problem by appropriately relaxing the rank and the ℓ_0-norm using either their convex envelopes or making use of factorisation methods (or a combination thereof). In this section, we will generalise all the methods introduced in chapters 3 and 4 such that a better (at least theoretically) approximation to the above two problems can be achieved.

We shall demonstrate our generalisation approach using matrix RPCA, but the following discussion pertains to all methods in the same way. Remember that RPCA was motivated by the usage of the nuclear norm and the elementwise ℓ_1-norm as the closest convex approximation to the rank and the ℓ_0-norm respectively, replacing the original intractable optimisation problem by the following convex one

$$\min_{\mathbf{A},\mathbf{E}} \|\mathbf{A}\|_* + \lambda \|\mathbf{E}\|_1 \quad \text{s.t.} \quad \mathbf{X} = \mathbf{A} + \mathbf{E} \tag{5.20}$$

Motivated by [65], we notice that, as explained in section 2.1, the nuclear norm is a special case of the Schatten p-norm and the elementwise ℓ_1-norm is a special case of the elementwise ℓ_q-norm, for $p = q = 1$. Therefore, we can generalise the above convex problem to the following

$$\min_{\mathbf{A},\mathbf{E}} \|\mathbf{A}\|_{S_p}^p + \lambda \|\mathbf{E}\|_q^q \quad \text{s.t.} \quad \mathbf{X} = \mathbf{A} + \mathbf{E} \tag{5.21}$$

which for $p = q = 1$ gives the convex problem as a special case.

The above generalisation becomes particularly interesting in the case where $0 < p < 1$ and $0 < q < 1$. As we have shown in Theorem 2.4, the following holds

$$\lim_{p \to 0^+} \|\mathbf{A}\|_{S_p}^p = \operatorname{rank}(\mathbf{A}) \tag{5.22}$$

$$\lim_{q \to 0^+} \|\mathbf{E}\|_q^q = \|\mathbf{E}\|_0 \tag{5.23}$$

The above means that by allowing p and q to become small, the generalised RPCA problem (5.21) can arbitrarily approximate the original intractable problem (5.18). What is more, the same idea can be used with all matrix or tensor methods, by simply replacing $\|\cdot\|_*$ with $\|\cdot\|_{S_p}^p$ and $\|\cdot\|_1$ with $\|\cdot\|_q^q$ in their optimisation problems. For instance, the generalised version of tensor RPCA becomes the following

$$\min_{\boldsymbol{\mathcal{A}},\boldsymbol{\mathcal{E}}} \sum_{i=1}^{N} a_i \left\|\mathbf{A}_{[i]}\right\|_{S_p}^p + \lambda \left\|\boldsymbol{\mathcal{E}}\right\|_q^q \quad \text{s.t.} \quad \boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{A}} + \boldsymbol{\mathcal{E}} \tag{5.24}$$

and it is straightforward to see that the above problem approximates arbitrarily the intractable problem (5.19) by allowing p and q to approach zero.

For the above generalisation to be of any practical use, we need to extend the algorithms presented in chapters 3 and 4 to be able to handle the general norms. The key to achieving this is to notice that the nuclear norm only appears in the form of singular value thresholding $\mathcal{D}_a\{\cdot\}$ and the elementwise ℓ_1-norm only appears in the form of shrinkage $\mathcal{S}_a\{\cdot\}$. That is—using the more general tensor case for demonstration—they only appeared in optimisation problems of the following form

$$\arg\min_{\mathbf{A}} a \left\|\mathbf{A}\right\|_* + \frac{1}{2}\left\|\mathbf{A} - \mathbf{A}_c\right\|_F^2 \tag{5.25}$$

$$\arg\min_{\boldsymbol{\mathcal{E}}} a \left\|\boldsymbol{\mathcal{E}}\right\|_1 + \frac{1}{2}\left\|\boldsymbol{\mathcal{E}} - \boldsymbol{\mathcal{E}}_c\right\|_F^2 \tag{5.26}$$

whose solutions were given by $\mathcal{D}_a\{\mathbf{A}_c\}$ and $\mathcal{S}_a\{\boldsymbol{\mathcal{E}}_c\}$ respectively. In the generalised version, the following problems have to be solved instead.

$$\arg\min_{\mathbf{A}} a \left\|\mathbf{A}\right\|_{S_p}^p + \frac{1}{2}\left\|\mathbf{A} - \mathbf{A}_c\right\|_F^2 \tag{5.27}$$

$$\arg\min_{\boldsymbol{\mathcal{E}}} a \left\|\boldsymbol{\mathcal{E}}\right\|_q^q + \frac{1}{2}\left\|\boldsymbol{\mathcal{E}} - \boldsymbol{\mathcal{E}}_c\right\|_F^2 \tag{5.28}$$

that is, we need to compute the proximal operators of $\|\cdot\|_{S_p}^p$ and $\|\cdot\|_q^q$ respectively. We will refer to these proximal operators as *generalised singular value thresholding* and *generalised shrinkage* respectively. The following three theorems show how to compute them.

Theorem 5.2 (Generalised shrinkage)**.** *Let $x \in \mathbb{R}$ and $f(x) = a|x|^q$ with $a > 0$ and $0 < q \leq 1$. Define $h(y) = \alpha|y|^q + \frac{1}{2}(x-y)^2$, $c_1 = [\alpha q(1-q)]^{\frac{1}{2-q}}$ and $c_2 = c_1 + \alpha q|c_1|^{q-1}$. Then, the proximal operator of f becomes*

$$\mathcal{P}_f\{x\} = \begin{cases} 0 & -c_2 \leq x \leq c_2 \\ \arg\min_{y\in\{0,y_+\}} h(y) & x > c_2 \\ \arg\min_{y\in\{0,y_-\}} h(y) & x < -c_2 \end{cases} \tag{5.29}$$

where y_+ and y_- are the roots of $h'(y) = \alpha q|y|^{q-1}\,sgn(y) - x + y = 0$ in $[c_1, x]$ and $[x, -c_1]$ respectively. We shall refer to the above proximal operator as generalised shrinkage and represent it by $\mathcal{S}_a^q\{x\}$.

Proof. The theorem can be proven by considering the monotony of h. See [65] for a complete proof. □

In order to compute $\mathcal{S}_a^q\{\cdot\}$, one needs a method for finding roots y_+ and y_-. Nie et al. [65] suggest using the Newton-Raphson root finding method [11], initialised at x.

Theorem 5.3 (Generalised shrinkage for tensors). *Let $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ be an N^{th}-order tensor and $f(\boldsymbol{\mathcal{X}}) = a \left\|\boldsymbol{\mathcal{X}}\right\|_q^q$ with $a > 0$ and $0 < q \leq 1$. Then, the proximal operator of f is a tensor of the same size as $\boldsymbol{\mathcal{X}}$ whose elements are given by*

$$\left(\mathcal{P}_f\left\{\boldsymbol{\mathcal{X}}\right\}\right)_{i_1 i_2 \cdots i_N} = \mathcal{S}_a^q\left\{X_{i_1 i_2 \cdots i_N}\right\} \tag{5.30}$$

For convenience, we will represent $\mathcal{P}_f\{\boldsymbol{\mathcal{X}}\}$ as $\mathcal{S}_a^q\{\boldsymbol{\mathcal{X}}\}$ and we will imply that $\mathcal{S}_a^q\{\cdot\}$ is applied elementwise.

Proof. The proximal operator can be written as

$$\begin{aligned}\mathcal{P}_f\left\{\boldsymbol{\mathcal{X}}\right\} &= \arg\min_{\boldsymbol{\mathcal{Y}}} a \left\|\boldsymbol{\mathcal{Y}}\right\|_q^q + \frac{1}{2}\left\|\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{Y}}\right\|_F^2 \\ &= \arg\min_{\boldsymbol{\mathcal{Y}}} \sum_{i_i=1}^{I_1}\sum_{i_2=1}^{I_2}\cdots\sum_{i_N=1}^{I_N}\left(a\left|Y_{i_1 i_2\cdots i_N}\right|^q + \frac{1}{2}\left(X_{i_1 i_2 \cdots i_N} - Y_{i_1 i_2 \cdots i_N}\right)^2\right)\end{aligned} \tag{5.31}$$

Since the objective function is separable in the elements of $\boldsymbol{\mathcal{Y}}$, we have

$$\begin{aligned}\left(\mathcal{P}_f\left\{\boldsymbol{\mathcal{X}}\right\}\right)_{i_1 i_2 \cdots i_N} &= \arg\min_{Y_{i_1 i_2 \cdots i_N}} a\left|Y_{i_1 i_2\cdots i_N}\right|^q + \frac{1}{2}\left(X_{i_1 i_2 \cdots i_N} - Y_{i_1 i_2 \cdots i_N}\right)^2 \\ &= \mathcal{S}_a^q\left\{X_{i_1 i_2 \cdots i_N}\right\}\end{aligned} \tag{5.32}$$

□

Theorem 5.4 (Genaralised singular value thresholding). *Let $\mathbf{X} \in \mathbb{R}^{m \times n}$ be a matrix of size $m \times n$ and $f(\mathbf{X}) = a\left\|\mathbf{X}\right\|_{S_p}^p$ with $a > 0$ and $0 < p \leq 1$. Assume the Singular Value Decomposition $\mathbf{X} = \mathbf{U}\mathbf{S}\mathbf{V}^T$. Then, the proximal operator of f becomes*

$$\mathcal{P}_f\left\{\mathbf{X}\right\} = \mathbf{U}\mathcal{S}_a^p\left\{\mathbf{S}\right\}\mathbf{V}^T \tag{5.33}$$

We shall refer to the above proximal operator as generalised singular value thresholding *and represent it by $\mathcal{D}_a^p\{\mathbf{X}\}$.*

Proof. This is Theorem 2 in in [65]. See proof therein. □

It is not difficult to see that, for $p = q = 1$, $\mathcal{S}_a^q\{\cdot\}$ reduces to $\mathcal{S}_a\{\cdot\}$ and $\mathcal{D}_a^p\{\cdot\}$ reduces to $\mathcal{D}_a\{\cdot\}$. The generalisation of all algorithms of chapters 3 and 4 to the Schatten p-norm and the elementwise ℓ_q-norm becomes now straightforward; simply replace $\mathcal{D}_a\{\cdot\}$ with $\mathcal{D}_a^p\{\cdot\}$ and $\mathcal{S}_a\{\cdot\}$ with $\mathcal{S}_a^q\{\cdot\}$.

Even though the above extensions are theoretically appealing, since they provide a way to arbitrarily approximate the original intractable optimisation problems involving the rank and the ℓ_0-norm, in practice they suffer from two major drawbacks that limit their applicability. Firstly, in order to compute the generalised shrinkage, we need to iteratively solve—using the Newton-Raphson method—one optimisation problem for each tensor element. This can make the algorithms considerably slower. Secondly, for $p < 1$ and $q < 1$, the Schatten p-norm and the elementwise ℓ_q-norm, and thus the optimisation problem, become non-convex. Even worse, this non-convexity becomes more pronounced as p and q approach zero. As a result, the algorithms converge more slowly and the obtained solution can be considerably suboptimal.

Empirically, we have found that the generalisation to Schatten p-norms and elementwise ℓ_q-norms works particularly well for matrix ORPCA. The generalised version of matrix ORPCA can be written as follows

$$\min_{\mathbf{V},\mathbf{E},\mathbf{U}} \left\|\mathbf{V}\right\|_{S_p}^p + \lambda\left\|\mathbf{E}\right\|_q^q \quad \text{s.t.} \quad \begin{array}{l}\mathbf{X} = \mathbf{U}\mathbf{V} + \mathbf{E} \\ \mathbf{U}^T\mathbf{U} = \mathbf{I}\end{array} \tag{5.34}$$

which can be solved by algorithm 3.6 provided that $\mathcal{D}_{\mu_k^{-1}}\{\cdot\}$ in step 2 be replaced by $\mathcal{D}^p_{\mu_k^{-1}}\{\cdot\}$ and $\mathcal{S}_{\lambda\mu_k^{-1}}\{\cdot\}$ in step 3 be replaced by $\mathcal{S}^q_{\lambda\mu_k^{-1}}\{\cdot\}$. For a small number of principal components r (i.e. number of columns of $\mathbf{U}$), matrices $\mathbf{U}$ and $\mathbf{V}$ are of small size and the problem is strongly regularised, due to the orthonormality constraint $\mathbf{U}^T\mathbf{U} = \mathbf{I}$. We believe that this is the reason why the algorithm behaves well even for very small values of p and q. We refer to this version of ORPCA as *Generalised Scalable Robust Principal Component Analysis*. We have shown experimentally that GSRPCA can outperform ORPCA in image denoising, producing crisper denoised images as p and q approach zero (unlike ORPCA which oversmooths the images). Our findings have been published in [69]. See our paper for further details.

5.3 Bringing Everything Together: Summary of Methods

The two extensions presented in the above sections (i.e. extensions to missing values and to general norms) are independent and therefore can be easily combined. For instance, considering both generalisations, the tensor RPCA problem becomes the following

$$\min_{\boldsymbol{\mathcal{A}},\boldsymbol{\mathcal{E}}} \sum_{i=1}^{N} a_i \left\|\mathbf{A}_{[i]}\right\|_{S_p}^p + \lambda \left\|\pi_\Omega\left(\boldsymbol{\mathcal{E}}\right)\right\|_q^q \quad \text{s.t.} \quad \boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{A}} + \boldsymbol{\mathcal{E}} \tag{5.35}$$

To complete the puzzle, we just need to solve problems of the following form

$$\arg\min_{\boldsymbol{\mathcal{E}}} a \left\|\pi_\Omega\left(\boldsymbol{\mathcal{E}}\right)\right\|_q^q + \frac{1}{2}\left\|\boldsymbol{\mathcal{E}} - \boldsymbol{\mathcal{E}}_c\right\|_F^2 \tag{5.36}$$

It is easy to see, by combining the proofs of theorems 5.1 and 5.3, that the proximal operator defined by the above problem is a combination of selective shrinkage $\mathcal{S}_{a,\Omega}\{\cdot\}$ and generalised shrinkage $\mathcal{S}^q_a\{\cdot\}$, which we shall refer to as *generalised selective shrinkage* and represent it by $\mathcal{S}^q_{a,\Omega}\{\cdot\}$. It is trivial to show that for a tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1\times I_2\times\cdots\times I_N}$, $\mathcal{S}^q_{a,\Omega}\{\boldsymbol{\mathcal{X}}\}$ is also a tensor of the same size as $\boldsymbol{\mathcal{X}}$ whose elements are given by

$$\left(\mathcal{S}^q_{a,\Omega}\{\boldsymbol{\mathcal{X}}\}\right)_{i_1 i_2\cdots i_N} = \begin{cases} \mathcal{S}^q_a\{X_{i_1 i_2\cdots i_N}\} & (i_1, i_2, \ldots, i_N) \in \Omega \\ X_{i_1 i_2\cdots i_N} & (i_1, i_2, \ldots, i_N) \notin \Omega \end{cases} \tag{5.37}$$

We are now in a position to present all the methods for robust low-rank modelling, both with matrices and tensors, capable of dealing with missing values and generalised to the Schatten p-norm and the elementwise ℓ_q-norm. Table 5.1 summarises all methods and the corresponding optimisation problems. By simply replacing

(i) $\mathcal{D}_a\{\cdot\}$ with $\mathcal{D}^p_a\{\cdot\}$ in the update of the low-rank component, and

(ii) $\mathcal{S}_a\{\cdot\}$ with $\mathcal{S}^q_{a,\Omega}\{\cdot\}$ in the update of the sparse component

where applicable in the algorithms of chapters 3 and 4, we can easily derive for each method of Table 5.1 the algorithm(s) that solve it.

	Method	Objective function	Constraints
Matrices	RPCA	$\|\mathbf{A}\|_{S_p}^p + \lambda \left\|\pi_\Omega\left(\mathbf{E}\right)\right\|_q^q$	$\mathbf{X} = \mathbf{A} + \mathbf{E}$
	BRPCA	$\frac{1}{2}\left(\|\mathbf{U}\|_F^2 + \|\mathbf{V}\|_F^2\right) + \lambda \left\|\pi_\Omega\left(\mathbf{E}\right)\right\|_q^q$	$\mathbf{X} = \mathbf{U}\mathbf{V} + \mathbf{E}$
	IRPCA	$\|\mathbf{P}\|_{S_p}^p + \lambda \left\|\pi_\Omega\left(\mathbf{E}\right)\right\|_q^q$	$\mathbf{X} = \mathbf{P}\mathbf{X} + \mathbf{E}$
	ORPCA	$\|\mathbf{V}\|_{S_p}^p + \lambda \left\|\pi_\Omega\left(\mathbf{E}\right)\right\|_q^q$	$\mathbf{X} = \mathbf{U}\mathbf{V} + \mathbf{E}$ $\mathbf{U}^T\mathbf{U} = \mathbf{I}$
	ROSL	$\|\mathbf{V}\|_{\text{row-1}} + \lambda \left\|\pi_\Omega\left(\mathbf{E}\right)\right\|_q^q$	$\mathbf{X} = \mathbf{U}\mathbf{V} + \mathbf{E}$ $\mathbf{U}^T\mathbf{U} = \mathbf{I}$
Tensors	RPCA	$\sum_{i=1}^{N} a_i \left\|\mathbf{A}_{[i]}\right\|_{S_p}^p + \lambda \left\|\pi_\Omega\left(\boldsymbol{\mathcal{E}}\right)\right\|_q^q$	$\boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{A}} + \boldsymbol{\mathcal{E}}$
	BRPCA	$\sum_{i=1}^{N} \frac{a_i}{2}\left(\|\mathbf{U}_i\|_F^2 + \|\mathbf{V}_i\|_F^2\right) + \lambda \left\|\pi_\Omega\left(\boldsymbol{\mathcal{E}}\right)\right\|_q^q$	$\boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{A}} + \boldsymbol{\mathcal{E}}$ $\mathbf{A}_{[i]} = \mathbf{U}_i\mathbf{V}_i$
	IRPCA	$\sum_{i=1}^{N} a_i \|\mathbf{P}_i\|_{S_p}^p + \lambda \left\|\pi_\Omega\left(\boldsymbol{\mathcal{E}}\right)\right\|_q^q$	$\boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{X}} \times_{i=1}^N \mathbf{P}_i + \boldsymbol{\mathcal{E}}$
	RHOSVD	$\sum_{i=1}^{N} a_i \left\|\mathbf{V}_{[i]}\right\|_{S_p}^p + \left\|\pi_\Omega\left(\boldsymbol{\mathcal{E}}\right)\right\|_q^q$	$\boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{V}} \times_{i=1}^N \mathbf{U}_i + \boldsymbol{\mathcal{E}}$ $\mathbf{U}_i^T\mathbf{U}_i = \mathbf{I}$
	RCPD	$\sum_{i=1}^{N} a_i \|\mathbf{U}_i\|_{S_p}^p + \lambda \left\|\pi_\Omega\left(\boldsymbol{\mathcal{E}}\right)\right\|_q^q$	$\boldsymbol{\mathcal{X}} = \mathbf{U}_1 \circ \cdots \circ \mathbf{U}_N + \boldsymbol{\mathcal{E}}$

Table 5.1: Summary of the generalised versions of all methods, for both matrices and tensors, that are capable of dealing with missing values and make use of general Schatten p-norms and elementwise ℓ_q-norms. For each method, the corresponding objective function and constraints are given.

Chapter 6

Implementation Details

In this chapter, we provide details on the implementation of the algorithms described in this thesis. We discuss our choice of convergence criteria, we explain how to best tune the parameters required by APG and ALM, and we describe the implementation of our software toolbox.

6.1 Convergence Criteria

All the algorithms described in chapters 3 and 4 are iterative; that is, they continue iterating until some condition, the *convergence criterion*, is satisfied. The convergence criterion is checked at the end of every iteration. If it is satisfied, then the optimal point has been reached and the algorithm terminates. Otherwise, the algorithm moves on to the next iteration.

In practice, with optimisation algorithms, three types of convergence criteria are commonly used, summarised as follows.

(i) **Value of the objective function**. The objective function is monitored after every iteration and when it ceases to decrease (or ceases to increase, for a maximisation problem) the algorithm terminates.

(ii) **Value of the decision variables**. The change in the decision variables is monitored after every iteration and when it becomes smaller than a threshold the algorithm terminates.

(iii) **KKT conditions**. The algorithm terminates when the Karush-Kuhn-Tucker necessary conditions for optimality [46] are satisfied.

One of the KKT conditions is the *feasibility* condition, which requires that the optimal solution be feasible, i.e. satisfy the constraints of the problem. In our implementation, the convergence criterion is based on the feasibility condition. In each iteration, we monitor the error in satisfying the constraints, in the ℓ_2 sense, and terminate the algorithm when this error is below some specified small threshold. For example, in matrix RPCA, where the constraint is $\mathbf{X} = \mathbf{A} + \mathbf{E}$, we measure the following relative error at the end of iteration k

$$E_k = \frac{\|\mathbf{X} - \mathbf{A}_k - \mathbf{E}_k\|_F}{\|\mathbf{X}\|_F} \tag{6.1}$$

and we terminate the algorithm when $E_k \leq \delta$, where δ is the specified threshold. For the rest of the algorithms, the process is similar. Table 6.1 summarises the convergence criterion used by each algorithm. Note than for the algorithms that have more than one constraints (such as tensor RPCA), the error of each constraint is measured and the algorithm terminates only if all of them are below δ. In our implementation, we used for all algorithms the value $\delta = 10^{-7}$.

	Algorithm	**Convergence criterion** ($\delta = 10^{-7}$)
Matrices	RPCA (apg)	$\left\lVert \mathbf{X} - \mathbf{A} - \mathbf{E} \right\rVert_F \leq \delta \left\lVert \mathbf{X} \right\rVert_F$
	RPCA (alm)	$\left\lVert \mathbf{X} - \mathbf{A} - \mathbf{E} \right\rVert_F \leq \delta \left\lVert \mathbf{X} \right\rVert_F$
	BRPCA	$\left\lVert \mathbf{X} - \mathbf{UV} - \mathbf{E} \right\rVert_F \leq \delta \left\lVert \mathbf{X} \right\rVert_F$
	IRPCA (sub)	$\max\Big(\left\lVert \mathbf{X} - \mathbf{PX} - \mathbf{E} \right\rVert_F , \left\lVert \mathbf{P} - \mathbf{J} \right\rVert_F \Big) \leq \delta \left\lVert \mathbf{X} \right\rVert_F$
	IRPCA (lin)	$\left\lVert \mathbf{X} - \mathbf{PX} - \mathbf{E} \right\rVert_F \leq \delta \left\lVert \mathbf{X} \right\rVert_F$
	ORPCA	$\left\lVert \mathbf{X} - \mathbf{UV} - \mathbf{E} \right\rVert_F \leq \delta \left\lVert \mathbf{X} \right\rVert_F$
	ROSL	$\left\lVert \mathbf{X} - \mathbf{UV} - \mathbf{E} \right\rVert_F \leq \delta \left\lVert \mathbf{X} \right\rVert_F$
Tensors	RPCA	$\max\Big(\left\lVert \boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{A}} - \boldsymbol{\mathcal{E}} \right\rVert_F , \Big\{ \left\lVert \mathbf{A}_{[i]} - \mathbf{J}_i \right\rVert_F \Big\} \Big) \leq \delta \left\lVert \boldsymbol{\mathcal{X}} \right\rVert_F$
	BRPCA	$\max\Big(\left\lVert \boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{A}} - \boldsymbol{\mathcal{E}} \right\rVert_F , \Big\{ \left\lVert \mathbf{A}_{[i]} - \mathbf{U}_i \mathbf{V}_i \right\rVert_F \Big\} \Big) \leq \delta \left\lVert \boldsymbol{\mathcal{X}} \right\rVert_F$
	IRPCA (sub)	$\max\Big(\left\lVert \boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{X}} \times_{i=1}^{N} \mathbf{P}_i - \boldsymbol{\mathcal{E}} \right\rVert_F , \Big\{ \left\lVert \mathbf{P}_i - \mathbf{J}_i \right\rVert_F \Big\} \Big) \leq \delta \left\lVert \boldsymbol{\mathcal{X}} \right\rVert_F$
	IRPCA (lin)	$\left\lVert \boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{X}} \times_{i=1}^{N} \mathbf{P}_i - \boldsymbol{\mathcal{E}} \right\rVert_F \leq \delta \left\lVert \boldsymbol{\mathcal{X}} \right\rVert_F$
	RHOSVD	$\max\Big(\left\lVert \boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{V}} \times_{i=1}^{N} \mathbf{U}_i - \boldsymbol{\mathcal{E}} \right\rVert_F , \Big\{ \left\lVert \mathbf{V}_{[i]} - \mathbf{J}_i \right\rVert_F \Big\} \Big) \leq \delta \left\lVert \boldsymbol{\mathcal{X}} \right\rVert_F$
	RCPD (sub)	$\max\Big(\left\lVert \boldsymbol{\mathcal{X}} - \mathbf{U}_1 \circ \cdots \circ \mathbf{U}_N - \boldsymbol{\mathcal{E}} \right\rVert_F , \Big\{ \left\lVert \mathbf{U}_i - \mathbf{J}_i \right\rVert_F \Big\} \Big) \leq \delta \left\lVert \boldsymbol{\mathcal{X}} \right\rVert_F$
	RCPD (lin)	$\left\lVert \boldsymbol{\mathcal{X}} - \mathbf{U}_1 \circ \cdots \circ \mathbf{U}_N - \boldsymbol{\mathcal{E}} \right\rVert_F \leq \delta \left\lVert \boldsymbol{\mathcal{X}} \right\rVert_F$

Table 6.1: Convergence criterion of each algorithm. During execution, an algorithm would continue to iterate until its convergence criterion is satisfied. In our implementation, we set $\delta = 10^{-7}$ for all algorithms.

Parameter	**Description**	**APG**	**ALM**
μ_0	Initial value of penalty parameter μ	10^3	10^{-3}
μ_{min}	Minimum value of μ (only in APG)	10^{-9}	–
μ_{max}	Maximum value of μ (only in ALM)	–	10^9
ρ	Multiplicative update factor of μ	0.9	1.2
δ	Convergence threshold	10^{-7}	10^{-7}

Table 6.2: Summary of the values used for the parameters of APG and ALM algorithms. Note that APG is only used with matrix RPCA whereas ALM is used with all methods. The values reported here are the same for all methods, with the exception of ROSL, for which $\mu_0 = 10^{-1}$.

6.2 Tuning Algorithmic Parameters

The algorithms described in chapters 3 and 4 were based on two optimisation methods, namely APG and ALM. APG was used only with matrix RPCA whereas ALM was used with all methods. APG and ALM require four parameters each that need to be tuned, three of which have to do with the penalty parameter μ and the fourth one is the convergence threshold δ discussed in the previous section. In the following, we explain how these parameters should be tuned and report the values that we used in our implementation.

(i) **Initial value of** μ. In APG, the penalty parameter μ is decreasing in each iteration, whereas in ALM it is increasing in each iteration. It is common to initialise μ with a large value for APG and with a small value for ALM. In our implementation, we use $\mu_0 = 10^3$ for APG and $\mu_0 = 10^{-3}$ for ALM, with the exception of the ALM algorithm for ROSL, in which we use $\mu_0 = 10^{-1}$.

(ii) **Final value of** μ. The purpose of this parameter is to set a limit in the decrease (in APG) or increase (in ALM) of μ, so as to avoid ill-conditioning. In our implementation, we use $\mu_{min} = 10^{-9}$ in APG and $\mu_{max} = 10^9$ in ALM.

(iii) **Multiplicative update factor of** μ. The penalty parameter μ is multiplied at each iteration by some factor ρ, for which we have $\rho < 1$ in APG and (typically) $1 < \rho < 2$ in ALM. We follow [51, 84] and we use $\rho = 0.9$ in APG. As for ALM, it is reported in [54] that the value of ρ is a trade-off between precision and speed, i.e. for ρ close to 1 the algorithm converges slowly but precisely whereas for ρ close to 2 it converges faster but less precisely. In our ALM implementation we use $\rho = 1.2$, which we have empirically determined to be a good choice for most cases.

(iv) **Convergence threshold** δ. This has been discussed in the previous section. A smaller value of δ leads to a more accurate result but takes more iterations to converge. In general, the selection of δ should be application-specific. In our implementation we use $\delta = 10^{-7}$.

A summary of the parameter values used is given in Table 6.2.

6.3 Software Toolbox Implementation

All the algorithms discussed in this thesis have been implemented in MATLAB [61] as a general-purpose toolbox for robust low-rank modelling. MATLAB is suitable for the implementation of this type of algorithms since

(i) it provides native support for matrix operations (of which the algorithms make heavy use),

(ii) its built-in matrix subroutines (such as the computation of the SVD) have efficient implementations, allowing for relatively fast computation, and

(iii) it is the most common choice in the literature of robust low-rank modelling, allowing for direct comparisons and ease-of-use by the community.

In the tensor algorithms, we have made extensive use of the open-source MATLAB tensor toolbox, which was (mainly) developed by Bader and Kolda [3, 4]. The tensor toolbox provides support for tensor operations, which are not native in MATLAB, such as tensor matricisation, the n-mode product and tensor decompositions, including the CP decomposition and the HOSVD.

Contrary to the rest of the algorithms having been implemented in MATLAB, the generalised shrinkage operator $\mathcal{S}_a^q\{\cdot\}$ was implemented in C and linked to the rest of the MATLAB code using the MEX compiler provided by MATLAB. This was done to maximise efficiency. The computation of $\mathcal{S}_a^q\{\cdot\}$, as described by Theorems 5.2 and 5.3, requires the application of the iterative Newton-Raphson method for *each* tensor entry. The implementation in MATLAB of such a nested iterative computation would be severely inefficient, so C was used instead.

To the best of our knowledge, our implementation is the most comprehensive in the literature of robust low-rank modelling. It covers the most important state-of-the-art methods for matrices and introduces the herein proposed novel methods for tensors. As part of this thesis, we plan to release our implementation as an open-source MATLAB toolbox, freely available for use by the community. It is our hope that this toolbox will facilitate and accelerate future research in the field of robust low-rank modelling.

Chapter 7

Experimental Evaluation

In this chapter, we experimentally evaluate the algorithms described in chapters 3 and 4 and investigate their practical applicability. We perform four sets of experiments; on synthetic data, on face image denoising, on background subtraction in videos and on reconstruction of missing images. We evaluate the algorithms based on their ability to correctly recover the low-rank and sparse components, their total execution speed, their convergence behaviour and their performance in real-life image analysis applications.

All the experiments were performed on a workstation equipped with an 8-core Intel i7 processor running at 3.40 GHz and having 16 GB of memory. The operating system was Linux kernel version 3.8.0 and the version of MATLAB used was 8.1.0 (release R2013a).

Note, finally, that in all experiments reported in this chapter, the norm parameters were set to $p = q = 1$, that is, the convex nuclear norm and elementwise ℓ_1-norm were considered.

7.1 Low-Rank Recovery with Synthetic Data

In this experiment, we will evaluate the ability of each algorithm to correctly recover a synthetically generated low-rank matrix/tensor from sparse corruption. This experiment is quite common in the literature as a "sanity check" for matrix methods and it has been performed with slight variations in [14, 18, 38, 51, 52, 54, 75, 93]. To the best of our knowledge, this thesis provides the most comprehensive, in terms of number of algorithms tested under the same conditions, version of this experiment.

7.1.1 Matrix Methods

For the evaluation of the matrix algorithms presented in chapter 3, we construct a low-rank matrix $\mathbf{A} \in \mathbb{R}^{1000\times 1000}$ and a sparse matrix $\mathbf{E} \in \mathbb{R}^{1000\times 1000}$ in the following way

- $\mathbf{A} = \mathbf{U}\mathbf{V}$, where $\mathbf{U} \in \mathbb{R}^{1000\times 50}$ and $\mathbf{V} \in \mathbb{R}^{50\times 1000}$. The entries of $\mathbf{U}, \mathbf{V}$ are independently sampled from a normal distribution with mean 0 and variance 1 and the entries of $\mathbf{A}$ are normalised to have variance 1. As a result, we will have $\operatorname{rank}(\mathbf{A}) \leq 50$, with the equality holding almost surely.
- With probability 0.1, each entry of $\mathbf{E}$ is sampled from a uniform distribution on the interval $[-100, 100]$, otherwise it is set to 0. As a result, around 10% of the entries of $\mathbf{E}$ will be non-zero.

Each algorithm is given as input the matrix $\mathbf{X} = \mathbf{A} + \mathbf{E}$ and produces as output a pair $\left(\hat{\mathbf{A}}, \hat{\mathbf{E}}\right)$, for each particular choice of λ. We run each algorithm for the following 7 values of λ

$$\lambda \in \{0.0032,\ 0.0068,\ 0.0147,\ 0.0316,\ 0.0681,\ 0.1468,\ 0.3162\} \tag{7.1}$$

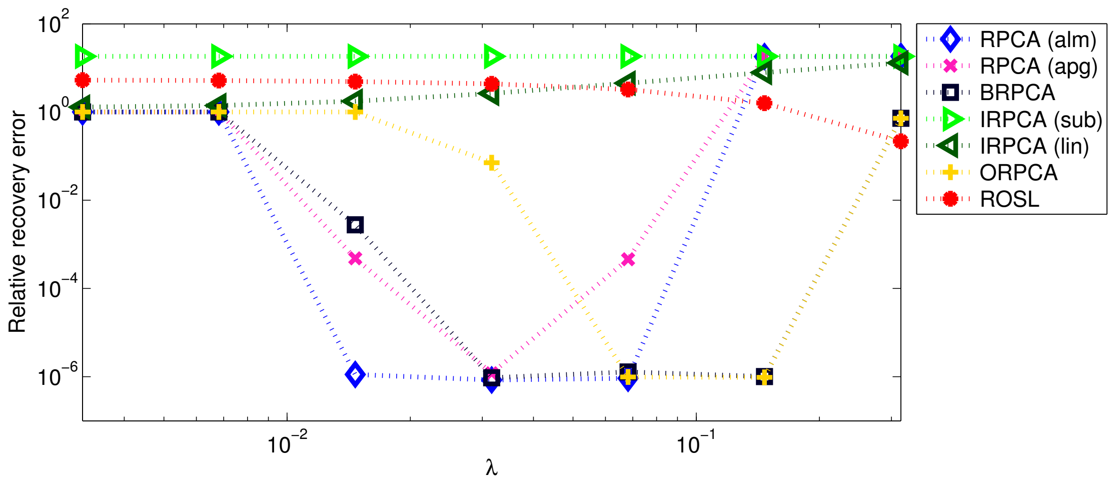


Figure 7.1: Relative recovery error against λ for all matrix algorithms on synthetic data. Both axes are on logarithmic scale. All algorithms except both versions of IRPCA and ROSL achieve exact recovery for a range of λ values.

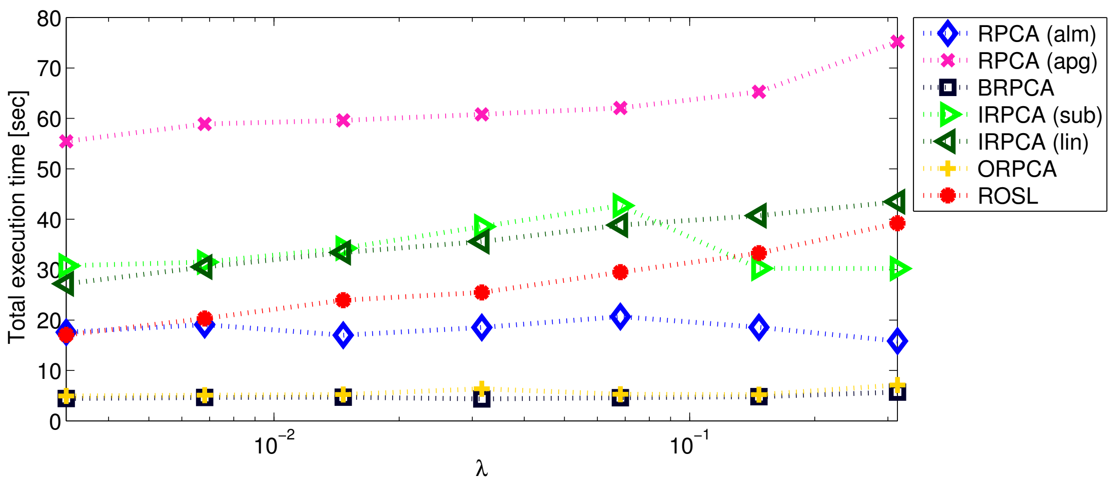


Figure 7.2: Total execution time in seconds against λ for all matrix algorithms on synthetic data. The horizontal axis is on logarithmic scale. RPCA via APG appears to be the slowest whereas the factorisation-based BRPCA and ORPCA are the fastest.

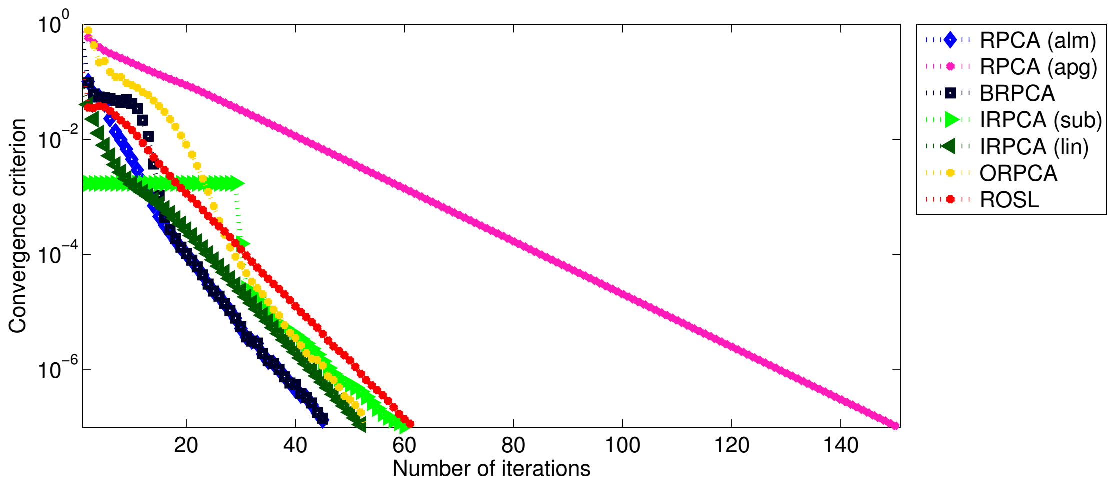


Figure 7.3: Convergence behaviour of each algorithm for its best performance in terms of relative recovery error. The convergence criterion (in logarithmic scale) is plotted against the number of iterations. All algorithms converge similarly fast, except RPCA via APG which converges more slowly.

Algorithm	**Error**	**Time [sec]**	**Iterations**	λ
RPCA (alm)	8.439×10^{-7}	18.536	46	0.0316
RPCA (apg)	1.198×10^{-6}	60.811	151	0.0316
BRPCA	9.484×10^{-7}	4.357	46	0.0316
IRPCA (sub)	18.271	30.789	61	0.0032
IRPCA (lin)	1.289	27.198	53	0.0032
ORPCA	9.694×10^{-7}	5.210	53	0.1468
ROSL	0.218	39.203	62	0.3162

Table 7.1: Summary of the best performance of each algorithm on synthetic data with respect to its relative recovery error. The λ value corresponds to the value for which this performance was achieved.

Finally, for the factorisation-based algorithms where an upper bound for the rank r is required, we set $r = 50$, which is (almost surely) the true rank of $\mathbf{A}$.

We evaluate each algorithm based on its accuracy in recovering $\mathbf{A}$, its total execution time and its convergence behaviour. In order to quantify the accuracy of recovering $\mathbf{A}$, we use the following *relative recovery error*

$$E = \frac{\left\|\mathbf{A} - \hat{\mathbf{A}}\right\|_F}{\|\mathbf{A}\|_F} \tag{7.2}$$

Fig. 7.1 plots the relative recovery error E against λ. We can see that there is a range of λ values for which most of the algorithms manage to recover $\mathbf{A}$ almost *exactly*, with the relative error being around 10^{-6}. The algorithms that fail to recover $\mathbf{A}$ are both versions of IRPCA and ROSL. It is interesting to note that the optimal value of λ for RPCA, which for an $m \times n$ matrix is proven in [18] to be $\lambda = 1/\sqrt{\min(m,n)}$ (here it is 0.0316), is within the range of exact recovery.

Fig. 7.2 plots the total execution time in seconds against λ. We can see that the only method based on APG (i.e. RPCA via APG) is the slowest. Particularly fast are the algorithms that are based on factorisation (i.e. BRPCA and ORPCA), since they operate on much smaller matrices. ROSL is also based on factorisation however it appears to be even slower than RPCA via ALM.

For each method, we select its best execution in terms of relative recovery error E and plot the corresponding convergence behaviour in Fig. 7.3. It becomes evident that the APG method (RPCA via APG) has slower convergence than the ALM methods, which explains its slow speed and justifies the preference of the state-of-the-art algorithms (including those proposed herein) towards ALM. The ALM algorithms on the other hand appear to have similarly fast convergence.

Finally, Table 7.1 summarises the best performance in terms of relative recovery error E for each algorithm. RPCA via ALM appears to achieve the most accurate recovery, and it is around 3 times faster than RPCA via APG. Among those algorithms that achieve exact recovery, the fastest are BRPCA and ORPCA. The herein proposed linearisation version of IRPCA is an improvement to the substitution-based IRPCA, however both of them, including ROSL, exhibit poor recovery performance.

7.1.2 Tensor Methods

For the evaluation of the tensor algorithms presented in chapter 4, we synthetically generate a low-rank tensor $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{100\times100\times100}$ and a sparse tensor $\boldsymbol{\mathcal{E}} \in \mathbb{R}^{100\times100\times100}$ as follows.

- $\boldsymbol{\mathcal{A}} = \mathbf{U}_1 \circ \mathbf{U}_2 \circ \mathbf{U}_3$, where $\mathbf{U}_1, \mathbf{U}_2, \mathbf{U}_3 \in \mathbb{R}^{100\times5}$. The entries of the factor matrices $\mathbf{U}_1, \mathbf{U}_2, \mathbf{U}_3$ were independently sampled from a normal distribution with mean 0 and variance 1 and the entries of $\boldsymbol{\mathcal{A}}$ were normalised to variance 1. This process ensures that $\operatorname{rank}(\boldsymbol{\mathcal{A}}) \leq 5$, with the equality holding almost surely. Note that due to Theorem 2.6, we also have that $\operatorname{rank}_i(\boldsymbol{\mathcal{A}}) \leq 5$ for $i \in \{1,2,3\}$.

- Similarly to the matrix case, each entry of $\boldsymbol{\mathcal{E}}$ was chosen with probability 0.1 from the uniform distribution on $[-100, 100]$ and otherwise was set to 0. As a result, around 10% of the entries of $\boldsymbol{\mathcal{E}}$ are non-zero.

Each algorithm was given $\boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{A}} + \boldsymbol{\mathcal{E}}$ as input, was executed for the same set of λ values (7.1) as in the matrix case and produced for each run a pair $\left(\hat{\boldsymbol{\mathcal{A}}}, \hat{\boldsymbol{\mathcal{E}}}\right)$. The rank parameters r_1, r_2, r_3 of BRPCA, IRPCA, RHOSVD and parameter r of RCPD were all set to 5. Finally, for all algorithms we set $a_1 = a_2 = a_3 = 1/3$.

Fig. 7.4 plots the *relative recovery error* defined by

$$E = \frac{\left\|\boldsymbol{\mathcal{A}} - \hat{\boldsymbol{\mathcal{A}}}\right\|_F}{\|\boldsymbol{\mathcal{A}}\|_F} \tag{7.3}$$

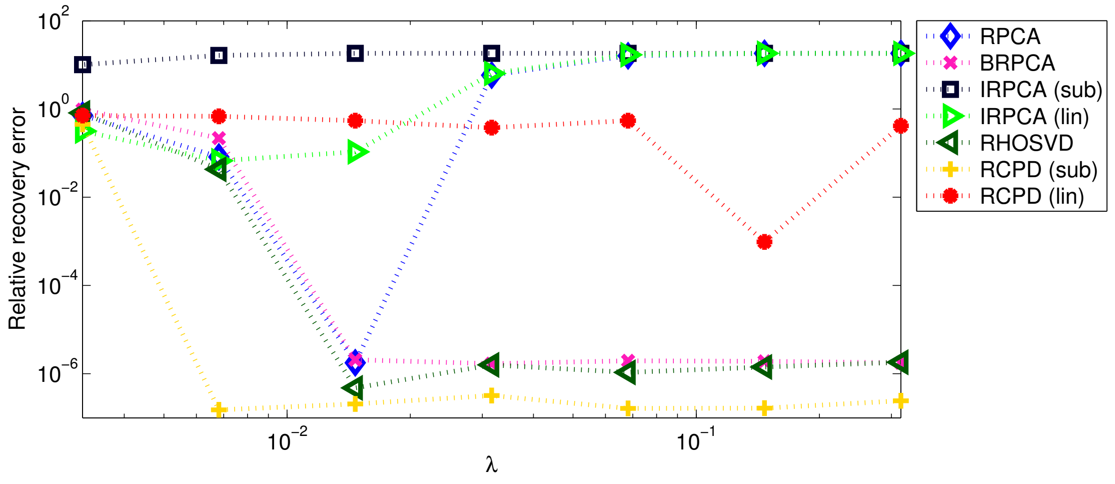


Figure 7.4: Relative recovery error against λ for all tensor algorithms on synthetic data. Both axes are on logarithmic scale. All algorithms except both versions of IRPCA and linearisation-based RCPD achieve exact recovery for a range of λ values.

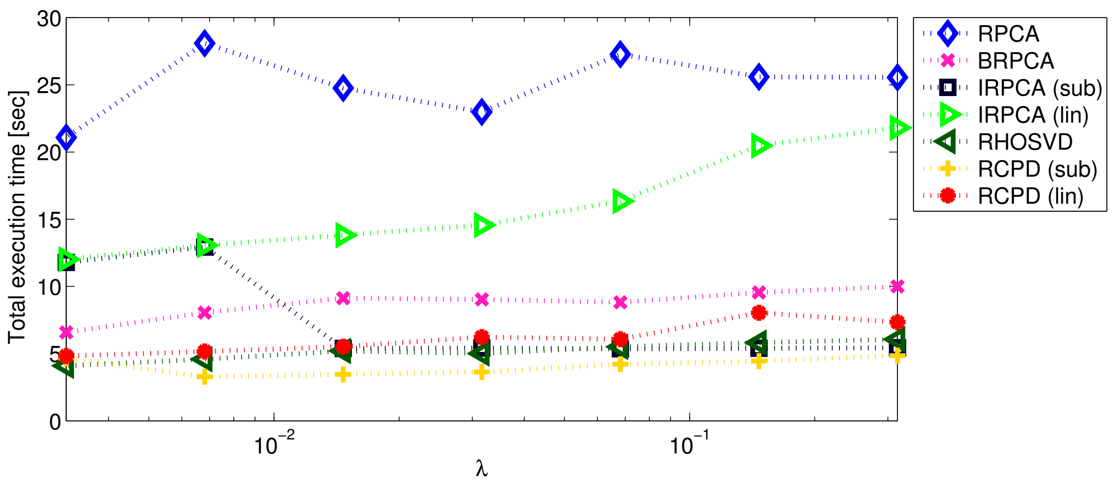


Figure 7.5: Total execution time against λ for all tensor algorithms on synthetic data. The horizontal axis is on logarithmic scale. RPCA appears to be the slowest whereas the factorisation-based BRPCA, RHOSVD and both versions of RCPD are particularly efficient.

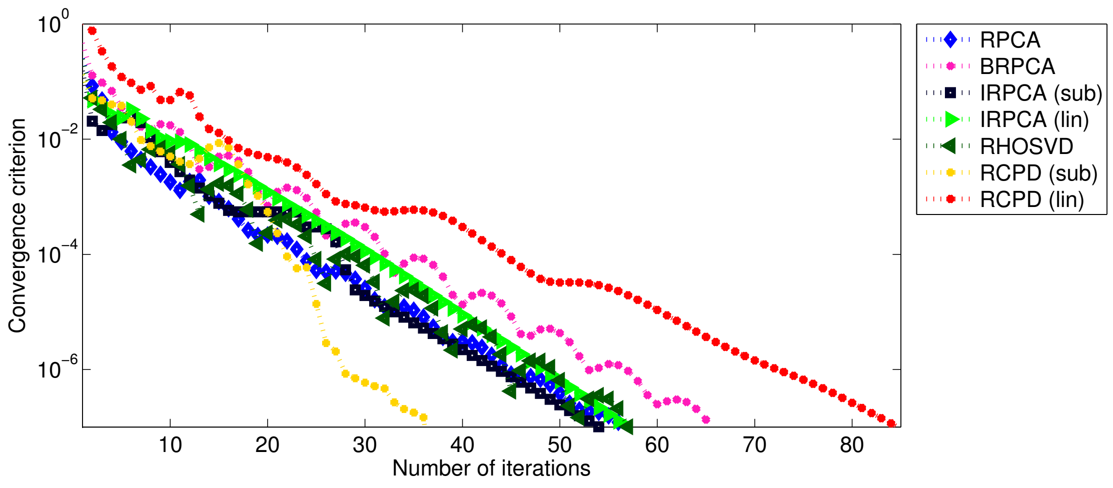


Figure 7.6: Convergence behaviour of each tensor algorithm for its execution that yielded the lowest relative recovery error. The convergence criterion (on logarithmic scale) is plotted against number of iterations. All algorithms exhibit similar convergence properties.

Algorithm	**Error**	**Time [sec]**	**Iterations**	λ
RPCA	1.770×10^{-6}	24.761	57	0.0147
BRPCA	1.682×10^{-6}	9.034	66	0.0316
IRPCA (sub)	10.141	11.791	55	0.0032
IRPCA (lin)	0.067	13.065	57	0.0068
RHOSVD	4.775×10^{-7}	5.205	58	0.0147
RCPD (sub)	1.510×10^{-7}	3.289	37	0.0068
RCPD (lin)	9.679×10^{-4}	8.043	85	0.1468

Table 7.2: Summary of the best executions of all tensor algorithms in terms of relative recovery error. For each algorithm the value of λ for which the best performance was achieved is shown. RPCA, BRPCA, RHOSVD and substitution-based RCPD achieve exact recovery, with the latter being the fastest followed shortly by RHOSVD.

against λ. We can see that there exists a range of λ values for which most algorithms achieve almost *exact* recovery of $\boldsymbol{\mathcal{A}}$. The best performance is achieved by RCPD via linearisation. The algorithms that fail to recover $\boldsymbol{\mathcal{A}}$ are both versions of IRPCA and the linearisation-based version of RCPD.

Fig. 7.5 plots the total execution time of each algorithm against λ. The factorisation-based BRPCA, RHOSVD and both versions of RCPD appear to be particularly efficient, in contrast to RPCA which is the slowest.

The convergence of the best executions in terms of relative recovery error are shown in Fig. 7.6. All algorithms appear to converge similarly fast, which means that the differences in execution speed are mostly due to the cost per iteration rather than the total number of iterations. The fastest and slowest algorithms to converge appear to be the substitution-based and linearisation-based versions of RCPD respectively.

Finally, Table 7.2 summarised the best achieved performances of each algorithm in terms of relative recovery error. We can see that both the lowest error and the highest speed are achieved by RCPD via substitution, with RHOSVD following shortly. It is important to note that the herein proposed BRPCA, RHOSVD and substitution-based RCPD appear to be better than RPCA both in terms of recovery performance and speed, with substitution-based RCPD being around 7 times faster than RPCA. However, both versions of IRPCA and the linearisation-based version of RCPD failed to recover the low-rank tensor. This behaviour of IRPCA is similar to the matrix case where IRPCA did not manage to recover the low-rank component either.

7.2 Denoising of Face Images

It is known [6] that an image of a convex Lambertian object lies approximately on a 9-dimensional subspace. Since faces can be approximated as convex and Lambertian, face images typically are of low rank and therefore constitute a suitable candidate for robust low-rank modelling. In this experiment, we will evaluate the ability of each algorithm to recover face images that have been corrupted by heavy non-Gaussian—but sparse—noise. Similar experiments have also been reported in [5, 69].

We use part of the Extended Yale Face Database B [31, 48], which is a collection of greyscale face images taken from 28 human subjects, each under 9 poses and 64 illuminations. All images are manually cropped and aligned, and each pixel ranges between 0 (black) and 1 (white). For the purposes of this experiment, we use the face images of a single subject in the frontal pose, i.e. a set of 64 images spanning various illuminations, and we resize each image to 48×42 pixels. Fig 7.7 shows the first 8 images of the total of 64 images used.

We corrupt each one of the 64 images with heavy, non-Gaussian, sparse noise. We consider two kinds of corruption, described as follows.

- **Salt & pepper noise**. Each pixel is considered for corruption with a probability p and set equiprobably to 0 or 1. That is, with probability $p/2$ is set to 0 (black), with probability $p/2$ is set to 1 (white) and with probability $1 - p$ remains unchanged. We consider 3 levels of salt & pepper noise, for $p \in \{10\%,\ 30\%,\ 60\%\}$.
- **Random patch**. For each image, a rectangular region of size $h \times w$ pixels is randomly chosen, where h and w are sampled uniformly from $\{1, 2, \dots, 40\}$ (thus producing a different patch size for each image). Then, all pixels within the selected region are set equiprobably to 0 (black) or 1 (white).

In total, we have 4 corruption scenarios, i.e. salt & pepper noise with (approximately) 10%, 30% and 60% of the pixels corrupted and random patch corruption with a maximum patch size of 40×40.

For the matrix methods, an input data matrix $\mathbf{X}$ of size 2016×64 is constructed, where each column contains a vectorised corrupted image. For the tensor methods, an input tensor $\boldsymbol{\mathcal{X}}$ of

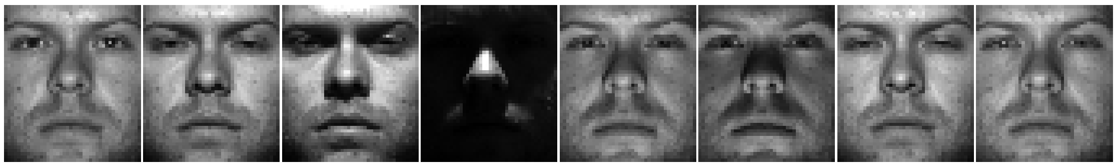


Figure 7.7: The first 8 images of the set of 64 face images used in the denoising experiment. The images correspond to the first subject, in frontal pose, of the Extended Yale Face Database B [31, 48], under 64 different illuminations.

size $48 \times 42 \times 64$ is constructed, where each frontal slice (i.e. the slice corresponding to the first two modes) contains a corrupted image. Notice that $\boldsymbol{\mathcal{X}}$ considers the image rows, the image columns and the illuminations as its three modes; due to the symmetry of the frontal pose and the alignment across illuminations it is expected that all 3 modes are low-rank. If $\mathbf{A}$ represents the corresponding uncorrupted matrix (or $\boldsymbol{\mathcal{A}}$ for tensor) and $\hat{\mathbf{A}}$ represents the recovered low-rank matrix (or $\hat{\boldsymbol{\mathcal{A}}}$ for tensor), then the *relative recovery error* is computed (for matrices) as

$$E = \frac{\left\| \mathbf{A} - \hat{\mathbf{A}} \right\|_F}{\left\| \mathbf{A} \right\|_F} \tag{7.4}$$

For tensors, E is computed the same way, by using $\boldsymbol{\mathcal{A}}$ and $\hat{\boldsymbol{\mathcal{A}}}$.

Each method is executed for a range of input parameters and its best performance in terms of the relative reconstruction error is reported. The following values for λ are used

$$\lambda \in \{0.0001,\ 0.0003,\ 0.0010,\ 0.0032,\ 0.0100,\ 0.0316,\ 0.1000\} \tag{7.5}$$

For matrix BRPCA, ORPCA, ROSL and both versions of RCPD, where an upper bound r of the rank is required, we use the following values

$$r \in \{10,\ 20,\ 30,\ 40,\ 50\} \tag{7.6}$$

From tensor methods, BRPCA, the two versions of IRPCA and RHOSVD require parameters r_1, r_2 and r_3 as input. Here we use the following setting

$$r_1 = 48\,\alpha \tag{7.7}$$
$$r_2 = 42\,\alpha \tag{7.8}$$
$$r_3 = 64\,\alpha \tag{7.9}$$
$$\alpha \in \{0.2,\ 0.4,\ 0.6,\ 0.8,\ 1.0\} \tag{7.10}$$

That is, r_1, r_2 and r_3 are set to a varying proportion of the corresponding tensor size. Finally, for all tensor algorithms we set $a_1 = a_2 = a_3 = 1/3$.

Fig. 7.8 and Fig. 7.9 show the reconstruction of the first image in the data achieved by the matrix and tensor methods respectively, when various levels of salt & pepper noise are considered. Most methods cope well with 10% and 30% corruption, however reconstruction in general becomes poor in the extreme case of 60% corruption. The substitution-based version of IRPCA appears to have the worst performance. In general, tensor methods produce a smoother result than the matrix methods. It is particularly impressive that the substitution-based version of RCPD achieves a high-quality reconstruction even for the extreme case of 60% corruption.

The numerical results for salt & pepper noise are given in Tables 7.3, 7.4 and 7.5 for 10%, 30% and 60% corruption respectively. We can see that in terms of relative recovery error, the tensor methods outperform the matrix methods in general. In both matrices and tensors, the factorisation-based methods appear to outperform the rest. In particular, ORPCA and RHOSVD

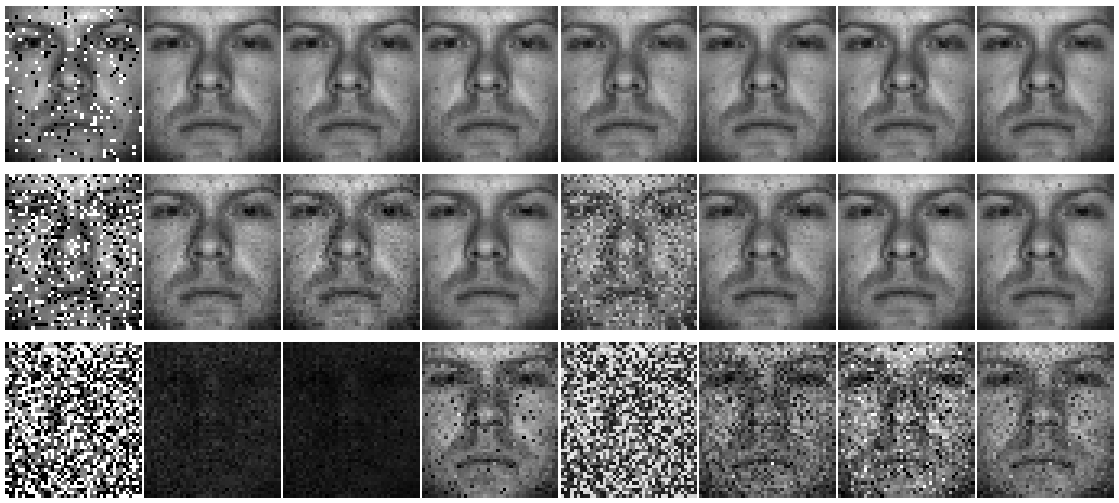


Figure 7.8: Reconstruction of the first image in the data using matrix methods. Corruption with salt & pepper noise with 10% (first row), 30% (second row) and 60% (third row). First column shows the corrupted input image and each following column shows its reconstruction by the corresponding matrix method.

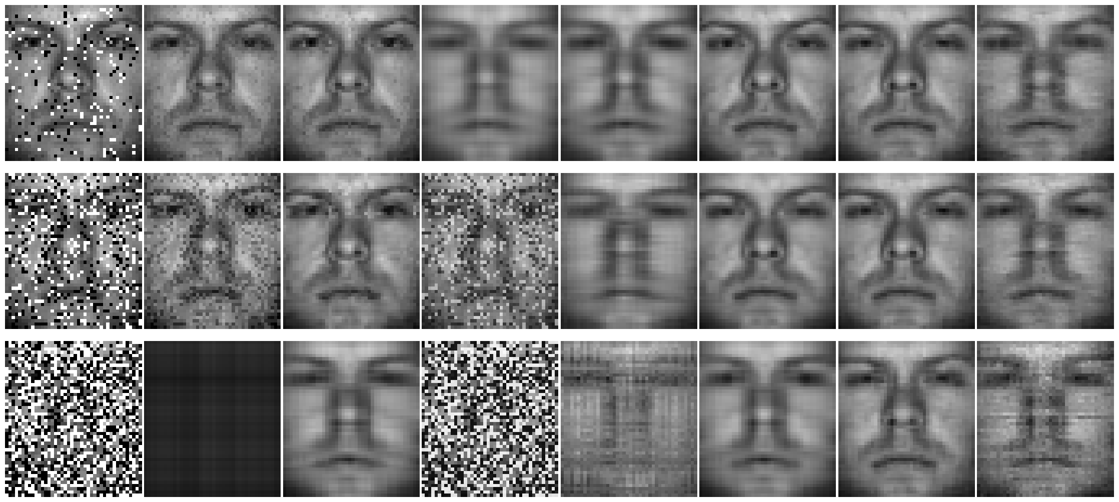


Figure 7.9: Reconstruction of the first image in the data using tensor methods. Corruption with salt & pepper noise with 10% (first row), 30% (second row) and 60% (third row). First column shows the corrupted input image and each following column shows its reconstruction by the corresponding tensor method.

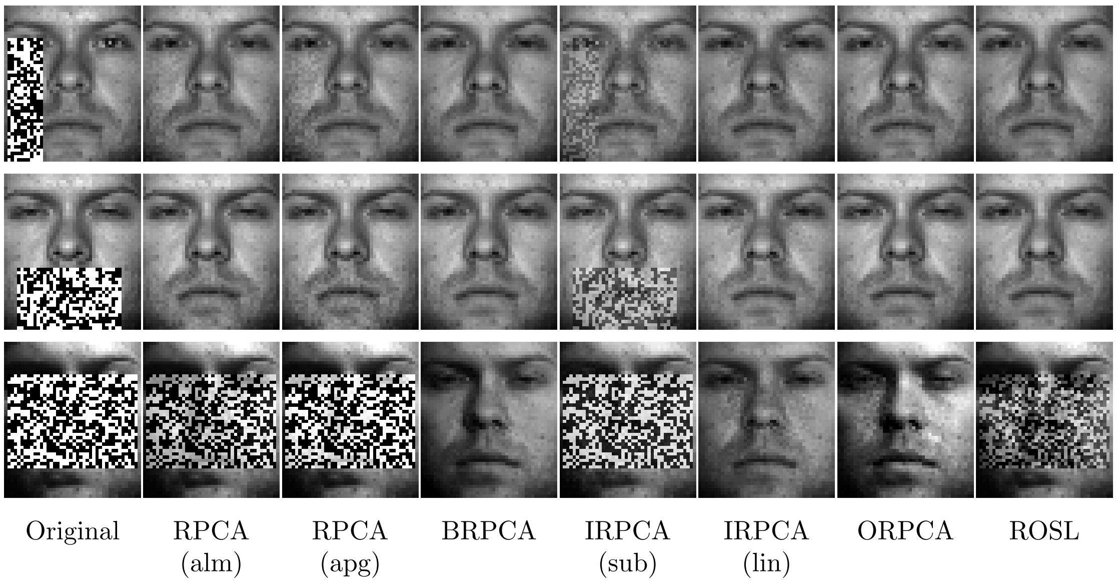


Figure 7.10: Reconstruction using matrix methods with random patch corruption of maximum size 40 × 40. Images number 1, 7 and 30 of the data are shown (first, second and third row respectively). First column shows the corrupted input image and each following column shows its reconstruction by the corresponding matrix method.

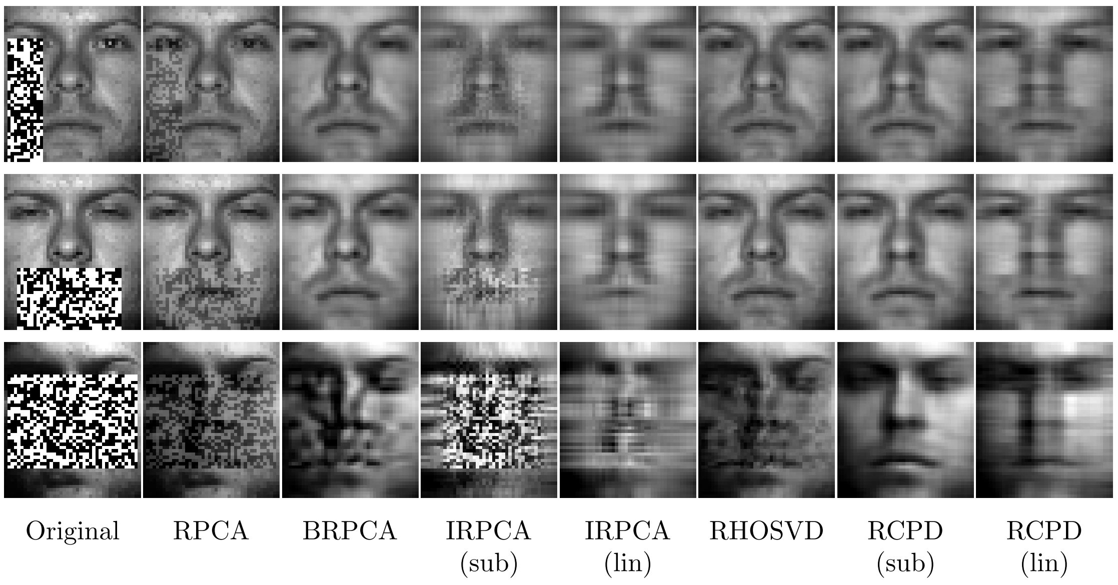


Figure 7.11: Reconstruction using tensor methods with random patch corruption of maximum size 40 × 40. Images number 1, 7 and 30 of the data are shown (first, second and third row respectively). First column shows the corrupted input image and each following column shows its reconstruction by the corresponding tensor method.

	Algorithm	**Error**	**Time [sec]**	**Iterations**	λ	r
Matrices	RPCA (alm)	0.131	0.414	59	0.0316	–
	RPCA (apg)	0.131	1.329	192	0.0316	–
	BRPCA	0.106	0.577	99	0.1000	50
	IRPCA (sub)	0.157	241.143	68	0.0003	–
	IRPCA (lin)	0.174	212.495	72	0.0032	–
	ORPCA	**0.105**	**0.698**	**103**	**0.1000**	**40**
	ROSL	0.137	0.951	75	0.1000	10
Tensors	RPCA	0.111	3.651	76	0.0316	–
	BRPCA	0.088	2.518	104	0.1000	0.8×
	IRPCA (sub)	0.189	2.879	92	0.0003	0.2×
	IRPCA (lin)	0.169	2.910	84	0.1000	0.2×
	RHOSVD	**0.084**	**4.431**	**101**	**0.1000**	**0.6×**
	RCPD (sub)	0.121	5.552	95	0.1000	50
	RCPD (lin)	0.220	3.570	86	0.1000	30

Table 7.3: Numerical results for 10% salt & pepper noise for both matrices and tensors. Best method of each category in terms of relative recovery error is shown in bold. For each algorithm, the values used for λ and r (if applicable) are reported.

	Algorithm	**Error**	**Time [sec]**	**Iterations**	λ	r
Matrices	RPCA (alm)	0.176	0.409	57	0.0316	–
	RPCA (apg)	0.200	1.411	190	0.0316	–
	BRPCA	0.151	0.609	99	0.1000	10
	IRPCA (sub)	0.508	194.946	53	0.0001	–
	IRPCA (lin)	0.220	225.636	73	0.0032	–
	ORPCA	**0.142**	**0.604**	**98**	**0.1000**	**20**
	ROSL	0.179	2.120	67	0.0316	30
Tensors	RPCA	0.237	3.659	74	0.0316	–
	BRPCA	0.123	2.491	98	0.1000	0.6×
	IRPCA (sub)	0.564	1.779	53	0.0001	1.0×
	IRPCA (lin)	0.213	2.994	84	0.1000	0.2×
	RHOSVD	**0.122**	**3.282**	**93**	**0.1000**	**0.4×**
	RCPD (sub)	0.128	5.363	94	0.1000	50
	RCPD (lin)	0.237	3.240	86	0.1000	30

Table 7.4: Numerical results for 30% salt & pepper noise for both matrices and tensors. Best method of each category in terms of relative recovery error is shown in bold. For each algorithm, the values used for λ and r (if applicable) are reported.

	Algorithm	Error	Time [sec]	Iterations	λ	r
Matrices	RPCA (alm)	0.829	0.492	71	0.0100	–
	RPCA (apg)	0.864	1.757	181	0.0100	–
	BRPCA	0.429	0.469	85	0.0316	50
	IRPCA (sub)	0.970	184.868	50	0.0001	–
	IRPCA (lin)	0.520	231.687	72	0.0032	–
	ORPCA	0.417	0.401	97	0.1000	10
	ROSL	**0.375**	**0.595**	**66**	**0.0316**	**10**
Tensors	RPCA	0.821	3.339	68	0.0100	–
	BRPCA	0.226	1.838	87	0.1000	$0.2\times$
	IRPCA (sub)	1.047	1.734	51	0.0001	$0.2\times$
	IRPCA (lin)	0.342	4.218	82	0.1000	$0.2\times$
	RHOSVD	0.193	2.475	87	0.1000	$0.2\times$
	RCPD (sub)	**0.173**	**3.440**	**83**	**0.0316**	**40**
	RCPD (lin)	0.292	3.292	84	0.1000	30

Table 7.5: Numerical results for 60% salt & pepper noise for both matrices and tensors. Best method of each category in terms of relative recovery error is shown in bold. For each algorithm, the values used for λ and r (if applicable) are reported.

	Algorithm	Error	Time [sec]	Iterations	λ	r
Matrices	RPCA (alm)	0.281	0.408	59	0.0316	–
	RPCA (apg)	0.328	2.065	191	0.0316	–
	BRPCA	0.215	0.483	91	0.0316	30
	IRPCA (sub)	0.463	351.084	94	0.0001	–
	IRPCA (lin)	0.300	200.991	64	0.0010	–
	ORPCA	**0.148**	**0.604**	**98**	**0.1000**	**20**
	ROSL	0.240	0.581	67	0.0316	10
Tensors	RPCA	0.214	3.660	74	0.0316	–
	BRPCA	**0.166**	**2.139**	**94**	**0.1000**	**$0.4\times$**
	IRPCA (sub)	0.441	3.293	91	0.0001	$0.2\times$
	IRPCA (lin)	0.304	2.721	72	0.0100	$0.2\times$
	RHOSVD	0.183	3.960	93	0.0316	$0.6\times$
	RCPD (sub)	0.188	2.958	89	0.0100	40
	RCPD (lin)	0.243	1.980	79	0.0316	20

Table 7.6: Numerical results for random patch corruption of maximum size 40×40 for both matrices and tensors. Best method of each category in terms of relative recovery error is shown in bold. For each algorithm, the values used for λ and r (if applicable) are reported.

are the best matrix and tensor method respectively for 10% and 20%, whereas ROSL and RCPD via substitution are the best matrix and tensor method respectively for 60%. IRPCA has the poorest performance in both matrices and tensors.

Fig. 7.10 and Fig. 7.11 show the reconstruction of 3 sample images in the case of corruption with a random patch. Notice that there are cases (such as the third image) where, although the corruption is particularly heavy, certain methods still achieve a high-quality reconstruction. Table 7.6 shows the numerical results for the random patch case. Both matrix and tensor methods achieve respectable reconstruction. The best methods are ORPCA for matrices (and overall) and BRPCA for tensors.

7.3 Background Subtraction

Given a set of video frames, background subtraction (or foreground segmentation) refers to segmenting (in each frame) the static background from any moving foreground objects. By modelling the background as low-rank and the foreground objects as sparse, robust low-rank modelling is well-suited for this task. Being one of the most common benchmarks for robust low-rank modelling, background subtraction has been used in [5, 14, 18, 38, 49, 75, 84].

In this experiment, we use the *highway* video taken from [34], which shows a highway where passing cars are the foreground objects. The video comes with manually created binary foreground masks, which indicate for each pixel whether it belongs to the background or the foreground. Six sample video frames and their corresponding foreground masks are shown in Fig. 7.12. For our purposes, we resize the video to a resolution of 48×64 pixels and use 400 consecutive frames. For the matrix methods, the input data matrix is $\mathbf{X} \in \mathbb{R}^{3072\times400}$, where each column is a vectorised video frame. For the tensor methods, the input data tensor is $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{48\times64\times400}$, where each frontal slice (i.e. the slice corresponding to the two first modes) is a video frame.

All algorithms return (ideally) a low-rank component containing the background and a sparse component containing the foreground. In order to calculate the *recovered foreground mask*, the values of the sparse component are thresholded. That is, an entry E in the sparse component is classified as background if $|E| \leq T$ and as foreground if $|E| > T$, with $T \geq 0$ being the chosen threshold. We evaluate each algorithm's performance by comparing the recovered foreground mask to the ground truth foreground mask. Our evaluation is based on *ROC analysis* [27, 60], a common framework for assessing the performance of binary classifiers. To better understand this type of analysis, the following terminology is used.

- **True positive**. A foreground pixel correctly classified as such. The number of true positives is represented by TP.
- **False positive**. A background pixel wrongly classified as foreground. The number of false positives is represented by FP.
- **True negative**. A background pixel correctly classified as such. The number of true negatives is represented by TN.
- **False negative**. A foreground pixel wrongly classified as background. The number of false negatives is represented by FN.
- **True positive rate**. The probability of a foreground pixel being classified correctly, calculated as $TPR = \frac{TP}{TP+FN}$.
- **False positive rate**. The probability of a background pixel being classified wrongly, calculated as $FPR = \frac{FP}{TN+FP}$.
- **False negative rate**. The probability of a background pixel being classified correctly, calculated as $FNR = 1 - FPR$.

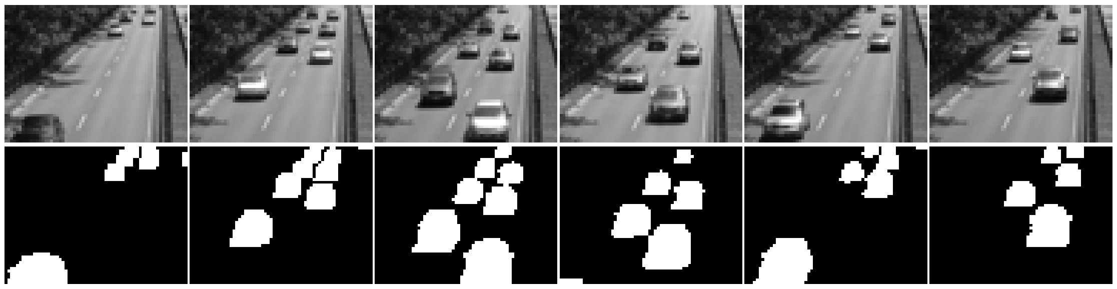


Figure 7.12: Six sample frames, together with their ground truth foreground masks, from the highway video used in the background subtraction experiment. The video is taken from [34].

Choosing the value of the threshold T is a trade-off between the number or true positives and the number of true negatives. That is, increasing T also increases FPR but decreases FNR, or in simpler words, it produces more background in expense to less foreground. Plotting TPR versus FPR for various values of T yields a curve known as *Receiver Operating Characteristic* [27]. The area under ROC, known as *Area Under the Curve*, will be used herein as an evaluation metric. This metric ranges from 0.5 (classification is no better than random choice) to 1.0 (classification is perfect).

Each method is executed for a range of input parameters and its best performance in terms of AUC is reported. The range of parameters is the same as in the denoising experiment (section 7.2). That is, λ ranges as follows

$$\lambda \in \{0.0001,\ 0.0003,\ 0.0010,\ 0.0032,\ 0.0100,\ 0.0316,\ 0.1000\} \tag{7.11}$$

For matrix BRPCA, ORPCA, ROSL and both versions of RCPD, r ranges as follows

$$r \in \{10,\ 20,\ 30,\ 40,\ 50\} \tag{7.12}$$

For tensor BRPCA, the two versions of tensor IRPCA and RHOSVD, r_1, r_2 and r_3 range as follows

$$r_1 = 48\,\alpha \tag{7.13}$$
$$r_2 = 64\,\alpha \tag{7.14}$$
$$r_3 = 400\,\alpha \tag{7.15}$$
$$\alpha \in \{0.2,\ 0.4,\ 0.6,\ 0.8,\ 1.0\} \tag{7.16}$$

Finally, for all tensor algorithms we set $a_1 = a_2 = a_3 = 1/3$.

Fig. 7.13 shows two example video frames together with their ground truth foreground masks (top row) and the background subtraction achieved by each matrix algorithm (subsequent rows). All matrix algorithms appear to perform excellently, apart from IRPCA via substitution. Similarly, Fig. 7.14 shows the same video frames as segmented by each tensor algorithm. Both versions of RCPD achieve excellent results. However, the results of the rest of the tensor algorithms are less good, as there are still foreground ‘shadows” in the recovered backgrounds.

For an objective numerical comparison, we plot for each algorithm the *Detection Error Tradeoff* curve [60]. The DET curve is obtained by plotting FNR against FPR for various values of T. It is semantically equivalent to the ROC curve but more visually pleasing. Fig. 7.15 and Fig. 7.16 show the DET curves for matrix and tensor methods respectively. The interpretation of the curves is that the closest they are to the origin, the better the performance is. We can see that all matrix methods perform similarly, with the notable exception of IRPCA via substitution. As for tensor algorithms, the DET curves confirm the visual observation that RCPD performs the best whereas IRPCA performs the worst.

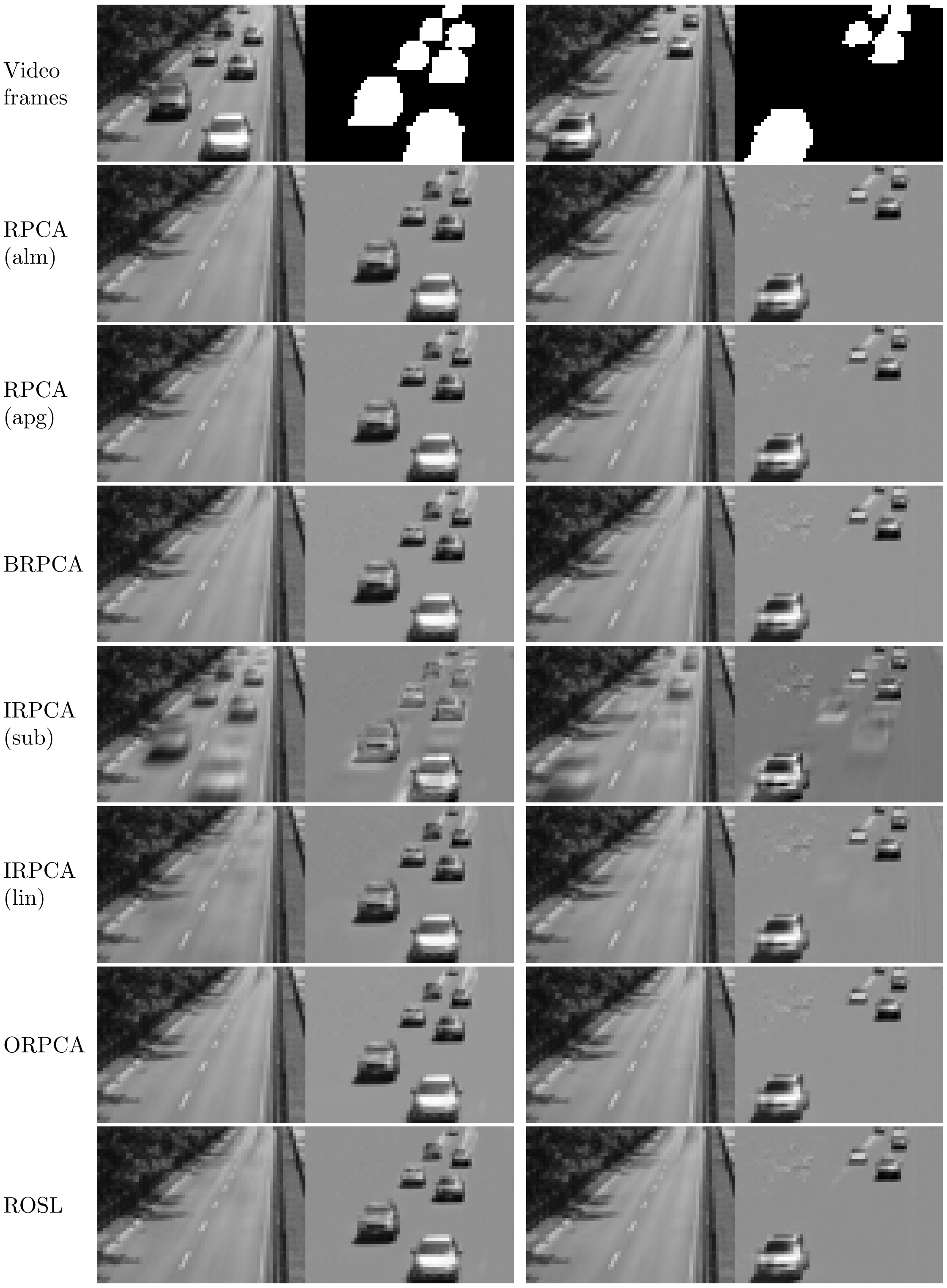


Figure 7.13: Background subtraction using matrix methods. Two sample video frames are shown. Top row shows the original video frames and the ground truth foreground masks. Each consecutive row shows the segmentation achieved by the corresponding matrix algorithm. Note that the foreground (sparse component) is rescaled to $[0, 1]$ for display purposes.

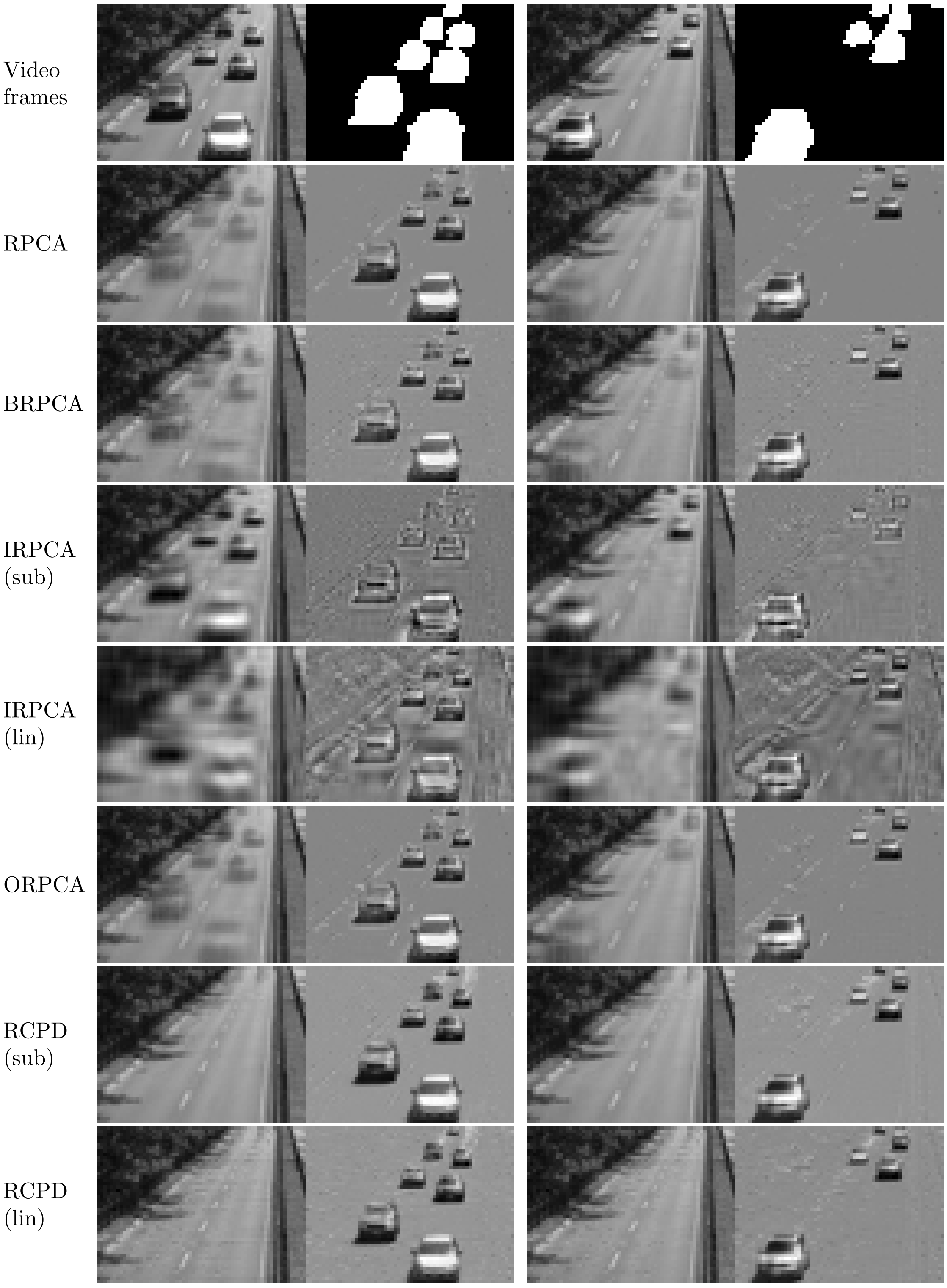


Figure 7.14: Background subtraction using tensor methods. Two sample video frames are shown. Top row shows the original video frames and the ground truth foreground masks. Each consecutive row shows the segmentation achieved by the corresponding tensor algorithm. Note that the foreground (sparse component) is rescaled to $[0, 1]$ for display purposes.

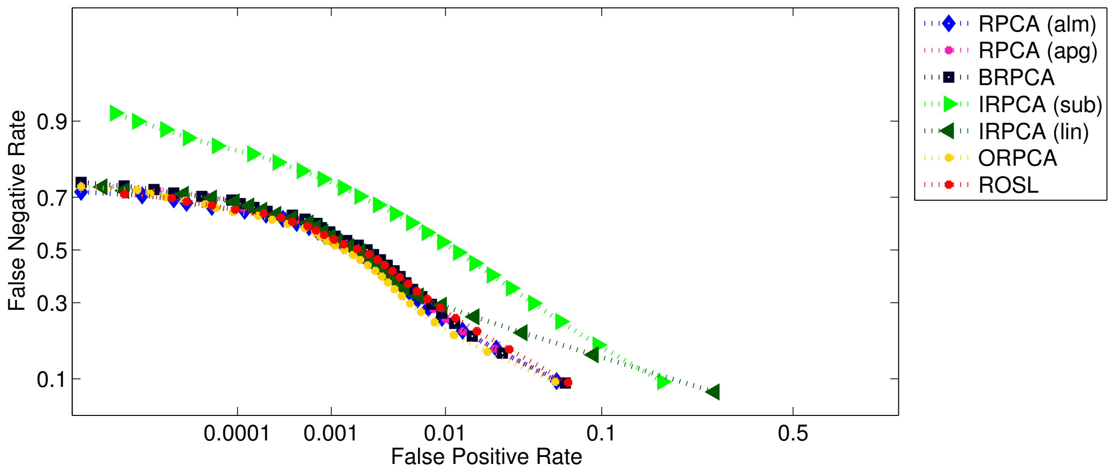


Figure 7.15: Detection Error Trade-off curves showing the performance of matrix algorithms in foreground segmentation. The closer a curve is to the origin, the better the performance. All algorithms exhibit similar performance, except IRPCA via substitution whose performance stands out as the worst.

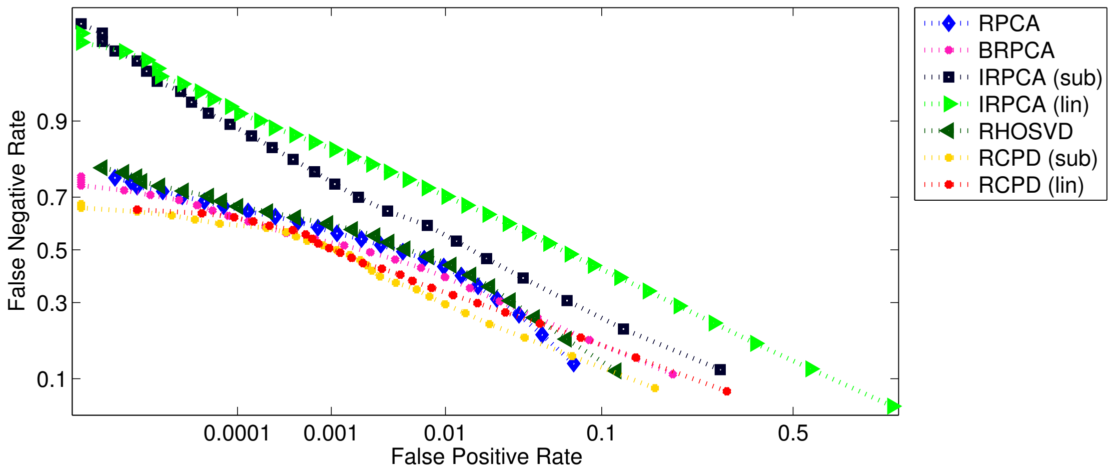


Figure 7.16: Detection Error Trade-off curves showing the performance of tensor algorithms in foreground segmentation. The closer a curve is to the origin, the better the performance. The two versions of RCPD perform the best, whereas the two versions of IRPCA perform the worst.

	Algorithm	**AUC**	**Time [sec]**	**Iterations**	λ	r
Matrices	RPCA (alm)	0.944	13.658	85	0.0100	–
	RPCA (apg)	0.944	30.138	183	0.0100	–
	BRPCA	0.945	3.430	82	0.0100	10
	IRPCA (sub)	0.918	1347.867	92	0.0001	–
	IRPCA (lin)	0.929	935.764	74	0.0100	–
	ORPCA	**0.946**	**5.737**	**88**	**0.0316**	**30**
	ROSL	0.944	16.590	58	0.0100	50
Tensors	RPCA	0.920	52.136	76	0.0100	–
	BRPCA	0.915	19.034	91	0.0316	0.6×
	IRPCA (sub)	0.884	34.748	85	0.0010	0.2×
	IRPCA (lin)	0.814	35.028	83	0.1000	0.2×
	RHOSVD	0.921	52.495	89	0.0100	0.8×
	RCPD (sub)	**0.937**	**37.039**	**82**	**0.0010**	**40**
	RCPD (lin)	0.921	46.206	88	0.1000	50

Table 7.7: Numerical results on foreground segmentation using both matrix and tensor methods. The first numerical column shows the Area Under the Curve (AUC). Higher AUC implies better performance. The algorithms with higher AUC for each category are shown in bold. The best performance for each algorithm is reported, for which the corresponding values of λ and r (if applicable) are shown.

Finally, Table 7.7 shows the AUC for all methods, together with their total execution time and number of iterations. For each algorithm its best performance is reported (as measured by AUC) and the values of λ and r (if applicable) for which it was obtained are shown. Matrix algorithms perform in general better than most tensor algorithms, with the exception of RCPD which performs as good as most matrix algorithms. In terms of time efficiency, matrix BRPCA and ORPCA are particularly fast, with a running time of less than 6 seconds. The slowest are the two versions of matrix IRPCA, with a running time of more than 15 minutes.

7.4 Reconstruction of Whole Missing Images

So far, the experiments on denoising and background subtraction have shown that matrix and tensor methods are comparable. In this last experiment, we will demonstrate the superiority of tensors over matrices in the problem of *reconstructing whole missing images*.

Robust low-rank modelling has been used extensively in missing value recovery, both with matrices (e.g. [16, 42, 50, 89]) and with tensors (e.g. [29, 56]). Nevertheless, in most experiments, the values to be missing were selected *at random* within the data matrix or the data tensor. Instead, in this experiment we will consider that the missing values are *whole data points* (in our case, images), similarly to the experimental setting used in [30]. To the best of our knowledge, it is the first time in the literature that this type of experiment is performed in the context of robust low-rank modelling.

For the purposes of this experiment, we will use part of the CMU Multi-PIE database of face images [35]. The Multi-PIE database consists of face images taken from various subjects, using a wide range of viewpoints, facial expressions and illuminations for each subject. Here, we use 10 subjects, 5 viewpoints ($-30°$, $-15°$, $0°$, $15°$, $30°$), 6 expressions (neutral, surprise, squint, smile, disgust, scream) and 5 illuminations, with a total of 1500 images. All images have been cropped, aligned and resized to a resolution of 40×30 pixels. Some examples of images that were used in the experiments of this section are shown in Fig. 7.17.

For the matrix methods, the data are represented by a matrix $\mathbf{X} \in \mathbb{R}^{1200\times1500}$, where each column is a vectorised image. For the tensor methods, the data are represented by a 6th-order tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{10\times5\times6\times5\times40\times30}$, where the modes correspond to subjects, viewpoints, expressions, illuminations, image rows and image columns respectively. The input data are then formed as follows.

- **Missing images**. With probability p, each of the 1500 images is considered to be *entirely missing*, in which case the corresponding entries in $\mathbf{X}$ or $\boldsymbol{\mathcal{X}}$ are set to zeros. Here we use $p \in \{0.2,\ 0.5,\ 0.8\}$.
- **Observed images**. With probability $1 - p$, the image is considered to be *observed* and is kept unaffected.
- **Binary observation mask**. A matrix $\mathbf{W}$ (or tensor $\boldsymbol{\mathcal{W}}$) is formed with the same size as $\mathbf{X}$ (or, respectively, $\boldsymbol{\mathcal{X}}$), with each entry W indicating whether the corresponding pixel is observed ($W = 1$) or missing ($W = 0$). This mask informs the algorithm about the location of the missing values, as explained in section 5.1.

The purpose of this experiment is to use the robust low-rank modelling algorithms in order to *reconstruct* the missing images, given the rest of the dataset.

Using any matrix method to reconstruct the missing images *fails dramatically*, since all reconstructed pixels turn out to be zero (i.e. black) or very close to zero. As we have shown in section 5.1, robust low-rank modelling—in principle—works by *freely* modifying the unknown entries such that the resulting matrix/tensor is low-rank. For matrices, this works quite well when the missing values are *randomly dispersed* throughout the data, as it has been demonstrated in the literature (see references above). In this experiment, however, the missing values are *entire matrix columns*; the rank of the resulting matrix can trivially be minimised by simply setting the missing columns to zero.

Although such a task is beyond the capabilities of matrix algorithms, tensor algorithms can still achieve respectable results, as we shall demonstrate below. The reason for this is that tensor algorithms can take advantage of the *higher-order structure* in the data. The Multi-PIE dataset is particularly structured, due to the contribution of various factors (subjects, viewpoints, expressions and illuminations). By representing the above factors as different modes and requiring that these modes be low-rank, the entire reconstruction of *previously unseen* images becomes possible.

In the following, we will evaluate the performance of all tensor algorithms in the reconstruction task. Each algorithm is run for the following values of λ

$$\lambda \in \{1,\ 10,\ 1000,\ 1000\} \tag{7.17}$$

and the best result in terms of reconstruction is reported. Since we wish to minimise the rank of the first four modes (i.e. subjects, viewpoints, expressions and illuminations) but not of the last two (i.e. rows and columns), we set $a_1 = a_2 = a_3 = a_4 = 0.25$ and $a_5 = a_6 = 0$. In order to achieve good reconstruction (and since no noise is being considered) the upper bounds for the ranks are set to high values. For BRPCA, both versions of IRPCA and RHOSVD we set

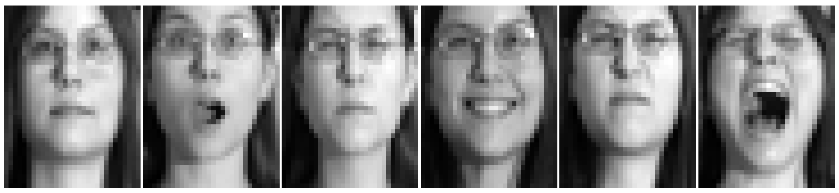
$$(r_1, r_2, r_3, r_4, r_5, r_6) = (10, 5, 6, 5, 40, 30) \tag{7.18}$$

that is, equal to the size of the tensor. For both versions of RCPD we set $r = 40$, that is, equal to the maximum mode size.

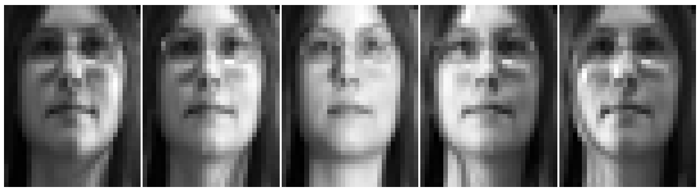
We evaluate the quality of the reconstruction using the *average correlation coefficient* across reconstructed images. Let $\mathbf{a}_i$ be a missing image (represented here in a vectorised form) and $\hat{\mathbf{a}}_i$

(a) The 10 different subjects

(b) The 5 different viewpoints



(c) The 6 different expressions



(d) The 5 different illuminations

Figure 7.17: Examples of images in the CMU Multi-PIE face database [35] that are used for the missing image reconstruction experiment.

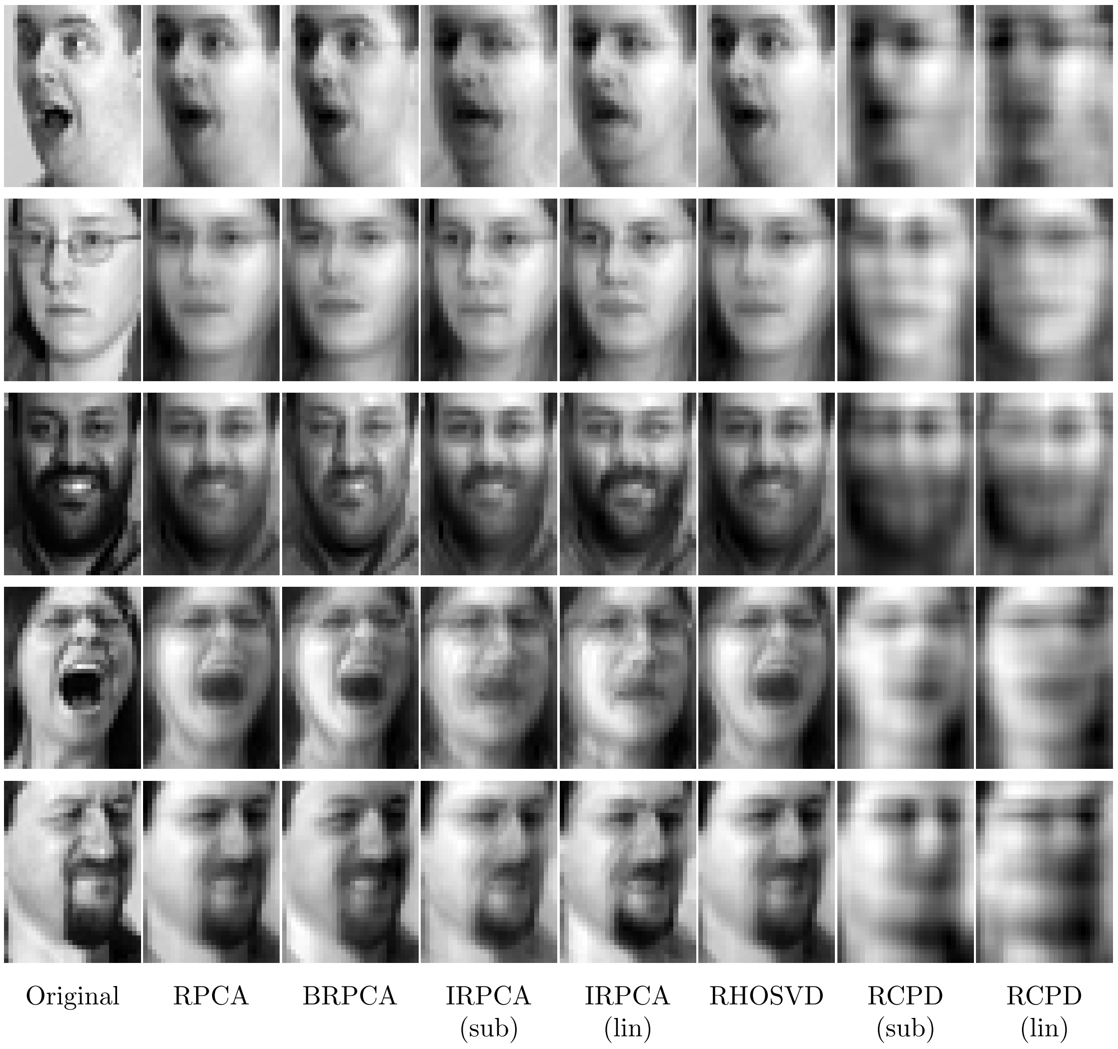


Figure 7.18: Examples of missing images (first column) and their reconstruction by each tensor algorithm. Here, 20% of the images are missing from the dataset. Five images are shown, corresponding to different subjects, expressions and viewpoints.

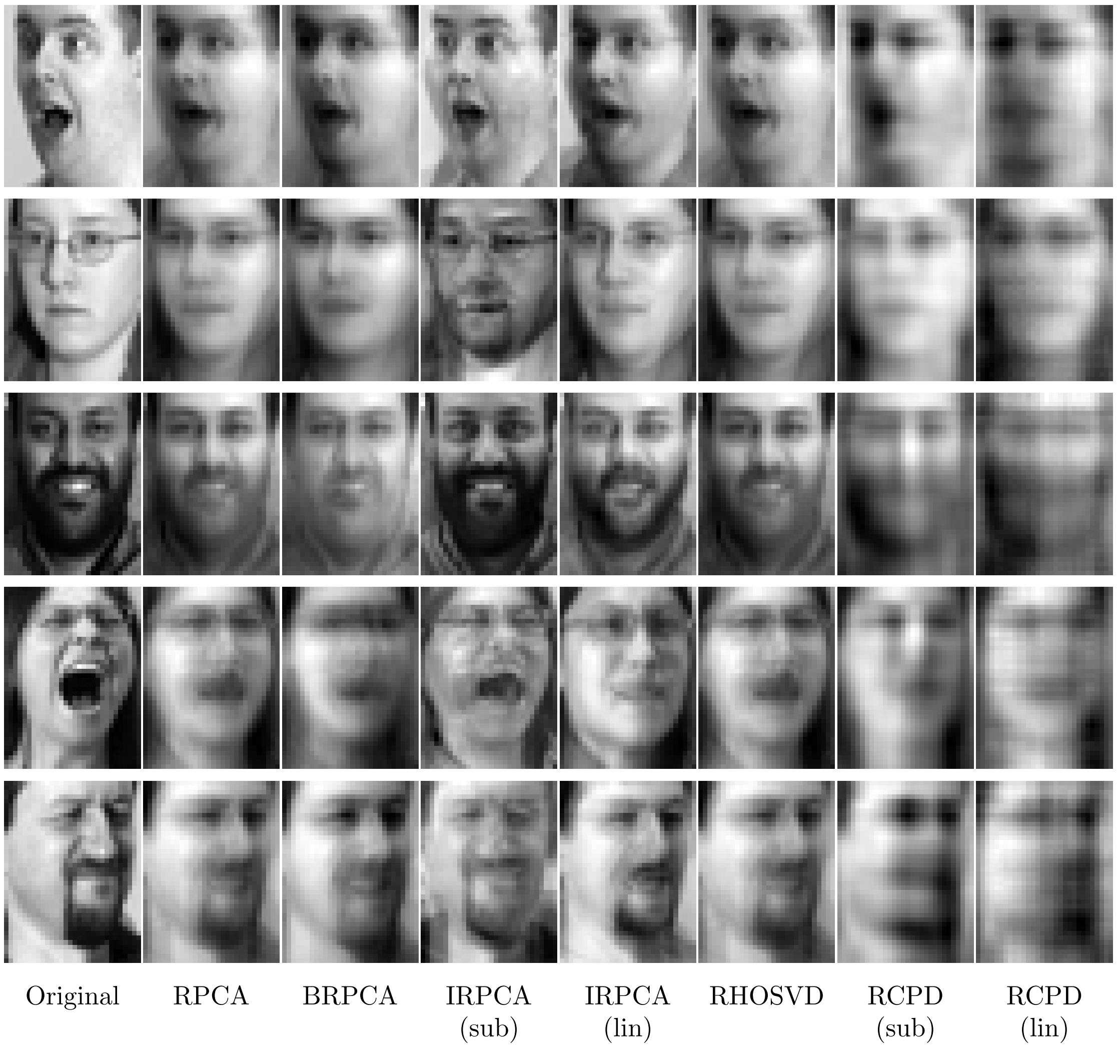


Figure 7.19: Examples of missing images (first column) and their reconstruction by each tensor algorithm. Here, 50% of the images are missing from the dataset. Five images are shown, corresponding to different subjects, expressions and viewpoints.

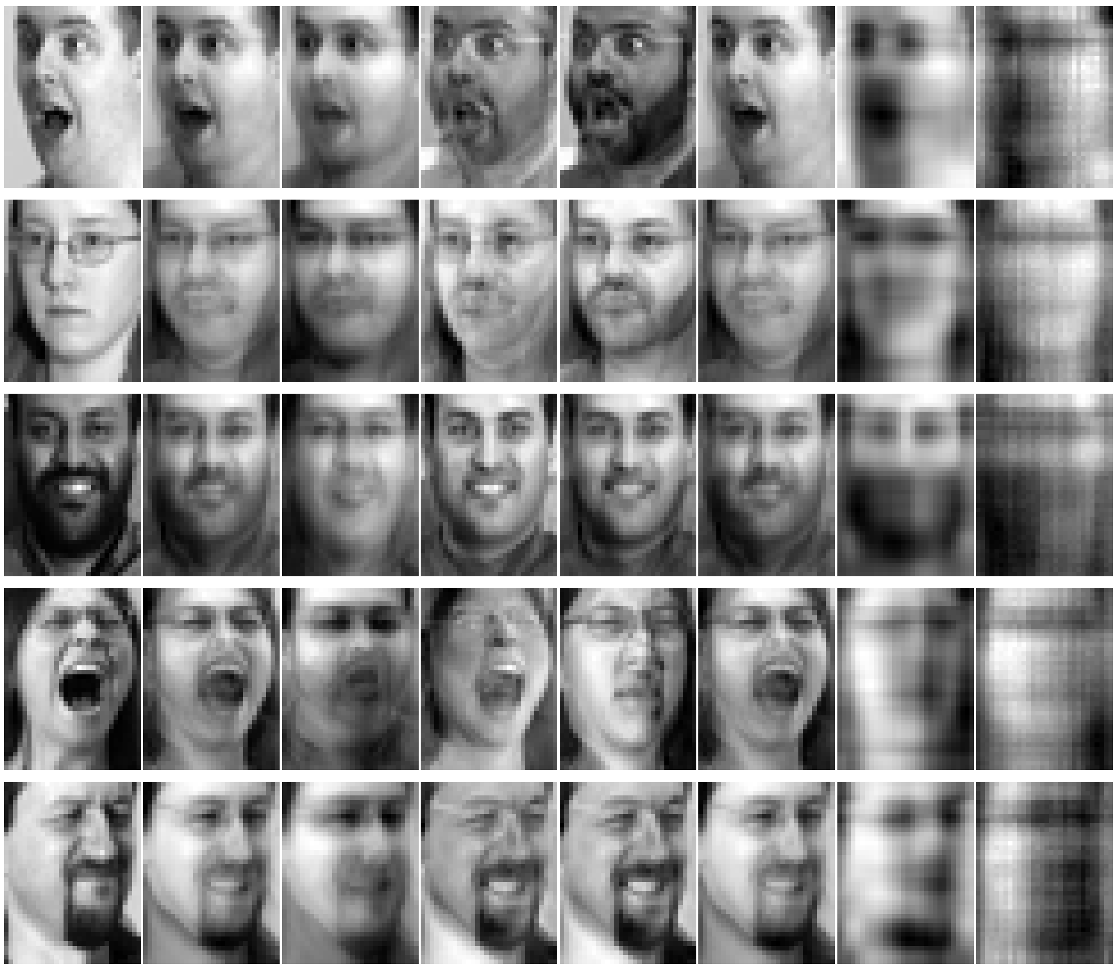


Figure 7.20: Examples of missing images (first column) and their reconstruction by each tensor algorithm. Here, 80% of the images are missing from the dataset. Five images are shown, corresponding to different subjects, expressions and viewpoints.

	Algorithm	Correlation	Time [sec]	Iterations	λ
20% missing	**RPCA**	**0.906**	**90.186**	**98**	**1**
	BRPCA	0.879	63.341	117	100
	IRPCA (sub)	0.835	86.445	84	1
	IRPCA (lin)	0.872	127.349	119	10
	RHOSVD	0.905	206.114	97	1
	RCPD (sub)	0.832	158.370	137	1000
	RCPD (lin)	0.801	139.213	109	10
50% missing	**RPCA**	**0.860**	**98.559**	**99**	**1**
	BRPCA	0.802	62.500	113	1
	IRPCA (sub)	0.590	95.278	93	1
	IRPCA (lin)	0.801	126.892	118	10
	RHOSVD	**0.860**	**216.610**	**99**	**1**
	RCPD (sub)	0.821	130.115	112	10
	RCPD (lin)	0.788	176.313	134	1000
80% missing	**RPCA**	**0.774**	**94.416**	**102**	**1**
	BRPCA	0.692	66.603	121	1000
	IRPCA (sub)	0.576	122.107	119	10
	IRPCA (lin)	0.640	152.437	141	1000
	RHOSVD	**0.774**	**199.691**	**94**	**1**
	RCPD (sub)	0.529	157.299	136	1000
	RCPD (lin)	0.739	180.912	134	1000

Table 7.8: Numerical results for the missing image reconstruction experiment with the Multi-PIE face database. The reconstruction performance is assessed by the average correlation coefficient across reconstructed images. For each missing percentage, the best performing algorithms are shown in boldface. For each algorithm, its best performance in terms of reconstruction is reported, together with the value of λ for which this was achieved.

be its reconstruction, for $i \in \{1, 2, \ldots, N\}$, and assume that both $\mathbf{a}_i$ and $\hat{\mathbf{a}}_i$ have been centred to have zero mean. The (absolute) correlation coefficient c_i between $\mathbf{a}_i$ and $\hat{\mathbf{a}}_i$ is computed as

$$c_i = \frac{|\langle \mathbf{a}_i, \hat{\mathbf{a}}_i \rangle|}{\|\mathbf{a}_i\|_2 \, \|\hat{\mathbf{a}}_i\|_2} \tag{7.19}$$

and its average value across reconstructed images is

$$\bar{c} = \frac{1}{N} \sum_{i=1}^{N} c_i \tag{7.20}$$

By definition, the value of $\bar{c}$ will be between 0 and 1, with a higher value indicating a better reconstruction.

Fig. 7.18, 7.19 and 7.20 show the reconstruction of 5 example missing images from the dataset, corresponding to different subjects, expressions and viewpoints, with 20%, 50% and 80% of the total number of images missing respectively. It is rather impressive that previously unseen images can be reconstructed with often very good results, even if the majority of the images from the dataset is missing. Visually, the best results are given by RPCA and RHOSVD, whereas the worst results are produced by both versions of RCPD.

Finally, Table 7.8 shows the numerical results in terms of average correlation coefficient $\bar{c}$, total execution time and total number of iterations. RPCA and RHOSVD yield the best results and are almost equivalent in terms of reconstruction performance. Both versions of IRPCA and both versions of RCPD have, in general, the poorest performance. BRPCA is in every case the fastest algorithm. Due to the use of high values for the r parameters, which was necessary to achieve a high-quality reconstruction, the efficiency of the factorisation-based methods was compromised. As a result, RPCA appears to be faster than the—otherwise more efficient—RHOSVD and RCPD.

7.5 Overall Evaluation of Algorithms

In this chapter we have evaluated all algorithms in a variety of tasks, both including simulated and real-life data. In general, all algorithms were found to perform respectably. In particular, the following conclusions can be drawn.

(i) **Ability of exact recovery**. Using synthetically generated low-rank data, we were able to evaluate whether the algorithms were capable of *exactly* recovering the low-rank data from sparse gross corruptions. It was found that most algorithms were indeed capable of doing so. The algorithms that failed to do so were IRPCA (both for matrices and tensors) and ROSL. In fact, the ability of exact recovery has been theoretically proven for matrix RPCA [18] and for tensor RPCA [40], which we experimentally verified. The fact that also other algorithms achieved exact recovery, including most of the tensor algorithms, suggest that the theoretical results for RPCA might be extensible to other cases as well.

(ii) **Convergence**. All algorithms converged in all cases. The rate of convergence relied mainly on the underlying optimisation algorithm. In general, three optimisation algorithms were used; APG, ALM and ALM with linearisation. The algorithms that were based on ALM and ALM with linearisation exhibited a similarly fast convergence behaviour. The algorithm based on APG (i.e. matrix RPCA via APG) converged more slowly, even though its rate of convergence is theoretically shown [7] to be $\mathcal{O}\left(k^{-2}\right)$. This is consistent with the findings of [52], where it was also reported that RPCA via ALM is faster than RPCA via APG.

(iii) **Speed**. Since all algorithms—except matrix RPCA via APG—converged with the same rate, the main factor in determining their speed is the computational cost per iteration. Our experimental results confirm our theoretical analysis in sections 3.6.2 (for matrix methods) and 4.6.3 (for tensor methods), where we predicted that the factorisation-based methods (i.e. BRPCA, ORPCA, ROSL, RHOSVD and RCPD) would outperform regularisation-based methods (i.e. RPCA and IRPCA) in terms of computational efficiency.

(iv) **Performance in image analysis tasks**. We evaluated all algorithms in three tasks of practical significance in image analysis and computer vision; image denoising, background subtraction and image reconstruction. Our results show that most algorithms achieved very good results, suggesting that robust low-rank modelling is a competitive candidate in such applications. Certain algorithms achieved impressive results even in cases of significantly heavy corruption. In general, it was observed that factorisation-based algorithms performed better than regularisation-based algorithms. We believe this is due to the increased flexibility of factorisation-based algorithms in the manipulation of ranks by the user. However, this comes at the cost of extra difficulty in tuning. Arguably, the method with the consistently worst performance was IRPCA, both with matrices and tensors.

(v) **Matrix versus tensor methods**. Matrix and tensor methods performed comparably well in the first three experiments (i.e. synthetic data recovering, denoising and background subtraction). Tensor methods in general outperformed matrix methods in image denoising whereas the opposite was true for background subtraction. However, tensor methods were far more superior than matrix methods in the last experiment, i.e. reconstruction of wholly missing images. This suggests that tensor methods are comparably good to matrix methods where both are applicable, however tensor methods have also the potential of further extending the applicability of robust low-rank modelling in cases where matrix methods are no longer applicable.

To conclude, it is difficult to say whether a particular algorithm is better or should be preferred, since this depends on the application under consideration and the practical requirements. Perhaps factorisation methods should be preferred, as they are faster and more flexible, provided the extra difficulty in tuning them can be afforded. Otherwise, RPCA should be preferred, provided that its slower speed is not a concern. Essentially, the final choice is a trade-off between speed and flexibility on the one hand and ease of use on the other hand.

It is easy to say though which algorithms should probably be avoided. There seems to be no reason to use matrix RPCA with APG, as its ALM version is faster and yields the same results. Also, the worst performance was consistently observed to be by IRPCA, both with matrices and tensors. Apart from performance, IRPCA has no computational advantage over other algorithms in terms of speed. Therefore, based on our evaluation and given the better alternatives, we consider matrix RPCA via APG and all versions of IRPCA to be now obsolete.

Chapter 8

Conclusions and Future Work

8.1 Conclusions

Robust low-rank modelling is a general-purpose framework based on two powerful assumptions; that the data are *low-rank* and the corruptions are *sparse*. These assumptions are general enough to cover a wide range of practical scenarios, including several applications in image analysis and computer vision.

The applicability of robust low-rank modelling relies upon the existence of practical methods for its computation. Recent advances in convex non-smooth optimisation have made the existence of such methods a reality. As a result, in recent years a multitude of approaches have been developed that focus on the simultaneous minimisation of the rank and the ℓ_0-norm of suitable matrices.

Part of this thesis was devoted in reviewing the state-of-the art methods for robust low-rank modelling on matrices. Our exposition was based on the realisation that all such methods can be viewed as different instantiations of the same generic problem, employing in each case a different approximation of the rank and the sparsity of suitable matrices. Such approximations involve surrogating the rank and the ℓ_0-norm with their convex envelopes and/or explicitly factorising the low-rank term. Hence, we made a distinction between regularisation-based and factorisation-based methods.

We contributed to the field of robust low-rank modelling by extending several state-of-the-art matrix-only methods to the more general case of *tensors*. Tensors are the multilinear generalisation of matrices and are capable of expressing higher-order structure in the data more faithfully than matrices. The significance of our contribution relies on the fact that we extended the applicability of robust low-rank modelling to real-life cases where the matrix-only methods were inadequate. On the theoretical side, we provided the first formulation of the two most important tensor decompositions (i.e. HOSVD and CP decomposition) within the framework of robust low-rank modelling.

We demonstrated the flexibility of our approach by showing how all methods under our consideration—both those already existing and those that were developed herein—can be easily extended to handle missing values in the data and incorporate more general norms. As a result, we developed a comprehensive and flexible framework for robust low-rank modelling on both matrices and tensors, whose applicability we demonstrated on a variety of real-life computer vision applications.

Finally, by making our implementation of our robust low-rank modelling framework available to the research community as a MATLAB toolbox, we hope to support and accelerate further developments in the area, both on the theoretical and the applications side.

8.2 Future Work

As a starting point for future research in the field of robust low-rank modelling, in this section we identify and propose a list of potential future research directions.

8.2.1 Theoretical Guarantees of Convergence and Optimality

Robust low-rank modelling relies on optimisation techniques, the convergence of which is crucial. Apart from the existing proofs of convergence for the already-established algorithms, our work lacks a theoretical analysis on the convergence of the newly proposed tensor methods. Empirically we observed that fast convergence was always achieved, which suggests that theoretical proofs of convergence may be possible to obtain. Ideally, a theoretical analysis of convergence should also characterise the rate of convergence (which is not known even for several of the already-established matrix algorithms) and the optimality of the result (i.e. local or global).

In addition to the above, it would be of notable significance to extend the theoretical results of [18, 40] regarding the guaranteed optimality of RPCA to other methods as well. Our experiments showed that most algorithms are also capable of exactly recovering low-rank synthetic data, which suggests the plausibility of such extensions. Ideally, such extensions should also provide suggestions for parameter tuning, which would be of great usefulness regarding practical applicability.

8.2.2 Non-Negative Factorisations

The factorisation-based methods described in this thesis are based on reformulating existing matrix or tensor decompositions within the framework of robust low-rank modelling. Matrix and tensor decompositions are important in that they provide insights on the structure of the data. For some applications (such as image analysis) the decomposition components need to have non-negative entries in order to make sense within the application domain. Such constrained decompositions are known as *non-negative matrix factorisations* [47] and *non-negative tensor factorisations* [88]. Reformulating such factorisations within the framework of robust low-rank modelling would be a novel and important extension. Note that the non-negativity is a convex inequality constraint and as such it would not compromise convexity. Some preliminary work was presented in [57], based on projections on the feasible non-negative space.

8.2.3 Other Tensor Decompositions and Low-Rank Models

The tensor decompositions on which this work has focused are the HOSVD and the CP decomposition, since they are probably the two most widely used. However, in the tensor literature, several other decompositions have been proposed—some of which are mentioned in section 2.2.2—which are worth being studied under a robust low-rank modelling framework. Other interesting tensor decompositions include CP with column-orthonormal factor matrices [22] and the newly-proposed tensor-SVD or t-SVD [92].

As a convex relaxation for the tensor rank, in this work we considered a convex combination of the nuclear norms of all matricisations. As we discussed in the beginning of chapter 4, this is indeed the most common choice in the literature. However, recently Mu et al. [62] showed that using the nuclear norm of only one matricisation that is reshaped so as to become as square as possible can achieve recovery from fewer samples. In another approach, in [32, 86] the so-called *mixture model* for describing a low-rank tensor was considered, in which an N^{th}-order low-rank tensor is written as the sum of N tensors, each of which is low-rank along a different mode. The authors argue that the mixture model allows for increased flexibility without the further introduction of parameters that require tuning. Both of these alternative approaches are promising and therefore worth investigating in future research.

8.2.4 Probabilistic Reformulation

An interesting development in robust low-rank modelling would be its probabilistic reformulation. Such a reformulation would provide a principled probabilistic interpretation of the optimisation problems used. More importantly, under a probabilistic framework the regularisation parameters could be treated as random variables and hence be determined automatically (see e.g. the relevant discussion in [93]). To this end, an interesting approach has been proposed by Babacan et al. [2], who reformulated BRPCA as a generative probabilistic model and solved it using variational Bayesian inference. The extension of such works to other methods and/or tensors would be of particular interest.

8.2.5 Novel Optimisation Techniques

This work uses in total three optimisation algorithms; APG, ALM and ALM with linearisation. Currently, ALM is considered state-of-the-art for robust low-rank modelling and is employed by most approaches in the literature. Even though ALM is empirically known to converge quickly, no theoretical description of its rate of convergence is known. Orabona et al. [66] have proposed a different approach for convex non-smooth optimisation to which they refer as *PRoximal Iterative SMoothing Algorithm* or *PRISMA* and which is based on replacing the non-smooth term with its smooth Moreau envelope and proceeding with APG updates. They prove that PRISMA's rate of convergence is $\mathcal{O}\left(L_1 k^{-2} + L_2 k^{-1} \log k\right)$, with L_1 and L_2 being certain Lipschitz constants, and apply it to solve matrix RPCA, in which they report faster convergence than ALM. This suggests that employing PRISMA in other matrix or tensor methods may be a promising development towards improving their speed.

A drawback of ALM with linearisation is the lack of a theoretical guarantee of convergence when the blocks of variables to be updated are more than two (even though in our experiments we always found it to converge). To address this problem, Liu et al. [57] showed that a certain reformulation of ALM with linearisation based on parallel updates is guaranteed to converge for any number of blocks of variables. Even though their proof is for convex problems only, it is worth investigating whether applying their ideas on the methods described in this work (especially in the tensor case, where the blocks of variables may be several) can improve their convergence.

Finally, as we saw in section 3.6.2, matrix methods have, in principle, a cost of $\mathcal{O}\left(n^3\right)$ per iteration, for an input matrix of size $n \times n$, which can be reduced to $\mathcal{O}\left(rn^2\right)$ using factorisation with $r \leq n$. Although this is a significant improvement compared to interior point methods that have a cost of $\mathcal{O}\left(n^6\right)$ per iteration, it still may be prohibitive for very large datasets. In a recent work, Mu et al. [63] showed that combining Frank-Wolfe and proximal optimisation techniques for solving matrix RPCA can yield *linear* cost per iteration. The drawback of their method is its convergence rate of $\mathcal{O}\left(k^{-1}\right)$, which is suboptimal compared to $\mathcal{O}\left(k^{-2}\right)$ achievable by APG. However, their method is worth of further investigation, and possible extension to factorisation-based and/or tensor methods, especially for very large datasets.

8.2.6 Real-Time Applications

So far, the computational cost of robust low-rank modelling has prevented it from being used in real-time applications, even though this would be an exciting practical development. As a potential application, consider the real-time subtraction of the background in video frames at real time. This could be used for e.g. real-time motion detection and tracking. Based on our experimental results, the most promising candidates for real-time performance are the factorisation-based BRPCA and ORPCA, due to their efficiency compared to other methods. However, applying them at real time would require a more efficient implementation choice than MATLAB, such as C or C++ combined with the high performance capabilities of graphics processing units.

Appendix A

Notations and Symbols

x, y, z, X, Y, Z	Scalars (lowercase or uppercase letters)
$\mathbf{x}, \mathbf{y}, \mathbf{z}$	Vectors (lowercase bold letters)
$\mathbf{X}, \mathbf{Y}, \mathbf{Z}$	Matrices (uppercase bold letters)
$\boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{Y}}, \boldsymbol{\mathcal{Z}}$	Tensors of order $N \geq 3$ (uppercase calligraphic bold letters)
x_i	Entry at position i of vector $\mathbf{x}$
X_{ij}	Entry at position (i, j) of matrix $\mathbf{X}$
$X_{i_1 i_2 \cdots i_N}$	Entry at position $(i_1, i_2, \ldots, i_N)$ of N^{th}-order tensor $\boldsymbol{\mathcal{X}}$
$\mathbf{I}$	Identity matrix of appropriate dimensions
$\mathbf{x}^T, \mathbf{X}^T$	Transpose of vector $\mathbf{x}$ and matrix $\mathbf{X}$
$\{\ldots\}$	A set, depending on context
$[x, y]$	Closed interval from x to y
$\mathbb{R}$	The set of real numbers
$\mathbb{R}^n$	The set of all real vectors of length n
$\mathbb{R}^{m \times n}$	The set of all real matrices of size $m \times n$
$\mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$	The set of all N^{th}-order real tensors of size $I_1 \times I_2 \times \cdots \times I_N$
$\lvert\cdot\rvert$	Absolute value of a real number
$\lVert\cdot\rVert$	Norm (in general)
$\lVert\cdot\rVert_{S_p}$	Schatten p-norm (matrices only)
$\lVert\cdot\rVert_*$	Nuclear norm (matrices only)
$\lVert\cdot\rVert_p$	Elementwise ℓ_p-norm
$\lVert\cdot\rVert_F$	Frobenius norm
$\lVert\cdot\rVert_0$	ℓ_0-norm
$\langle\cdot,\cdot\rangle$	Inner product
$\operatorname{rank}(\cdot)$	Matrix or tensor rank
$\operatorname{rank}_n(\cdot)$	Tensor n-rank
$\operatorname{vec}(\boldsymbol{\mathcal{X}})$	Vectorisation of tensor $\boldsymbol{\mathcal{X}}$
$\mathbf{X}_{[n]}$	n-mode matricisation of tensor $\boldsymbol{\mathcal{X}}$
$\mathbf{X} \otimes \mathbf{Y}$	Kronecker product between matrices $\mathbf{X}$ and $\mathbf{Y}$

$\bigotimes_{i=1}^{N} \mathbf{U}_i$	Shorthand for $\mathbf{U}_1 \otimes \mathbf{U}_2 \otimes \cdots \otimes \mathbf{U}_N$
$\mathbf{X} \odot \mathbf{Y}$	Khatri-Rao product between matrices $\mathbf{X}$ and $\mathbf{Y}$
$\bigodot_{i=1}^{N} \mathbf{U}_i$	Shorthand for $\mathbf{U}_1 \odot \mathbf{U}_2 \odot \cdots \odot \mathbf{U}_N$
$\boldsymbol{\mathcal{X}} \times_n \mathbf{U}$	n-mode product between tensor $\boldsymbol{\mathcal{X}}$ and matrix $\mathbf{U}$
$\boldsymbol{\mathcal{X}} \times_{i=1}^{N} \mathbf{U}_i$	Shorthand for $\boldsymbol{\mathcal{X}} \times_1 \mathbf{U}_1 \times_2 \mathbf{U}_2 \times_3 \cdots \times_N \mathbf{U}_N$
$\mathbf{x} \circ \mathbf{y}$	Outer product between vectors $\mathbf{x}$ and $\mathbf{y}$
$\mathbf{U}_1 \circ \mathbf{U}_2 \circ \cdots \circ \mathbf{U}_N$	CP decomposition with factor matrices $\{\mathbf{U}_i\}$
$\nabla f(\mathbf{x}), \frac{\partial f}{\partial \mathbf{x}}$	Gradient of function f with respect to $\mathbf{x}$
$\mathcal{C}_f$	Convex envelope of function f
$\mathcal{P}_f\{\cdot\}$	Proximal operator of function f
$\mathcal{S}_a\{\cdot\}$	Shrinkage operator
$\mathcal{S}_{a,\Omega}\{\cdot\}$	Selective shrinkage
$\mathcal{S}_a^p\{\cdot\}$	Generalised shrinkage
$\mathcal{S}_{a,\Omega}^p\{\cdot\}$	Generalised selective shrinkage
$\mathcal{D}_a\{\cdot\}$	Singular value thresholding (matrices only)
$\mathcal{D}_a^p\{\cdot\}$	Generalised singular value thresholding (matrices only)
$\mathcal{M}_a\{\cdot\}$	Magnitude shrinkage
$\pi_\Omega(\cdot)$	Sampling operator
$\mathcal{O}(\cdot)$	Asymptotic “big O” notation
$\min(x_1, x_2, \ldots, x_N)$	The smallest among scalars $\{x_i\}$
$\min_{\mathbf{x}} f(\mathbf{x})$	The minimum value of real function f with respect to $\mathbf{x}$
$\arg\min_{\mathbf{x}} f(\mathbf{x})$	The minimiser of real function f with respect to $\mathbf{x}$

Appendix B

List of Abbreviations

ALM	Augmented Lagrange Multipliers
APG	Accelerated Proximal Gradient
AUC	Area Under the Curve
BCD	Block Coordinate Descent
BRPCA	Bilinear Robust PCA
CANDECOMP	CANonical DECOMPosition
CP	CANDECOMP/PARAFAC
DET	Detection Error Trade-off
GSRPCA	Generalised Scalable Robust PCA
HOSVD	Higher Order SVD
IRPCA	Inductive Robust PCA
KKT	Karush-Kuhn-Tucker
LADM	Linearised Alternating Direction Method
LRR	Low-Rank Representation
ORPCA	Orthonormal Robust PCA
PARAFAC	PARallel FACtors
PCA	Principal Component Analysis
PRISMA	PRoximal Iterative SMoothing Algorithm
RCPD	Robust CP Decomposition
RHOSVD	Robust Higher Order SVD
ROC	Receiver Operating Characteristic
ROSL	Robust Orthonormal Subspace Learning
RPCA	Robust PCA
SVD	Singular Value Decomposition

Bibliography

[1] E. Acar and B. Yener. Unsupervised multiway data analysis: A literature survey. *Knowledge and Data Engineering, IEEE Transactions on*, 21(1):6–20, Jan. 2009.

[2] S. D. Babacan, M. Luessi, R. Molina, and A. K. Katsaggelos. Sparse bayesian methods for low-rank matrix estimation. *Signal Processing, IEEE Transactions on*, 60(8):3964–3977, Aug. 2012.

[3] B. W. Bader and T. G. Kolda. Algorithm 862: MATLAB tensor classes for fast algorithm prototyping. *ACM Transactions on Mathematical Software*, 32(4):635–653, Dec. 2006.

[4] B. W. Bader, T. G. Kolda, et al. MATLAB Tensor Toolbox Version 2.5. Available online, Jan. 2012. URL `http://www.sandia.gov/~tgkolda/TensorToolbox/`.

[5] B.-K. Bao, G. Liu, C. Xu, and S. Yan. Inductive Robust Principal Component Analysis. *Image Processing, IEEE Transactions on*, 21(8):3794–3800, Aug. 2012.

[6] R. Basri and D. Jacobs. Lambertian reflectance and linear subspaces. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 25(2):218–233, Feb. 2003.

[7] A. Beck and M. Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM J. Img. Sci.*, 2(1):183–202, Mar. 2009.

[8] S. Bengio, F. Pereira, Y. Singer, and D. Strelow. Group Sparse Coding. In *Advances in Neural Information Processing Systems*, 2009.

[9] M. Bertalmio, G. Sapiro, V. Caselles, and C. Ballester. Image inpainting. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, pages 417–424, 2000.

[10] D. P. Bertsekas. *Constrained Optimization and Lagrange Multiplier Methods*. Athena scientific series in optimization and neural computation. Athena Scientific, 1996.

[11] D. P. Bertsekas. *Nonlinear Programming*. Athena Scientific, 2nd edition, Sept. 1999.

[12] T. Bouwmans and E. H. Zahzah. Robust PCA via principal component pursuit: A review for a comparative evaluation in video surveillance. *Computer Vision and Image Understanding*, 122:22–34, 2014.

[13] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Found. Trends Mach. Learn.*, 3(1):1–122, Jan. 2011.

[14] R. S. Cabral, F. De la Torre, J. P. Costeira, and A. Bernardino. Unifying nuclear norm and bilinear factorization approaches for low-rank matrix decomposition. In *Computer Vision (ICCV), 2013 IEEE International Conference on*, pages 2488–2495, Dec. 2013.

[15] R. S. Cabral, F. De la Torre, J. P. Costeira, and A. Bernardino. Matrix completion for weakly-supervised multi-label image classification. *IEEE Transactions Pattern Analysis and Machine Intelligence (PAMI)*, 2014.

[16] J.-F. Cai, E. J. Candès, and Z. Shen. A singular value thresholding algorithm for matrix completion. *SIAM J. on Optimization*, 20(4):1956–1982, Mar. 2010.

[17] E. J. Candès and B. Recht. Exact matrix completion via convex optimization. *Commun. ACM*, 55(6):111–119, June 2012.

[18] E. J. Candès, X. Li, Y. Ma, and J. Wright. Robust Principal Component Analysis? *J. ACM*, 58(3):11:1–11:37, June 2011.

[19] J. D. Carroll and J.-J. Chang. Analysis of individual differences in multidimensional scaling via an n-way generalization of "Eckart-Young" decomposition. *Psychometrika*, 35(3):283–319, 1970.

[20] T. F. Chan. An improved algorithm for computing the singular value decomposition. *ACM Trans. Math. Softw.*, 8(1):72–83, Mar. 1982.

[21] C.-F. Chen, C.-P. Wei, and Y.-C. Wang. Low-rank matrix recovery with structural incoherence for robust face recognition. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 2618–2625, June 2012.

[22] J. Chen and Y. Saad. On the tensor SVD and the optimal low rank orthogonal approximation of tensors. *SIAM J. Matrix Anal. Appl.*, 30(4):1709–1734, Jan. 2009.

[23] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Third Edition.* The MIT Press, 3rd edition, 2009.

[24] D. Donoho. For most large underdetermined systems of equations, the minimal ℓ_1-norm near-solution approximates the sparsest near-solution. *Communications on Pure and Applied Mathematics*, 59(7):907–934, 2006.

[25] L. Eldén and B. Savas. A Newton-Grassmann method for computing the best multilinear rank-(r_1, r_2, r_3) approximation of a tensor. *SIAM Journal on Matrix Analysis and Applications*, 31(2):248–271, 2009.

[26] N. M. Faber, R. Bro, and P. K. Hopke. Recent developments in CANDECOMP/PARAFAC algorithms: a critical review. *Chemometrics and Intelligent Laboratory Systems*, 65(1): 119–137, 2003.

[27] T. Fawcett. An introduction to ROC analysis. *Pattern Recogn. Lett.*, 27(8):861–874, June 2006.

[28] M. Fazel. *Matrix Rank Minimization with Applications.* PhD thesis, Dept. Electrical Engineering, Stanford University, CA, USA, 2002.

[29] S. Gandy, B. Recht, and I. Yamada. Tensor completion and low-n-rank tensor recovery via convex optimization. *Inverse Problems*, 27(2):025010, 2011.

[30] X. Geng, K. Smith-Miles, Z.-H. Zhou, and L. Wang. Face image modeling by multilinear subspace analysis with missing values. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 41(3):881–892, June 2011.

[31] A. S. Georghiades, P. N. Belhumeur, and D. J. Kriegman. From few to many: Illumination cone models for face recognition under variable lighting and pose. *IEEE Trans. Pattern Anal. Mach. Intelligence*, pages 643–660.

[32] D. Goldfarb and Z. Qin. Robust low-rank tensor recovery: Models and algorithms. *SIAM Journal on Matrix Analysis and Applications*, 35(1):225–253, 2014.

[33] G. H. Golub and C. F. Van Loan. *Matrix Computations (4th Ed.)*. Johns Hopkins University Press, Baltimore, MD, USA, 2013.

[34] N. Goyette, P. Jodoin, F. Porikli, J. Konrad, and P. Ishwar. Changedetection.net: A new change detection benchmark dataset. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2012 IEEE Computer Society Conference on*, pages 1–8, June 2012.

[35] R. Gross, I. Matthews, J. Cohn, T. Kanade, and S. Baker. Multi-PIE. *Image Vision Comput.*, 28(5):807–813, May 2010.

[36] R. A. Harshman. Foundations of the PARAFAC procedure: Models and conditions for an "explanatory" multi-modal factor analysis. *UCLA Working Papers in Phonetics*, 16:1–84, 1970.

[37] J. Håstad. Tensor rank is NP-complete. *J. Algorithms*, 11(4):644–654, Dec. 1990.

[38] R. He, T. Tan, and L. Wang. Robust recovery of corrupted low-rank matrix by implicit regularizers. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 36(4):770–783, Apr. 2014.

[39] H. Hotelling. Analysis of a complex of statistical variables into principal components. *J. Educ. Psych.*, 24, 1933.

[40] B. Huang, C. Mu, D. Goldfarb, and J. Wright. Provable low-rank tensor recovery. Available online, 2014. URL `http://www.optimization-online.org/DB_FILE/2014/02/4252.pdf`.

[41] P. J. Huber, J. Wiley, and W. InterScience. *Robust statistics*. Wiley New York, 1981.

[42] H. Ji, C. Liu, Z. Shen, and Y. Xu. Robust video denoising using low rank matrix completion. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 1791–1798, June 2010.

[43] I. T. Jolliffe. *Principal Component Analysis*. Springer, second edition, Oct. 2002.

[44] T. Kolda and B. Bader. Tensor decompositions and applications. *SIAM Review*, 51(3): 455–500, 2009.

[45] Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, Aug. 2009.

[46] H. W. Kuhn and A. W. Tucker. Nonlinear programming. In *Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability*, pages 481–492. University of California Press, 1951.

[47] D. D. Lee and H. S. Seung. Algorithms for non-negative matrix factorization. In *Advances in Neural Information Processing Systems 13*, pages 556–562. 2001.

[48] K. C. Lee, J. Ho, and D. Kriegman. Acquiring linear subspaces for face recognition under variable lighting. *IEEE Trans. Pattern Anal. Mach. Intelligence*, pages 684–698.

[49] Y. Li, J. Yan, Y. Zhou, and J. Yang. Optimum subspace learning and error correction for tensors. In *Computer Vision - ECCV 2010*, volume 6313 of *Lecture Notes in Computer Science*, pages 790–803. Springer Berlin Heidelberg, 2010.

[50] X. Liang, X. Ren, Z. Zhang, and Y. Ma. Repairing sparse low-rank texture. In *Proceedings of the 12th European Conference on Computer Vision - Volume Part V*, ECCV'12, pages 482–495, 2012.

[51] Z. Lin, A. Ganesh, J. Wright, L. Wu, M. Chen, and Y. Ma. Fast convex optimization algorithms for exact recovery of a corrupted low-rank matrix. In *In Intl. Workshop on Comp. Adv. in Multi-Sensor Adapt. Processing, Aruba, Dutch Antilles*, 2009.

[52] Z. Lin, M. Chen, and Y. Ma. The Augmented Lagrange Multiplier Method for Exact Recovery of Corrupted Low-Rank Matrices. arXiv, Mar. 2011.

[53] Z. Lin, R. Liu, and Z. Su. Linearized alternating direction method with adaptive penalty for low-rank representation. In *Advances in Neural Information Processing Systems 24*, pages 612–620. 2011.

[54] G. Liu and S. Yan. Active subspace: Toward scalable low-rank learning. *Neural Comput.*, 24(12):3371–3394, Dec. 2012.

[55] G. Liu, Z. Lin, S. Yan, J. Sun, Y. Yu, and Y. Ma. Robust recovery of subspace structures by low-rank representation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 35(1):171–184, 2013.

[56] J. Liu, P. Musialski, P. Wonka, and J. Ye. Tensor completion for estimating missing values in visual data. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 35(1): 208–220, Jan. 2013.

[57] R. Liu, Z. Lin, and Z. Su. Linearized alternating direction method with parallel splitting and adaptive penalty for separable convex programs in machine learning. *arXiv preprint arXiv:1310.5035*, 2013.

[58] H. Lu, K. N. Plataniotis, and A. N. Venetsanopoulos. MPCA: Multilinear Principal Component Analysis of Tensor Objects. *IEEE Trans. Neural Networks*, 19(1):18–39, Jan. 2008.

[59] D. Luenberger and Y. Ye. *Linear and Nonlinear Programming.* International Series in Operations Research & Management Science. Springer, 2008.

[60] A. F. Martin, G. R. Doddington, T. Kamm, M. Ordowski, and M. A. Przybocki. The DET curve in assessment of detection task performance. In *EUROSPEECH*. ISCA, 1997.

[61] MATLAB. *version 8.1.0 (R2013a).* The MathWorks Inc., Natick, Massachusetts, 2013.

[62] C. Mu, B. Huang, J. Wright, and D. Goldfarb. Square Deal: Lower Bounds and Improved Relaxations for Tensor Recovery. In *International Conference on Machine Learning (ICML 2014)*, June 2014.

[63] C. Mu, Y. Zhang, J. Wright, and D. Goldfarb. Scalable robust matrix recovery: Frank-Wolfe meets proximal methods. *ArXiv e-prints*, Mar. 2014.

[64] Y. Nesterov. A method of solving a convex programming problem with convergence rate $\mathcal{O}\left(1/k^2\right)$. *Soviet Mathematics Doklady*, 27(2):372–376, 1983.

[65] F. Nie, H. Wang, H. Huang, and C. Ding. Joint schatten p-norm and ℓ_p-norm robust matrix completion for missing value recovery. *Knowledge and Information Systems*, pages 1–20, 2013.

[66] F. Orabona, A. Argyriou, and N. Srebro. PRISMA: PRoximal Iterative SMoothing Algorithm. *ArXiv e-prints*, June 2012.

[67] O. Oreifej, X. Li, and M. Shah. Simultaneous video stabilization and moving object detection in turbulence. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 35(2): 450–462, Feb. 2013.

[68] Y. Panagakis, M. Nicolaou, S. Zafeiriou, and M. Pantic. Robust canonical time warping for the alignment of grossly corrupted sequences. In *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, pages 540–547, June 2013.

[69] G. Papamakarios, Y. Panagakis, and S. Zafeiriou. Generalised Scalable Robust Principal Component Analysis. In *Proceedings of the British Machine Vision Conference*, Sept. 2014.

[70] N. Parikh and S. Boyd. Proximal algorithms. *Foundations and Trends in Optimization*, 1 (3), 2014.

[71] K. Pearson. On lines and planes of closest fit to systems of points in space. *Philosophical Magazine*, 2(6):559–572, 1901.

[72] Y. Peng, A. Ganesh, J. Wright, W. Xu, and Y. Ma. RASL: Robust alignment by sparse and low-rank decomposition for linearly correlated images. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 34(11):2233–2246, Nov. 2012.

[73] B. Recht, M. Fazel, and P. Parrilo. Guaranteed minimum-rank solutions of linear matrix equations via nuclear norm minimization. *SIAM Review*, 52(3):471–501, 2010.

[74] C. Sagonas, Y. Panagakis, S. Zafeiriou, and M. Pantic. RAPS: Robust and Efficient Automatic Construction of Person-Specific Deformable Models. In *IEEE Int. Conf. on Computer Vision and Pattern Recognition (CVPR 2014)*, June 2014.

[75] X. Shu, F. Porikli, and N. Ahuja. Robust Orthonormal Subspace Learning: Efficient recovery of corrupted low-rank matrices. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, June 2014.

[76] M. Signoretto, L. De Lathauwer, and J. A. K. Suykens. Nuclear Norms for Tensors and Their Use for Convex Multilinear Estimation. Technical report, ESAT-SISTA, K. U. Leuven (Leuven, Belgium), 2010.

[77] L. Sirovich and M. Kirby. Low-Dimensional Procedure for the Characterization of Human Faces. *Journal of the Optical Society of America A*, 4(3):519–524, 1987.

[78] G. Strang. *Linear Algebra and Its Applications.* Thomson Brooks/Cole Cengage learning, 2006. ISBN 9780534422004.

[79] G. Tomasi and R. Bro. A comparison of algorithms for fitting the PARAFAC model. *Computational Statistics & Data Analysis*, 50(7):1700–1734, 2006.

[80] L. R. Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31 (3):279–311, 1966.

[81] M. Turk and A. Pentland. Face recognition using eigenfaces. In *Computer Vision and Pattern Recognition, 1991. Proceedings CVPR '91., IEEE Computer Society Conference on*, pages 586–591, June 1991.

[82] M. Vasilescu and D. Terzopoulos. Multilinear analysis of image ensembles: TensorFaces. In *Computer Vision - ECCV 2002*, volume 2350 of *Lecture Notes in Computer Science*, pages 447–460. Springer Berlin Heidelberg, 2002.

[83] R. Vidal. Subspace clustering. *Signal Processing Magazine, IEEE*, 28(2):52–68, Mar. 2011.

[84] J. Wright, A. Ganesh, S. Rao, Y. Peng, and Y. Ma. Robust principal component analysis: Exact recovery of corrupted low-rank matrices via convex optimization. In *Advances in Neural Information Processing Systems 22*, pages 2080–2088. 2009.

[85] L. Yang, Z.-H. Huang, and X. Shi. A fixed point iterative method for low n-rank tensor pursuit. *Signal Processing, IEEE Transactions on*, 61(11):2952–2962, June 2013.

[86] Y. Yang, Y. Feng, and J. A. K. Suykens. Robust low rank tensor recovery with regularized redescending m-estimator. 2014.

[87] X. Yuan and J. Yang. Sparse and low-rank matrix decomposition via alternating direction methods. Technical report, 2009.

[88] S. Zafeiriou. Algorithms for nonnegative tensor factorization. In *Tensors in Image Processing and Computer Vision*, Advances in Pattern Recognition, pages 105–124. Springer London, 2009.

[89] D. Zhang, Y. Hu, J. Ye, X. Li, and X. He. Matrix completion by truncated nuclear norm regularization. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 2192–2199, June 2012.

[90] X. Zhang, D. Wang, Z. Zhou, and Y. Ma. Simultaneous rectification and alignment via robust recovery of low-rank tensors. In *Advances in Neural Information Processing Systems*, pages 1637–1645. Curran Associates, Inc., 2013.

[91] Z. Zhang, A. Ganesh, X. Liang, and Y. Ma. TILT: Transform Invariant Low-Rank Textures. *International Journal of Computer Vision*, 99(1):1–24, 2012.

[92] Z. Zhang, G. Ely, S. Aeron, N. Hao, and M. Kilmer. Novel methods for multilinear data completion and de-noising based on tensor-SVD. In *IEEE Int. Conf. on Computer Vision and Pattern Recognition (CVPR 2014)*, June 2014.

[93] X. Zhou, C. Yang, H. Zhao, and W. Yu. Low-rank modeling and its applications in image analysis. *arXiv preprint arXiv:1401.3409*, 2014.

[94] H. Zou, T. Hastie, and R. Tibshirani. Sparse principal component analysis. *Journal of Computational and Graphical Statistics*, 15(2):265–286, 2006.

Index